DISEÑO DE BASE DE DATOS 1985.

Fecha	Tema	Horario	Profesor
Del 17 de mayo al 15 de junio	INTRODUCCION	Viernes de 17 a 21 h Sábados de 9 a 14 h	ING. DANIEL RI ZERTUCHE

Los Datos como un recurso de la empresa

El administrador de la Base de Datos.

Diccionario de Datos.

ANALISIS DE REQUERIMIENTOS

Funciones Operativas
Funciones de Control y Planeación

DISEÑO CONCEPTUAL

Modelos de datos
Análisis de entidades
Síntesis de atributos

DISEÑO LOGICO

Formulación del esquema
Formulación de los subesquemas
Evaluación del esquema

DISEÑO FISICO

Métodos de acceso
Diseño de registros y agrupamiento
Diseño de métodos de acceso
Seguridad e integridad
Evaluación del desempeño

CASOS PRACTICOS

'edcs.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

DISEÑO DE BASES DE DATOS

A R T I C U L O

MANAGEMENT MISINFORMATION SYSTEMS

EXPOSITOR:
Ing. Daniel Ríos Zertuche

MAYO, 1985

MANAGEMENT SCIENCE
Vol. 14, No. 4, December, 1967
Printed in U.S.A.

6

MANAGEMENT MISINFORMATION SYSTEMS*

RUSSELL L. ACKOFF

University of Pennsylvania

Five assumptions commonly made by designers of management information systems are identified. It is argued that these are not justified in many (if not most) cases and hence lead to major deficiencies in the resulting systems. These assumptions are: (1) the critical deficiency under which most managers operate is the lack of relevant information, (2) the manager needs the information he wants, (3) if a manager has the information he needs his decision making will improve, (4) better communication between managers improves organizational performance, and (5) a manager does not have to understand how his information system works, only how to use it. To overcome these assumptions and the deficiencies which result from them, a management information system should be imbedded in a management control system. A procedure for designing such a system is proposed and an example is given of the type of control system which it produces.

The growing preoccupation of operations researchers and management scientists with Management Information Systems (MIS's) is apparent. In fact, for some the design of such systems has almost become synonymous with operations research or management science. Enthusiasm for such systems is understandable: it involves the researcher in a romantic relationship with the most glamorous instrument of our time, the computer. Such enthusiasm is understandable but, nevertheless, some of the excesses to which it has led are not excusable.

Contrary to the impression produced by the growing literature, few computerized management information systems have been put into operation. Of those I've seen that have been implemented, most have not matched expectations and some have been outright failures. I believe that these near- and far-misses could have been avoided if certain false (and usually implicit) assumptions on which many such systems have been erected had not been made.

There seem to be five common and erroneous assumptions underlying the design of most MIS's, each of which I will consider. After doing so I will outline an MIS design procedure which avoids these assumptions.

Give Them More

Most MIS's are designed on the assumption that the critical deficiency under which most managers operate is the *lack of relevant information*. I do not deny that most managers lack a good deal of information that they should have, but I do deny that this is the most important informational deficiency from which they suffer. It seems to me that they suffer more from an *over abundance of irrelevant information*.

* Received June 1967.

This is not a play on words. The consequences of changing the emphasis of an MIS from supplying relevant information to eliminating irrelevant information is considerable. If one is preoccupied with supplying relevant information, attention is almost exclusively given to the generation, storage, and retrieval of information: hence emphasis is placed on constructing data banks, coding, indexing, updating files, access languages, and so on. The ideal which has emerged from this orientation is an infinite pool of data into which a manager can reach to pull out any information he wants. If, on the other hand, one sees the manager's information problem primarily, but not exclusively, as one that arises out of an overabundance of irrelevant information, most of which was not asked for, then the two most important functions of an information system become *filtration* (or evaluation) and *condensation*. The literature on MIS's seldom refers to these functions let alone considers how to carry them out.

My experience indicates that most managers receive much more data (if not information) than they can possibly absorb even if they spend all of their time trying to do so. Hence they already suffer from an information overload. They must spend a great deal of time separating the relevant form the irrelevant and searching for the kernels in the relevant documents. For example, I have found that I receive an average of forty-three hours of unsolicited reading material each week. The solicited material is usually half again this amount.

I have seen a daily stock status report that consists of approximately six hundred pages of computer print-out. The report is circulated daily across managers' desks. I've also seen requests for major capital expenditures that come in book size, several of which are distributed to managers each week. It is not uncommon for many managers to receive an average of one journal a day or more. One could go on and on.

Unless the information overload to which managers are subjected is reduced, any additional information made available by an MIS cannot be expected to be used effectively.

Even relevant documents have too much redundancy. Most documents can be considerably condensed without loss of content. My point here is best made, perhaps, by describing briefly an experiment that a few of my colleagues and I conducted on the OR literature several years ago. By using a panel of well-known experts we identified four OR articles that all members of the panel considered to be "above average," and four articles that were considered to be "below average." The authors of the eight articles were asked to prepare "objective" examinations (duration thirty minutes) plus answers for graduate students who were to be assigned the articles for reading. (The authors were not informed about the experiment.) Then several experienced writers were asked to reduce each article to $\frac{1}{2}$ and $\frac{1}{4}$ of its original length only by eliminating words. They also prepared a brief abstract of each article. Those who did the condensing did not see the examinations to be given to the students.

A group of graduate students who had not previously read the articles were then selected. Each one was given four articles randomly selected, each of which

was in one of its four versions: 100%, 67%, 33%, or abstract. Each version of each article was read by two students. All were given the same examinations. The average scores on the examinations were then compared.

For the above-average articles there was no significant difference between average test scores for the 100%, 67%, and 33% versions, but there was a significant decrease in average test scores for those who had read only the abstract. For the below-average articles there was no difference in average test scores among those who had read the 100%, 67%, and 33% versions, but there was a significant *increase* in average test scores of those who had read only the abstract.

The sample used was obviously too small for general conclusions but the results strongly indicate the extent to which even good writing can be condensed without loss of information. I refrain from drawing the obvious conclusion about bad writing.

It seems clear that condensation as well as filtration, performed mechanically or otherwise, should be an essential part of an MIS, and that such a system should be capable of handling much, if not all, of the unsolicited as well as solicited information that a manager receives.

The Manager Needs the Information That He Wants

Most MIS designers "determine" what information is needed by asking managers what information they would like to have. This is based on the assumption that managers know what information they need and want it.

For a manager to know what information he needs he must be aware of each type of decision he should make (as well as does) and he must have an adequate model of each. These conditions are seldom satisfied. Most managers have some conception of at least some of the types of decisions they must make. Their conceptions, however, are likely to be deficient in a very critical way, a way that follows from an important principle of scientific economy: the less we understand a phenomenon, the more variables we require to explain it. Hence, the manager who does not understand the phenomenon he controls plays it "safe" and, with respect to information, wants "everything." The MIS designer, who has even less understanding of the relevant phenomenon than the manager, tries to provide even more than everything. He thereby increases what is already an overload of irrelevant information.

For example, market researchers in a major oil company once asked their marketing managers what variables they thought were relevant in estimating the sales volume of future service stations. Almost seventy variables were identified. The market researchers then added about half again this many variables and performed a large multiple linear regression analysis of sales of existing stations against these variables and found about thirty-five to be statistically significant. A forecasting equation was based on this analysis. An OR team subsequently constructed a model based on only one of these variables, traffic flow, which predicted sales better than the thirty-five variable regression equation. The team went on to explain sales at service stations in terms of the

customers' perception of the amount of time lost by stopping for service. The relevance of all but a few of the variables used by the market researchers could be explained by their effect on such perception.

The moral is simple: one cannot specify what information is required for decision making until an explanatory model of the decision process and the system involved has been constructed and tested. Information systems are subsystems of control systems. They cannot be designed adequately without taking control in account. Furthermore, whatever else regression analyses can yield, they cannot yield understanding and explanation of phenomena. They describe and, at best, predict.

Give a Manager the Information He Needs and His Decision Making Will Improve

It is frequently assumed that if a manager is provided with the information he needs, he will then have no problem in using it effectively. The history of OR stands to the contrary. For example, give most managers an initial tableau of a typical "real" mathematical programming, sequencing, or network problem and see how close they come to an optimal solution. If their experience and judgment have any value they may not do badly, but they will seldom do very well. In most management problems there are too many possibilities to expect experience, judgement, or intuition to provide good guesses, even with perfect information.

Furthermore, when several probabilities are involved in a problem the unguided mind of even a manager has difficulty in aggregating them in a valid way. We all know many simple problems in probability in which untutored intuition usually does very badly (e.g., What are the correct odds that 2 of 25 people selected at random will have their birthdays on the same day of the year?). For example, very few of the results obtained by queuing theory, when arrivals and service are probabilistic, are obvious to managers; nor are the results of risk analysis where the managers' own subjective estimates of probabilities are used.

The moral: it is necessary to determine how well managers can use needed information. When, because of the complexity of the decision process, they can't use it well, they should be provided with either decision rules or performance feed-back so that they can identify and learn from their mistakes. More on this point later.

More Communication Means Better Performance

One characteristic of most MIS's which I have seen is that they provide managers with better current information about what other managers and their departments and divisions are doing. Underlying this provision is the belief that better interdepartmental communication enables managers to coordinate their decisions more effectively and hence improves the organization's overall performance. Not only is this not necessarily so, but it seldom is so. One would hardly expect two competing companies to become more cooperative because

the information each acquires about the other is improved. This analogy is not as far fetched as one might first suppose. For example, consider the following very much simplified version of a situation I once ran into. The simplification of the case does not affect any of its essential characteristics.

A department store has two "line" operations: buying and selling. Each function is performed by a separate department. The Purchasing Department primarily controls one variable: how much of each item is bought. The Merchandising Department controls the price at which it is sold. Typically, the measure of performance applied to the Purchasing Department was the turnover rate of inventory. The measure applied to the Merchandising Department was gross sales; this department sought to maximize the number of items sold times their price.

Now by examining a single item let us consider what happens in this system. The merchandising manager, using his knowledge of competition and consumption, set a price which he judged would maximize gross sales. In doing so he utilized price-demand curves for each type of item. For each price the curves show the expected sales and values on an upper and lower confidence band as well. (See Figure 1.) When instructing the Purchasing Department how many items to make available, the merchandising manager quite naturally used the value on the upper confidence curve. This minimized the chances of his running short which, if it occurred, would hurt his performance. It also maximized the chances of being over-stocked but this was not his concern, only the purchasing manager's. Say, therefore, that the merchandising manager initially selected price P_1 and requested that amount Q_1 be made available by the Purchasing Department.

In this company the purchasing manager also had access to the price-demand curves. He knew the merchandising manager always ordered optimistically.

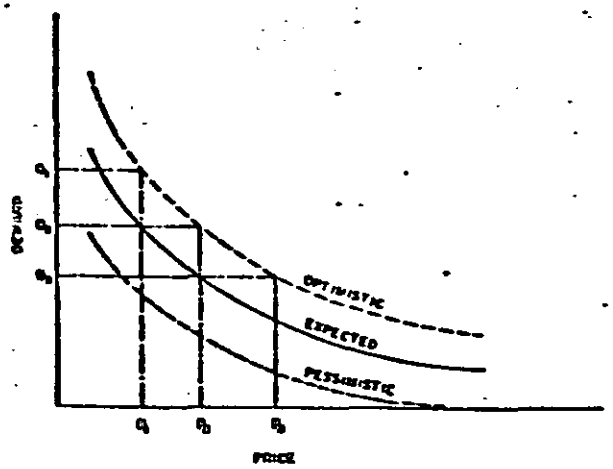


FIGURE 1. Price-demand curves

Therefore, using the same curve he read over from Q_1 to the upper limit and down to the expected value from which he obtained Q_2, the quantity he actually intended to make available. He did not intend to pay for the merchandising manager's optimism. If merchandising ran out of stock, it was not his worry. Now the merchandising manager was informed about what the purchasing manager had done so he adjusted his price to P_2. The purchasing manager in turn was told that the merchandising manager had made this readjustment so he planned to make only Q_2 available. If this process—made possible only by perfect communication between departments—had been allowed to continue, nothing would have been bought and nothing would have been sold. This outcome was avoided by prohibiting communication between the two departments and forcing each to guess what the other was doing.

I have obviously caricatured the situation in order to make the point clear: when organizational units have inappropriate measures of performance which put them in conflict with each other, as is often the case, communication between them may hurt organizational performance, not help it. Organizational structure and performance measurement must be taken into account before opening the flood gates and permitting the free flow of information between parts of the organization. (A more rigorous discussion of organizational structure and the relationship of communication to it can be found in [1].)

A Manager Does Not Have to Understand How an Information System Works, Only How to Use It

Most MIS designers seek to make their systems as innocuous and unobtrusive as possible to managers lest they become frightened. The designers try to provide managers with very easy access to the system and assure them that they need to know nothing more about it. The designers usually succeed in keeping managers ignorant in this regard. This leaves managers unable to evaluate the MIS as a whole. It often makes them afraid to even try to do so lest they display their ignorance publicly. In failing to evaluate their MIS, managers delegate much of the control of the organization to the system's designers and operators who may have many virtues, but managerial competence is seldom among them.

Let me cite a case in point. A Chairman of a Board of a medium-size company asked for help on the following problem. One of his larger (decentralized) divisions had installed a computerized production-inventory control and manufacturing-manager information system about a year earlier. It had acquired about $2,000,000 worth of equipment to do so. The Board Chairman had just received a request from the Division for permission to replace the original equipment with newly announced equipment which would cost several times the original amount. An extensive "justification" for so doing was provided with the request. The Chairman wanted to know whether the request was really justified. He admitted to complete incompetence in this connection.

A meeting was arranged at the Division at which I was subjected to an extended and detailed briefing. The system was large but relatively simple. At the heart of it was a reorder point for each item and a maximum allowable

stock level. Reorder quantities took lead-time as well as the allowable maximum into account. The computer kept track of stock, ordered items when required and generated numerous reports on both the state of the system it controlled and its own "actions."

When the briefing was over I was asked if I had any questions. I did. First I asked if, when the system had been installed, there had been many parts whose stock level exceeded the maximum amount possible under the new system. I was told there were many. I asked for a list of about thirty and for some graph paper. Both were provided. With the help of the system designer and volumes of old daily reports I began to plot the stock level of the first listed item over time. When this item reached the maximum "allowable" stock level it had been reordered. The system designer was surprised and said that by sheer "luck" I had found one of the few errors made by the system. Continued plotting showed that because of repeated premature reordering the item had never gone much below the maximum stock level. Clearly the program was confusing the maximum allowable stock level and the reorder point. This turned out to be the case in more than half of the items on the list.

Next I asked if they had many paired parts, ones that were only used with each other; for example, matched nuts and bolts. They had many. A list was produced and we began checking the previous day's withdrawals. For more than half of the pairs the differences in the numbers recorded as withdrawn were very large. No explanation was provided.

Before the day was out it was possible to show by some quick and dirty calculations that the new computerized system was costing the company almost $150,000 per month more than the hand system which it had replaced, most of this in excess inventories.

The recommendation was that the system be redesigned as quickly as possible and that the new equipment not be authorized for the time being.

The questions asked of the system had been obvious and simple ones. Managers should have been able to ask them but—and this is the point—they felt themselves incompetent to do so. They would not have allowed a handoperated system to get so far out of their control.

No MIS should ever be installed unless the managers for whom it is intended are trained to evaluate and hence control it rather than be controlled by it.

A Suggested Procedure for Designing an MIS

The erroneous assumptions I have tried to reveal in the preceding discussion can, I believe, be avoided by an appropriate design procedure. One is briefly outlined here.

1. Analysis Of The Decision System

Each (or at least each important) type of managerial decision required by the organization under study should be identified and the relationships between them should be determined and flow-charted. Note that this is *not* necessarily the same thing as determining what decisions *are* made. For example, in one company I found that make-or-buy decisions concerning parts were made only at the time when a part was introduced into stock and was never subsequently reviewed. For some items this decision had gone unreviewed for as many as twenty years. Obviously, such decisions should be made more often; in some cases, every time an order is placed in order to take account of current shop loading, underused shifts, delivery times from suppliers, and so on.

Decision-flow analyses are usually self-justifying. They often reveal important decisions that are being made by default (e.g., the make-buy decision referred to above), and they disclose interdependent decisions that are being made independently. Decision-flow charts frequently suggest changes in managerial responsibility, organizational structure, and measure of performance which can correct the types of deficiencies cited.

Decision analyses can be conducted with varying degrees of detail, that is, they may be anywhere from coarse to fine grained. How much detail one should become involved with depends on the amount of time and resources that are available for the analysis. Although practical considerations frequently restrict initial analyses to a particular organizational function, it is preferable to perform a coarse analysis of all of an organization's managerial functions rather than a fine analysis of one or a subset of functions. It is easier to introduce finer information into an integrated information system than it is to combine fine subsystems into one integrated system.

2. An Analysis Of Information Requirements

Managerial decisions can be classified into three types:

(a) Decisions for which adequate models are available or can be constructed and from which optimal (or near optimal) solutions can be derived. In such cases the decision process itself should be incorporated into the information system thereby converting it (at least partially) to a control system. A decision model identifies what information is required and hence what information is relevant.

(b) Decisions for which adequate models can be constructed but from which optimal solutions cannot be extracted. Here some kind of heuristic or search procedure should be provided even if it consists of no more than computerized trial and error. A simulation of the model will, as a minimum, permit comparisons of proposed alternative solutions. Here too the model specifies what information is required.

(c) Decisions for which adequate models cannot be constructed. Research is required here to determine what information is relevant. If decision making cannot be delayed for the completion of such research or the decision's effect is not large enough to justify the cost of research, then judgment must be used to "guess" what information is relevant. It may be possible to make explicit the implicit model used by the decision maker and treat it as a model of type (b).

In each of these three types of situation it is necessary to provide feedback by comparing actual decision outcomes with those predicted by the model or decision maker. Each decision that is made, along with its predicted outcome

should be an essential input to a management control system. I shall return to this point below.

3. Aggregation Of Decisions

Decisions with the same or largely overlapping informational requirements should be grouped together as a single manager's task. This will reduce the information a manager requires to do his job and is likely to increase his understanding of it. This may require a reorganization of the system. Even if such a reorganization cannot be implemented completely what can be done is likely to improve performance significantly and reduce the information loaded on managers.

4. Design Of Information Processing

Now the procedure for collecting, storing, retrieving, and treating information can be designed. Since there is a voluminous literature on this subject I shall leave it at this except for one point. Such a system must not only be able to answer questions addressed to it; it should also be able to answer questions that have not been asked by reporting any deviations from expectations. An extensive exception-reporting system is required.

5. Design Of Control Of The Control System

It must be assumed that the system that is being designed will be deficient in many and significant ways. Therefore it is necessary to identify the ways in which it may be deficient, to design procedures for detecting its deficiencies, and for correcting the system so as to remove or reduce them. Hence the system should be designed to be flexible and adaptive. This is little more than a platitude, but it has a not-so-obvious implication. No completely computerized system can be as flexible and adaptive as can a man-machine system. This is illustrated by a concluding example of a system that is being developed and is partially in operation. (See Figure 2.)

The company involved has its market divided into approximately two hundred marketing areas. A model for each has been constructed as is "in" the computer. On the basis of competitive intelligence supplied to the service marketing manager by marketing researchers and information specialists he and his staff make policy decisions for each area each month. Their tentative decisions are fed into the computer which yields a forecast of expected performance. Changes are made until the expectations match what is desired. In this way they arrive at "final" decisions. At the end of the month the computer compares the actual performance of each area with what was predicted. If a deviation exceeds what could be expected by chance, the company's OR Group then seeks the reason for the deviation, performing as much research as is required to find it. If the cause is found to be permanent the computerized model is adjusted appropriately. The result is an adaptive man-machine system whose precision and generality is continuously increasing with use.

Finally it should be noted that in carrying out the design steps enumerated

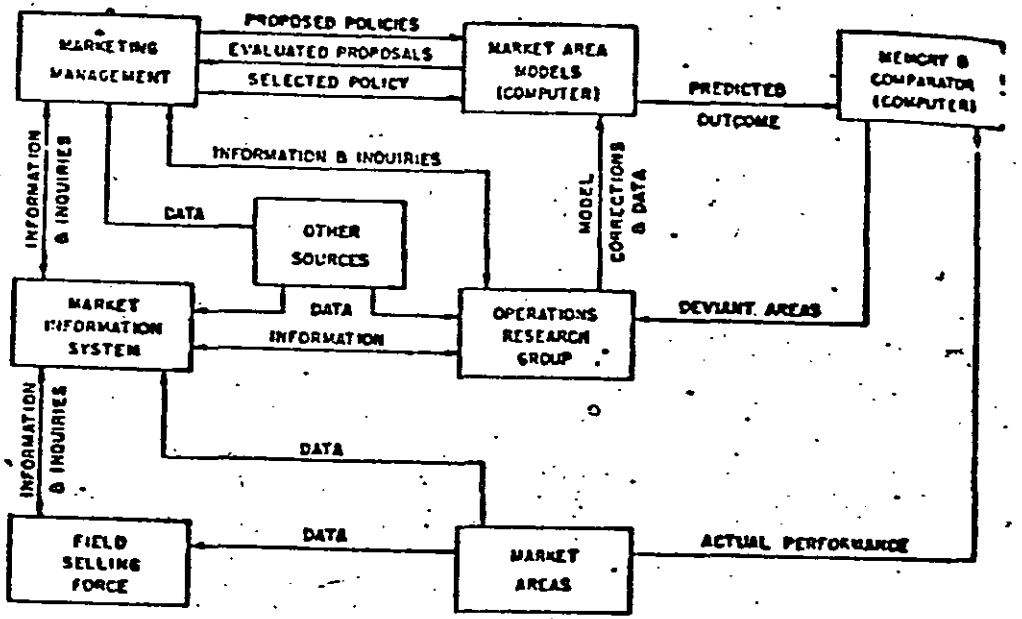


FIGURE 2. Simplified diagram of a market-area control system

above, three groups should collaborate: information systems specialists, operations researchers, *and managers*. The participation of managers in the design of a system that is to serve them, assures their ability to evaluate its performance by comparing its output with what was predicted. Managers who are not willing to invest some of their time in this process are not likely to use a management control system well, and their system, in turn, is likely to abuse them.

Reference

1. SENGUPTA, S. S., AND ACKOFF, R. L., "Systems Theory from an Operations Research Point of View," *IEEE Transactions on Systems Science and Cybernetics*, Vol. 1 (Nov. 1965), pp. 9-13.

DISEÑO DE BASES DE DATOS

A R T I C U L O

WHY RESTRICT THE MODELING CAPABILITY OF CODASYL DATA STRUCTURE SETS

EXPOSITOR:
Ing. Daniel Ríos Zertuche

MAYO, 1985

Why restrict the modelling capability of codasyl data structure sets?

by CHARLES W. BACHMAN
Honeywell Information Systems
Billerica, Massachusetts

ABSTRACT

Several issues have been raised concerning changes to the capabilities of the CODASYL Data Description Language specifications for data structure sets. The paper argues against new restrictions suggested and for removal of existing restrictions. The issues are:

allow recursive set declaration
keep multiple member declaration
allow alternate owner declarations

The concept of "record-roles" is introduced to justify the need for these capabilities. The expanded capabilities described offer an alternate means of achieving the same end result without the need to introduce the "record-role" into the CODASYL Data Description Language.

INTRODUCTION

The concept of data structure sets has been well established through the publicity and use of I-D-S, the Honeywell Integrated Data Store system.[1-3] In recent years this concept has been adopted by the various CODASYL committees and imbedded in the CODASYL Data Description Language[4] and the COBOL Data Manipulation Language.[5] A number of hardware and software suppliers have implemented the data structure sets of these languages (DDL/DML) as part of their systems. They include:

IDMS	(Cullinane for IBM 360/370)
I-D-S	(Honeywell GE200, GE400, GE600, H6000)
I-D-S II	(Honeywell for H66, H64)
EDMS	(Xero Sigma 6/7/8)
DMS 1100	(Univac for Univac 1100 Series)
PHOLAS	(Philips for Unidata 7000, P1000)
PHOLAS	(Siemens for S4004)

Other implementations have been reported for CDC and DEC.

The capabilities of the data structure set, as developed in I-D-S and now defined in both the CODASYL DDL and the COBOL DML, provide for set-type declarations which:

(1) restrict the record type declared as owner to be different from any of the record types declared as member,
(2) permit declaration of one or more record types to serve as member records of an occurrence of a set type, and
(3) restrict to one the number of record types which can be declared to serve as owner records of occurrences of a set type.

THE PROPOSALS

This paper is a refinement of a working paper written in response to an assignment accepted at the IFIP-TC2 meeting on Data Description Languages held in Namur, Belgium in January 1975. There were three closely related proposals for changes to the CODASYL DDL discussed at that meeting. These proposals relate directly to three points enumerated in the prior paragraph. Assignments were given to defend a number of such proposals. The proposals interrelated and my working paper treats them as a single concept.

The first proposal, which was unanimously supported at the meeting, was to remove the restriction that the record-type declared as owner could not also be one of those declared as member. I strongly concurred with this proposal as a removal of an unnecessary restriction.

The second proposal was to add a restriction that only one record-type could be declared as member of a particular set-type. This proposal received mixed support. I strongly disagreed with this proposal, for essentially the same reasons that I support the first and third proposals. It adds an unnecessary restriction.

The third proposal was to remove the restriction that only one record-type can serve as owner of a particular set-type. This proposal received scant attention. The meeting did not express an opinion on the subject. I strongly recommended it, as I have done to the DBTG at least seven years ago. It is the removal of a restriction.

ARGUMENTS

There are several arguments for permitting a set-type to permit the record-type declared as member to be the same as the record-type declared as owner. There is a specific argument which will be treated first and then a general argument which relates to all three proposals mentioned earlier.

The specific argument treats the need for tree-like data structures, catalogues, organization structures and parsing trees. For these structures, it is necessary to support recursive sets, which provides the capability to build trees, with branches which have branches, which have branches, etc. An example of this is illustrated in Figure 1.

If all the straight lines in Figure 1 are considered to be "branches," then this structure can be built with a single record type and a single set-type. However, the membership of the "branch" record-type in the "fork" set-type must not be mandatory. A record declared to be as the lowest level branch is never a member of a "fork" set-occurrence. This special branch is characterized as the trunk. Figure 2 is a data structure diagram[8] which illustrates the branch/fork structure.

In this data structure diagram, the "fork" set-type is illustrated with a broken line, meaning that the "branch" record-occurrences are sometimes members of it. That is, they are members if they are not the "trunk" branch of the tree. Note that all branch record-occurrences, whether or not they are the trunk, own "fork" set-occurrence with zero, one, or more subordinate "branches" records.

The third Namur proposal, which supports alternative owners in data structure sets, would permit the trunk branch to be treated as a distinctly different type of entity. Figure 3 is a redrawing of Figure 1.

It illustrates a tree with one trunk and many forks and branches. When this is drawn as a data structure diagram, the illustration of Figure 4 is developed. This structure has some advantages.

The "branch" record-occurrences can now be treated as mandatory members of the "fork" set-type, i.e., no branches are floating in the air. This can be very important

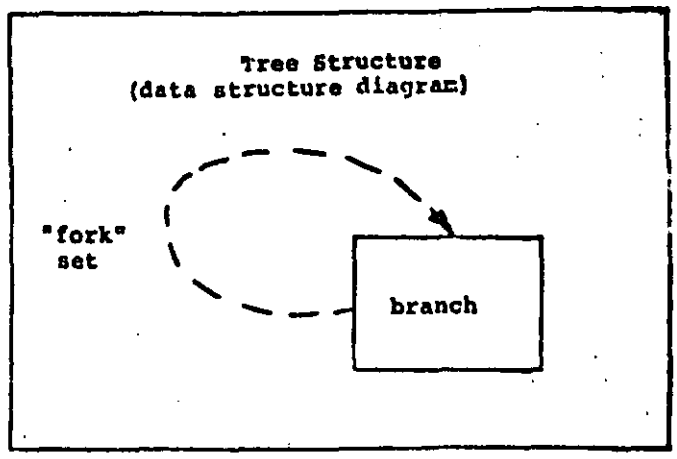


Figure 2

from a naming point of view, as each branch needs a field for its "branch name" while the trunk does not need such a name. Branches are frequently named with an articulated grammar. All the branch names at a single fork of the tree must have unique local names. The higher level branches (i.e., the branches which are farther from the trunk), are uniquely named by concatenating their unique local names to the tree unique name of the branch immediately below, This is expressed below in BNF (Backus/Nauer/Form)

⟨branch-name⟩::=⟨character-string⟩ |
⟨branch-name⟩⟨articulation-character⟩
⟨character-string⟩
⟨character-string⟩::=⟨character⟩ | ⟨character-string⟩
⟨character⟩
⟨character⟩::=a|b|c|.....|z|0|1|.....|9
⟨articulation-character⟩::=any symbol which is not a blank or a character

In our information systems today, there are many examples of tree structures. In catalog (file) systems, we find

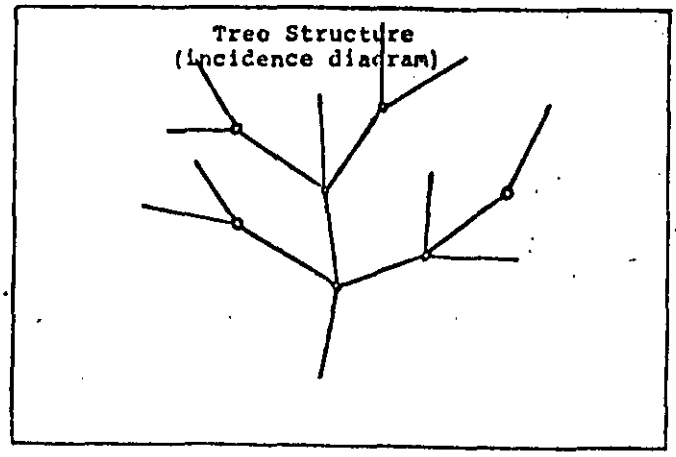


Figure 1

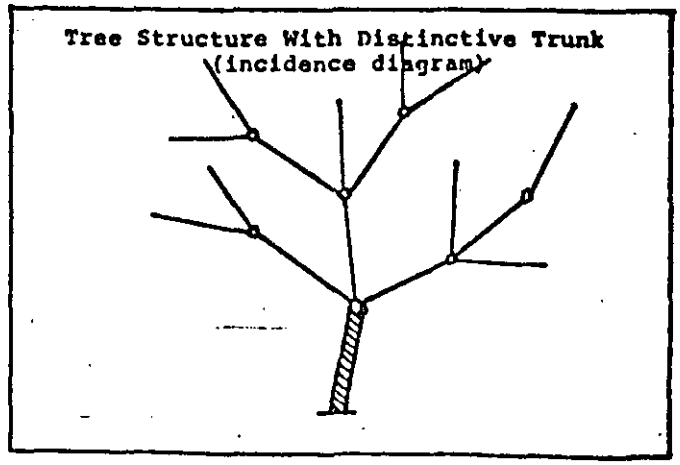


Figure 3

directories of directories of directories of. . . of named files. In corporate organizations we find companies which have departments, which have departments, etc. Figure 5 is a data structure diagram which illustrates this.

This example illustrates a specific need for this type of structure and supports the proposal. Some will argue that this is an incorrect approach to the fundamental corporate organization structure and that they are really hierarchies of different types of organizational-units. At one time, the General Electric Company had a well defined hierarchical organization structure which is illustrated in Figure 6.

If you were a "section" manager, you knew exactly where you were in the management hierarchy. This is an easier structure to handle manually than by computer, as people did not get quite as upset when someone thought it appropriate for a particular "unit" manager to report directly to a "section" or "department" manager. If this organization structure were declared to an I-D-S database system with the structure illustrated in Figure 6, then no "unit" could directly report to a "section" or "department", it would have to be assigned to a "subsection."

The proposal to support alternative owner record-types for a single set-type should be accepted because it is useful. It does not require that the database administrator use either the structure of Figure 5 (alternative owners) or Figure 6 (unique owners). It should be supported because it allows each administrator the choice.

The more general argument for the support of alternative owner record-types for a set-type, also supports the need to retain the capability for multiple member record-types for a set-type. In an information system, real world entities are represented by records. These entities are classified by entity-type in order to facilitate the processing of data concerning them. Further, each entity-type may portray several concurrent roles or behavior patterns and sometimes these roles are shared by other distinctively different entity-types. For example, a person, or a company, or a governmental unit may serve in the role of "employer" of people, and as an "owner" of property. Within such a designated role, the record-types representing the entity-types should be capable of being the owner or member of a

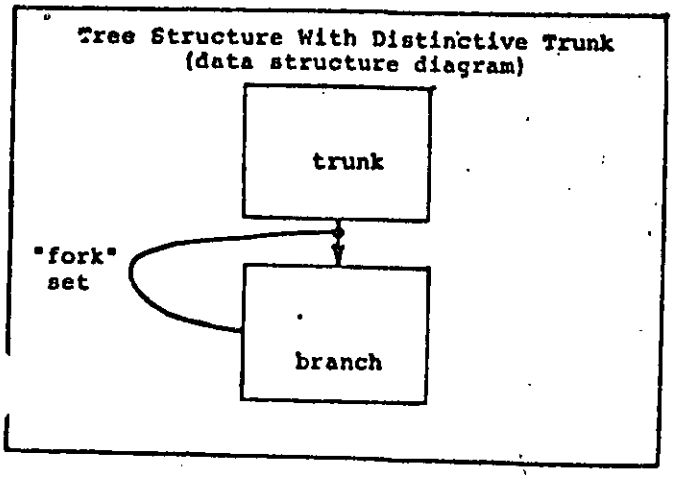


Figure 4

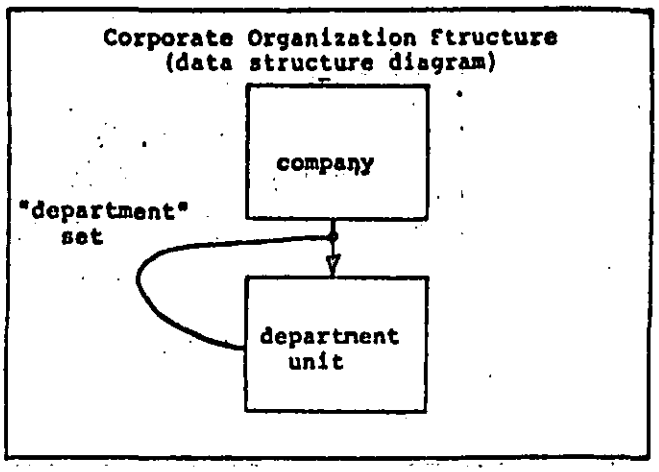


Figure 5

set-type which is role related, and be the holder of a field which is role related. Figure 7 is a data structure diagram which illustrates the employer role played by persons, companies and government units.

To model this structure, it is necessary to declare the "person" record-type, "company" record-type and "gov-

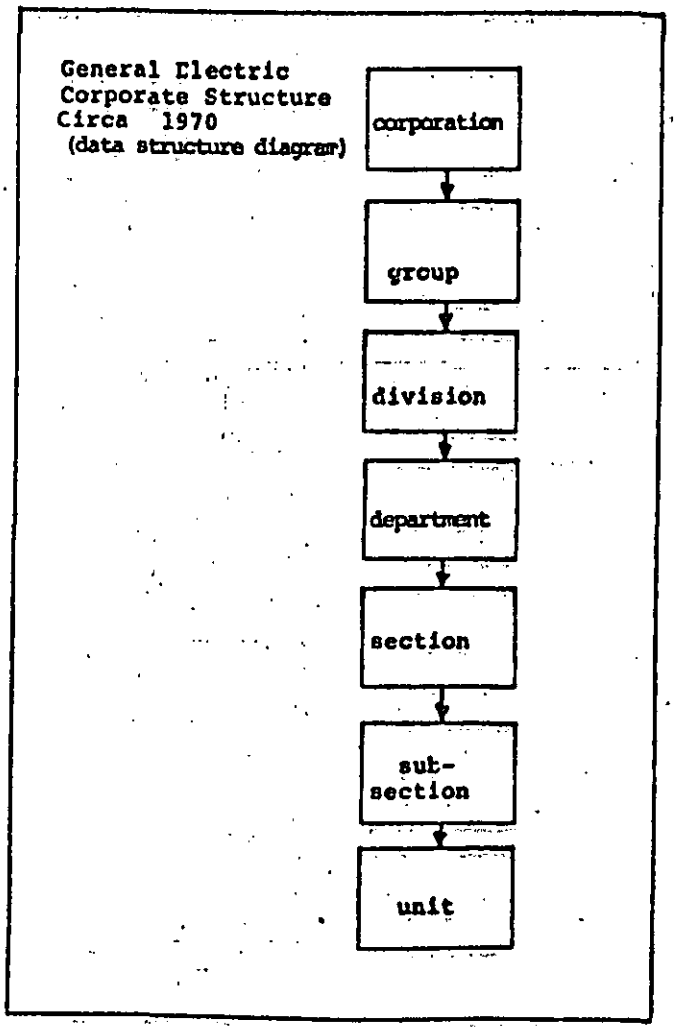


Figure 6

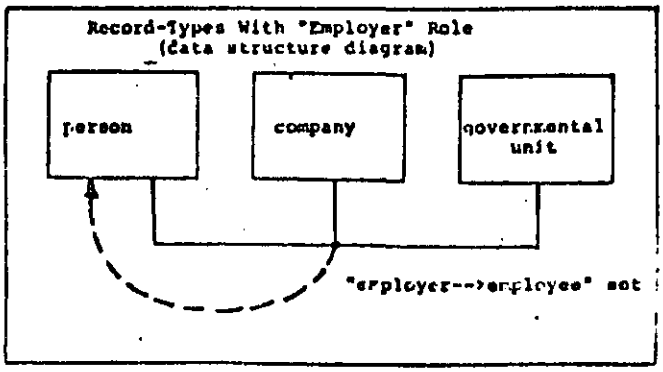


Figure 7

ernmental unit" record-type such that all are able to assume the rule of owner of the "employer⟶employee" set. It is necessary to declare the "person" record-type as a sometime set member, "sometime," since all persons are not necessarily employers.

In the case of a person who is self-employed, the same "person" record would be the owner and member of the same occurrence of the "employer⟶employer" set. Both the alternative owner (prop. 3) and recursive set (prep. 1) proposals would need to be accepted to support this structure.

The reader should glance back to Figure 6, one of the possible means of implementing the organizational unit aspects of a corporate structure. Given this structure, now imagine how the organization-to-employee relationship would have to be handled. Each organizational record, from the "corporation" record through to the "unit" record, must be able to handle the role of employer. All seven of the organizational records need to be declared as alternative owner types to the "organization⟶employee" set. Figure 8 illustrates this extension to Figure 6.

If one assumes that each of the units needs to have a manager, who is a person, then each of these seven organizational unit role record-types must also be declared as a member in the "managed" set. The support of the "managed" set gives an example of the usefulness of the ability to declared multiple record-types as members of the same set-type. (Proposal 2). Figure 9 illustrates the further extension of Figure 6 to include the "manager-⟶organization" set.

RECORD-ROLE CONCEPT

For the theoreticians (and it is they who have largely argued for reducing the number of member record-types declarable for a set-type to one, and keeping the owner types declarable to one) the introduction of the "record-role" concept may be of great importance. This is because there will be no argument from the practitioners over having only one role declared as the owner of a set-type and only one role declared as the member of set-type if roles become declarable entities. Furthermore, the owner and the member declarations could be restricted to be different "roles."

In the work at Honeywell Information Systems on this subject, the word "record-role" has been used to characterize the role concept introduced above. The following definitions apply:

- A "record-occurrence" is the database representation of a real world entity.
- A "record-role" is a declaration of the a collection of the properties (fields and sets) which a record-occurrence may represent on behalf of one role of a real world entity.
- A "record-type" is a declaration of a collection of one or more record-roles which a record occurrence may represent, while roles are all played concurrently by a real world entity.
- A "record-class" is the collection of all record-occurrence of a particular record-type. A record-occurrence is always in only one record-class, defined by a record-type.
- A "role-class" is the collection of all record-occurrences of a particular record-role. A record-occurrence

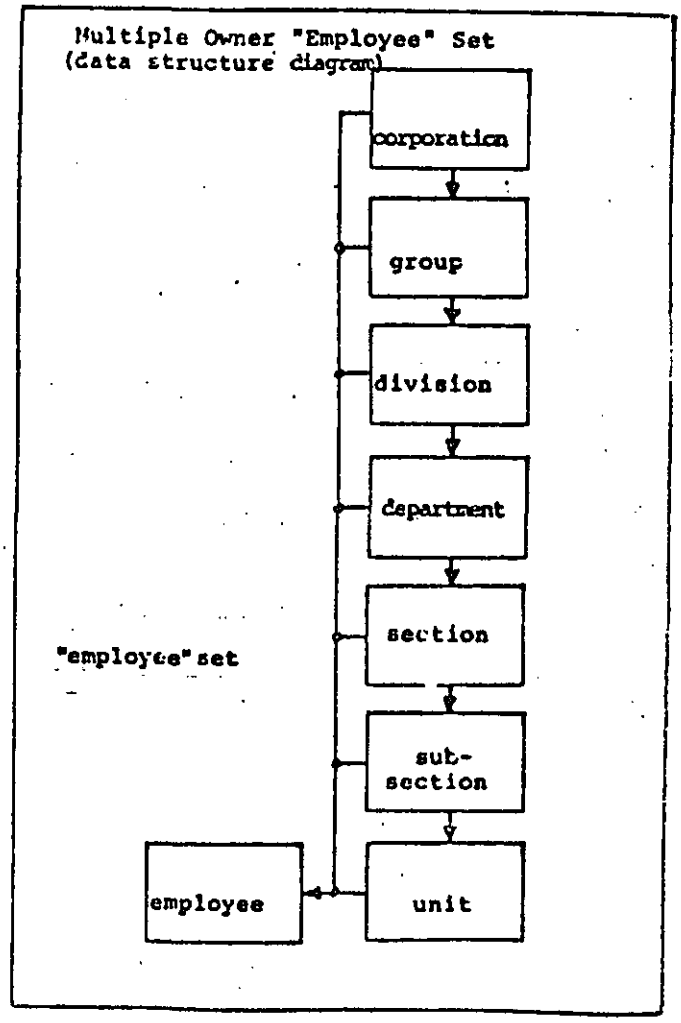


Figure 8

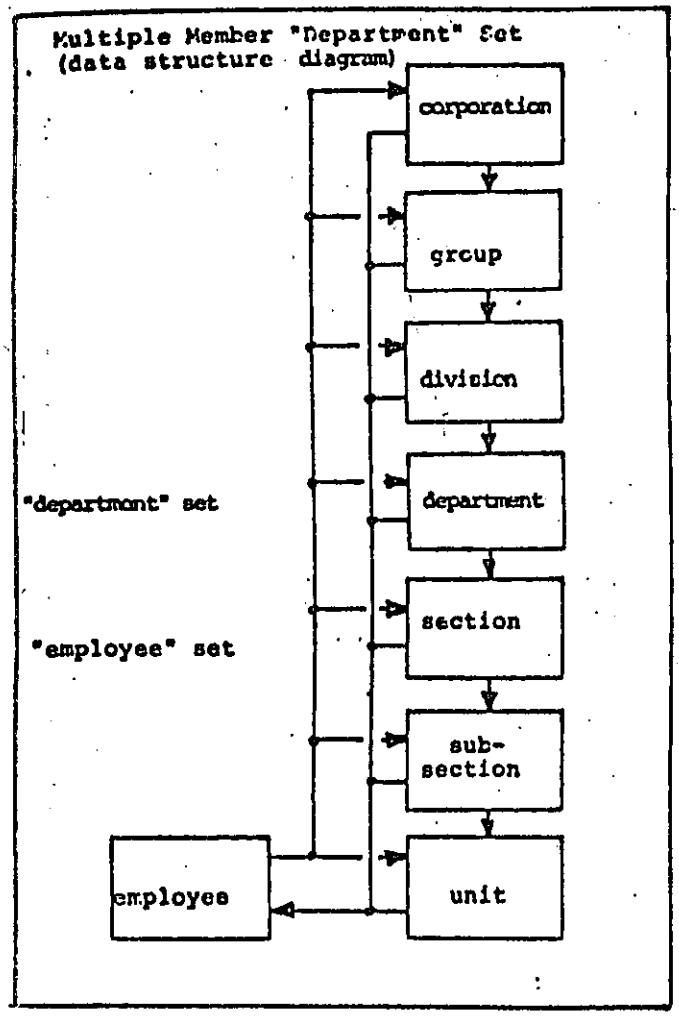


Figure 9

is in one or more many role-classes depending upon its record-type declaration.

- A "role-occurrence" is a subdivision of a record-occurrence which is the representation of that record-occurrence playing one role.

This distinction between record-role and record-type has not been made in existing database systems. While a record-type may have represented several record-roles, there was no mechanism of sharing the record-role declarations between two or more record-types. Thus the same fields and sets, relative to the role, had to be multiply declared, once for each record-type which played the role. This led to the requirement for multiple member declarations and alternate owner declarations for sets.

With the record-role concept, the declaration of fields, groups and sets are all associated with the record-role declaration. Field may be accessed using field-names which are qualified by record-role-name rather than record-type-names. Sets are ordered by role declared fields. Set owner selection is based upon role declared fields. Record occurrences of different record-types coexist within the same set type as owners or members when the record-types share the same record-role with the declared function.

In the paper "The Evolution of Data Structures," there was a sequence of data structure diagrams which were used to illustrate the progressive introduction of new meta objects and new inter-object relationships into data structuring capabilities. The first two of the following three data structure diagrams are reprinted from that article. Figure 10 illustrates the meta entities: record, field, group, owner, member and data-structure-set, where there may be an unlimited number of member record-types and owner record-types declarable for a set-type. The diagram of Figure 10 is the meta object structure which I recommend to DBTS and have recommended and used for a number of years.

Figure 11 is a simplification of Figure 10 where the restrictions of a one owner entity-role and a one member entity-role have been placed on the set-type. The set-owner and set-member meta entities have been merged with the set-type meta entities, as they exist on a 1:1:1 basis. This yields the restricted structural capability which was recommended by some of the attendees at the Namur Conference.

The data structure diagram of Figure 12 introduces the meta entity "record-role". In this structure, the "record-role" meta entity has displaced the "record-type" meta entity in its direct relationship to the set, group and field. The record-type concept is now at the side, associated by a declared relationship with one or more "record-roles".

"Record-role" declaration may be associated with one or more record-types. From the viewpoint of the set-type, there is only one "owner" record-role and only one "member" record-role. This fits more easily into the viewpoint of both the relational model and the Data Independent Access

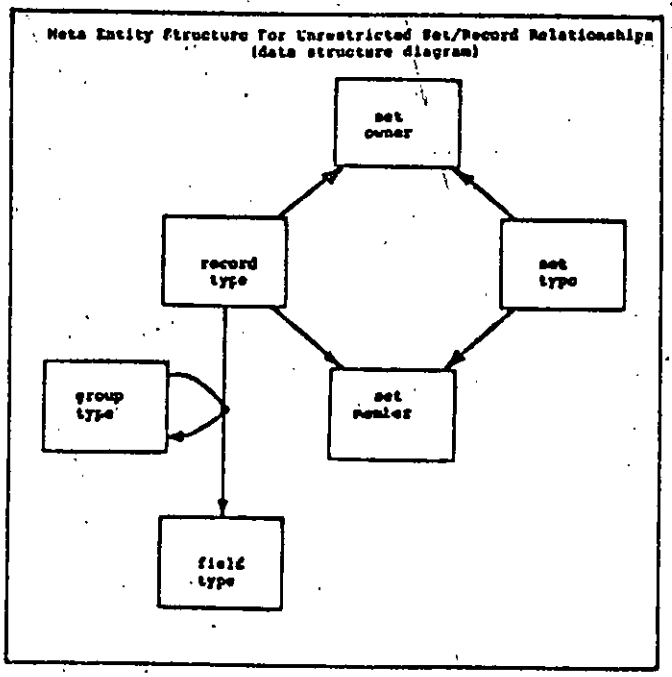


Figure 10

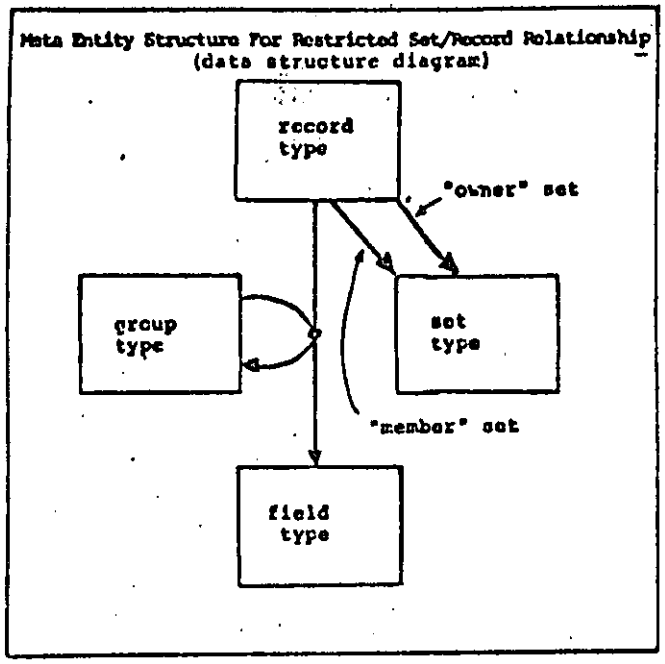


Figure 11

Model. However, the requirements of the real world which we wish to model can be satisfied as each real world entity can be recognized acting in one or more roles and its record-occurrences are combined with the declared role occurrences.

At this time I have no interest in trying to introduce the "record-role" concept into existing data description languages and data manipulation languages. Rather, I wish to provide the rationale, within these languages, for:

(1) recursive set-types,
(2) multiple member set-types, and
(3) multiple owner set-types

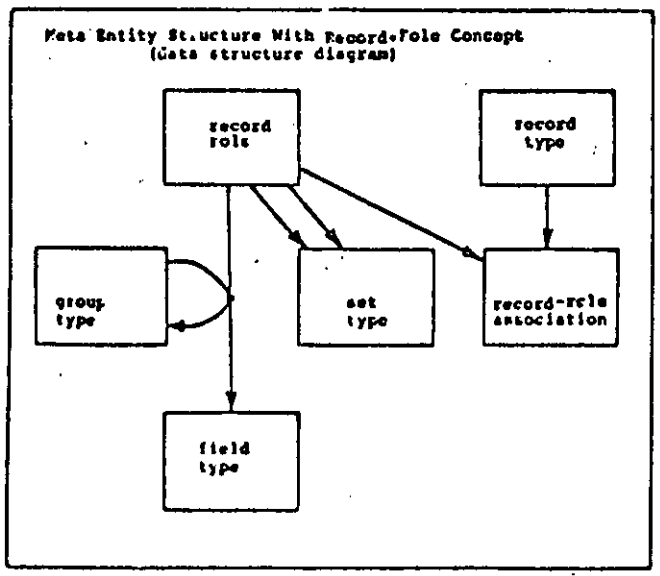


Figure 12

which provide an alternate means for achieving the objectives achieved by the record-role concept while remaining within the limitations of the presently available meta entities. However, the eventual introduction of the "record-role" may be the unifying factor that we seek.

The data structure diagramming technique has been extended to support the concept of record-role. Figure 13 is a redrawing of Figure 7 with focus on the record-roles of employer and employee.

They are illustrated by the two hexagons so designated. The record-types with which the roles are associated are designated by the background boxes. The employer role is played by the company, person and governmental-unit record-types. If classical data structure diagrams were thought to represent record-types and the relationships between them, then the new diagrams illustrate record-roles, their associations with record-types and their relationships with other record-roles. Record-roles are illustrated by hexagons and the background boxes name the record-types which play the role. Each record-type is considered to have one or more roles. In this example "person," "company" and "government-unit" are the record-types. "Employer" and "employee" are record-roles. A complete data structure diagram would show each record-type once as a box on top of a stack of hexagons. Each hexagon representing a record-role played by the record-type. It would also show each record-role, once as a hexagon at the top of a stack of boxes. Each record-type would appear once more as a background box behind each record-role it plays. Figure 13 is thus an incomplete data structure diagram as it shows the record-types playing each record role but does not graphically illustrate each record-

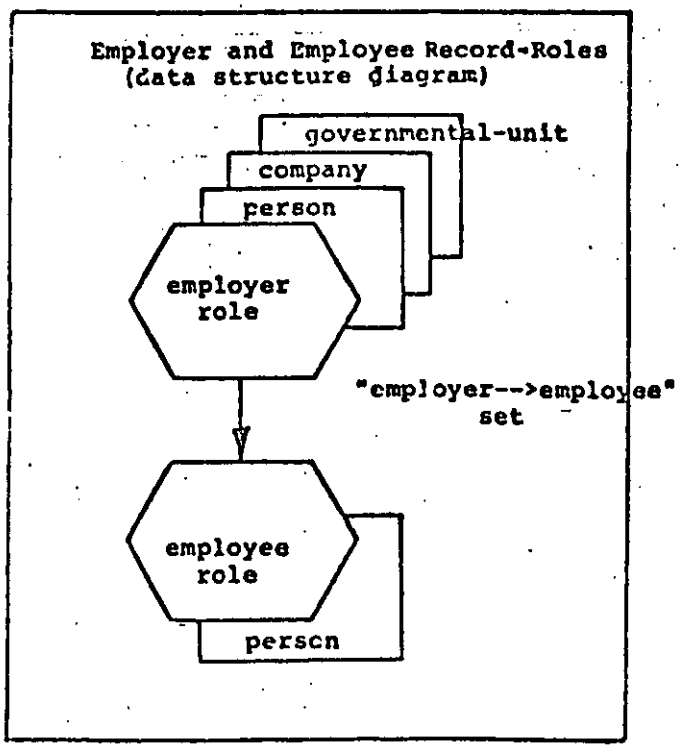


Figure 13

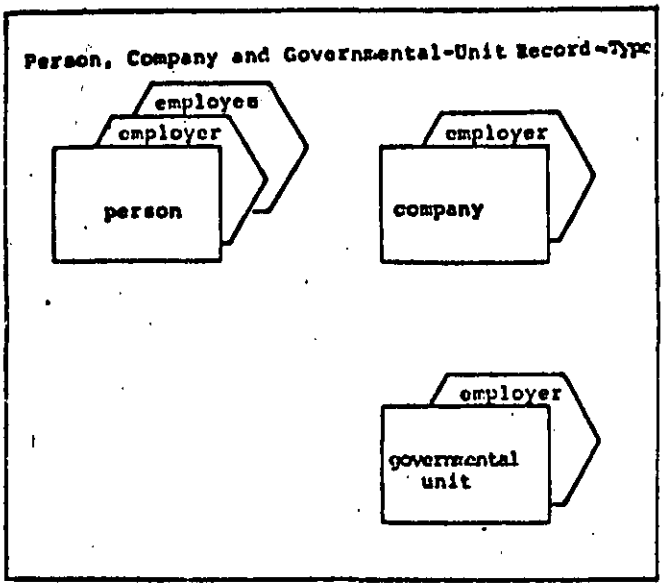


Figure 14

type with its record-roles. Figure 14 illustrates each record-type with the record-role that it plays. Thus most old data structure diagrams can be considered as being examples where the record had only one record-role. Thus no role factoring is necessary. If alternative owners or multiple members exist in those diagrams, then the record-roles for them has not yet been factored. The importance of the record-role concept to data structure diagrams may not be immediately obvious at the first comparison of Figures 7 and 13. However, consider the following analogy. If identical programming code appears in several parts of a computer program, it is common to factor this code out as subroutines or at least as macro procedures so that the documentation is more easily understood. The record-role concept is the data structure diagram equivalent of a subroutine call. The diagram illustrates the shared aspects of the record-role and also all the places it has been invoked. For data structure diagrams, representing complex organizations with many record-types, record-roles and their relationships, the record-role has proven to be extremely useful in simplifying the diagrams. They are much more readable. Of necessity, these diagrams are only effective after the new concept has been understood, used awhile and accepted.

SUMMARY

The data structure set is almost the only structural tool currently available to the database administrator to represent the relationships between entities in his enterprise. At this time when all of its usages are unknown, it seems desirable not to place any restrictions upon its application. Proposal 1, to permit recursive sets (where owner and member are of the same record class), is a proposal to remove a restriction. Proposal 2, to prohibit multiple member record-types, is a proposal to add a restriction. Proposal 3, to permit alternative owner declarations, is a proposal to remove a restriction. These facilities, within today's record-network model would provide a workable implementation of the role concept, an expression of the evident and important multiple behavior patterns which are characteristic of real world entities.

REFERENCES

1. Bachman, C. W., and S. B. Williams, "A General Purpose Programming System for Random Access Memories," *Proceeding AFIPS Conference Proceeding*, FJCC Volume 26 AFIPS Press Montvale N. J., 1964 pages 411–422.
2. "Integrated Data Store," *DPMA Quarterly*, January 1965.
3. "Software for Random Access Processing," *Datamation* April 1965, pages 36–41.
4. *CODASYL Data Description Language Journal of Development*, June 1973, (c13.6/22:113) Superintendent of Documents, U. S. Government Printing Office, Washington D. C. 20402.
5. *CODASYL COBOL Journal of Development*, January 1976, (110-GP-1D) Material Data Management Branch, Dept. of Supply and Service, 5th Floor, 88 Metcalfe Street, Ottawa, Ontario, Canada, K1A 0S5.
6. "Data Structure Diagrams, Data Base 1, 2," 1969, *Quarterly Newsletter of ACM SIGBD*, pages 4–10.
7. "The Evolution of Data Structures," *Proc NordDATA Conference*, August 1973, Copenhagen, Denmark, pages 1075–1093.

DISEÑO DE BASES DE DATOS

A R T I C U L O

THE ENTITY - RELATIONSHIPS MODEL = A BASIS FOR THE ENTERPRISE VIEW OF DATA

EXPOSITOR:
Ing. Daniel Ríos Zertuche

MAYO, 1985

The entity-relationship model—A basis for the enterprise view of data

by PETER PIN-SHAN CHEN
Massachusetts Institute of Technology
Cambridge, Massachusetts

ABSTRACT

The concept of the enterprise view of data is very useful in the database design process and in the construction of conceptual schema. This paper discusses the use of the entity-relationship approach in describing and maintaining the enterprise view of data. Fundamental operations for changing the enterprise schema are presented. Finally, an example is given to show the differences between the entity-relationship approach and the data-structure approach in modeling the enterprise view of data.

INTRODUCTION

The subject of the logical view of data has attracted considerable attention in the past ten years. However, most researchers have focused on the user view of data. The need for studying the enterprise view of data was not recognized until recently. Different users of a database may have different views of the database, but the enterprise should have a unique and consistent view of the database. This is particularly important in designing a logically meaningful and consistent database. The concept of the enterprise view of data is very useful in the database design process and in the design of conceptual schema.

Enterprise view and database design

Database design is a process to organize data into a form which matches the underlying data model of the database management system. There are three major types of database management systems: network, hierarchical, and relational. In the network database management systems, which include Honeywell's IDS and UNIVAC' DMS-1100, data will be organized into different types of records and can be represented by a data-structure diagram[1] (see Figure 1). In the hierarchical database management systems, which include IBM's IMS, data will be organized into a form similar to but more restricted than the data-structure diagram. In the relational database management systems,[2] data will be organized into a set of tables (or "relations"). In general, to design a database is to decide how to organize data into specific forms (record types, tables) and how to access them. Up to now, there are very few tools available to aid the database design process. Usually, the database designer relies on his own intuition and experience. Thus, the resulting database may not satisfy company's objectives and may cause problems in company's operations.

Another related problem in database design is that the output of the database design process—the user schema (a description of the user view of data)—is not a "pure" representation of the real world. One of the reasons is that the database designer is restricted by the limited capabilities of the database management system. For example, the many-to-many relationships between entities are difficult to represent directly in some database systems. Another reason is that the user schema may contain some features related to the storage representation of the database. For instance, it may describe which record types can be directly accessed and how to access other record types. In addition, the user schema is usually designed to be efficient for a certain type of data processing operations. For example, the data about employees may be grouped into two record types, employee-master and employee-detail, to improve the retrieval performance. Therefore, the user schema is usually not a direct representation of the real world. This makes the user schema difficult to understand and difficult to change.

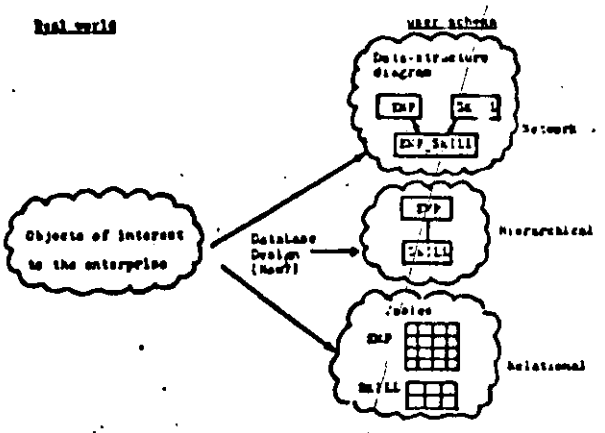


Figure 1—Conventional database design process

A possible solution to the above problems is to introduce an intermediate stage in the database design process: defining the enterprise schema, which is a "pure" representation of the real world and is independent of storage and efficiency considerations. The enterprise schema will then be translated into different types of schemata for different database management systems (see Figure 2). It can also be translated into several schemata for the same database management system to optimize different types of data processing operations. There are several advantages of this approach:

(1) The enterprise schema is easier to understand than a user schema since the former does not have the restrictions of the underlying database management system;
(2) The enterprise schema is more stable than the user schema, since some types of changes in the user schema may not require any change in the enterprise schema. If the enterprise schema needs to be changed to reflect the changes in the enterprise environment, the changes can be performed easily since efficiency and storage issues are not considered.

Enterprise view and conceptual schema

What is the difference between the enterprise schema and the conceptual schema proposed by the ANSI/X3/SPARC group?[24] Basically, they are very similar since both are descriptions of the enterprise view of data. In the SPARC's approach, the conceptual schema serves as the interface between the external schema (user view of data) and the storage schema (physical view of data) (see Figure 3). The requirement of serving as an interface between two other schemata may introduce some undesirable features into the conceptual schema. If this restriction on the conceptual schema is ignored, there is almost no difference between the conceptual schema and the enterprise schema. Therefore, the techniques discussed in this paper are also suitable for describing and maintaining the conceptual schema in the SPARC's architecture.

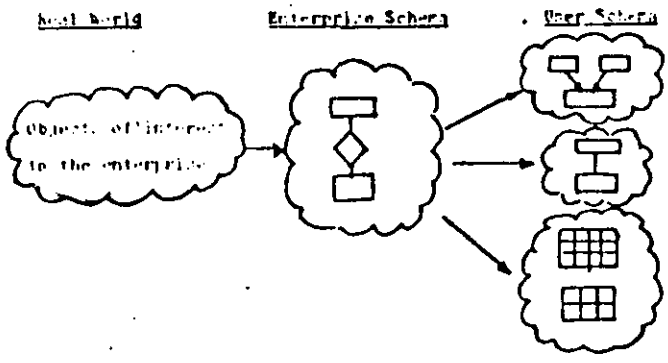


Figure 2—Enterprise schema as an intermediate step in database design

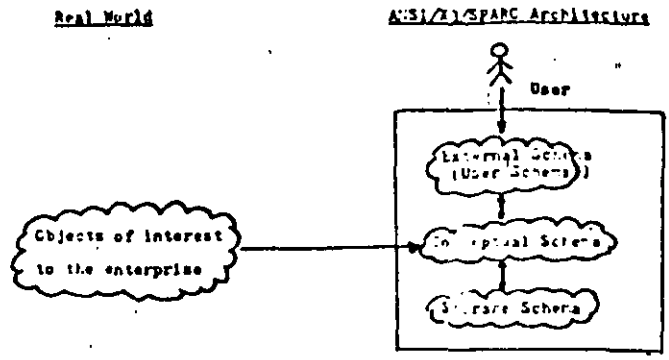


Figure 3—Enterprise view and conceptual schema

Approach used in the paper

In order to describe the enterprise view of data, a mental framework to model the real world is needed. Different people may be used to different mental frameworks. The mental framework used in this paper is the Entity-Relationship (E-R) model.[4,5] The E-R model and similar approaches[6-9] have been found useful in modeling the real world. A diagrammatic technique called the Entity-Relationship (E-R) diagram will be used in this paper to represent the enterprise view of data.

This paper is divided into three parts. The first part discusses how to use the E-R model and diagrammatic technique to describe the enterprise view of data. This is an extension of the work reported in Reference 5. The second part describes fundamental operations for changing the enterprise view of data. This is an area where very little work has been done. The operations proposed in this paper will be useful in maintaining the enterprise schema. The third part uses the E-R approach to analyze an example given by Bachman[10] concerning changes in the conceptual schema.

MODELING THE REAL WORLD USING THE ENTITY-RELATIONSHIP MODEL AND DIAGRAMMATIC TECHNIQUE

In this section, we shall use examples to show how to use the Entity-Relationship (E-R) model and diagrammatic technique to describe the enterprise view of data. A more formal definition of the model can be found in Reference 5.

It is assumed that the responsibility of defining and maintaining the enterprise schema belongs to a person called the *enterprise administrator*. The following is the suggested procedure for the enterprise administrator to define the enterprise schema:

(1) *identify entity sets of interest to the enterprise*

An *entity* is a "thing" which can be distinctly identified. According to the needs of the enterprise, entities can be classified into different *entity types* such as EMPLOYEE, STOCK_HOLDER. An *entity set* is a group of entities of the same type. In the E-R

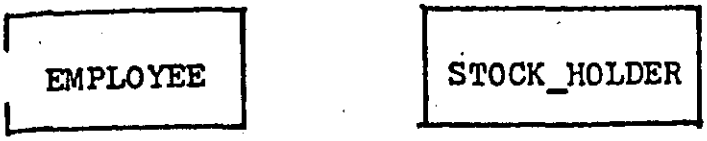


Figure 4—Entity sets

diagram, an entity set is represented by a rectangular-shaped box (see Figure 4). The terms, "set" and "type," can be interchanged in the E-R diagram. The reader may use either one to interpret the E-R diagram.

There are many "things" in the real world. In addition, different enterprises may view the same thing differently. It is the responsibility of the enterprise administrator to select the entity types which are most suitable for his company.

(2) *identify the relationship sets of interest to the enterprise*

Entities are related to each other. Different *types* of relationships may exist between different types of entities. A *relationship set* is a set of relationships of the same type. For example, PROJ_EMP, which describes the assignment of employees to projects, is a relationship set defined on two entity sets, EMP and PROJ. A relationship set can also be defined on more than two entity sets. For example, PROJ_SUPP_PART is a relationship set defined on three entity sets PROJ, SUPP, and PART. In the entity-relationship diagram, a relationship set is represented by a diamond-shaped box with lines connecting to the related entity sets (see Figure 5). The "m" and "n" associated with the PROJ_EMP relationship in the E-R diagram indicate that the relationship is an m:n mapping. That is, each employee may be associated with several projects, and each project may have several employees. In certain companies, each employee belongs to at most one project, and the PROJ_EMP relationship is a 1:n mapping.

There are many types of relationships between entities. The responsibility of the enterprise administrator is to select the relationship sets (or types) which are of interest to the enterprise. He also has to specify the type of mappings (1:1, 1:n, m:1, or m:n) of the relationships.

(3) *identify relevant properties of entities and relationships (i.e., define value sets and attributes)*

Entities and relationships have properties, which can be expressed in terms of *Attribute-value* pairs. "Blue," and "4" are examples of *values*. Values can be classified into different *types* such as COLOR or QUANTITY. A *value set* is a group of values of the same type. An *attribute* is a mapping from an entity set (or a relationship set) to a value set (or a group of value sets). For example, "address" is an attribute which maps entities in the entity set EMP to values in the value set NAMES_OF_LOC. Note that we relax the constraint imposed in Reference 5 that the mapping from the entity set to the value set has to be a function (i.e., m:1 mapping). In other words, we now allow that an attribute (such as address) can have several values (such as locations) for the same entity (employee). This relaxation in the definition of attribute will make the changes in the enterprise view simpler. This point will become clear in the next section.

In the E-R diagram, a value set is represented by a circle, and an attribute is represented by an arrow directed from the entity set (or the relationship set) to the desired value set(s) (see Figure 6). After selecting entity sets and relationship sets, the enterprise administrator identifies the attributes and value sets which are relevant to the company's operations.

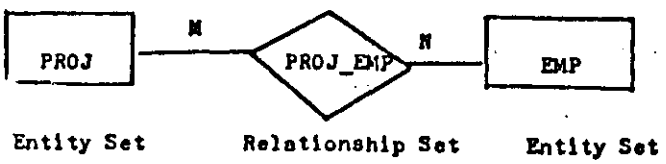


Figure 5—Relationship set

The three steps stated above cover a major part of the enterprise schema. For simplicity, we shall not discuss in this paper other issues related to the enterprise schema such as integrity constraints.

To design a database, the enterprise administrator first draws an E-R diagram such as the one shown in Figure 7. He then drew the attributes and value sets for each entity set and relationship set. The E-R diagram is then translated into a data-structure diagram or a set of tables ("relations") (see Figure 2). The rules and procedures used in the translation process were discussed in Reference 5. Here, we shall investigate how to change the enterprise schema (the E-R diagram) itself.

MODIFICATION OF THE ENTERPRISE VIEW

Although the enterprise schema is more stable than a user view, it still needs to be changed from time to time

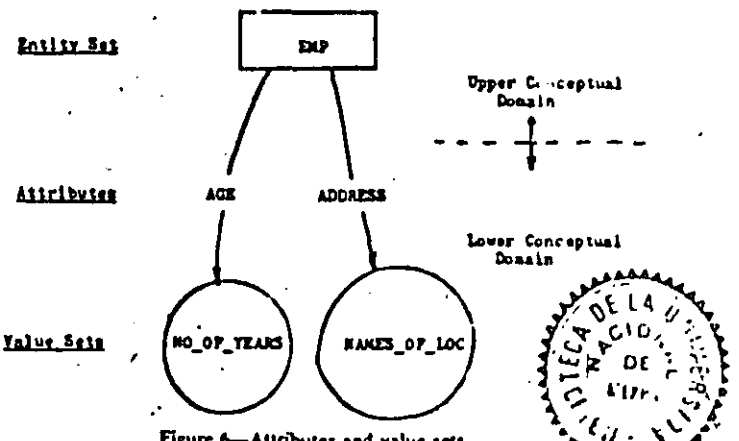


Figure 6—Attributes and value sets

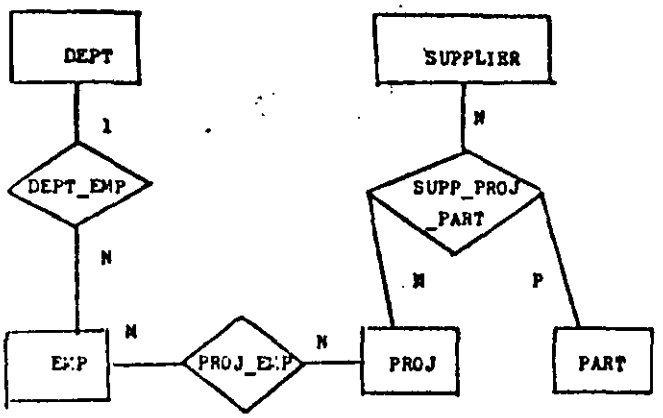


Figure 7—An entity-relationship diagram (with entity sets and relationship sets only)

to reflect the changes in the enterprise environment. Excepting a paper by Bachman,[10] very little work has been done in this area. In this paper, we use the E-R model as a basis for analyzing different types of changes in the enterprise view of data. We not only propose a set of operations but also analyze the consequences of these operations.

There are five basic types of operations: *add*, *delete*, *split*, *merge*, and *shift*. The first four operations are applicable to entity sets, relationship sets, attributes, and value sets. The *shift* operation is used when the enterprise administrator would like to view a value set in the old enterprise schema as an entity set in the new schema or vice versa. It is useful to think that the E-R diagram consists of two conceptual domains: (1) the upper conceptual domain which consists of entity sets and relationship sets; (2) the lower conceptual domain which consists of attributes and value sets. We shall discuss the first four operations in both the upper and lower conceptual domains. Finally, we shall discuss the shifting an entity set from the upper conceptual domain to the lower conceptual domain and the shifting a value set in the opposite direction.

Operations in the upper conceptual domain

The following are the basic operations applicable to entity sets and relationship sets:

(1) *Split an entity into several subsets*
For instance, the entity set EMP in Figure 8a can be split into two entity sets: MALE_EMP AND FEMALE_EMP in Figure 8b. The consequence of this operation is that the relationship sets associated with the entity set may also have to be split. For example, PROJ_EMP is split into PROJ_M_EMP and PROJ_F_EMP (see Figure 8b).

(2) *Merge several entity sets into one entity set*
This is the opposite operation of (1). The consequence is that the related relationship sets may have to be merged.

(3) *Split a relationship set into several subsets*
An example of this operation is: the relationship set PROJ_EMP in Figure 8a can be split into two relationship sets, PROJ_MANAGER and PROJ_WORKER, in Figure 8c. Note that the type of mapping in the new relationships may be different from that in the original relationship. For instance, the mapping in PROJ_MANAGER is 1:n while the mapping in PROJ_EMP is m:n.

(4) *Merge several relationship sets into one set*
This is the opposite operation of (3). Note that these relationship sets have to be defined on the same group of entity sets.

(5) *Add a new entity set*
For example, a new entity set called SUPPLIER may be added to the E-R diagram in Figure 8a. The result is shown in Figure 9a. Note that it is possible to have stand-alone entity sets in the enterprise schema, although in many cases relationships between the new entity set and the existing entity sets are established immediately (see the next operation).

(6) *Add a new relationship set*
We may add a new relationship set for the new entity set such as the relationship set PROJ_SUPP in Figure 9b. We may also add a new relationship set for existing entity sets such as the relationship set PROJ_MANAGER in Figure 9b.

(7) *Delete an entity set*
For instance, after deleting the entity set EMP in Figure 9b, we have Figure 9c. The consequences are:
(i) the relationship sets related to the entity set are

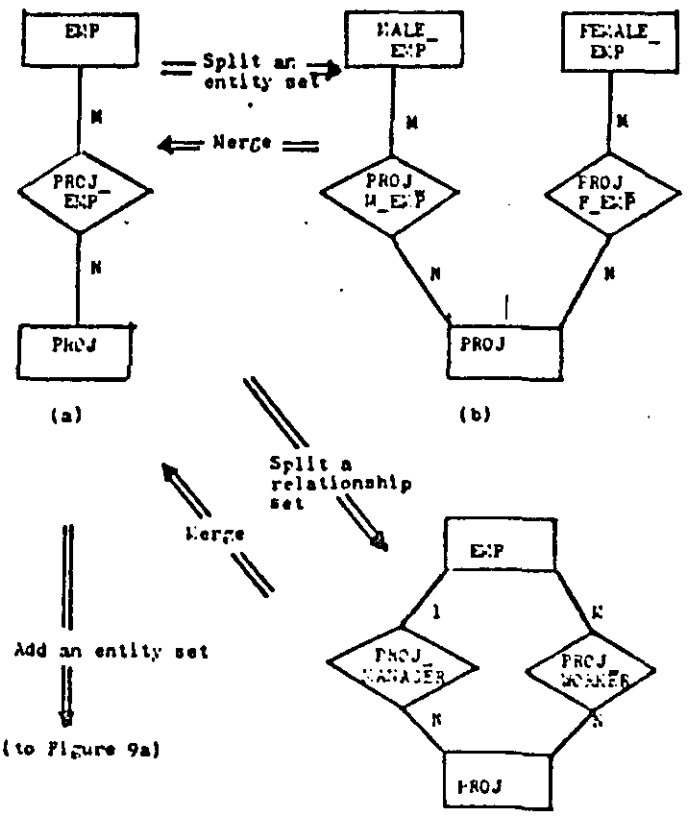


Figure 8—{Split / Merge} {Entity / Relationships} Sets

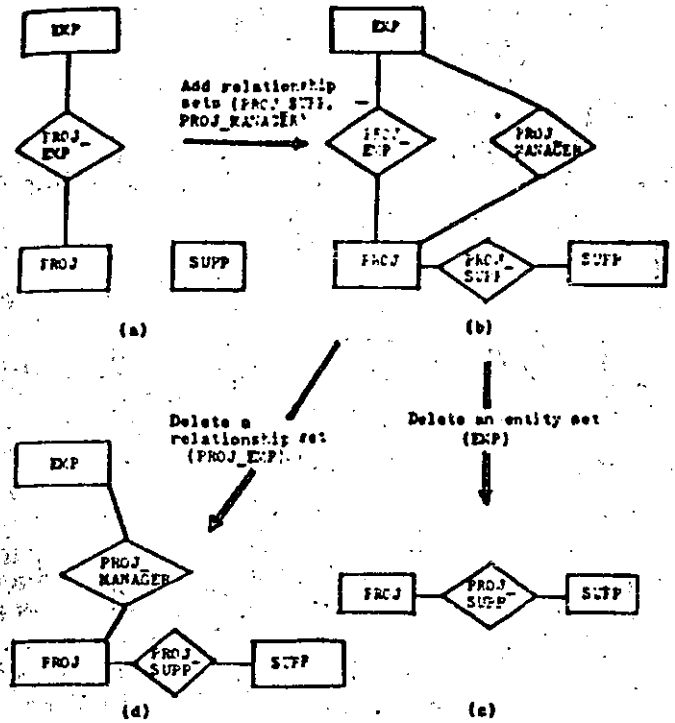


Figure 9—{Add, Delete} {Entity, Relationship} Sets

also deleted; (ii) attributes related to the deleted entity set and related relationship sets are also deleted.

(8) *Delete a relationship set*
An example is: delete the relationship set PROJ_EMP in Figure 9b, and the result is shown in Figure 9d. The consequence of this operation is that the attributes of the relationships are deleted (not shown in Figure 9d).

Operations in the lower conceptual domain

Assume that the entities in the entity set EMP have two attributes, LEGAL_NAME and PHONE, which map the entities to the value sets NAME and PHONE_# (see Figure 10a). We shall use these attributes and value sets as the basis for the discussion of the following operations:

(1) *Add a value set*
For example, a new value set called DOLLARS may be added to Figure 10a. The result is shown in Figure 10b. Usually, this operation is followed by an "add attribute" operation.

(2) *Delete a value set*
After deleting the value set PHONE_# in Figure 10a, we get Figure 10c. The consequence is that all attributes associated with this value set will be deleted.

(3) *Split a value set into several subsets*
The value set NAMES in Figure 10a may be split into two value sets: FIRST_NAMES and LAST_NAMES in Figure 10d. The consequence is that attributes related to the value set may have to be adjusted. Although the attribute LEGAL_NAME is not split in Figure 10d, it is possible to split it into two attributes: LEGAL_FIRST_NAME and LEGAL_LAST_NAME. It is the responsibility of the enterprise administrator to make this decision.

(4) *Merge several value sets into a value set*
This is the opposite operation of (3).

(5) *Add an attribute*
For instance, Figure 11b is obtained by adding the attribute OTHER_NAME to Figure 11a.

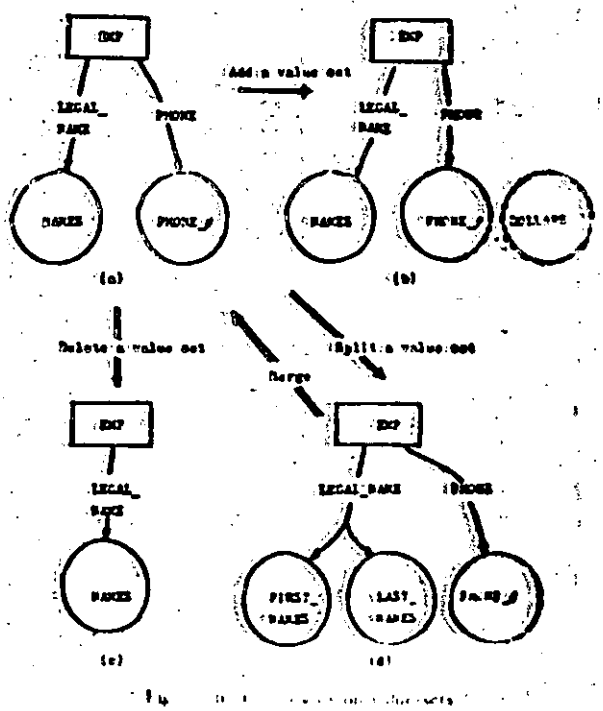
(6) *Delete an attribute*
Deleting the attribute LEGAL_NAME from Figure 11a, we have Figure 11c. The value set associated with the attribute will be deleted by another operation ("delete value set") if desired. In some cases, the value set may be still associated with other attributes (see Figure 11c).

(7) *Split an attribute into several attributes*
For example, Figure 11d is obtained by splitting the attribute PHONE in Figure 11a into two attributes, OFFICE_PHONE and HOME_PHONE.

(8) *Merge several attributes into one attribute*
This is the opposite operation of (7). The attributes have to be defined on the same entity set (or relationship set).

Operations between two conceptual domains

Assume that there are two entity sets (EMP and PROJ), one relationship set (PROJ_EMP), four value sets (NAMES_OF_PLACES, SOC_SEC_#, PHONE_#, and



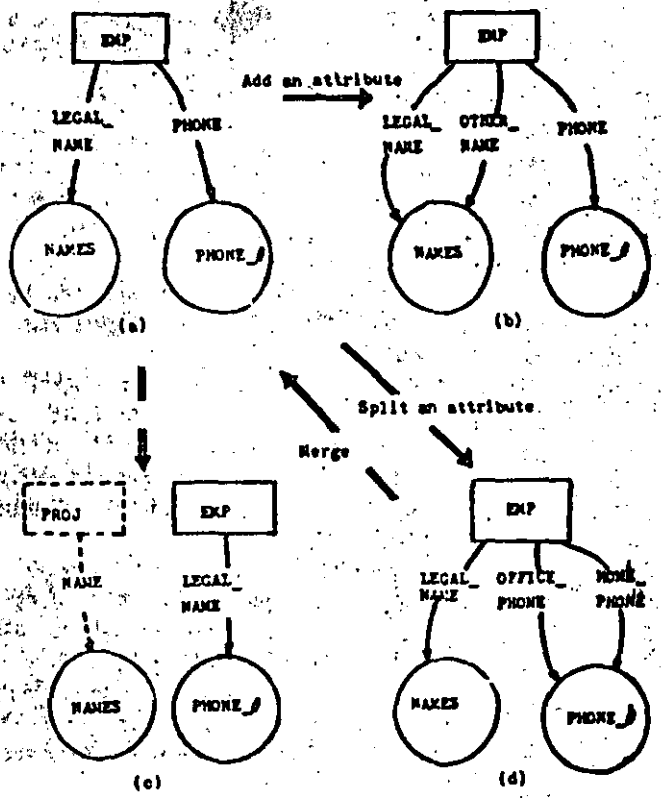


Figure 11—Operations on attributes

PROJ_NAMES), and four attributes (ADDRESS, SOC_SEC_NO, PHONE, and NAME) as shown in Figure 12a. We shall use them as the basis for the discussion on the following operations:

(1) *Shift a value set from the lower conceptual domain to the upper conceptual domain*
When the enterprise environment changes, it may become natural to view PLACE as an entity set instead of a value set. Thus, in Figure 12b "ADDRESS" becomes a relationship set, and "PLACE" has an attribute "NAME" which points to the value set NAMES_OF_PLACES. Since PLACE is an entity set, we may establish new relationships of it with other entity sets such as PROJ or add more attributes and value sets to describe properties of "places."

(2) *Shift an entity set from the upper conceptual domain to the lower conceptual domain*
When the enterprise environment changes again, it may become natural to view PROJ as a value set instead of an entity set. In Figure 12c, PROJ is deleted from the upper conceptual domain, and the relationship set PROJ_EMP becomes the attribute INVOLVED_PROJ. The entity set PROJ in Figure 12b may have been associated with several value sets, but only the value set PROJ_NAMES which is used to identify the entities PROJ remains in the lower conceptual domain.

ANALYSIS OF AN EXAMPLE

In a recent paper, Bachman[10] uses data-structure diagrams to illustrate the changes in a conceptual schema. In this section, we shall first state his example and then use E-R diagrams to interpret his example.

Description of the example using data-structure diagrams

The following is a simplified version of Bachman's example:

(a) In the beginning, the enterprise administrator declared a conceptual schema as shown in Figure 13a. The reader is assumed to have some knowledge of the data-structure diagram.[1] Simply speaking, a rectangular-shaped box represents a record type, and an arrow represents a data-structure-set (i.e., 1:n relationship between record types). In Figure 13a, there are two types of conceptual records, COMPANY and PERSON, and a data-structure-set "a" representing the fact that each person is associated with exactly one company and that each company has a set of personnel.

(b) Later, the enterprise administrator recognized that the personnel of the company were persons in their own right. This fact may be discovered at the manager

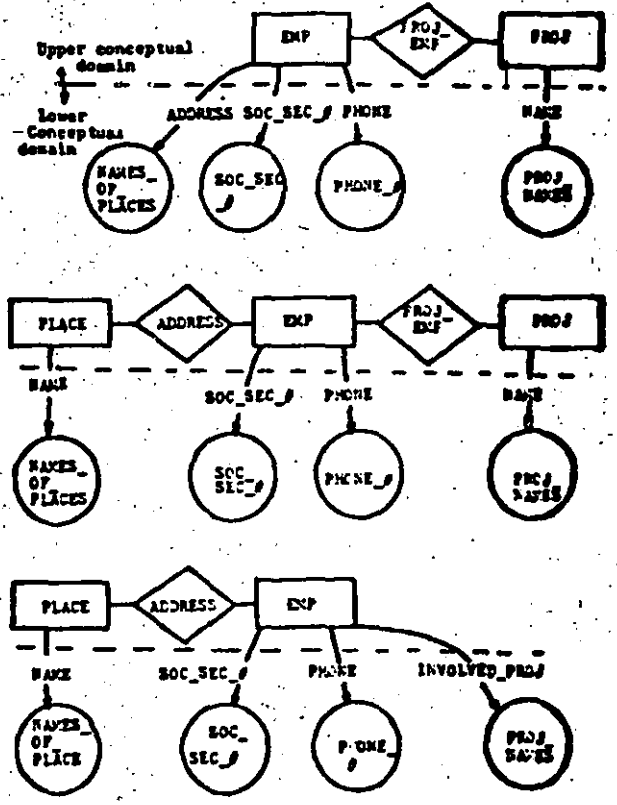


Figure 12—Shifting a set from the upper conceptual domain to the lower conceptual domain and vice versa

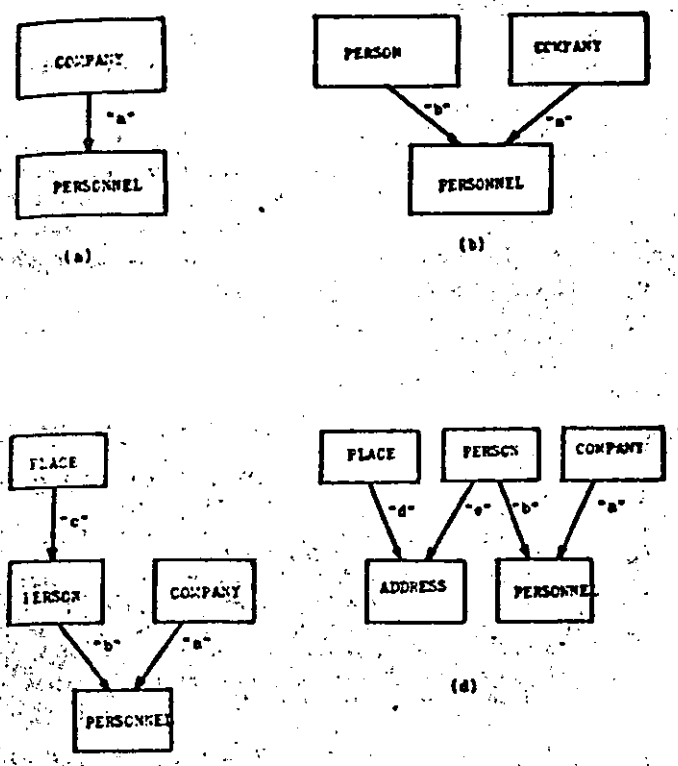


Figure 13—Expressing changes in the enterprise view using data-structure diagrams

of several companies that some of the persons held two jobs and were personnel to two of the merged companies. Figure 13b illustrates the data-structure diagram for the new conceptual schema. Basically, the old personnel type record has been split into two record types, PERSONNEL and PERSON. The "PERSON" has attributes NAME and ADDRESS (not shown in the figure).

(c) After a while, the enterprise administrator decided to factor the address of residence out of the person record. Figure 13c illustrates the addition of the "PLACE" conceptual record type and the data-structure-set type "c." It was also assumed that each person has a unique address (place).

(d) It is now recognized that people move from place to place and that it is desirable to know current address as well as past addresses. Another reason may be: it is discovered that a person may have more than one address. In either case, a new conceptual record type ADDRESS is added to the conceptual schema (see Figure 13d).

Analysis using entity-relationship diagrams

In the following, we shall use E-R diagrams to explain the above example:

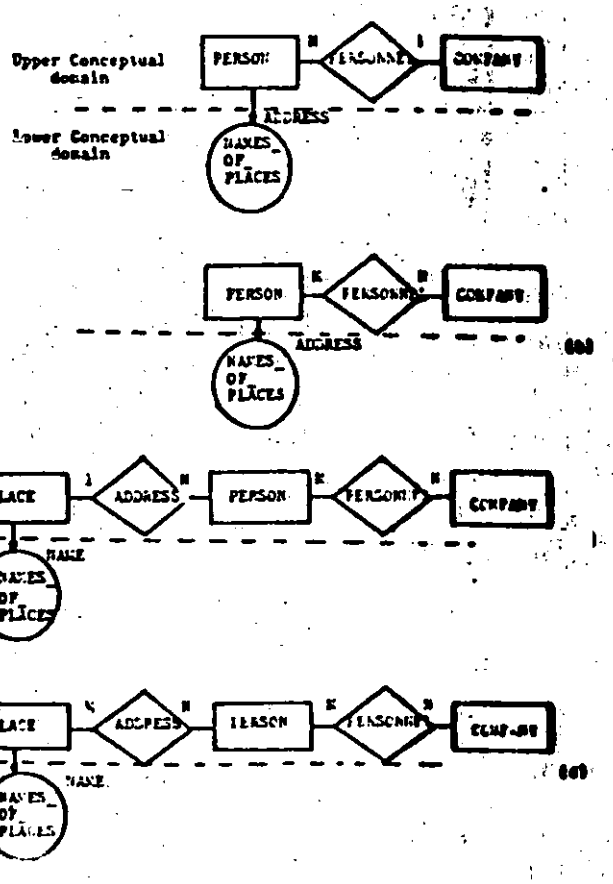
(a) The E-R diagram in Figure 14a is corresponding to

two types of entities, PERSON and COMPANY, in the enterprise view. Since the mapping between COMPANY and PERSON is 1:n, the relationship set PERSONNEL is represented by a data-structure-set "a" in Figure 13a.

(b) Figure 14b is the corresponding E-R [illegible]gram for Figure 13b. Since the relationship set PERSON[illegible] is an m:n mapping, it is represented by a relationship record type PERSONNEL and two data-structure-sets "a" and "b" in Figure 13b. Note that F[illegible] 14a and 14b have the same entity sets and relationship set in the upper conceptual domain, and the difference is the type of mapping between the [illegible] sets.

(c) Now the enterprise administrator prefers to "PLACE" as an entity set rather than a value. Thus, we have Figure 14c. The attribute ADDRESS in Figure 14b becomes a relationship set in Figure 14c. Since the mapping between PLACE and PE[illegible]SON is 1:n, the relationship set ADDRESS is represented by the data-structure-set "c" in Figure 13c.

(d) The enterprise administrator discovers that the mapping between PLACE and PERSON is an m:n mapping instead of a 1:n mapping. The new enterprise view is represented by Figure 14d. Since the mapping is m:n, the relationship set ADDRESS is represented by the record type ADDRESS and two data-structure-sets "d" and "e." Note that Figures 14c and 14d



are almost the same except that the type of mapping between PLACE and PERSON is different.

In general, the E-R diagram is easier to use to analyze the changes in the enterprise view than the data-structure diagram. Bachman also raised the issue of the ambiguity in Figure 13d: If one wants to modify a person's address, does he have to create a new "address" record or to change the name of the place where the person is living? This question can be easily answered using the E-R approach. Consider Figure 14d. Since the PLACE is an entity set, to change a person's address is to change the relationship between the person and "his place." We should not change the name of the place where the person is living since "NAME" and "NAMES_OF_PLACES" are used to describe a property of the PLACE entities (see Figure 14d).

SUMMARY

The enterprise schema is useful as an intermediate step in database design. In this paper, we have shown how to use the entity-relationship model and diagrammatic technique to describe the enterprise schema. Since the enterprise environment changes from time to time, the enterprise schema will have to change to reflect these changes. Five basic types of operations (add, delete, split, merge, and shift) which are useful in modifying the enterprise schema have been presented, and the consequences of these operations have been discussed. Finally, we have used an example to analyze the differences between the entity-relationship approach and the network approach in modeling the enterprise view of data.

REFERENCES

1. Bachman, C. W., "Data Structure Diagrams," *Data Base 1*, 2, Summer 1969, pp. 4–10.
2. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM 13*, 6, June 1970, pp. 377–387.
3. ANSI, *Interim Report of ANSI/X3/SPARC Group on Database Management Systems*, ANSI, February 1975.
4. Chen, P. P., "The Entity-Relationship Model," (abstract), *Proc. 1st Very Large Database Conf.*, Framingham, Mass., Sept. 1975, ACM.
5. Chen, P. P., "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Tran. on Database Systems 1*, 1, March 1976, pp. 9–36.
6. Moulin, P., J. Randon, M. Teboul, et al., "Conceptual Model as a Database Design Tool," *Proc. IFIP TC-2 Working Conf.*, Jan. 1976, Black Forest, Germany, pp. 459–479.
7. Hall, P., Todd S. Owlett, "Relations and Entities," *Proc. IFIP TC-2 Working Conf.*, Jan. 1976, Black Forest, Germany, pp. 430–458.
8. Deheneffe C. and H. Hennebert, "NUL: a Navigational User's Language for a Network Structured Data Base," *Proc. ACM 1976 SIGMOD Conf.*, Washington, D.C., June 1976, pp. 135–142.
9. Tozer, E. E., "Database Systems Analysis and Design," Technical report, Software Sciences Limited, England, April 1976.
10. Bachman, C. W., "Trends in Database Management—1975," *Proc. AFIPS 1975 NCC*, Vol. 44, AFIPS Press, Montvale, N. J., pp. 569–576.

DISEÑO DE BASES DE DATOS

A R T I C U L O

DESIGN AND IMPLEMENTATION OF AN INFORMATION BASE FOR DECISION MAKERS

EXPOSITOR:
Ing. Daniel Ríos Zertuche

MAYO, 1985

15

Design and implementation of an information base for decision makers*

by R. H. BONCZEK, C. W. HOLSAPPLE and A. B. WHINSTON
Purdue University
West Lafayette, Indiana

ABSTRACT

When considering the design and implementation of systems for decision support, a crucial point is the power and flexibility of available tools for representing data contexts. The value of such systems is constrained by the "richness" of patterning allowed by their data structure mechanisms. We introduce the notion of an information base as a natural step forward in the continuing evolution of data structures. The outstanding features of the information base are (1) its accommodation of the horizontal and vertical integration of information parcels into a single semantic mechanism, and (2) the integration of operators into this semantic structure.

INTRODUCTION

A topical and decidedly significant area of research involves the identification of those criteria which a computerized information system must satisfy if it is to be of value to non-programming decision makers. The ensuing discussion focuses upon such criteria and their implications for system design and implementation. In particular, we introduce the notion of an *information base* and demonstrate how it may be developed and implemented as an extension to the CODASYL DBTG[1] approach to data management. We commence with the characterization of an information base as a semantic network. It is then shown that this semantic network may be realized as an extension to an approach that has been used in commercial environments. Moreover, we illustrate how the information base serves as the cornerstone for a generalized decision support system.

Within the scope of this paper a distinction is drawn between the terms information and data. Observe, first of all, that information is an abstraction; it is not something which can be pointed to or seen. However it may be conveyed by patterns of "matter-energy,"[2] i.e., by configurations of symbols, by data. Data and information invariably accompany one another. The words on this piece of paper are not information, but rather a pattern of matter-energy which as a consequence of certain activities (e.g., inputing, transmitting, decoding, associating, storing, deciding, etc.) conveys information.[3] The important point is the patterning of data; the "richness" of a notation in terms of the kinds of data relationships which it can represent has obvious implications for its power in conveying information. With this in mind, we can note a pronounced trend in the history of information systems from the relatively impoverished linear data structure to the tree and network data structures, capable of a greater variety of data configuration; correspondingly the ease with which comparatively complex information can be conveyed has also grown. Summarizing, "... we can say that data is an objective notation which has no significance in itself, versus information as a subjective concept which relates a datum to a context."[4]

In order to understand the varieties of contexts or configurations in which data must appear if there is to be a comprehensive conveyance of information, we examine the field of semantics. Of special interest is the notion of a semantic net. The results of this examination constitute a basis for the specification of *information base* features which permit the unambiguous representation of all types of information pertinent to decision support applications. This representation must configure data such that all significant relationships among parcels of information (e.g., among facts, procedures, empirical information, etc.) are accommodated. Furthermore, these objectives for information base features must be met in a manner that is amenable to processing for the purposes of inference and deduction.

Since semantics deals with the *relationships* between symbols and what they denote or mean,[5] what we call the information base may be viewed as a semantic mechanism capable of representing meanings in terms of data configurations. Its storage technique must be general enough to handle the basic kinds of information involved in decision making regardless of the specific decision application. These types of information are: directive information, conceptual information, empirical information, stimulatory information, information about expectations, information concerning valuations, and procedural information. In addition the information base must be flexible enough to represent the often intricate interrelationships among information parcels, relating them so as to capture their full meaning

* Research supported by Office of Water Research and Technology Grant No. 6538-62-1310.

and impact with respect to other parcels of information. This latter point is particularly significant in that it furnishes a basis for the synthesis of separate parcels of information that are all related to the same object, concept, observations, etc.

Woods[5] defines a semantic network to be an attempt to combine into a single mechanism both the ability to store factual knowledge and the ability to model associative connections which render certain parcels of information accessible from certain others. Moreover he indicates three criteria which must be satisfied by a notation used for semantic representation:

1. Logical adequacy. The notation must provide an exact, formal and unambiguous representation of any particular interpretation that may be given to a sentence.
2. There must be an algorithm for translating an initial sentence into this notation.
3. There must be algorithms capable of using the semantic representation in order to perform needed inferences and deductions.

The information base detailed in the subsequent discussion will be shown to satisfy the definition of a semantic network. A query language will be described which, in conjunction with the information base, will be shown to satisfy the three requirements of notations for semantic representation.

FEATURES OF THE INFORMATION BASE

A specific design and implementation of the information base is described in a later section; this design and implementation is based in part upon the idea of a network data base advanced in the CODASYL DBTG Report of 1971[1] and subject to extensions and modifications outlined in Reference 6. The term information base, rather than data base, is used to emphasize its incorporation of two fundamental features which do not appear in the general data base management literature. Both features concern ways of patterning data that can convey information not commonly treated in the guise of data base management, but of value to decision makers; they furnish methods for introducing two novel kinds of context into data structures.

In observing the progression from linear structures to trees, to networks, we note increased facility for relating a datum with other data; there is an increased capacity for specifying the context of a datum in terms of data structures. Though there is little context inherent in linear data structures, the data content of groups of such structures may be used to represent trees. This becomes complex and cumbersome as the tree to be represented grows in size. Similarly, though it is possible to twist tree structures to the task of representing large or complex networks by using collections of tree-like structures, this cannot be accomplished in a facile, straightforward manner. This analogy may be continued with respect to the two features being introduced in this section. That is, they can, in some sense, be represented within network data structures, but such an approach leads to certain asymmetries (with respect to processing) and difficulties akin to those encountered when representing trees in linear structures. Since the two features are not inherent in the common notion of a network data base, we introduce the information base as a mechanism which encompasses both while allowing full network capabilities.

The first feature involves the introduction of the concept of resolution levels within the mechanism for information organization. A simple example of this is described by Winograd.[7] Consider data about cars in which specific weights and colors are related (linked) to each car; on a higher level of resolution, we may want to somehow store information about what the properties of cars are. So on one level of resolution we are interested in specific attributes of specific cars and on another level we are concerned with properties of cars. Thus two distinctive characteristics of the information base are links which integrate individual information parcels on a given level of resolution into a single network structure and secondly, the integration of information of varying levels of resolution into a single structure. We term the former characteristic "horizontal integration" and the latter "vertical integration." So horizontal refers to linkage of entities on the same level; whereas vertical denotes linkage among different levels via information parcels that participate in both levels (though the nature of participation is different on each level). A subsequent section of this paper describes both an implementation and the implications of this feature.

The second outstanding feature of the information base involves its ability to handle the integration of programs into its logical structure. Not only does this permit the linkage of a datum with a program that uses it; it allows the construction of networks (in both the horizontal and vertical sense) of programs. This capacity has two primary effects. First, it provides the basis for model formulation. Second, it furnishes a more comprehensive mechanism for semantic representation.

The aspect of model formulation involves the action of relating certain modules into a desired configuration. This necessitates a knowledge of which configurations are meaningful and which are not. Such knowledge is stored in the information base's semantic network. This approach has much in common with the notion of structured programming. Programs devised according to the tenets of structured programming[8] are readily amenable to storage within the information base; indeed there is also the ability to store alternative modules (e.g., alternative functional forms) for performing a particular role within the context of either other modules or a higher resolution level. The advantages of structured programming in terms of maintainability and extensibility[9] are also apparent in the strategy of integrating program modules into the logical structure of an information base. That is, it is possible to add, replace or delete a module in the same manner that one would add, replace or delete an occurrence of data.

It is useful at this juncture to point out a distinction

between program modularity and program resolution. The idea of resolution level also goes under the name of level of abstraction. Dijkstra[9] indicates that each level of a system's software hierarchy constitutes an abstract resource which participates in the next higher level and which has available to it the resources of lower levels. So "... at one level the programming amounts to manipulation of the abstract resources supported by the next lower level of the hierarchy. The programs at that level manipulate abstractions—the abstractions of the resource, whatever it may be—and at the same time participate in generating a higher level of abstraction for the next level of the hierarchy to manipulate."[10] Furthermore, Miller and Lindamood suggest that a "... highly modular implementation is one in which specific functions are performed by specific modules (and nowhere else); on the other hand, a system which preserves a hierarchy of abstract resources would appear to require modularity as a minimum, and perhaps a great deal more 'structure'."[10] Such a structure is effectively treated by the information base feature of resolution levels which allows the arrangement of program resources into levels of abstraction.

The second effect of allowing the integration of programs into the structure of the information base is the more comprehensive semantic representation that is permitted. Much literature about semantic networks is concerned with the network representation of English sentences (e.g., see References 5 and 11). These sentences consist of patterns of verbs and arguments. The typical decision maker who queries the information base requests the execution of some model (i.e., operators, verbs) using certain data (i.e., operands, arguments) as inputs. The usual data base structures handle information about arguments only; the meaningful operator contexts in which such arguments may appear is not represented in standard types of data base structures. A more detailed discussion and practical application of this feature of representing programs in an information base is presented in Reference 12. The remainder of this paper focuses on details and examples of the resolution level feature and on the utility of the information base as a device for semantic representation.

REQUIREMENTS FOR A DECISION SUPPORT SYSTEM

Recall that, in this paper, we are principally concerned with the information base from the standpoint of its contribution to the realization of a general decision support system. Although there are several facets involved in reaching decisions, we investigate three in particular: information access, model formulation, and analysis. The efficacy of a decision support system may be evaluated in terms of its flexibility, facility, scope, timeliness and cost in supporting these three facets.

With respect to information access there must be a mechanism for the systematic, integrated storage of all pertinent information. The information base outlined above provides just such a mechanism, through both horizontal and vertical integration and through its capacity to relate operators with each other and with arguments. Given such a storage mechanism there must be a technique for interrogating (and modifying) it that can be used by decision makers who are not computer experts or programmers. The query language for accomplishing this is presented later.

The second facet which must be supported is the activity of model formulation. This facet refers both to models that are subsequently used for purposes of analysis and to models in the sense of plans to be implemented. This is a crucial aspect for resolving unstructured problems and for supporting the exploratory aspects of decision making. In short, the decision support system must have a component for the generation and evaluation of alternatives for achieving a stated goal. As already indicated, the information base contributes to such an end.

The decision support system must also provide for the activity of analysis; i.e., the fitting of data with models and models with data, thereby resulting in some expectation, beliefs or knowledge. Implicit in the very nature of the planning activity is the dynamic quality of the interface between model and data; for even though a collection of data may be comparatively stable over some time period, both the problems and the models used for problem solving may be subject to frequent alteration. Notice that a model operates on a particular subset of the entire collection of operands available, and it requires a certain configuration of this data as input. We contend that the tedious, cumbersome task of interfacing data and models for purposes of analysis should be automatically handled by the decision support system in response to the commands of a non-programming user. The method for accomplishing this is discussed in subsequent sections.

FORMALIZATION OF THE INFORMATION BASE

We now present a formal description of what is meant by the term "information base." We define a record occurrence to be a uniquely labeled aggregate of data (i.e., string of symbols). Where I^+ is the set of positive integers, let X_0 be the set of labels associated with a finite set of record occurrences, such that $X_0 \subset I^+$. A record type, uniquely denoted by the label p_i, may be described by a function r_i as follows. Define R_k as the set of all $r_i: X_k \rightarrow \{0,1\}$ such that:

(1) $\forall x \in X_k,\ r_i(x) = \begin{cases} 1 & \text{If } x \text{ is of the type labeled } p_i \\ 0 & \text{otherwise} \end{cases}$

(2) $\forall x \in X_k,\ \sum_i r_i(x) \leq 1$

(3) $\forall r_i \in R_k,\ \sum_j r_i(x_j) > 0 \quad$ where $X_k = \{x_j\}$

Property (1) states that r_i defines the collection of $x \in X_k$ of the type labeled p_i. Property (2) indicates that each $x \in X_k$ can belong to at most one p_i. Property (3) states that each r_i is non-trivial.

Before defining X_k for $k>0$, we note that $P_k = \{p_i\}$ is the set of all labels associated with the elements of R_k. Since X_0

is finite, we can define these labels such that $P_k \subset I^+$, $P_k \cap X_0 = \emptyset$; furthermore we can define each of these sets of labels such that it has no elements in common with any other P_k. Define:

$$X_1 = \{p_1 \in P_0 \mid \exists x \in X_0 : r_1(x) \neq 0\} \cup X_0$$

$$X_2 = \{p_2 \in P_1 \mid \exists x \in X_1 : r_2(x) \neq 0\} \cup X_1$$

$$\vdots$$

$$X_k = \{p_k \in P_{k-1} \mid \exists x \in X_{k-1} : r_k(x) \neq 0\} \cup X_{k-1}$$

It follows from the definition of X_0 and R that there must exist a K such that $X_K = X_{K+1} = \ldots$; then let $X = X_K$. Observe that X is the set of labels of all record occurrences within an information base; these labels are unique identifiers, thereby serving as information base keys. All occurrences of a record type denoted by the label p can be determined by successive applications of the function r to the set X. The magnitude of K indicates the levels of resolution inherent in the information base. The reader will notice that P is always a subset of X; if it were not desired to treat all record types as record occurrences, one could define $X = X_{K-1}$. There are advantages to defining $X = X_K$, especially for purposes of altering the logical structure of an information base after it has been loaded. This will be elaborated in a subsequent section.

Continuing, we now formally define the information-set (in-set). This construct, as implemented in the information base, is drawn in part[8] from the "set" idea of the CODASYL DBTG Report,[1] hence the term "in-set." It is important to differentiate this from the familiar notion of a mathematical set. Let $Q_i = \{x \in X \mid r_i(x) \neq 0\}$. If a function associates each element of its domain with no more than one element of its range it is said to be a functional relation. Then each functional relation $f: Q_j \rightarrow Q_i$ uniquely defines an in-set of which the record type r_i is said to be the owner and the record type r_j is called the member. It is important to make several observations about the in-sets of an information base. It is permissible, and sometimes useful,[8] to allow $i = j$. Second, an in-set may be used to associate record types of different levels of resolution. Third, the set F of in-sets of an information base must be carefully defined so that its elements are consistent; e.g., one should exercise caution in defining both $f_1: Q_i \rightarrow Q_j$ and $f_2: Q_j \rightarrow Q_i$ as elements of F. Finally if $f_1: Q_j \rightarrow Q_i$ and $f_2: Q_i \rightarrow Q_k$, then we can form the composite in-set $f_1 \circ f_2: Q_j \rightarrow Q_k$ defined by

$$(f_1 \circ f_2)(x) = f_1(f_2(x)) \; \forall x \in Q_j.$$

This is sometimes desirable from the standpoint of access efficiency; it also allows us to attach special significance or meaning to certain groups of sets.

The foregoing is a formal description of the major features of the information base. It accounts for both the horizontal integration (via in-sets) and vertical integration (via resolution levels) of information into a single mechanism. In order to illustrate the use of resolution levels, we apply the above formalisms to the problem (see Winograd[7]) of representing information about cars. In this problem cars are to be described in terms of color and weight; in addition we would like to denote that color and weight are properties. Suppose we have record occurrences as shown in Figure 1a; these are identified by the respective labels in X_0. The set R_0 is also shown; by inspection we see that R_0 satisfies the needed conditions as given at the beginning of this section. The function r_1 determines whether or not an element of X_0 is of the type color. Similarly r_2 is associated with the type weight and r_3 is associated with the type car. In our implementation each r_i defines (and is defined by) a linked list of occurrences of its type. Given X_0, R_0, and P_0 we apply the rule for defining X_1 to obtain the result shown in Figure 1b. R_1 is also given and clearly satisfies the necessary conditions for its definition. Application of r_4 to elements of X_1 can be used to determine which elements are vehicle properties. Figure 1c gives the X_2 that follows from the definition. If we take $R_2 = \emptyset$, then $X = X_2$.

The occurrences and their "vertical" relations with each other are diagrammed in Figure 2. Also depicted are two in-sets: f_1 and f_2. Using the definitions of Q_1, Q_2, and Q_3 given in Figure 2, $f_1: Q_3 \rightarrow Q_1$ and $f_2: Q_2 \rightarrow Q_3$. The arrows in the diagram point from the owner of the in-set to the member; i.e., each arrow points in the direction opposite to that in the notation of its corresponding functional relation. Using the formalisms introduced here it is a simple matter to represent an extended problem including other kinds of vehicles,[13] more properties, subclassifications of properties (e.g., structural, functional, etc.) and even properties of properties.

An information base for water quality management

More detailed discussions of the water quality management problem may be found in References 14 and 15. The objective of the example presented in this section is to demonstrate the applicability of the information base as a

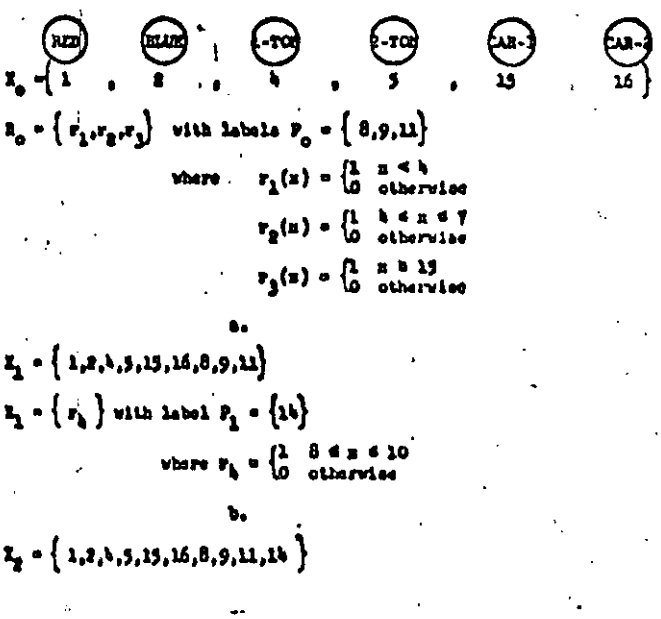


Figure 1—Resolution levels for representing information about cars

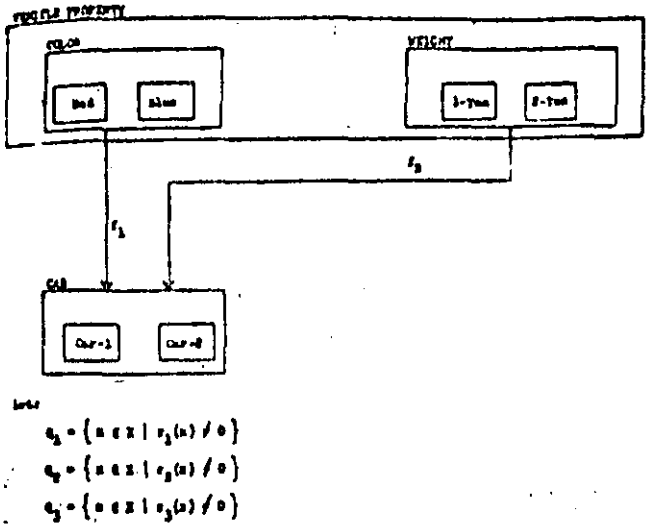


Figure 2—Occurrences in our information base

device for capturing the semantics used to support practical decision problems. At this point, we presume that the reader has a sufficient concept of what an information base entails to obviate the need for complete formalistic description. So for the sake of economy, the following example is presented in a less formal manner than the previous one. It will be used to depict certain implementational details (e.g., languages in which the information base is specified and with which it is utilized).

Consider the record type POLLUTER, displayed in Figure 3a. This aggregate of data item types represents measures of types of polluter activity for a given date. So occurrences of this record type correspond to measurements taken on various dates. In order to build a semantic network, we must indicate how this concept of POLLUTER fits into the pattern of knowledge concerning water quality management. A polluter is properly characterized as being a property of a river reach. Other properties of a reach include reach parameters, headwater, incremental flow, and treatment plan. So a reach is characterized in terms of these properties as follows: a reach is a portion of a river in which certain water quality parameters are relatively invariant; which has no more than one (point-source) polluter, one incremental flow or one headwater; and which must possess treatment plans. This could be represented in the information base by occurrences of the REACH PROPERTY record type displayed in Figure 3b. However, observe that each occurrence of the data item NAME (e.g., "POLLUTER," "HEADWATER," "PARAMETER," etc.) is also the label of a record type which is itself an aggregate of item types and which may have numerous occurrences. So, for instance, "POLLUTER" denotes an occurrence of REACH PROPERTY; but it also denotes a record type (shown in Figure 3a). The same circumstance holds for the other reach properties, though their record types are not depicted here. The resultant logical structure is illustrated in Figure 3c; a record type enclosed by another record type indicates that the enclosed record type is also an occurrence of the enclosing record type.

We continue the example by examining general water quality modeling characteristics. In order to simulate water quality we need information about the following: the rivers involved, the reaches which are in each river, each reach's properties, junctions, piping plans, and model parameters. This is shown in the structure of Figure 4. Note that the record type GMC has two item types: CHAR (characteristic) and IMPT (a measure of the relative importance of each characteristic). Five occurrences of GMC are shown: RIVER, REACH, JUNCTION, MODEL and PIPE PLAN. General Modeling Characteristic is not the only property of

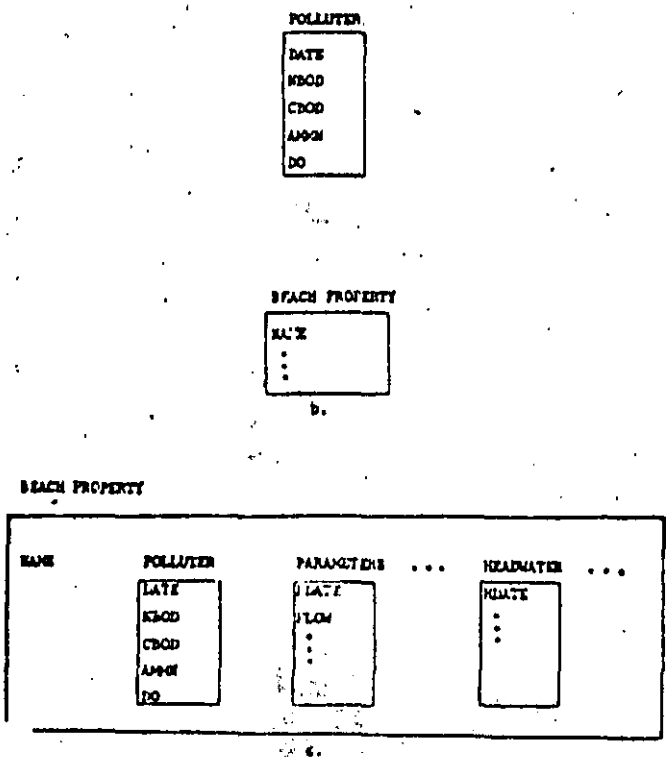


Figure 3—Attributes of a logical structure

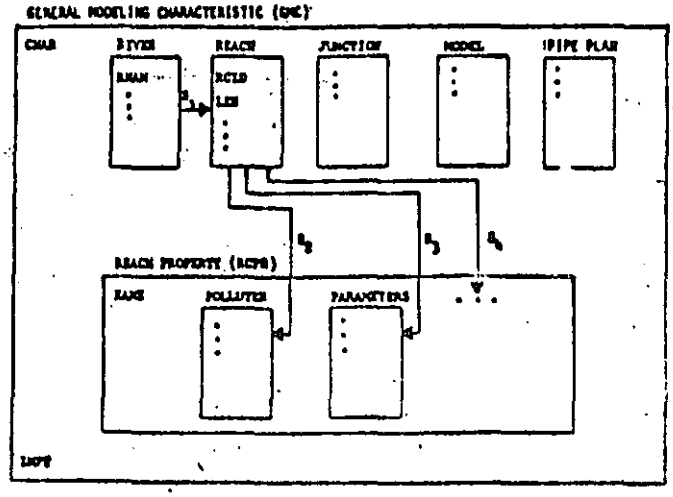


Figure 4—Example of logical structure (GMC)

a segment that needs to be represented; Local Modeling Characteristics (LMC) are also needed. (The term segment is used to indicate a particular area of a river basin.) The details of the high level record type LMC are not shown here, but they describe information about non-point sources of pollution, permits for point-source pollution, treatment plant construction status, permit violation data, etc. As shown in Figure 5, GMC and LMC are occurrences of SEGMENT PROPERTY which is itself an occurrence of the record type WQMA; BASIN and SEGMENT are also occurrences of this record type. The information base could be further extended to incorporate aspects of land use planning since they influence and are influenced by water quality management.

The foregoing logical structures are initially defined in terms of an Information Description Language (IDL). Use of the IDL to define the logical structure of Figure 4 is presented in Figure 6. The specification shown is largely self-explanatory. Each record type is followed by the item types which compose it. If the record type is of a high level, then its item types are followed by a specification of those record types which are its occurrences. Definition of an in-set must be preceded by specifications of its owner and member record types. For simplicity, details of the type and size of items are not shown; also the ordering criterion of each in-set is not shown.

A LANGUAGE FOR DECISION SUPPORT

The reader will observe from the preceding discussion that the decision support system has two basic components: an information base and a query language. Clearly the usability of a semantic network depends upon implementation of a language with which one can extract (insert) meanings that are held in the semantic net. Not only are semantics conveyed by a particular language, they are limited by it as well. The language is used to express meanings, but it also delineates the kinds of meanings which are expressed. We can devise arbitrarily complex semantic networks, but their usability is (from the practical standpoint) constrained by the languages (and language processors) which can be interfaced with them. Observe then that there is a fundamental duality of (1) the language in which ideas are expressed, and (2) the structural representation of ideas in an information base. On the other hand the semantic mechanism must be capable of taking full advantage of the language's power. In the case of the

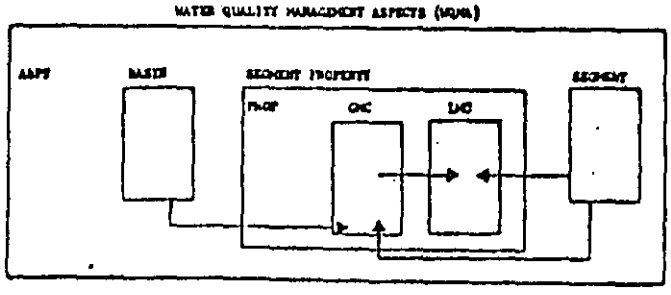


Figure 5—Example of logical structure (WQMA)

```
RECORD      GMC
ITEM        CHAR
ITEM        IMPT
  :
IN          GMC
            RECORD      RIVR
            ITEM        RNAM
              :
            RECORD      RECH
            ITEM        RCID
            ITEM        LEN
              :
            RECORD      JUNC
            ITEM        JID
              :
            RECORD      MODL
            ITEM        MID
              :
            RECORD      PIPE
            ITEM        PPID
              :
SET         S1
OWNER       RIVR
MEMBER      RECH
RECORD      RCPR
ITEM        NAME
  :
IN          RCPR
            RECORD      PLTR
            ITEM        DATE
              :
            RECORD      PARA
              :
            RECORD      HDW
              :
SET         S2
OWNER       RECH
MEMBER      PLTR
SET         S3
OWNER       RECH
MEMBER      PARA
  :
```

Figure 6—Example of IDL for Figure 4

implementation described in this paper, the query language is the constraining factor since it is intended primarily for the practical support of decision activities of managers in both the public and private sectors.

Implementation of a natural language (e.g., English) processor is certainly a noble objective. It is our experience that the typical decision maker neither uses, nor needs, a complete facility for conversing in a natural language. It often happens that phrases or clauses are sufficient to convey an idea; there are grammatical constructs (e.g., reflexive, passive) which are not particularly germane to the decision activities of information access, model formulation, and analysis. In addition the decision maker is more prone to desire information conveyed in a tabular or graphical fashion than in a narrative mode. It has also been found that the user sitting at a computer terminal has a tendency to use abbreviations and concise mathematical notation.

With these factors in mind, the query language to be outlined here has been designed to meet the needs of decision makers for flexibility and brevity of expression, while at the same time being easy to learn and utilize. The query language is effectively a subset of English that has been extended to include standard mathematical operators (i.e., relational, arithmetic, and univariate and multivariate functions). The focus here is upon use of this language for interrogation, though it may be used for data creation and modification as well.[8]

The standard framework of the language consists of a collection of operators (used in the capacity of verbs, adverbs and adjectives) relating to operations typically performed by most decision makers: these operators are of two kinds: commands (e.g., LIST, PLOT, STAT, REGRESS, etc.) and mathematical operators (e.g., MAXIMUM, AVERAGE, =, +, <, etc.). In addition to this standard framework the user may define arguments (used in the capacity of nouns), synonyms for arguments and operators, and any further operators (i.e., programs to be integrated into the information base) that are mundane to the particular decision making application to be supported. This definition is effected in terms of an Information Description Language (IDL) which establishes the context(s) of all data, arguments and operators. That is, it defines the semantic net.

Details of the query processor are not discussed in this paper but may be found in References 16–18. Briefly, the query language has a context-sensitive grammar; inverse transformations are used to take a surface structure query into a deep structure expression in a language having a context-free grammar. This deep structure expression is compiled using well-known methods of syntax-directed analysis. Parts of the compiled expression are used as input to network traversal routines which make extractions from the information base for use in analysis indicated by the query's command (verb).

The query's syntax appears as follows:

(COMMAND)(FIND clause)(CONDITIONAL clause)

or alternatively,

(CONDITIONAL CLAUSE)(COMMAND)(FIND CLAUSE).

So some sample queries are:

LIST REACH.NAME,REAERATION.PARAMETER AND REAERATION.EXPONENT, FOR DATE=110175 AND REACH.LENGTH<.9

WHEN DATE=110175, PLOT REACH.NUMBER VERSUS AMMONIA.CONCENTRATION AND DO.CONCENTRATION/LOG(TEMPERATURE)

LIST GENERAL.MODELING.CHARACTERISTIC IF IMPORTANCE>3

The language allows any meaningful configuration of arguments and mathematical operators to appear in the FIND and CONDITIONAL clauses.

PROCESSING HIGHER LEVEL RECORD TYPES

Upon receipt of a query, the query processor generates appropriate commands for traversal of a multi-level network. These commands are operators in an Information Manipulation Language (IML). We use the term IML to distinguish from the Data Manipulation Language (DML) proposed in the CODASYL DBTG Report.[4] The DML is intended to permit access, modification and retrieval for a single level network data base. The IML has the more extensive function of furnishing tools for manipulation of the information base. Thus the IML contains operators for handling traditional DML functions[16] and operators for processing higher level record types. The latter are discussed here.

In the DML, routines exist for creating a record occurrence at a unique location denoted by its key. The IML includes analogous routines for specifying that an existing record occurrence be treated as a record type as well. There are four such commands:

CRTK—Create Record Type based on a given Key

CRTR—Create Record Type based on the current occurrence of another Record type

CRTO—Create Record Type based on the current Owner of a given set

CRTM—Create Record Type based on the current Member of a given set

Another traditional DML operator (AMS) adds a specified record occurrence as a Member of a given Set. A similar IML operator is used for adding an existing occurrence of one record type as an occurrence of another record type. Note that utilization of this operator must be preceded by a generalization of the definition of a record type which was introduced above (i.e., the definition is generalized by removing the restriction that $\Sigma_i\, r_i(x) \leq 1$, $\forall x \in X_k$, where $r_i \in R_k$). This operator is AORT, Add Occurrence to Record Type, and it uses the key of the occurrence to be added. In conjunction with commands for the logical restructuring of a network data base,[6] AORT provides the ability to add and delete higher level record types and add existing occurrences to higher level record types; and this is accomplished without dumping and reloading data.

Finally operators are needed for determining the key of a record type, given an occurrence of the record type. These commands are:

GKRR—Get Key of the Record type for the current Record occurrence of that type

GKRO—Get Key of the Record type whose occurrence is the current Owner of some set

GKRM—Get Key of the Record type whose occurrence is the current Member of some set

These operators provide the capacity to proceed from a lower level occurrence to a higher level occurrence, when used in conjunction with traditional DML operators.

It must be emphasized that the typical user of the query system needs to have no knowledge of the IML operators, for they are automatically set up and executed by the query processor in response to a user query.

ADVANTAGES OF THE RESOLUTION LEVEL FACILITY

We contend that the concept of resolution levels effectively adds a new dimension to the field of information storage. The preceding discussion has suggested a means for operationalizing this concept as an extension to the traditional single-level network approach. One advantage is that multi-level semantic networks may be stored without introducing asymmetry in the interpretation and processing of in-sets and record types. Since a record type may also be defined to be an occurrence of a higher level record type, the addition of a record type is treated by creating a new record occurrence at the next higher level. That is, we remove the distinction between data values and the structural pattern according to which data is organized. In other words, the terms "attribute" and "value" are recognized as being relative, so that what is a value on one level is an attribute on another and vice versa.

From one viewpoint this abolishes the special status of an IDL specification by permitting record type definition to be a dynamic process. That is, the creation of a new record type is synonymous with the creation of a new record occurrence of a higher level record type. Thus the IDL specification of the highest level of resolution is effectively reduced to the definition of three record types (one describing information about record types, another relating to information about sets, and one with various system information[6]) and some in-sets between them. This definition is always the same regardless of the content and structure of lower resolution levels.

A second advantage, already mentioned in connection with integration of programs into the information base, concerns a mechanism for handling levels of abstraction in software. A third advantage is that higher level record types may be used to characterize areas of an information base by assigning record types of a particular area to be occurrences of a higher level record type; these areas may be defined for a variety of reasons (e.g., for information security, to denote scenarios, to delimit functional areas—which may overlap, etc.). As the information base becomes large and varied in content, this technique may also be used to realize efficiencies in path determination processing by limiting the scope of network traversal to a particular information base area.

THE INFORMATION BASE AS A DEVICE FOR SEMANTIC REPRESENTATION

With the foregoing background, we can now address the three criteria proposed by Woods,[5] which must be satisfied by a notation used for semantic representation. First observe that the information base is a tool for the representation of a semantic network (i.e., a single mechanism with both the ability to store factual knowledge and the ability to model associative connections which render certain parcels of information accessible from certain others).

The first criterion of a notation for semantic representation is logical adequacy. The notation must provide an exact, formal and unambiguous representation of any particular interpretation that may be given to a sentence. Recall that the sentences with which we are concerned are those allowed in the query language for decision makers.

The information base allows a given query to have a multitude of interpretations. The query specifies a group of data items which may be related to each other in many ways via vertical and horizontal linkages in the information base. Each path of linkages on which these items lie corresponds to a particular interpretation of the query. Upon receiving a query which is subject to multiple interpretations the query processor prompts the system's user in order to ascertain which interpretation (i.e., path) is intended. Details of the manner in which this has been implemented may be found in References 16 and 18.

The second criterion is that there must be an algorithm for translating an initial query into the notation of the information base. This is the central function of the query processor whose operation has already been described; implementational details appear in Reference 13. The third criterion, concerning algorithms capable of using the semantic representation, has also been addressed in the discussion of the query language. Observe that the IML provides the means for interfacing algorithms with the semantic representation.[16] Algorithms which have been used range from relatively commonplace report generators to large scale water quality simulation models.[15]

CONCLUSION

In the beginning we observed that it is the patterning of symbols which can convey information; a datum's meaning derives from its context, from its relationships with other data. Thus when considering the design and implementation of systems for decision support, a crucial point is the power of available tools for representing contexts. The value of such systems is constrained by the "richness" of patterning allowed by their data structure mechanisms. Observing the progression from relatively impoverished linear structures to trees and networks, we note that each stage has provided a more powerful and flexible tool for semantic representation. In this paper we have introduced the notion of an information base as a natural step forward in the continuing evolution of data structures. An outstanding feature of the information base is its accommodation of both the horizontal and vertical integration of information parcels into a single mechanism. An information base implementation which builds upon network concepts was discussed. A topic for future research is the investigation of an information base implementation which builds upon the relational data base notions.[19] A second distinctive feature of the information base, namely the integration of operators into its structure, was briefly described. The information base is utilized by a non-procedural, English-like query language, that has been designed for decision support applications. This language, in conjunction with the information base, satisfies the requirements for a notation for semantic representation.

ACKNOWLEDGMENTS

The authors are indebted to Professor Walter Reitman of the University of Michigan, Professor Tibor Vamos of the Hungarian Academy of Sciences and Dr. Bertram Raphael of SRI for useful discussion on semantic network and data base management.

REFERENCES

1. CODASYL, Data Base Task Group Report, ACM, April 1971.
2. Miller, J., "Living Systems: The Organization," *Behavioral Science*, Vol. 17, January 1972.
3. Kneitel, A. M., "Hard vs. Soft Information," *Management Datamatics*, June 1976.
4. Albarda, J. D., *Structures and Relations in Information*, Groningen, Druk: V.R.B. Offsetdrukkerijl, Rotterdam, 1974.
5. Woods, W. A., "What's in a Link: Foundations for Semantic Network," in *Representation and Understanding* (ed. Bobrow, D. G. and A. Collins), Academic Press, New York, 1975.
6. Bonczek, R. H., C. W. Holsapple, and A. B. Whinston, "Extensions and Corrections for the CODASYL Approach to Data Base Management," *International Journal of Information Systems*, Vol. 2, 1976.
7. Winograd, T., *Understanding Natural Language*, Academic Press, New York, 1972, pp. 23–27.
8. Dijkstra, E. W., "Notes on Structured Programming," T.H.E. Report No. EWD-248, 70-WSK-03, 2nd Edition, April 1970.
9. Donaldson, J. R., "Structured Programming," *Datamation*, December, 1973.
10. Miller, E. F. and G. E. Lindamood, "Structured Programming: Top-down Approach," *Datamation*, December 1973.
11. Heidorn, G. E., "Automatic Programming Through Natural Language Dialogue: A Survey," *IMB Journal of Research and Development*, July 1976.
12. Bonczek, R. H., C. W. Holsapple and A. B. Whinston, "Implementation of a Decision Support System for Regional Water Quality Planning," Krannert Institute Paper No. 570, Purdue University, West Lafayette, Ind., September, 1976.
13. Bonczek, R. H., "Theoretical Description of Access Language for a General Decision Support System," Doctoral Dissertation, Purdue University, 1976.
14. Haseman, W. D., C. W. Holsapple and A. B. Whinston, "Implementation of a Large Scale Water Quality Data Management System," *Socio-Economic Planning Sciences*, Vol. 10, March 1976.
15. Holsapple, C. W. and A. B. Whinston, "Decision Support System for Area-wide Water Quality Planning," *Socio-Economic Planning Sciences*, Vol. 10, 1976.
16. Haseman, W. D. and A. B. Whinston, *An Introduction to Data Management*, Richard D. Irwin Co., Homewood, Illinois, 1977.
17. Bonczek, R. H., W. D. Haseman and A. B. Whinston, "Structure of a Query Language for a Network Data Base," Technical Report, Krannert Graduate School of Management, Purdue University, April 1976.
18. Bonczek, R. H., W. D. Haseman and A. B. Whinston, "Automatic Path Determination in a Network Data Base," Technical Report, Krannert Graduate School of Management, Purdue University, April 1976.
19. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications ACM* 13 (6), June 1970.

DISEÑO DE BASES DE DATOS

A R T I C U L O

PHYSICAL OBJECTS, HUMAN DISCOURSE AND FORMAL SYSTEMS

EXPOSITOR:
Ing. Daniel Ríos Zertuche

MAYO, 1985

19

... in Data Base Management Systems, G.M. Nijssen, (ed.).
... Publishing Company, 1977

PHYSICAL OBJECTS, HUMAN DISCOURSE AND FORMAL SYSTEMS

Ronald Stamper
London School of Economics
Houghton Street, London WC2

The central problems of database design, for the information analyst, are how to select the appropriate universe of discourse and how to relate the data structures, within the information system, to the real entities in the world. These are the problems of operational semantics and they are quite different from the formal semantic problems which arise in the study of data management. Their solution depends upon our being able to rely upon the stable norms of human discourse which are established when people use language and other signs for some practical purpose. A conceptual schema embodying the operational semantics of the system should not be confused with a general schema to contain a canonical data structure. The issues raised are illustrated by concentrating upon the simplest of semantic problems: the identifying of physical objects. The analysis presented arises from the LEGOL Project[a].

Introduction - Information Analysis

By examining the simplest problem of information analysis we shall see that a database is not a model of reality but an embodiment of a myth. What could be simpler than the use of data elements to represent the physical objects, the tangible reality represented by a database? That is the problem we shall examine.

The information analyst specifies what data are required to solve some class of organisational problems. The designer of the information system must ensure that the data in the formal system are correctly linked to what he will call the 'real-entities' in some 'object system'. They create an appropriate myth. The reality lies in the operational success of the system. If the system helps us effectively to solve some set of practical problems, then the myth upon which it is based is real enough. Change the problem, change the purpose of the system, and the myth may become inappropriate. A database, for all the logical precision of its structure rests on this quicksand. Only a continual analytical vigilance can keep it afloat.

Information analysis is being studied at the London School of Economics in a study of administrative systems based on complex rules. This, the LEGOL Project, uses statute law as experimental material. By attempting to devise a formalism which can express the kinds of rules that might appear in a statute defining a tax system, for example, it is hoped to discover a way of specifying an information system at a very general level. A second prototype interpreter for this language is now being designed, an essential part of which is a semantic model. The result of information analysis may be viewed as a semantic model for an application, possibly in some area of law. It appears that the semantic model of LEGOL is the

[a] The LEGOL Project is supported by the UK Scientific Research Council with assistance from IBM(UK) Scientific Centre, Peterlee.

same thing as the conceptual schema of database studies[a]. If this is so, then the LEGOL Project suggests that recent discussions of the conceptual schema have failed to recognise some important ideas and to distinguish two quite different lines of enquiry. The purpose of this paper is to explain these ideas and the method of enquiry being employed in the research.

Real World and Formal System

In the discussions of conceptual schemas there is evidence that two quite different problems have not been adequately separated. Fig. 1 is intended to emphasize some major features of our problem.

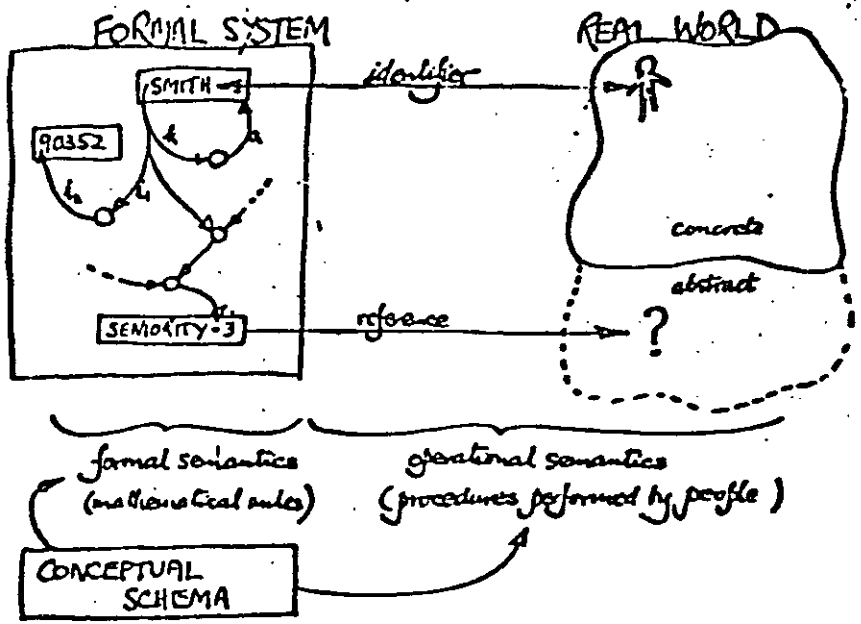


Fig1: Two aspects of semantics, formal & operational

The formal system contains strings of symbols - ink marks on paper or electromagnetic traces - and some of these may constitute an identifier of a person. The real world may contain the real person himself - you may meet him and shake him by the hand.

[a] This was not obvious to us on reading the ANSI-SPARC Interim Report (1) but this is the firm opinion of Dr. T.B. Steel, Chairman of this Committee, expressed at IFIP TC2 meetings in Pont-a-Mousson, September 1975, and Nice, January 1977

To a manager or administrator, who wants, say, to train his staff appropriately the identifier is the link to the person he can shake by the hand. It is easy to put into a database many millions of strings of symbols but the manager wants to know that the 'facts' they express are a reliable basis for action. To imagine that a database contains knowledge in this sense, merely because its structures of hierarchies, relations or networks can yield strings of printed symbols which may be construed as statements is naive. The manager's is an epistemological problem. No mathematical analysis can guarantee to the manager that the statements elicited from a database by his enquiries constitute 'real knowledge'. This can only be done by ensuring that there are operationally effective procedures for giving things names and for finding the real thing, given the name. By 'operationally effective' is meant that the procedure can be carried out, by the people who must use the identifier, with sufficient reliability to enable the information system to do a useful job*. The epistemological semantics of the database are embodied in these procedures which people perform. They may be difficult to establish and costly to support.

Mathematical methods, however, are appropriate to formal semantics. If we ask for the meaning of one string of symbols in terms of others, we need never leave the formal system in which mathematics can legitimately describe these relationships. In Fig. 1 the formal equivalence of 'SMITH' and the employee number '90352' is one such relationship. Other formal relationships, as the figure suggests, may link an abstract concept such as 'SENIORITY' to the more concrete concepts. These formal operations must be added to the operational procedures in order to establish the epistemological semantics of an abstract concept. This can be seen from the notion of 'SENIORITY' which is only connected to the concrete world via the formal procedures **. Hence formal semantics is a proper subset of epistemological semantics. The conceptual schema must therefore contain descriptions of both the internal, formal operations and the external procedures to establish the meanings of data.

Two Conceptual Schemas

Within Fig. 1 there is lurking a second, quite different problem. The formal system depends upon mapping the strings of symbols it contains upon suitable storage structures. This is as true of a clerical system as of a computer system but in the latter case it is crucial to the design of expensive general purpose software. This is where the second notion of a conceptual schema arises. It is a quest for a canonical structure to which all mappings of sub-sets of the data onto physical storage may be related. This view was exemplified in the statement by Nijssen (11) which placed the one conceptual schema between many external schemas (serving different sets of programs and many internal schemas (serving groups of physical groups of physical storage devices)). This kind of conceptual schema is naturally sought if one is preoccupied with the computer, the efficient use of hardware and the writing of programs. These problems are large enough to warrant the undivided attention of some researchers.

A serious mistake is made if we confuse these two notions:

CS(A) a schema of application concepts
CS(D) a canonical structure of data relationships

Both are essential but they serve quite different purposes and, of course, they are related. Confusion, regrettably, is more common than a careful differentiation of CS(A) and CS(D). One reason is our habit of using mnemonic labels to talk about formal data elements. By discussing CS(D) in terms of examples where data elements are called EMPLOYEE NO., DEPARTMENT NO., and so on, we import into the discussion notions that belong to CS(A). With care we can avoid cheating by using our intuitive knowledge of the semantics of such elements but the ease with which we feel we can handle the epistemological problems of EMPLOYEE NO., DEPARTMENT NO. and so on leads us to suppress the problems of CS(A). The result is to imagine that a canonical structure of suitably labelled data elements will capture both the contents of CS(D) and those of CS(A) as well. This assumption is implicit in the majority of papers on database architecture (12, 13, 14). The view adopted here, however, is that the application concepts, whilst embodying structures that must be reflected in the canonical data structure, must embody a great deal more having nothing to do with the formal properties of the database.

To exemplify this, consider a description of a house. We want to know:

(a) Numbers of bedrooms and reception rooms
(b) Is it fit for human habitation?
(c) Is it free of land charges?

For the information system at large and the epistemological semantics of the data it is essential to know that

(a) is a relatively objective attribute which can be supplied by any normal person who goes to check,
(b) is a value judgement which must be made subject to certain constraints, by a qualified surveyor, whilst
(c) can be determined by computer search of Land Registry files where all such charges must be registered to be valid.

Such semantics are irrelevant to CS(D). For the designer of the information system these concepts are fundamental and they belong in the CS(A).

Following the information analysis phase and the specification of CS(A) a limited amount of information about the application must be passed to the designer of the computer system. As it relates to the data, this limited information may be lodged in CS(D). Essential questions we need to ask are: what information can be discarded as we move from CS(A) to CS(D)? and in CS(D) what canonical forms are appropriate for the data structures?

To see how these questions can be answered we must remind ourselves of the technical problem which CS(D) help us to solve: how to navigate through storage volumes to find the data elements wanted by programs. Obviously this entails naming the data elements so that they can be allocated to storage in any way appropriate to the storage topology (this allocation is contained in the internal schema). The names used in the CS(D) could be mnemonics but they need not be. Let us suppose that they are not, so that there is no temptation to transfer subconsciously or covertly any knowledge of the epistemological semantics. What then do we understand about the external schema which maps the data names used in application programs into the canonical form of the CS(D)? A CS(D) full of data names without self-evident meanings will not help the application programmer to set up a mapping of his data elements into those of the canonical data model. He will be forced back to CS(A) where he may find that the meanings of data already in the database are not appropriate to his new task.

---

* Notice how this statement emphasises the importance of knowing the purpose of the system, a recurrent theme in this paper.

** Abstract concepts e.g. 'GUILTY' may not be entirely formal but, to a greater or lesser extent, the embodiment of someone's value judgement or prescription. This is dealt with in general terms in (3) and briefly in the context of LECOL in (1). Space does not permit the exploration of this topic in this paper.

Not only names of data elements but relationships among these elements must be contained in CS(D). To investigate the signification of these relationships, we can strip the problem bare by not permitting mnemonics to be carried across from CS(A) to CS(D). If we do this, it is clear that 'logical' notions such as 'entity', 'attribute', 'role' and so on, cannot be used as in CS(A) when talking of real things. Our database software is intended to help us 'navigate' through storage to the data we want. The CS(D) is the 'chartroom', 'Logical' relationships are significant because they define routes among data elements that must be available and may be used quite often. From this point-of-view, the dispute between the devotees of binary and n-ary models falls into perspective. If the computer permits only sequential processing then one must travel between single data elements and the binary model is natural basis for the canonical form such as Senko describes in (15). If the computer is more appropriately used as a parallel processor, then groupings of elements which are necessarily and only necessarily related will be important structures to identify. This leads to Codd's third normal form. Something is gained and something is lost which ever canonical form is used. The psychological discomfort of the binary model is pointed out by Senko in his paper. The artificiality of the binary model is an advantage if you wish to avoid confusing CS(A) and CS(D). The relational view of data is convenient but dangerous at the CS(D) level but at the CS(A) level relational structures seem to be essential. At least this is strongly suggested by our studies of information analysis for legal systems.

Having attempted in this section to draw a demarcation line between application concepts, CS(A) and canonical data structures, CS(D) we shall turn to the CS(A) without the risk of becoming involved simultaneously in two sets of difficult and complex problems. We introduce these ideas as they arise in the LEGOL Project.

The concrete and the abstract worlds

The experimental material used by the LEGOL Project is legislation of the kind upon which routine administrative systems are based. These systems of rules are constructed so that they are grounded, as far as possible, upon simple, reliable concepts such as physical objects, events and relationships (see Fig. 2) whilst the abstract notions such as 'a work of literature', 'copyright', 'ownership of copyright' and so on, are precisely explicated through formal rules.

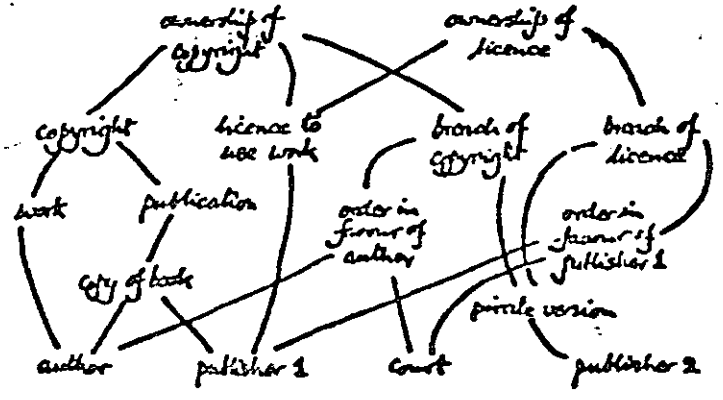


Fig 2: Abstract entities grounded upon physical ones

LEGOL is intended to describe such a structure. Both the analysis of concepts and the expression of legal prescriptions is included and both belong, strictly speaking, to the conceptual schema (by which is meant CS(A) in the rest of this paper). One may imagine that the legal draftsman is trying to write a program to govern the behaviour, not of a computer, but of the real world. In a sense, the LEGOL version of the rules and their concepts have the same function and, therefore, by the earlier analysis, the conceptual schema has this role of 'programming' the real world. Fig. 3 on the left hand side shows this.

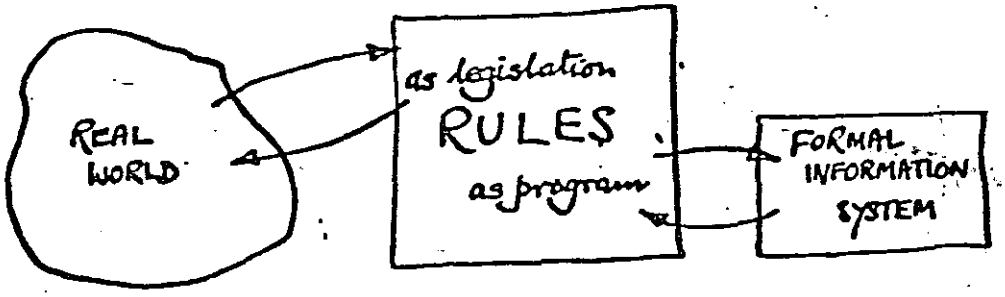


Fig 3: Two functions of LEGOL as conceptual model

Simultaneously, the LEGOL version serves as a computer program because there is a computer system[a] which will interpret it. This is represented on the right hand side of Fig. 3. The LEGOL Project has effectively escaped the extra complications with which CS(D) must deal by accepting the solutions built into the software employed by the prototype, namely PRTV and MP/3 (16 and 17).

From this analysis, it would seem that a conceptual schema, expressed by LEGOL or in some other way, would deal fully with the link between formal system and real world. This is not so. We are also dependent upon the ways in which people understand the words, numbers, sentences and so on, which cross the formal system's boundary. This neglected topic is the one we shall now examine.

Discourse Systems

Formal information systems are not products of nature. They inhabit the system of natural discourse which human beings have created as their most important 'tool'. Natural language serves as the meta-language in which formal systems are defined. This discourse system complicates the picture which was described above but it also provides the essential means of escape from the infinite regress of fetile logical analysis.

[a] albeit a crude, limited prototype at present

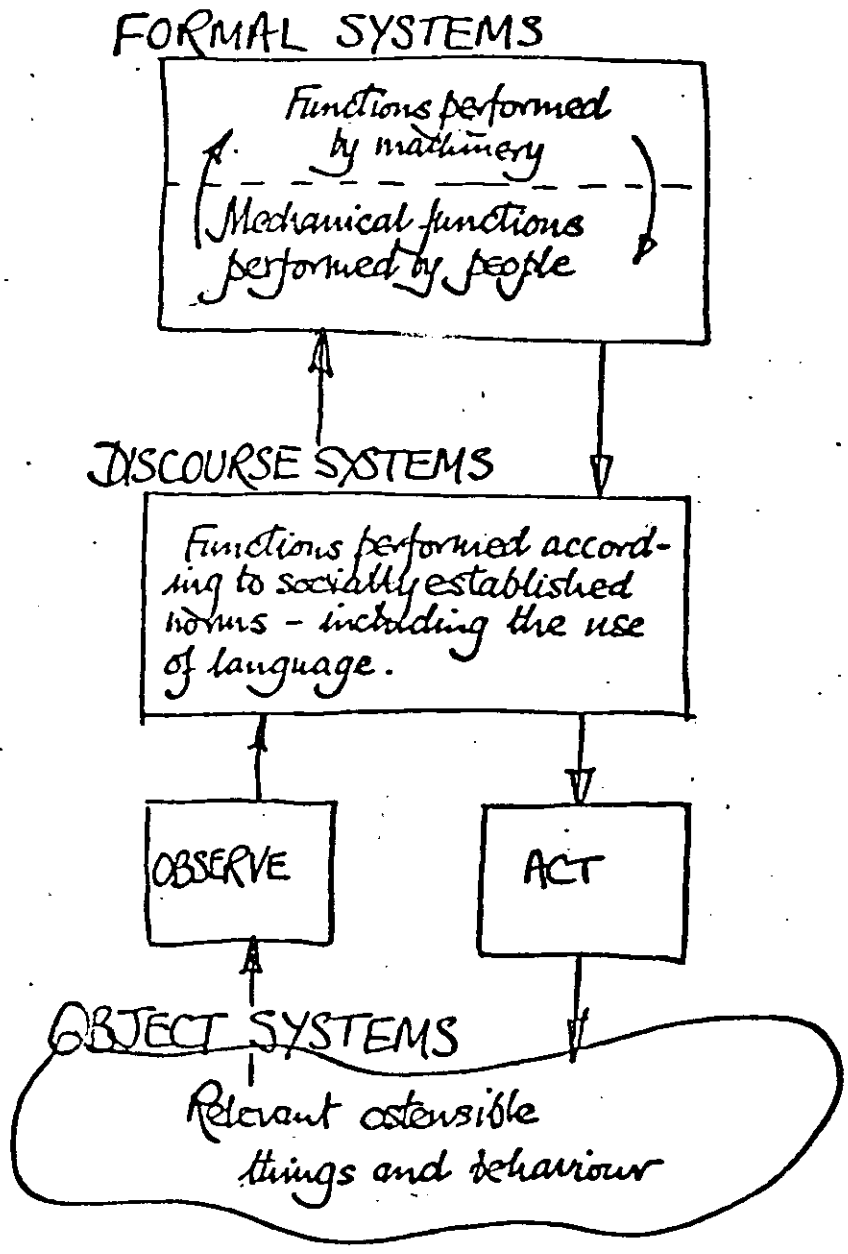


Fig 4: The intermediate role of the discourse systems

The symbols in the formal system are related to the things in the object-system by people* using natural language. See Fig. 4. It is convenient to call the system in which the meanings of signs depend upon linguistic and other norms, the discourse system*(4). It corresponds to the informal part of an organisational information system. It evolves through human, socio-linguistic intercourse, unlike the formal system which depends upon explicit definitions. When observations are made, resulting in data-values being inserted in a database, or when datasare retrieved, as a basis for action, the link will be made by words or other signs which are meaningful both in the formal system and the discourse system; e.g. 'CUSTOMER, J. SMITH LTD'. When symbols of the formal system are used without their having a place in the discourse system (e.g. an account number '3708X455') their use must be explained to the relevant people, in natural discourse*.

Operationally, the symbols we call 'identifiers' will be linked to the relevant physical objects through the discourse system. The quest for logical certainty in this link is a blind alley. We shall always be dependent upon the stability of human norms in the use of signs. These constantly shifting norms are the slightly insecure but only available foundation upon which a practical database can be built. Surprisingly, despite its central role in explaining any large data-processing system, the concept of a 'discourse system' has been overlooked by information system theorists.

Even when dealing with the apparently simple semantic problem of how to identify physical objects, the information analyst cannot rely upon the norms of the discourse system. The way in which the real world is partitioned into objects depends upon the use of language within the discourse system, when people are solving practical problems. It is a mistake to assume that there is a unique, objectively-given set of physical objects to which language refers. The spatial and temporal limits of the objects we talk about are a function of why we are referring to them, of the shared problem we are trying to solve through the use of verbal and other symbols. If the purpose is changed, so are the objects. The norms of the discourse system can shift in subtle ways to match the shifting context of different problems.

This thesis needs to be demonstrated and this is done below. If it is true then it implies that formal systems call for the use of words in stable and uniform ways that differ from the usage familiar in everyday discourse. It will be a task of the conceptual schema to embody these special meanings of data in the formal system and to help users to employ them correctly.

Entity and Purpose

Presumably we want to build formal information systems which enable people to cooperate and do useful things in the real world. The basis of human cooperation is the use of signs, including words and other symbols (5), so that one person may observe and others may know or one person may instruct and others may act. In any use of information there will always be a closed loop from observation, through decision, back to action affecting the thing observed. In an informal use of words to solve a practical problem, very few people will be involved in this cycle and there will be no long delay between observation and action. Formal systems tend to supersede informal discourse when many people have to cooperate in a complex task which requires precise action over long distances and extended time periods. Formal systems are the essence of many kinds of organisation.

* Note that discourse involves more than natural language; it depends upon learning how to use words, numbers or other signs to get things done.

The operations of observing and acting are the interfaces between the object system and the information system. The computer specialist is familiar with the problems of organising and labelling symbolic structures within the formal information system. In this paper, we are concerned with the quite different problem of how to organise and label the contents of the object system[a]. The semantic problem may be regarded as that of establishing correspondence between entities and data. Just as we distinguish and label data to navigate through storage space within the computer, so do we distinguish and label entities to navigate through real space and time. Just as we wish to update a record consistently and later retrieve it, so do we wish our observations and acts in the real world to be consistent. For example, if traffic warden A observes a vehicle obstructing the road, then that vehicle and no other should be towed away by constable B and its owner and no one else should be fined. Any non-computer specialist would be amazed that the identifier problem should be presented this way round, but discussions of identifiers by data-processing specialists are almost invariably in terms of locating records in files and dealing with them consistently. In the literature, we find that discussions, ostensibly about identifying entities, usually turn out to be about identifying and locating records in files: the information retrieval problem.

The semantic problem is much more difficult than the information retrieval problem, despite their formal similarities. The fundamental difficulty about identifying things in the real world is that there is always room for dispute. Testing for sets of matching symbols in storage is relatively simple. Identifying some entity in the real world has no absolute outcome. We must aim to provide a person with an identifier which enables him to find the entity with sufficient precision for some specific purpose, such as levying tax. We must be prepared to redefine the entities when we change our purpose, for example, 'factory' for tax purposes, would be differently interpreted from 'factory' in the context of industrial safety legislation. In computing terminology, one might say that for entities there are many 'subschemas' but no general 'schema'.

The danger of supposing that one, universal, entity-structure can be imposed upon the real world is illustrated by the philosophical problems that this supposition will generate. By abandoning the belief in a universally valid picture of the world in favour of a series of different pictures, each serving some practical purpose, these philosophical problems evaporate (6). The effect of purpose on the entity-structure is copiously illustrated later, in the section on entity-boundaries, meanwhile it may be enough to consider the classical problem of Heraclitus's river. One cannot bathe twice in same river, though, in a sense, it is still there. This paradox is like the confusion of two databases, one to be used by bridge builders and one for environmentalists.

The same, dangerous supposition of a universal entity-structure is embodied in a database management system which requires a single, general schema to be established. A mathematical or computer-centred approach to database design tends to encourage the view that the world has a self-evident, unambiguous structure which can be recorded in files. The distinction made earlier between CS(A), application concepts, and CS(D), the canonical data structure, is basic to solving the problems of database semantics. We must constantly remind ourselves that our discussions about 'customer-records' and 'product records' and 'stock-records' usually have nothing to do with the real world of customers, products and stocks. These terms refer to groups of symbols to which we should attach codes such as '7364', '8421' and '2020', instead of 'CUSTOMER', 'PRODUCT', and 'STOCK' with their extraneous

[a] The physical objects which themselves carry information (e.g. documents) are themselves part of the object-system. In so far as we are interested in them as objects rather than signs, they may be treated in the same way.

and strictly irrelevant connotations. The symptom of this kind of error is the use of a term, such as 'customer', without qualification. In a real company, 'customer' will have many meanings to suit many users - lawyer, accountant, salesman, market researcher, production controller. The analysis of the entities referred to in legislation is made clear because there are frequent shifts in meaning which the draftsman must make explicit. The LEGOL system, therefore, requires every entity definition to be given a context. This is a difficult problem for the designer of database software but it is better that we face up to it. I conjecture that a monolithic database can easily become an embarrassing organisational white-elephant.

One task for the semantic model is to keep track of the varied purposes for which data are used. This information is typical of that used initially by the system designer and then forgotten once it is only implicit in the final physical design. It should be held in CS(A) and discarded for the narrower purposes of CS(D).

Entity and Stability

Our ability to use computers correctly within an organisational system depends upon our ability to use signs, such as words, numbers and various codes, in a stable way so that the observer of say, a faulty component, can name it in a production report knowing that a colleague in another factory will be able to identify it and repair it. Many people may observe and work upon the same component in ways that are coordinated through the exchange of signs. The component may change location, colour, texture, shape, function and so on yet, through this flux, the concept of that component will be sustained by a group of people who are able to identify it with a constant name. When Quine refers to 'the myth of physical objects as a device for working a manageable structure into the flux of experience', (7) to some people he may seem to be adopting a quaint philosophical stance, but not to anyone who has thought carefully about the problems of designing a database for a production system. Materials change their shape and appearance; batches form, merge and disperse; assemblies are created and their components changed; throughout this flux a limited but effective entity-structure must be used; the formal system could not economically recognise every possible discernible physical object.

The effectiveness of any information-system depends upon stable patterns of human verbal or symbolic behaviour known as 'perceptual norms'. These norms belong to the discourse-system. They come into existence because people share problems which they solve cooperatively through the use of language. Given the context and the problem, a group of people will evolve a way of attaching words to reality so that common perception of the problem can be obtained; failure on the part of an individual to perceive the problem in the same way as the majority will result in his acting inappropriately; criticism from his colleagues will lead him to adjust his perceptions into line with the established norm. Through this social mechanism (and not through the application of explicit definitions) words are assigned to things in a stable way and entity-structures are imposed upon the world about us. These structures of perceptual norms may be sustained through social interaction. They may lead to poor solutions to practical problems, particularly when the interaction is purely verbal or symbolic; norms are exposed to objective criticism only when they give rise to events in the real world of physical objects and people. For example, the impounding of the wrong car and the charging of the innocent owner for obstructing the highway would suggest that the symbolic 'image' of the real world used by the police in the files of the road-licence and traffic-offences system is 'out of focus'. Many information systems interpose abstract concepts between observation and action. These, such as the legal object 'company' or legal relationship 'ownership' help us to create a stable picture of a changing world, but their meanings are ultimately dependent upon the semantics of physical objects and events. Normally it is easier to resolve disputes about physical things than disputes about other entities. This

justifies for example a careful review of how we attach names to physical objects as a prelude to the study of more tenuous concepts. This example of a semantic problem is about the simplest we could choose.

The functions of names

Dewey (8) suggests that a name serves three functions: as a fence, as a label, and as a vehicle. We partition the world into physical objects by naming them; changes may take place within the boundaries but we keep the fences intact. This partitioning of the world provides us with a reference-frame, a kind of generalised coordinate system by which we can navigate. Names are the coordinate labels. We attach a label to any point to which we ourselves might wish to return or to which we might wish to direct someone else. Once the object has been given a name, we can transport it, as it were, symbolically, and place it in new juxtapositions with other objects, without the need to shift the object physically. Names, as vehicles, are the means by which we discover, and impose upon the world, new and perhaps preferable arrangements by means of our conjectures, hypotheses, theories and plans. Let us examine each of these functions more closely from an information analyst's point-of-view. Information systems depend for their effectiveness upon the correct assignment of names to things, the appropriate manipulation of those names followed by the correct association of things with the names.

Let us first observe that the information analyst is concerned with the definition of the formal information system in which the meanings of words, numbers and codes are formally defined. The meta-language upon which his definitions depend is the natural language of the discourse system which, in turn, is rooted in the perceptual norms arising from social interaction.

The fencing off is done by the discourse system. Normally, we employ two or more stages, beginning with the recognition of an entity-class before we name the [illegible] ordinary use of words is generally adequate for [illegible], the normal usage is not precise enough; we then have recourse to definitions (as in the Copyright Act 1965 c73 where, for example, "'written matter' includes any writing, sign or visible representation".) The first step is intended to provide operationally effective procedures for differentiating the physical object from its surrounding. It may involve specifying who should make the observation in those cases where the special skills of, say, a physician or valuer or tax inspector, may be required. The instruments to be used, the place of observation and other circumstances may have to be included in the definition. Thus the physical object is identified initially.

Having found* the object, we must attach a label to it. In the everyday situations dealt with by our discourse systems this is simple because the number of objects we are concerned with is small. In a formal information system employing a computer, the number of occurrences may run into millions. The construction of these labels for use in an information system is complicated in ways that will be discussed below. The identifier must not only correspond mathematically with the unique physical object it represents, but it must correspond operationally. The information system will not be able to do useful work unless there are procedures for locating and for checking the identity of the object specified by any identifier. (Consider, by way of illustration, the operational problems of locating an object in a large museum or of associating code-numbers with billets of red-hot steel) These are central issues in the design of codes for identifiers, a topic discussed further below.

* Objects that are too large or too small to present themselves for observation in a simple way (e.g. the solar-system or a molecule) must be treated as abstractions. See Popper (9) on observationally-testable statements.

Having established an operationally effective, one-to-one relationship between physical objects and identifiers, we can begin to use these carefully constructed signs as vehicles. But we shall need 'traffic' rules. By re-arranging the signs which represent the objects in the world, we conjecture new arrangements of the world itself (for example, orders sorted by product type may be assigned to appropriate machines). New arrangements conjectured in this way will only be translatable into reality if the identifiers have not been moved beyond the bounds within which they are meaningful. The information analyst must establish these boundaries if the data-processing system is to be constrained to produce meaningful outputs. (Thus: engine 1234 cannot be used simultaneously in two distinct cars but an abstract object such as the play, 'Hamlet', may be used simultaneously by several theatres) This is the central issue in the semantic problem of updating files which is touched later. Each of the functions of a name will now be examined in more detail.

Entity-boundaries in a formal system

A formal system must rely on a given entity structure and normally it relies on the discourse system to assign the names it uses. Reliance on the discourse system is ultimately unavoidable but there may be a converse influence of formal rules on the the norms of discourse when everyday terminology needs to be tightened; rules may help to make the naming structure more widely consistent than it would be under an unaided discourse system. For example, in the 1965 Rent Act, "'the occupier', in relation to any premises, means any person lawfully residing in the premises or part of them at the termination of the former tenancy"; this restriction removes ambiguity from the word 'occupier' as it is employed in normal discourse. Another reason for formalising the naming of things, common in technical situations, is the lack of any established perceptual norms upon which to rely; for example, a particular subassembly for an engine may never have been made before so what it constitutes must be defined, ad hoc, by an engineer. His definition will ultimately rest upon normal usage of words in the discourse system and any formal definition will be anchored similarly in the perceptual norms, deriving what stability it has from these cultural reference points, not from the logic of the defining rules themselves.

The information analyst has a responsibility for partitioning the world appropriately into its relevant component entities so that data about them may be manipulated meaningfully and consistently by the formal system. Let us look at some of the problems he might encounter.

Changes in the properties of putative entities may transform them beyond easy recognition and raise problems of when the entity ceased to exist, being transformed into another entity, perhaps. For example, since the advent of transplant surgery, the formal definition of the moment of death has become a matter of keen interest. In a manufacturing situation one can imagine a piece of steel being worked into many varied shapes, its metallurgical properties and its physical continuity being the only aids to its identification. The piece of steel, through all its transitions, will be regarded as the same entity, the reason being that we do not change our intentions about what to do with the steel as a result of the transformations it undergoes. The human body will be used very differently alive and dead. In general, the partitioning of the world into its physical entities will depend upon what we intend to do with the entities so categorised. Transplant surgery has made semantics a very lively subject!

Assembly-subassembly relationships offer innumerable alternative entity structures from which one must be selected, within some simple constraints. The components of each assembly must be mutually exclusive, otherwise instructions for making and dismantling are likely to be ambiguous; also the list of components must be exhaustive, otherwise the formal system will not be able to describe the manufacturing process completely.

The detail into which the assembly is decomposed will be limited by the need to act upon components individually, to manufacture them, order them, or redesign them. All these activities may share the same entity-structure but other structures are possible. For example, subassemblies may be distinguished on the grounds of the functions they perform. This criterion will suit the design and maintenance engineers who must understand the working of the whole assembly. It may be impossible to select one decomposition that will serve all purposes. The design engineer may prefer a decomposition based upon function whilst the production engineer may be more concerned with physical structure; maintenance may require elements of both, function to aid diagnosis, physical structure to aid repair. The problems of how to name the subassemblies in a complex structure obtrude even more severely when the main assembly is an organisation and the decomposition is an accounting framework. This difficult problem, aggregation/disaggregation or assembly/subassembly, may provide a major justification for using a computer in certain applications; the computer may permit information based on many different structures to be consistently related to a single, very detailed underlying entity-structure but the solution may be uneconomic if this common structure is too detailed.

Systemic continuity may be sufficient reason for naming, as a physical entity, something with changing physical composition. The proverbial example might again be the river that Heraclitus pointed out would not be the same one next time you stepped into it; the human body is much like the river in this respect but, more prosaically, there are, for example, production lines which, through maintenance and development, are physically entirely replaced whilst retaining their systemic identity. Physical objects are not always physically unchanging. Their permanence arises not from what they are made of, but from the constancy of the problems they pose and the ways we wish to use them.

Change or use or role may signal the end of the existence of one object and the beginning of another. Natural discourse would tend to acknowledge continuity in cases where a formal system would have to make more exact distinctions. For example, riding a bicycle up to the traffic lights leads to my son stopping, in conformity with the law relating to vehicles; but carrying the bicycle home to him for Christmas, I was carrying a parcel, so I was governed by different laws. Similarly, many a bottle has become an offensive weapon. This kind of shift can be treated semantically as the reclassifying of the same particular object under various generic names. Such an explanation will do for the discourse system, which has a restricted span of attention, as it were, the object may only be relevant when its role has been decided: the traffic law did not apply to my son's bicycle until Christmas morning when my parcel became his vehicle.

Different objects may be made from the same material presenting a problem similar to the one above. A sword may be turned into a ploughshare. We might record this event as the end of the sword's existence and the beginning of a ploughshare. The transformation may be quite a subtle one, as when an artist selects a shapely piece of wood as an objet-trouvé; the act of selection would bring this natural object within the terms of copyright law which would treat it as any other sculpture more elaborately fashioned.

From the above analysis, it must be concluded that our names for things do not embody any knowledge of the intrinsic nature of the world, rather do they reflect the nature of our problems, our purposes and our linguistic means of interacting with the physical world. If we are on such shifting ground when considering the semantics of physical objects, how much more cautious we shall need to be when we come to consider more abstract entities and properties! At least we should be prepared to look closely at the glib assumptions made about shared databases. Data obtained for different purposes are likely to be semantically different. They should be shared only with the utmost caution. For example, 'child' seems a simple notion but in data-processing systems for family allowances and for medical record linkage the definitions would be difficult to reconcile.

For the purpose of the LEGOL Project, the above analysis gives some useful direction to our treatment of semantics. If we are to define data groupings which are irreducible ones from a logical point-of-view, then it appears that we should make them correspond to physical entities, containing information which will be constant throughout its existence. For a physical object, the minimum set of elements is a name and times for start and end of existence. The invariance of the logical data-structure seems an appropriate way of reflecting the semantic invariance of the real entity-structure to which we are referring. All the time-varying properties would be treated separately. Surprisingly, it may be more appropriate to include purpose as one of the properties of an entity class. This conjecture raises the difficulty of defining purpose appropriately; in LEGOL, the nearest we come to doing this is to define each entity-class within a limited context such as a branch of law, an Act, a section or even a subsection.

The formation and assignment of names

In the discourse system, the giving of names is usually quite simple. the number of objects involved may be quite small and the names may be assigned temporarily for solving a limited problem. People supply their own names to avoid confusion; things are labelled by demonstrative pronouns, genitive nouns, possessive pronouns or by demonstrative or possessive adjectives plus a common noun. (Common nouns, e.g. 'book' are labels for entity-classes) Seldom in normal discourse do we need to attach proper names to large numbers of objects.

The naming of large numbers of objects is, by contrast, a major problem in formal information systems. Indeed, the reason for having a formal system is often the need to keep track of multitudes of things in a manner that is consistent over distance and time. Licencing and registration systems are typical examples. Normal discourse is adapted for more localised use. Before any object can become a part of the subject-matter 'known' to a formal system, it must be given a name in a way that satisfies some simple criteria. The names may also be chosen to reflect real structural patterns. These are the problems of forming names.

Each name must be unique. This can be achieved in various ways, the commonest of which is to assign names from a sequential list, keeping note of the last values used, e.g. accession numbers to library books. Names might be generated in more elaborate ways from names of related objects; for example, subassembly names as elaborations of their parent assembly names. Uniqueness may be guaranteed by the rules so that identical names will only be generated for identical entities. If this is not the case, the formal system must include a register of names, some means of checking that new names are unique and rules for modifying any homonyms. For example, a person's name may be enough to identify him in a personnel system, but provision may have to be made for the occasion when a new employee's name is already used by the system. Uniqueness is the only necessary characteristic of the name that is chosen for an object.

Structural properties of names are sometimes useful, but they are not necessary; sometimes they can be a source of dangerous problems. For example, plots of land might be named according to their parish, within local government area, and serially numbered within parish; this structure, reflecting relationships among the plots, would be useful until a change of local government boundaries. Accession numbers of books will indicate roughly when the library acquired them. More elaborate structures can be represented if a suitable isomorphic name-structure can be found.

Reassignment of names may be forced upon a formal system. Within one and the same system, structured names may have to be reassigned if the underlying, real structure is no longer correctly represented by the name structure. The local government reorganisation would require such effort, in the example cited above. The merging of two formal systems performing the same functions for different

populations of objects will require reassignment of names if there is any risk of the same name being used for objects previously dealt with by separate systems. This reassignment can be an entirely formal operation. But this would not be so when merging systems which deal with overlapping populations. Such a situation would call for procedures to check any likely synonyms resulting from the merger; these procedures may be expensive and slow, requiring positive physical identification of the objects and reference to both information systems. Problems would be minimised if naming always followed standards based upon underlying, stable structures; for example, using grid references for naming plots instead of the method suggested above. Standard rules for forming names would not, of course, solve the problem if the entity boundaries were drawn differently in the system being merged.

Identification of objects

Whereas the assignment of a name is an operation performed only on the accession of an object into the formal system's universe of reference, identification is the continuing use of the name to link the real object and the symbols that describe it. The link is two-way. For the formal information system to serve any organisational purpose, it must be possible for users to move reliably from object to symbol and from symbol to object.

The name of the object, to be an identifier, must be supplemented by the name of the entity class of which it is an instance. Often enough, the name itself is formed in a way that indicates its entity class (one may have an employee named 'FRED'; (yet women's names have been taken from flowers and given to furnaces!)) The intrinsic identification of class from the form of a name will be a useful local device, say, for handling some limited streams of messages from which the chance of ambiguity can be excluded, but in general, we have to assume that the name gives no clue to the entity class.

The first step from object to symbol is to distinguish and name the entity class and subclass down to the point in the relevant hierarchy of classes beyond which individual unique names are given. This can often be done by relying upon the familiar norms of the discourse system, if not, the information analyst will need to ensure that the relevant personnel are trained or advised how to apply the special classification rules which distinguish an entity from its surroundings and establish which naming procedure is applicable. The second step is to derive the name from the object. Often this is simple, you read the number printed on the document or on the ring round a bird's leg, for example. Less reliably you may take a number from a tag attached to a garment; less reliably because there are high risks of the tag having been removed and reattached incorrectly, or incorrectly attached in the first instance. As in the case of a credit card, one may use additional information such as a signature, to reduce the risk of incorrect use of a name; more information is used on a passport which includes a description and photograph. In some cases, the name cannot be 'tied' to the object, perhaps because it is too small (insect), inaccessible (star), a liquid (batch of dye) too hot (metal) or likely to be embarrassed (spy). In such cases, the observation of the object must supply enough information to enable one to construct its name. The same problem arises when an extrinsic label becomes detached from its owner. Systems are sometimes disastrously designed with insufficient regard for this problem.

The problem just cited is akin to the problem of going from symbol to object. Used in this way, we see identifiers as generalised coordinates for locating things in the world. If the identifier does not convey appropriate topographical information to the discourse system, as in general it will not, then the name serves only as a check that one has arrived at the correct point. Locating the object would, in general, call for the scanning of possibly all the objects of a particular type referred to by the system. Objects are usually found more easily by applying various established techniques. In the simplest case, the

formal system relies upon the discourse system in which the people expected to use the identifier can be presumed to know the location of the object. Sometimes this knowledge is not readily available and the formal system must keep track of persons who may be expected to know (e.g. next of kin, owner). To establish the formal name of a person correctly, the system may have to be able to conduct a dialogue based on information obtained during earlier transactions with him, e.g. asking him to supply his mother's maiden name, a technique which is not, incidentally, universally applicable. In other cases, the formal system may keep track of the actual location of the object, perhaps noting a person's changes of address every year, or even by following movements second by second, as in an air-traffic control system. Sometimes, movements can be tracked if certain decisions about movements and consequent actions are recorded; in this category are batches of chemicals in a system of tanks or billets of steel rushing through a mill into specified locations in the cooling-beds. The problem, mentioned in the previous paragraph, of how to go from object to identifier when it can carry no label, cannot readily be solved unless the physical location of the object can be treated as one of its known identifying properties. There may be other, time-dependent properties necessary for identification, such as stage of manufacture; they must also then be logged by the formal system. Either the logging of these characteristics will have to take place literally continuously or the formal system will have to employ certain rules of continuity that will enable it, with sufficient accuracy, to extrapolate, from suitable, intermittant observations, to the current locations and states of the objects in its universe. These continuity rules are an essential part of the semantics of these objects. In a rigorous analysis they need to be made explicit. If there is any significant risk of the tag being lost from an explicitly labelled object then, to remain effective, the system must employ similar, but less exacting, continuity rules. Detecting errors in identifiers would be impossible without them.

There can never be one hundred per cent certainty that labels remain attached to the objects or that continuity rules are wholly reliable, hence the risk of identification error cannot entirely be eliminated. This risk has two components: finding the wrong object given its name and associating data about a known object with the wrong identifier. These are two important design parameters for the information system. The formal system must have checks built into it to ensure the desired levels of semantic reliability.

Incidentally, it should be noted that the case of the object which cannot be explicitly labelled serves to illustrate the idea that a name is an abbreviation for a detailed description of the object, detailed enough for the object to be selected uniquely among others of its type. The name might actually be formulated by encoding one such standardised description. See (10). Rules which, in a formal system, enable one to convert one identifying description into another, probably via the name, might be regarded, along with rules of continuity, as a means of strengthening the links between object and symbol. They too will have to be made explicit for use by a formal system. A suitable name might be rules of synonymy.

Rules of continuity and rules of synonymy are used when updating files to check for inconsistencies. These are examples of the traffic rules for Dewey's names as 'vehicles'.

Application in LEGOL

The foregoing analysis is incorporated in the semantic model of the LEGOL system where the idea of an identifier is not to be confused with the notion of a 'key' for information-retrieval. The LEGOL formalism is intended for expressing rules about the conduct of a business or a society. An identifier enables one to find the real objects to which the information system refers, a process dependent upon discourse system.

Structurally, the semantic model is surmounted by a number of entity-types which embody the rules for identifiers. The simplest type is the 'object' and the physical object the prototype for those abstractions which we treat conceptually as objects and identify, symbolically, in ways analogous with the identifying of physical objects. Characteristic of an object is that a name, unique to the individual, is a sufficient identifier. By way of contrast, we may consider the relation, another type, which links other entities. The prototype is the physical relation among physical objects. To identify a relation, we must know which objects are related, when and in what manner, such as 'on', 'under', 'inside'. A relation is not uniquely identifiable unless the period of its existence is known. So, for a relation we have the following identifier elements:

Relation: TYPE, OBJECT 1, ... OBJECT n, START, END.

In the case of an object, we need only the unique name as an identifier, but an object in the semantic model also incorporates the period of the object's existence:

Object: NAME, START, END.

These basic templates can then be used to define some of the 'traffic rules' of the semantic model. For example, the relation cannot exist outside the period of existence of any of the objects involved. A knowledge of when an object starts and ends its existence is also essential for its semantics to be precisely defined. Unless the three elements of an object are precisely established, it will normally prove impossible to build a formal information system to exercise control over those objects. For example, an analysis of the Family Allowance Act 1965 failed to reveal definition of the period of a family's existence.

Object names are assumed to be unique for all time. This can always be realised by suitably adapting a naming system which does not provide unique names. In this, 'unique' need only be interpreted within a context of a given problem. If that context is a dispute among three individuals, the naming of those parties raises no difficulty. When large numbers of individuals are involved, there may be formal rules for registration which govern the assignment of names. If so, the LEGOL analysis could include them. If not, the semantic analysis will include sufficient information to enable the discourse system to act as a reliable interface between the object system and the formal system. Information of this kind is essential for building an application though it has no place in the programmer's view of a database system.

Before the assignment of individual names, the object class must be specified with sufficient precision. In the LEGOL system, this is done by defining a hierarchy of 'entity-categories', each of them limited to a specified context or problem. Thus, an instance of an object class may be 'building' but this entity-category will be given different meanings in the contexts of revenue law and planning law. Although the semantic models in these two contexts may be almost identical, it will always be clear that the data-values have different meanings.

At the basic level of analysis, the person who is formulating rules in the LEGOL language may give a particular interpretation to an entity-category in an appropriate context. At this point, he must be responsible for establishing the mechanics of the naming process.

Summary

This is the first of a series of papers about the LEGOL semantic model which treats the concepts of entity and identification from a point-of-view of systems analysis rather than as a problem of information retrieval. This leads to a conceptual scheme more complex than envisaged in previous discussions of NDMS. The added complexities make explicit characteristics of object systems and discourse systems which are later submerged within the design of a particular physical information system.

Only the narrow topic of how to link physical objects to the signs or symbols that identify them has been considered in this paper. Philosophical problems are avoided in system design because we use symbols only with sufficient precision for a specific purpose. The fact that philosophical questions bedevil any attempt to use language in universal ways, suggests that when we extend the scope of a formal system, we must check that the semantic assumptions upon which it is based can cope with its extension.

At least, philosophy has helped us to realise, contrary to popular opinion, that the world is not composed of so many well-defined objects ready to be named but rather, that it is structured into entities through the agency of language. An entity-structure depends for its stability upon the conceptual norms established and maintained by natural discourse in a social system. The choice of norm is a reflection of the problem shared by the people who sustain the relevant norms. To change their problems is to change their perception of reality and their use of words.

Formal systems can only use signs in ways that are rooted in the norms of the discourse system. Physical objects have the simplest kinds of identifiers: just a class of entity and a name. We must start by clarifying, for a specific formal system, how the physical objects it is to recognise are to be fenced off from one-another. Many examples were given to demonstrate the difficulties of doing this and to show that the difficulties could be resolved by considering the purpose for which the object was to be distinguished as a stable entity despite possible vast physical changes occurring to it.

The analyst must design the method of forming and assigning names to the objects identified by the formal system. Uniqueness is the only structural requirement of a system of names though it may, with both advantages and disadvantages, embody the structural information.

Finally, the analyst must establish precisely how to link object to identifier and identifier to object, if the input to and output from his system are to relate meaningfully to the world. To some extent, this can be done with labels but rules of continuity and rules of synonymy are necessary too.

The LEGOL language and system have been continuously in the background to the analysis. One of the purposes of LEGOL is to make explicit the semantics upon which a formal system is based, so that they may be designed consciously, rather than by default, as at present, through want of a language in which to talk about the problem.

Acknowledgements

Many people must be thanked for their helpful comments on this paper. In particular, I will mention my colleagues at the London School of Economics, Frank Land, Sam Waters, Bob Davenport and Peter Mason, and my colleagues who, whilst I was working on the paper, were at IBM Peterlee, especially Terry Rogers, Peter Hitchcock, Pat Hall and Nigel Martin. Also I must thank the BCS Study Group on Databases for their helpful discussion of a draft version. Finally, the participants at the IFIP TC2 Working Conference in Nice 1977 led me to add ten pages to the opening of this paper in response to the discussions during the week, this being the last to be presented.

References

1. Stamper R.K. The LEGOL Project - A Survey IBM UKSC Report No.0081, Peterlee, 1976.

2. Kent, W. Describing Information (Not Data, Reality?) IBM Technical Report TR 03.012, Palo Alto 1976.

3. ANSI-SPARC/DBMS Study Group Interim Report American National Standards Institute, 1975

4. Stamper, R.K. 'Logical Structures of Files' in Data Organisation for Maintenance and Access The British Computer Society, 1970.

5. Stamper R.K. Information Batsford, London; Wiley, New York, 1973.

6. Cavehay-Williams, R. The Double Criterion of Empirical Judgement, March 1961.

7. Quine, W.V.O. From a Logical Point-of-View Harper and Row, New York, 1953 (revised edition, 1961)

8. Dewey, J. How We Think Heath, London, New York, 1933

9. Popper, K.R. Logic of Scientific Discovery, Harper and Row, New York, 1935 (revised edition 1965)

10. Neiland, J.W. Talking about Particulars Routledge and Kegan Paul Ltd, London, New York 1970.

11. Nijssen, G.M. A Gross Architecture for the Next Generation of Database Management Systems in (12).

12. Nijssen, G.M. (ed) Modelling in Database Management Systems North-Holland, Amsterdam and Oxford 1976.

13. Nijssen, G.M. (ed) Database Description North-Holland, Amsterdam and Oxford, 1975.

14. Nijssen, G.M. (ed) Concepts for the Conceptual Schema North-Holland, Amsterdam and Oxford, 1977.

15. Senko, M.E. DIAM as a Detailed Example of the ANSI SPARC Architecture in (12)

16. Todd, S.J.P. 'The Peterlee Relational Test Vehicle - a System Overview' IBM System Journal No4, 1976.

17. Mandil, S.M. The MP/3 Macro-Processor IBM UKSC0044, December 1973.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

DISEÑO DE BASES DE DATOS

A R T I C U L O

MIS IS A MIRAGE

EXPOSITOR:
Ing. Daniel Ríos Zertuche

MAYO, 1985

Harvard Business Review
January-February 1972

5

John Dearden

MIS is a mirage

Can a single, integrated system be devised to fill all of management's information needs? Only if Superman lends a helping hand

Foreword

Every company of any size has many information systems, both formal and informal. The formal systems it uses cover such a variety of territory that one man cannot possibly comprehend the mass of details and principles required to design a single supersystem that embraces them all. Even a group of systems experts cannot create such a supersystem, the author argues, because the components that must be amalgamated are too different in their natures to be fused together effectively. After demonstrating the futility of the MIS approach, the author recommends practical steps for reforming defective information systems.

Mr. Dearden is Professor of Business Administration at the Harvard Business School. He is well known to HBR readers, especially for his significant contributions to the theory of divisional control. (A list of Mr. Dearden's previous HBR articles on information systems appears in the ruled insert on page 98.)

Some years ago I expressed the opinion that "of all the ridiculous things that have been foisted on the long-suffering executive in the name of science and progress, the real-time management information system is the silliest." [1]

I no longer believe this statement is true. We now have something even sillier: the current fad for "*the* management information system," whether it is called the Total System, the Total Management Information System, the Management Information System, or simply MIS.

I certainly do not mean to suggest that a company does not need good management information systems—nothing could be further from the truth. But the notion that a company can and ought to have an expert (or a group of experts) create for it a single, completely integrated supersystem—an "MIS"—to help it govern every aspect of its activity is absurd.

For many businessmen, it is probably inconceivable that the lofty phrases and glittering promises surrounding the MIS conceal a completely unworkable concept. Yet this is exactly what I propose to demonstrate—that a company that pursues an MIS embarks on a wild-goose chase, a search for a will-o'-the-wisp.

Let me first try to explain what I understand by the "MIS concept" and examine its alleged advantages, and then show why the concept is unworkable. Then I shall be in a position to recommend some practical remedies for defective management information systems, which certainly constitute a real problem for executives today.

Confusion between terms

It is difficult even to describe the MIS in a satisfactory way, because this conceptual entity is embedded in a mish-mash of fuzzy thinking and incomprehensible jargon. It is nearly impossible to obtain any agreement on how MIS problems are to be analyzed, what shape their solutions might take, or how these solutions are to be

1. "Myth of Real-Time Management Information," HBR May-June 1966, p. 123.

However, in practice, no such limitations are intended. Kenneron's inclusive definition of the MIS approach is quite consistent with the nearly universal benefits claimed for it.

The MIS approach

Given this inclusive definition, how is management to apply it? In other words, how should management think about the problem of setting up an MIS?

Fundamental assumptions

First, it appears that if management wishes to subscribe to the theory of the MIS, it must make up its mind to accept two fundamental (if highly questionable) assumptions that are quite different from traditional ones made in this area:

1. Management information is a subject for study and specialization. That is, it is sufficiently homogeneous so that a set of principles and practices can be esablished for evaluating all management's information needs and satisfying them. In short, the MIS approach attacks all the problems of management information as a whole, rather than by individual areas, such as finance and marketing. This homogeneity is a necessary assumption, since without it there is no reason why general solutions to a management's information requirements can be found.

2. The systems approach can and should be used in analyzing management's information requirements. Proponents claim the systems approach is necessary for mastering the sprawl of requirements and for synthesizing the general MIS solution. (I shall have more to say about the systems approach later.)

Diagnosis & development

Once management has accepted these two assumptions, it can begin to develop an MIS program. As the theory goes, there seem to be two techniques for setting to work:

☐ Management can hire an MIS expert to act as a superconsultant to the president of the company. This expert studies the types of problems that the president must solve, the decisions that he must make, and so forth, and recommends methods for satisfying the president's total information requirements. He then drops to lower levels of management and provides the same services there.

In general, the expert depends on others to implement his recommendations. For example, the controller becomes responsible for changing the cost accounting system in the way the consultant recommends.

☐ Management can create a staff department that reports to the top. This group is responsible for the company's computer-based systems but also provides the same type of diagnoses and evaluations as the superconsultant.

The staff group, unlike the consultant, usually has responsibility for implementation.

Its alleged advantages . . .

Under this approach, then, either a single person or a group of persons is responsible for developing and overseeing the construction of the entire management information system. This concentration of authority and responsibility in the hands of systems experts supposedly creates a number of significant advantages:

☐ Experts schooled in the MIS "discipline" can analyze management's information needs more effectively than can the people traditionally responsible for satisfying them. Moreover, these experts can better determine which techniques will best meet these needs.

☐ Because the MIS is developed as a unified, single system, rather than as a number of separate systems, it is completely coordinated and completely consistent.

☐ Information needs are determined from the top down. Hence the top will be in better control; the frequent practice of letting lower management decide what information will pass upward is eliminated.

☐ The company reduces its direct information costs by eliminating systems. Also, the MIS itself is cheaper to run because it has been designed by information experts who know the most economical means for satisfying management's information needs.

☐ Since one expert or group is responsible for the system, management's desire that the system be kept up-to-date can readily be satisfied.

In short, the proponents promise, experts can design an MIS that is more effective, more efficient, more consistent, and more dynamic than the haphazard aggregate of individual systems a company would otherwise employ.

These are impressive advantages that any manager would enjoy, and doubtless this ap-

implemented. This confusion makes it very difficult to attack the concept, because no matter what assumptions a critic makes about the nature of the MIS approach, a proponent can always reply that *his* use of the term is different from others'.

But there is a common thread which runs through the various uses of the term, a thread that at once unifies but also subverts the MIS literature. This thread is the computer-based information system.

Computer-based activity . . .

Wherever the MIS is discussed, it is almost invariably stated that a management information system does not necessarily require a computer and that many forms of management information are not computer-based.

Yet, if one looks at what is actually being discussed, he quickly discovers that the term "MIS" is used, essentially, to stand for "computer-based information systems." For example, a recent article in *Business Week* read as follows:

"[Some], concerned that systems analysts are . . . a 'mixed bag' whose training and knowledge are a hit-or-miss proposition, are convinced that management information systems (MIS) is *the* emerging field in business administration. Both Wharton and MIT have tailored programs especially for systems specialists, but no school has gone further than the University of Minnesota, whose B-school now offers MS and PhD degrees in management information systems and has launched an MIS research center. Since the center's opening three years ago, MIS Director Gordon B. Davis and his staff have worked to develop 12 new systems-related courses—from on-line, real-time systems to a seminar on software. In addition, the program's 50 MS and 22 PhD candidates spend a good portion of their time alone and in teams at work on actual computer problems in industry." [2]

It seems evident to me that MIS education as described here is principally education in computer-based information systems.

It is vital to note, first of all, that the information generated by this kind of system does not include a great deal of the information that is most important to management—especially, important *qualitative* information. Second, a specialist group that develops such a system is usually responsible for implementing only one part of any of a company's individual management information systems—namely, that part that interfaces directly with the computer. For example, such a group has little (if anything) to do with specifying the nature of an accounting and financial control system, although it may be responsible for the computer programming this system employs.

My conclusion, therefore, is that such a group has little impact on most of the information supplied to management, particularly at upper levels. Consequently it is ridiculous to say that it creates (or *can* create) a total management information system.

. . . vs. MIS

To the extent that MIS refers only to company information systems that use a computer base and to the extent that everyone understands this limitation, I have no serious quarrel with the trend to MIS; it is vital that management tightly control its computer-based information systems, and in general the so-called MIS groups seem designed to guarantee a tight rein to management.

In my experience, however, such a limited definition of MIS is *not* what advocates of this approach to information systems mean when they use the term. They intend something novel and far more global, some entity that can provide revolutionary benefits we cannot derive from the traditional approach. Walter Kenneron suggests this definition of the MIS:

"A management information system is an organized method of providing past, present and projection information relating to internal operations and external intelligence. It supports the planning, control and operational function of an organization by furnishing uniform information in the proper time-frame to assist the decision-maker." [3]

This is approximately what I perceive most people to mean by MIS. And if this definition seems grandiose, I can only remark that "*the* management information system" describes a grandiose idea. If the definition were less global in its scope, it would not measure up to the term. If, for example, one were to limit the definition to the context of a company's financial accounting programs, he would have to speak of the *financial* MIS of the company, rather than its general MIS.

2. June 5, 1971, p. 56.

3. "MIS Universe," *Data Management*, September 1970.

proach was developed to solve the real problems of poor information that have been plaguing management with increasing frequency. The growing complexity and the pace of change of modern business, especially in the last ten years, have surely made many information systems obsolete and many more inadequate for present tasks.

Equally, the last ten years have seen the extensive development of information technology, management science, and systems analysis—a development that has been accompanied by rapid growth in the number of experts working in information systems.

To some—that is, the proponents of MIS—it seemed logical to centralize the development and control of information systems in the hands of these experts. After all, the problems that beset information systems have been the result of change and growth, they reasoned, and these problems could perhaps be solved by using the new information technology that had been developing simultaneously.

Several companies have tried this approach, and many people currently advocate it. In spite of its apparent logic, however, I know of no company in which it has worked out. This fails to surprise me because, as I have already implied, I believe the whole MIS approach is fundamentally fallacious.

. . . & its real fallacies

There are four fallacies and one serious misconception inherent in the MIS approach, as I have described it. The fallacies are these:

◊ Management information is sufficiently homogeneous so that it can be made an area of specialization for an expert.

◊ If the different information systems ordinarily used by a company are developed separately, the resulting management information system will necessarily be uncoordinated and therefore inefficient and unsatisfactory.

◊ The "systems" approach is a new boon to business administration.

◊ It is practicable to centralize the control over a company's entire management information system.

The misconception is this:

◊ The specialist expertise that creates a good logistics system for a company can extend its talents into the broad domain of general company activity and create a general management information system.

There is no reason to suppose an MIS group can actually do this—in fact, there is good reason to think it cannot.

Let me refute these errors one by one.

1. The true MIS expert does not and cannot exist.

A complete management information system consists of such a huge assortment of different types of activities that no man can possess a broad enough set of special skills to apply to even a small proportion of them. Consider the skills required to build any one of these individual information systems.

The financial accounting and control system: This includes preparation of financial statements, development of budgets and long-range plans, analyses of capital investments, publication of product costs, and so forth.

Traditionally, the controller is responsible for all these financial subsystems; with respect to the financial information systems, he plays the role that the MIS expert is supposed to play in the general management information systems. In complementary fashion, the MIS expert must have a thorough understanding of the controller's systems function.

The logistics information system: This system controls the flow of goods from the purchase of raw materials to the physical distribution of the finished products. Next to the financial control system, it is probably the most comprehensive information system in the typical manufacturing business.

A logistics system normally consists of several subsystems of varying degrees of independence. For example, there could be distinct systems for different product lines. Within each product line, furthermore, there could be subsystems for procurement, production scheduling, finished goods, inventory control, and so forth, and still others for plant utilization and expansion. Depending on its industry, a company has a larger or smaller number of complex, interrelated logistics information subsystems.

The critical point to note here is that the logistics information system is almost completely different from the financial information system. In point of fact, most of the skills needed to develop financial information systems are of

no use in developing logistics information systems and vice versa. Even the user relationships are different. In building a financial information system, the controller develops a system that provides information for management outside the finance function, whereas logistics information is normally developed and used by the people directly concerned with logistics.

Furthermore, logistics subsystems frequently have little in common with each other, so that an expert in one type of subsystem might not be able to transfer his expertise to a different type. For example, there may be little or no similarity between a procurement information system and a finished-goods distribution system. Like the financial system, the logistics information system or subsystem is a job for a specialist.

The marketing information systems: Like the two systems just described, the marketing information system can also consist of a number of subsystems. A company may maintain separate subsystems for separate product lines; and within a product line, it may maintain further subsystems for advertising and sales promotion, short-term sales forecasting, long-term sales forecasting, product planning, and so forth.

Again, the critical point is this—a marketing information system is almost completely different from the other two systems. Consequently, expertise in either or both of the other systems would be of limited value in developing a marketing information system and vice versa.

Legal services, industrial relations, and public relations: One of the major purposes of each of these staff functions is to provide top management with specialized information different from that provided by any other staff office and different from that provided by the three information systems previously described.

R&D reporting: The information system management requires in this area is distinct from all others, and expertise in these other areas offers limited help in developing an R&D information system.

In short, except in the small company (which probably needs only simple systems), there are several information systems that have very few similarities and many wide differences. Consequently, it makes no sense to regard the processes of developing and implementing these several management information systems as constituting a single and homogeneous activity.

I conclude that few, if any, individuals have the training to call themselves experts in management information systems. Indeed I believe it is much more practical to teach the new information technology to the functional experts than to teach information technologists functional specialties. After all, the man who could master all the functional specialties—the true MIS expert—would have to be an intellectual superman; and hence he does not and cannot exist except, perhaps, as a very rare exception.

If an MIS can be implemented at all, it can only be implemented by a staff group, and one of considerable size.

2. Coordinated systems for functional areas can be developed without a 'total systems approach.'

"Unless you develop the MIS as a single, integrated system, all you will get is a bunch of unrelated, uncoordinated, ineffective systems." If I have heard this statement once, I have heard it a hundred times; and it still is not true.

I have seen many systems that have intricate interfaces with one another and that are still efficient and effective. In the automobile industry, for example, the development of a new model car involves many functions—styling, engineering, product planning, finance, facility planning, procurement, and production scheduling. Each functional unit develops its internal information system for controlling its part of the operation; in addition, at each interface, the functional units exchange the information necessary to maintain coordination between them.

If an information system is ineffective, the cause is very likely to be the incompetence of the people responsible for it, *not* the absence of the general MIS approach. In this connection I might quote William M. Zani:

"Most companies have not conceived and planned their management information system with any significant amount of attention to their intended function of supporting the manager as he makes his decisions." [4]

Zani goes on to suggest a new approach to developing an MIS as a solution to this situation. My solution would be to make some personnel changes, because anyone who fails to design an information system for its users is incompetent.

Such incompetence is very prevalent. I have seen dozens of companies where management is

4. "Blueprint for MIS," HBR November-December 1970, p. 95.

not receiving half the relevant accounting information that could be made available if the financial information system had been properly designed in the first place. And although I am not sufficiently expert in other types of information systems to know whether the same situation exists there, I have no reason to believe accounting is worse than the others.

To assert that such problems as these result from the independent development of different information systems, rather than from sheer and ordinary incompetence, is simply ridiculous—and to recommend the "MIS cure" is even more ridiculous. To ensure that a company has efficient information systems which are well coordinated with one another, management need only bear down on the personnel in the various functional areas who are responsible.

3. 'The systems approach' is merely an elaborate phrase for 'good management.'

There are many definitions of the systems approach, but the following is representative:

"The systems approach to management is basically a way of thinking. The organization is viewed as an integrated complex of interdependent parts which are capable of sensitive and accurate interaction among themselves and with their environment." [5]

What does this mean? It took me some time to figure it out.

When the systems approach first appeared in the literature, I had a great deal of difficulty understanding the concept; and my confusion increased until I started asking people this question: "What would an executive do differently if he were to adopt the systems approach in place of the traditional one?"

Without exception, the replies I received made assumptions about the traditional approach that simply are not valid. For example, some assumed that the executive perceives his organization as static; others, that he fails to consider the interaction of related variables. In other words, the replies were predicated on an incompetent, even a stupid, executive.

Thus I concluded that the alleged advantages of the systems approach really result from the difference between an adequate and an inadequate manager. If you doubt this, I invite you to ask the question I did the next time you hear someone champion the systems approach to management.

It is therefore not surprising that good managers follow the systems approach, because this approach is merely the ancient art of management. Would a competent business executive plan a major expansion program without considering the sources and timing of funds, the availability of people, the possible reactions of competitors, and so forth? Certainly not. And he would consider them in relation to one another.

My conclusion, then, is that the systems approach is precisely what every good manager has been using for centuries. The systems approach may be new to science and to weapons acquisition, but it is certainly not new to business administration.

At this point, let me summarize briefly. First, an MIS would have to be developed by a *group* composed of experts in the various types of information systems used by management. This must be so because the possibility that a single individual will be expert in *all* types of information is remote. Second, the approach taken by the MIS group would be approximately the same as that taken by any competent and expert manager working in one of the functional information systems.

How, then, does the MIS approach differ from the traditional approach to information systems?

The only difference I can see is that a company's management information system would be the responsibility of one centralized group; whereas, traditionally, the information systems experts have been located in the various functional areas. This brings me to the last fallacy—that such centralization is practicable.

4. Centralizing the control of a company's information systems in a staff group creates problems that are insoluble; therefore it is simply not feasible.

It is theoretically possible to assemble a staff MIS group that is sufficiently large and diversified to have expertise in all the formal information systems described earlier—marketing, manufacturing (logistics), finance, and so forth. But to organize this group properly, the company should appoint an executive vice president for information to supervise the work of the group—that is to say, the systems of the staff vice presidents, the controller, the logistics information group, the marketing information group, and so

5. Spyros Makridakis, "The Why's and Wherefores of the Systems Approach," *European Business*, Summer 1971.

forth. But what would this accomplish? Let me ignore the fact that no sane manufacturing or marketing executive would delegate the responsibility for his information system.

One result might be that this executive vice president for information would promote better coordination between functional areas. On the other hand, of course, the problems of coordination would drastically increase in the manufacturing and marketing areas because the responsibility for the information systems had been separated from the people who hold the line responsibility. And in any event, simply having all of the information groups, including the MIS group, report to a single executive would hardly change the *approach* to developing information systems. Thus the special value of the MIS approach is still obscure.

In short, it seems to me that if any of the MIS people are competent to tell the functional experts what to do, they should be in the functional area. I see no logical way to centralize the responsibility for all the management information systems.

Significant misconception

If the MIS approach is as fallacious as I believe it to be, how has it been able to maintain even a superficial credibility?

The answer, as I have hinted earlier, is this: the early success of information technology in renovating logistics systems has been so great that there is a natural inclination to try the same methods on the company information systems as a whole.

This misconception has evolved in a natural enough way. Responsibility for a logistics system has traditionally been divided among several executives—e.g., in purchasing, in manufacturing, and in marketing. This divided responsibility has often resulted in poor coordination throughout the system. Furthermore, the people responsible for the system have often been old-fashioned in their methods and relatively unskilled in information techniques. Thus a vacuum has frequently existed with respect to the responsibility for a company's logistics information system into which the burgeoning information technology has moved easily and successfully.

However, as we have seen, there is no reason to suppose that the principles of information technology used so successfully in the logistics area can be generalized to apply to the other management information systems within a company or to the management information system considered as a whole.

Thus, when a group of experts has completed its overhaul of the logistics system, it will *not* be in a position to attack the financial, marketing, or any other system. First, the group will not have the specialist expertise required. Second, the type of problems the group may have found in the logistics area will almost certainly not exist in other areas if the staffs in these other areas are competent. Third, there will be no responsibility vacuum as in the logistics area; the MIS group will not be in a position to take over by default.

If you have any doubt about the validity of these statements, I suggest that you examine the kinds of things that any MIS group is doing. Outside of the routine computer systems, you will almost certainly find them concerned basically with parts of the *logistics* information system only.

Roots of poor information

So far this article has been quite negative. Now I should like to suggest some positive actions to mitigate the information crisis, if it can be called that. Before I propose these actions, however, it is appropriate to review the causes of management information problems.

As I have pointed out, the principal cause of poor information systems is that we have put incompetent or ineffective people in charge of these systems.

The secondary causes are somewhat more complicated.

Growing use of computers

Computers and computer-related systems activities have been growing very rapidly, and currently the cost of these activities has become very significant in many companies. In spite of large expenditures, however, the quality of the information available to management appears unimproved.

One reason is, of course, that some computer installations are not run effectively. Another is that the computer-based information systems have been oversold; management has been led to expect much more than it has received. In other words, management's dissatisfaction with its information occurs, not from any deteriora-

tion in its information systems, but from its inflated expectations.

Interface conditions

Individual systems change and improve at different rates, and this creates problems at the interfaces between them. For example, operations research techniques, used in modern logistics systems, require much more sophisticated cost accounting information than traditional cost accounting techniques can generate. Problems can also occur at the interface between production and marketing, because production-scheduling techniques are frequently much more sophisticated than the techniques ordinarily used in market forecasting.

In general, the benefits of advanced techniques may be largely lost where they are dependent on primitive ones. (To some extent, of course, the problem of proper coordination at the interfaces reflects the competency of the staff involved. Other things being equal, only an incompetent would use an advanced technique whose effectiveness would be undermined by inadequate support.)

Rapidity of change

Many companies are changing very rapidly, and it is necessary that their information systems keep pace. In some companies, information systems are *not* keeping pace. To some extent, this is caused by the inability of the staff personnel traditionally responsible for information systems to react to change. After all, many people who were once perfectly adequate in a relatively static situation become ineffective in a dynamic situation.

Greater management challenge

Management must always operate with insufficient information. And frequently, the more important the decision, the greater the uncertainty. In many areas the truth of these statements is becoming more salient because, while the role of management is becoming more complex, the new information technology is not helping significantly.

For example, I have spent many years working on control systems for decentralized companies. The problems of control in such companies today are much more difficult than they were ten years ago—increases in size, complexity, and geographical dispersion have made control much more difficult. Yet the new information technology has been of little help in this area, simply because the problems of controlling decentralized divisions do not lend themselves to computerized or mathematical solutions.

Accordingly, it is important to realize that part of our information crisis results from the nature of the present business environment. We shall simply have to live with it. This does not mean, of course, that we should not continue trying to improve the situation.

Toward real solutions

Any company that believes it is facing genuine management information problems and wants to solve them should consider the following measures.

1. Place competent people in each of the formal information systems.

To my mind there is no question that incompetency is the leading cause of problems in many management information systems. Hence the obvious answer is to retrain or replace the incompetents.

2. Examine the interfaces.

This is best done in connection with system evaluation, and the examination should focus on these evaluative questions:

O Is there adequate communication between individual groups at all important interfaces?

The executive might bear in mind formal techniques such as scheduled meetings and formal agreements.

O Does each group involved in an interface know enough about the other interfacing systems to do its job effectively?

This is a question of education. For example, cost accountants should know enough about company operations-research models to be sure these models are providing correct information; or, at the very least, they should be able to explain to the OR group the relevant limitations of the information their group can supply. On the other hand, the OR people should know enough about cost accounting to ask for the right type of data and to appreciate the limitations in the data they receive.

But although this is principally a matter of

education, it may well be that some staff members are not intellectually capable of handling interface requirements, and they may have to be replaced.

3. Examine the logistics system.

Originally many logistics systems were organized for manual data processing and have never been changed. Equally, the procurement, production, and distribution functions typically report to different executives, and consequently no one is formally responsible for the logistics information system. Since it is here that computers and information technology are most applicable, management should evaluate its logistics area and, where appropriate, reorganize it and make a staff unit, responsible for its logistics information system, report to the company officer who directs the logistic system itself.

Readers particularly interested in this topic may wish to consult these previous HBR articles by Professor Dearden:

"Can Management Information Be Automated?" March-April 1964, p. 128.

"Computers: No Impact on Divisional Control," January-February 1967, p. 99.

"How to Organize Information Systems," March-April 1965, p. 65.

"Myth of Real-Time Management Information," May-June 1966, p. 123.

For more perspective on the CBIS-MIS controversy, readers may also find these HBR articles helpful:

Warren F. McFarlan, "Problems in Planning the Information System," March-April 1971, p. 75.

William M. Zani, "Blueprint for MIS," November-December 1970, p. 95.

---

4. Organize a central computer group for systems control.

Computer use will continue to expand, and it is vital that management maintain central control over computers and computer-based information systems.[6] Such a group should be responsible for overseeing all computer-related work—for long-range planning, coordination, and control of all computer acquisitions and applications. In addition, it should be responsible for coordinating computer-based systems and might even undertake the systems and implementation work in a situation where several organization groups use the same data base.

Most companies already have such groups. Some are even called "MIS groups," although, in reality, they have authority only over computer-related work.

5. Create an administration vice president, if one does not already exist.

I recommend the creation of an office to which the following report:

- The controller.
- The treasurer.
- The computer and systems group.
- The legal office.
- The industrial relations office.
- Other offices for company relations (that is, public and governmental).
- Organization planning.

The marketing, manufacturing, and R&D groups would continue to be independent.

Such an office has several advantages:

☐ It provides better control over the staff activities. The increasing number of staff operations, together with their increasing specialization, has made it nearly impossible for the president to exercise real control here. An administrative vice president can exercise much more effective control over the size and direction of these activities.

☐ It provides a practical alternative to locating the computer and systems group in the controller's office. An administrative vice president can provide effective supervision and, at the same time, maintain an objectivity that a controller often finds difficult because of his involvement with specific computer applications.

☐ It allows the company to handle miscellaneous projects easily—for example, an evaluation of a functional information system or an analysis of the formal information entering the president's office. To take care of nonrecurring or particularly pressing information systems problems, frequently the best arrangement is to organize temporary task forces that report to the administrative vice president.

☐ It simplifies the process of coordinating staff offices.

However, I would not make the administrative vice president or the offices reporting to him

6. See Warren F. McFarlan, "Problems in Planning the Information System," HBR March-April 1971, p. 75.

responsible for the *entire* management information system. Marketing, manufacturing, and R&D would all be responsible for their own information systems. Also, the different activities reporting to his office would develop their information systems in relative independence except where interface communications are in question.

Questions for my critics

Inevitably, I shall be accused of setting up a straw issue in this article and then demolishing it.

If the MIS approach really embraces only computer-based information systems or centralized logistics systems, then I *have* set up a straw issue. No harm has been done, however, because I have at least clarified the meaning of "MIS."

But I cannot believe the concept is meant to embrace only this. I have done my best to discover what the MIS approach really is, through talking with its proponents and studying its literature; and this article honestly represents my best understanding.

If I am correct in believing that the approach pretends to embrace more than computerized systems and logistics, then I have *not* set up a straw issue. And those who doubt my conclusions, negative as these may be, would be wise to ask themselves the following questions before they take up the pen of protest:

- Which information systems are to be included in the MIS?
- What kinds of experts are to be included in an MIS group, and what training shall they have?
- Where is this group to fit into the corporate organization? In particular, what will happen to the staff groups from the controller's office, the legal department, the marketing research department, and so forth?
- What authority is the MIS group to have? Is it to have authority to design and implement systems, or is it to serve in an advisory function only?
- What can this group accomplish that cannot be better accomplished by placing information specialists under functional groups?

Arguing the viability of the MIS approach is pointless unless answers to these questions are set forth clearly. And the clearer the answers, I believe, the more transparent the MIS mirage.

Scientists & critics

The criticism of science in the twentieth century is a kind of *lèse majesté* somewhat equivalent to criticizing the Roman Catholic Church in the twelfth century. It is seldom realized that every form of intellectual endeavor with the exception of science has both practitioners and informed critics. These critics in the fields of poetry, fiction, drama, painting, sculpture, music, etc., not only function as interpreters of the practitioners to the general public, but as critics who compare and evaluate the work of the practitioners. A critic who never wrote a poem, composed a score, or painted a picture may perform the valuable service of noting that a particular poem, score, or painting is uninspired, shoddy, or imitative. Furthermore, no one seriously supposes that the work of such critics constitutes censorship or restriction on the free creative spirits of poets, musicians, or painters. But scientists have insisted that any criticism of their work does constitute censorship or a failure to appreciate the necessity for "basic" (which sometimes should be read as trivial or useless) research. . . .

The way things are now, if someone can get by with preempting the use of the term "science" he is relatively free from exposure as a charlatan even if he is one.

Mortimer Taube,
Computers and Common Sense,
New York, Columbia University Press, 1961, pp. 121, 124.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

DISEÑO DE BASES DE DATOS

A R T I C U L O

A METHODOLOGY FOR MULTI-CRITERIA INFORMATION SYSTEM DESIGN

EXPOSITOR:
Ing. Daniel Ríos Zertuche

MAYO, 1985

A methodology for multi-criteria information system design*

by JOHN S. CHANDLER and THOMAS G. DeLUTIS
The Ohio State University
Columbus, Ohio

ABSTRACT

The design dilemma faced by the designer is to satisfy a set of conflicting user demands and resolve a set of conflicting resource requirements concurrently. In light of the complexity of modern systems, it is assumed that good system design need only produce satisfactory performance for both criteria. Current evaluative techniques, however, concentrate on either the user criterion or the system criterion aspect of the total design problem, but not both. A methodology has been developed that establishes a formal liaison between the evaluation of user goals as a function of system activity and the evaluation of resource utilization as a function of user demand, thereby creating a design/evaluation process that encompasses both criteria. The methodology employs three stages in an iterative manner to produce a "satisfactory" design. The IPSS simulator models and measures system activity, multiple goal programming evaluates both user and system goals, and heuristic procedures determine design modifications to improve performance. A functional description of the methodology and an example of its use will be presented.

INTRODUCTION

In March of 1973, ACM and NBS sponsored a Workshop** on computer performance evaluation. One of the major results of that Workshop was a consensus that there have been two separate approaches to the evaluation of information systems performance—one which focuses on the computer system domain and the other whose attention is directed at the application system (user) domain. Each have their own goals and measures: the computer system domain measures are based on resource queueing and utilization statistics and the user domain is evaluated through the performance of requested services. Measures such as throughput and response time are common for the latter. The Workshop also concluded that any performance analyses "should recognize both the costs of a computer installation and the needs of users for service."

The complexity of the design problem for modern computer based information systems has increased significantly over its predecessors due to:

a. the servicing of an expanding range of user or uses with corresponding diverse performance goals and resource requirements, and
b. the dynamic and unpredictable behavior of the system as a function of design decisions and load mix.

Thus, it is quite possible to improve the performance of the system with respect to one or more users at the expense of others. Likewise, because system resources are used by different users, improving the performance characteristics of one or more resources for the benefit of specific users may have an overall detrimental effect on performance. The problem presented to the designer is to configure a system which satisfies the user criterion while achieving system resource related performance criteria.

A computer based information processing system can be viewed as a symbiotic relationship between the system's users and its hardware, software and data resources. Ideally, the system will perform "optimally" when it achieves its user oriented objectives within a minimum cost system. However, optimal solutions are seldom achieved when systems are complex, ill defined or constrained for reasons outside the control of the designer, and thus, the designer usually settles for a satisfactorily behaving system. Hopefully, systematic procedures are employed to achieve system configurations which concurrently meet the user objectives while obtaining efficient utilization of its resources. Current evaluative technologies focus on only one criterion in the system design equation, either the user or the system's resource performance. The ability to simultaneously ascertain the impact of resource performance on user goals or vice versa is not readily achievable through these methodologies. The purpose of this paper is to describe a methodology which establishes a formal liaison between the evaluation of user goals as a function of system behavior

This research is being conducted with the support of The National Science Foundation, Grant No. SIS75-21648.

** This was one in a series of Workshops sponsored jointly by ACM and NBS to examine the major issues involving computers. Performance evaluation was chosen as the topic of this Workshop because of its significant impact on computer usage. A summary of the conclusions appears in Reference 1.

and the analysis of resource performance as a function of user activity.

User oriented analyses with objective functions based on response time, throughput, and cost have been (and are continuing to be) reported in the literature. Most frequently, analytic approaches use queueing models as their basis (References 2-4 are representative of this type of analysis). Due to the necessity to maintain tractable models, many simplifications are required for a model's analytical solution. Simulation models have also been applied to user oriented analysis.[5,6] Unfortunately, these models yield only average and/or aggregate measures of system response. As a result of these simplifications, the analyses produced by both of the approaches fail in many cases to identify the relationship between users and resources. Therefore, they are suspect when used to predict the impact on system performances of modifying the current environment.

Alternatively, performance analyses can be made from the system's standpoint, treating the user and his goals in the aggregate. The most common approach is a subsystem study, where a particular part of the information system complex is isolated, with the subsystem user(s) represented by a stochastic generator, both analytic and simulative. The most emphasized areas of research has been the I/O subsystem[7-10] and CPU utilization.[11-13] The problem with this level of evaluation is that, although providing valuable local intuitive insight, these models rarely relate to the ultimate information system user, and, therefore, do not provide realistic insight into global performance.

Examinations of complete systems have also been made. Exhaustive hardware/software measurements have been analyzed by Gonzales and Cantrell[14,15] while simulation models, including an aggregate user component, have been built by Reeves and Pooch, Norland, and Lum.[16-18] Although results of the evaluations include resource utilization statistics and user oriented measures such as response time, there is little attempt in these models to relate particular resource usage to the effort on user goal attainment. (Two exceptions are Lindsay's study of the KRONOS system[19] and Hall's data base investigations.[20]) But from practical experience it is evident that there is indeed a relationship between user goals and resource usage. In fact, Buzen[21] has recently proposed some fundamental laws for computer performance which relate resource activity to global system/user measures such as response time and throughput.

It is assumed that the objective of good system design is to satisfy both performance related criteria. However, in light of the complexity of modern systems, many design decisions tend to be made without proper supportive evidence on performance. The crux of the problem is to establish a causal relationship between user goal attainment and system resource expenditures. The methodology to be discussed has been designed to establish such a liaison and will be shown to allow for the collection of heretofore hard to obtain evaluative information. The methodology measures the impact of individual user classes on internal system performance and identifies system bottlenecks which inhibit

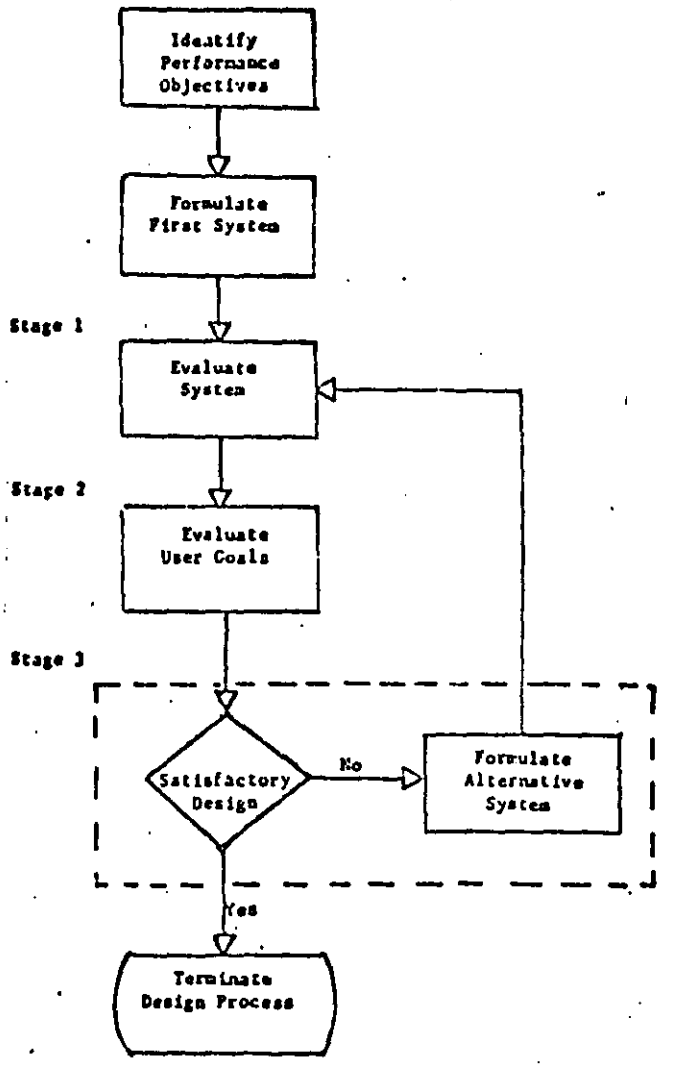


Figure 1—Stages in system design process

the attainment of user goals. This is achieved by maintaining resource utilization statistics on a user class basis. This methodology presents an evaluative framework which is capable of eliminating many of the numerous nonsatisfactory designs by directing the designer to the most advantageous ones. This methodology is an iterative one with each iteration involving three separate but integrated stages. Figure 1 illustrates the activities for an iteration. Briefly the responsibilities for each stage shown in this figure are:

Stage 1: System Evaluation

This stage is responsible for evaluating the behavior of a specific information system model. It does this by associating the hardware, software and data activities belonging to a specific design with the system's user activities. The outputs of Stage 1 are performance statistics for the resources in the aggregate and for their behavior with respect to identified users (or uses). To perform this function, the IPSS Simulator is employed.†

† IPSS is a special purpose discrete event simulator whose development was conducted with the support of the National Science Foundation, Grant No. GN-36622.

Stage 2: User Goal Evaluation

Stage 2 has two purposes. The first is to ascertain whether the user goals are being either over or under achieved. The second purpose is to determine the "best" set of guidelines for altering the *current* system configuration in order to obtain the user goals with minimum penalty for either under or over achievement. Multiple goal programming is used for this purpose. As will be seen, "best" is a function of the assigned penalty coefficients in the goal programming objective function.

Stage 3: Design Evaluation

Stage 3 has two functions. The first is to ascertain whether or not the current design's performance is satisfactory with respect to both the user criteria and the system criteria. If the design is not satisfactory, then this stage's second goal is to define a new system based upon the current design, prior alterations, and the results of the Stage 1 and Stage 2 analyses. Heuristic procedures are currently employed for Stage 3.

The focus of this paper is on the Stage 2 formulation and its formal liaison to Stage 1. The paper also identifies the unique features of IPSS which permit this multi-stage multi-criteria methodology to be achieved. The paper concludes with a discussion of the use of the Stage 1 and Stage 2 results in the Stage 3 heuristics.

EVALUATIVE REQUIREMENTS

For the purposes of this methodology, an information system is viewed as the sum of its users and their goals, and the system's services and their subordinate activities. This is illustrated in Figure 2. It is assumed that the system's analyst can identify and classify the system's users according to their service request characteristics and according to the performance constraints imposed upon the system (i.e., goals) when honoring their requests. It is also assumed that the analyst can identify those information system activities which are critical to system performance. Obviously, the complexity of the problem is increased substantially when a system supports diverse users or provides a wide spectrum of services. Whether or not the system is complex or simple, the criteria for identifying system activities should be based upon the sensitivity of the system's performance with regard to changes in their behavior.

Information system services are viewed as being a series of distinct yet interconnected activities which are invoked during the processing of a stream of user requests for the

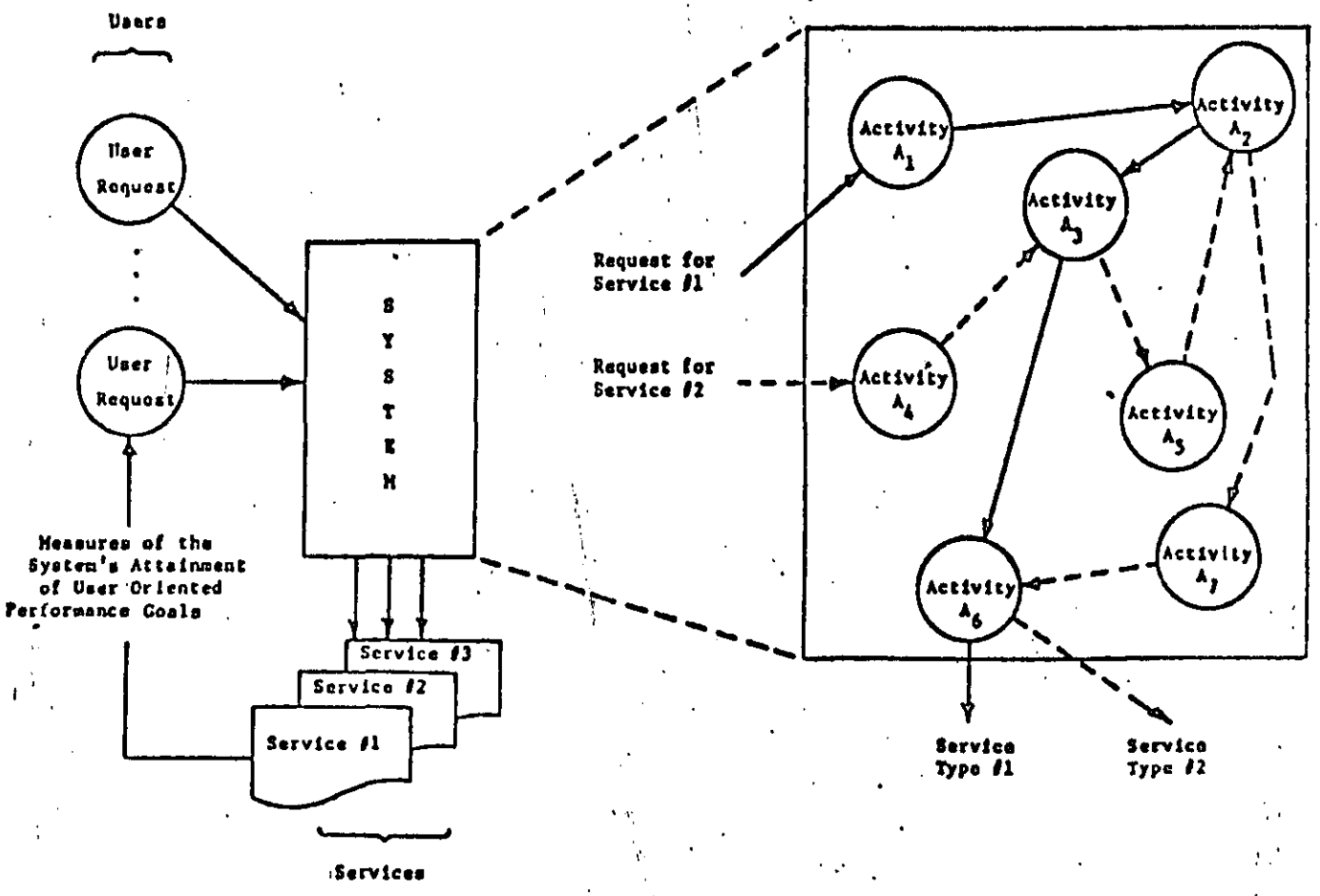


Figure 2—The methodology's view of an information system

service. Again, Figure 2 illustrates this view of a system. Most likely, system activities are aggregations of one or more traditional computer system functions that perform the following tasks:

1. request (job) scheduling,
2. task management,
3. resource allocation,
4. secondary storage I/O processing, and
5. application processing.

The choice of what constitutes an activity is part of the art of performance evaluation, however, a necessary condition for their selection is that they be measurable and that these measurements distinguish the service rates for separate classes of system services. It is also assumed that the role of performance measurement is to determine the current processing rate for the jth activity with respect to the ith service.

Figure 3 is a schematic of the functional composition of system activities. Each is viewed as an individual queueing system containing one or more priority queues and one or more identical servers. Additionally, the performance measure for the activity in processing a request type is the sum of both the queue performance and service functions of the activity. Throughout this paper, the variable $R_j(i)$ is employed to identify this performance of activity A_j with respect to Service S_i. It is assumed to be the average of performance for all the executions of A_j for S_i. Also associated with each activity A_j is the performance factor β_j which is interpreted as the scaling factor to be applied to the $R_j(i)$ to obtain the level of performance for the jth activity which minimizes the goal programming objective function. It should be noted that the problem of identifying a "good" level of performance for activities, i.e., determining the appropriate values of the $R_j(i)$'s is compounded by the multiple use of the activity by different and possibly conflicting services. Therefore, the modification of an activity's processing rate to achieve one goal may be counterproductive to the attainment of another goal. It is on this

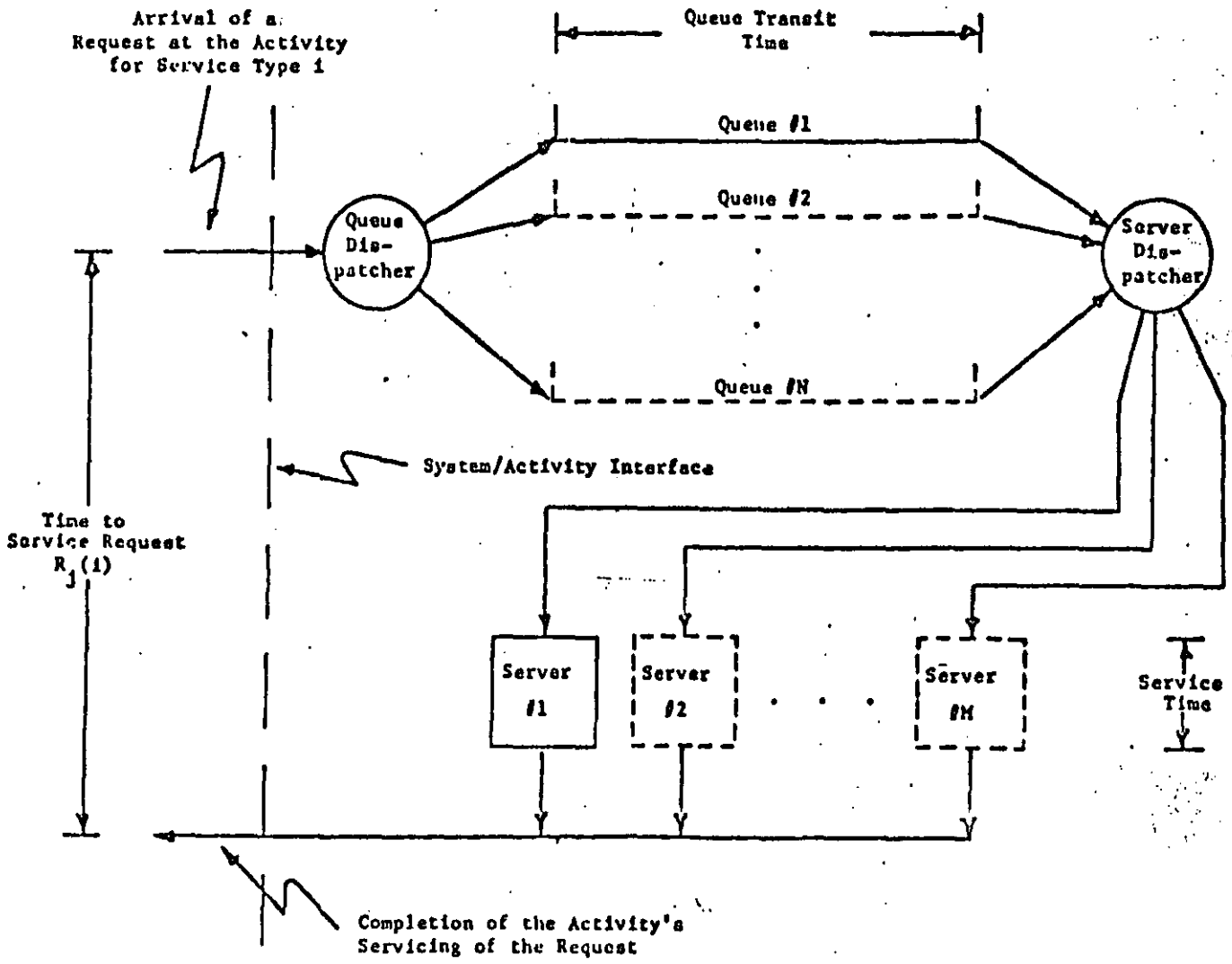


Figure 3—A conceptual view of an activity

possibility of multiple conflicting interactions and goals that this methodology is focused.

FORMULATION OF THE STAGE 2 EVALUATIVE PROCEDURE

Stage 2 is based on an evaluative procedure commonly called multiple goal programming (MGP). The procedure was first formulated by Charnes and Cooper[22] in 1961 to solve linear programming problems that had conflicting constraints. Ijiri[23] developed the details of the procedure within the framework of mathematical programming. This technique has been used to solve problems in the areas of strategic management planning such as accounting control,[23] advertising-media planning (Charnes and Cooper[24]), and resource allocation.[25] The employment of goal programming in conjunction with information system performance evaluation is a new use of the procedure.

There are three reasons for choosing multiple goal programming for use in this stage of the methodology. First, this approach can evaluate linear and ordinal multiple goal situations, both of which are inherent to information systems evaluation. For example, one user class may pay twice as much for its service, and, therefore, satisfaction of its goals may be worth twice as much as others; a linear relation. On the other hand, certain users, such as a critical patient monitoring application, may have incomparable importance relative to other classes; an ordinal relation. Second, multiple goal programming produces a solution that not only evaluates the total goal situation, but also evaluates each goal, individually. One of the purposes of this methodology is to determine the critical user classes and associated activities. Third, MGP derives the "best" design under the given goal constraints. Other design approaches such as weighting, sequential elimination, and spatial proximity,[26] are based on selecting the "best" design from a finite set of alternatives. The purpose of the overall methodology, however, is to *design* an appropriate system to satisfy the user and resource constraints. The standard formulation of a multiple goal programming problem is:

[A] Minimize $P \cdot D$
Subject to $A \cdot X + D = G$
where
- A = a matrix of technological coefficients which can be thought of as the rates at which the ith service uses the jth resource
- X = the array of resulting system resource allocation levels
- G = the array of service goals
- D = the array of discrepancies from these goals
- P = the array of penalties associated with the discrepancies in D

and where the objective function is to minimize the product of the discrepancies and their associated penalties. The solution to a multiple goal programming problem represents the best set of levels for the resource allocation vector X such that the objective function is minimized. The remainder of this section discusses the specific formulation for the Stage 2 component of the methodology.

Figure 4 illustrates the relationship between MGP, the information system activities, and its servicing of user requests. The servicing of a request type i is a sequence of activities, $A_1, A_2, \ldots, A_n$, each assumed to be measurable by $R_j(i)$. In general, the measure can be a function $M(R_j(i))$ of the service time, however, just $R_j(i)$ will be employed in the following discussion.

The performance of the information system for service type i is given by the relation

$$T_s(i) = \sum_{j=1}^{n} (R_j(i)) \tag{1}$$

where $T_s(i)$ is the average system response time to service requests of type i. Assuming that the performance goal for the service is $T_d(i)$, then the discrepancy between performance and goal is given by

$$D(i) = T_d(i) - T_s(i). \tag{2}$$

The objective of MGP is to determine new performance levels for each activity in such a manner that the weighted discrepancy, P·D, is minimized (hopefully to zero). Letting β_j be a scaling factor to be applied to the jth activity, then the new performance level for the activity is $R_j(i)\beta_j$. Incorporating the β_j's into equation (1) results in the following expression for the discrepancies:

$$D(i) = T_d(i) - \sum_{j=1}^{n} (R_j(i) \cdot \beta_j). \tag{3}$$

Observe that both positive and negative discrepancies are possible, and, therefore, the formulation of the user goal evaluation as a MGP problem becomes:‡

[B] Minimize $P_D^+ \cdot D^+ + P_D^- \cdot D^-$
s.t. $R \cdot \beta + D^+ - D^- = G$
where
- R is a matrix of service rates
- β is the array of scaling factors
- G^+ is the array of user goals
- D^+ and D^- are the arrays of, respectively, the positive and negative discrepancies from the user goals
- P^+ and P^- are the arrays of penalties associated with the corresponding positive and negative discrepancies.

The formulation serves two purposes: it evaluates goal achievement and produces the β_j's. By setting the values of the β's to reflect only the current configuration (i.e., $\beta_j = 1$),

‡ The complete derivation appears in a previous paper by the authors presented at the Annual Conference of the Computer Measurement Group, November, 1976.[27]

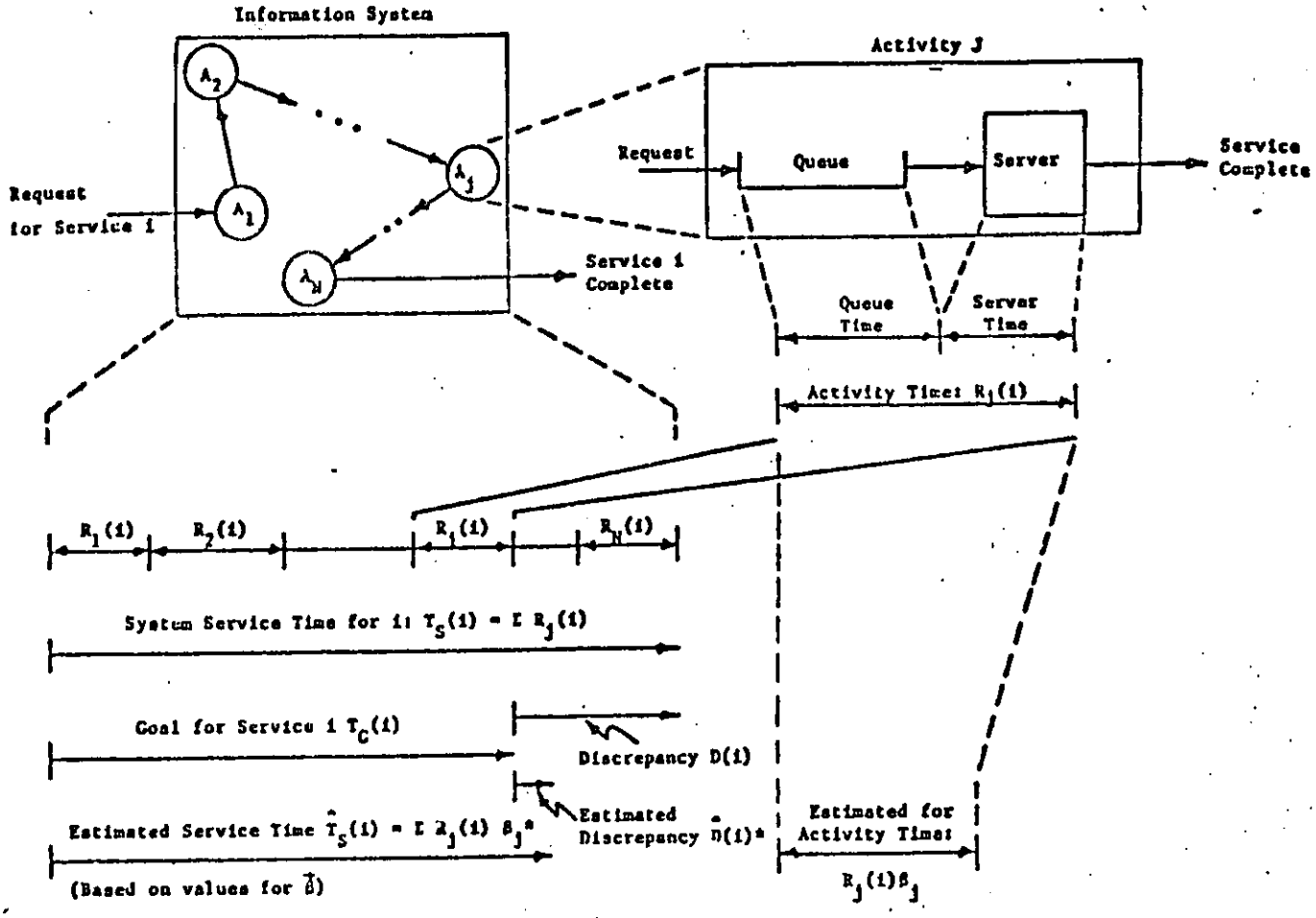


Figure 4—Relationship between goal programming and information systems characterization

the evaluation of the system's attainment of the user goals is accomplished.

Experiments with formulation [B] produced valid, but impractical sets of β's. The MGP problem as stated allowed for the possibility of solutions where a β_j could equal 0, clearly an unacceptable situation. In order to inhibit this type of solution, limits were placed on the range of possible β_j values. This was accomplished with the following set of additional constraints:

$$\beta_j + \mu_j^- - \mu_j^+ = L_j \quad (4)$$

$$\beta_j + \nu_j^- - \nu_j^+ = H_j \quad (5)$$

where

$$0 < L_j \leq 1$$

$$1 \leq H_j$$

L_j is used to restrict the alternative possibilities for the case that $\beta_j < 1$ while H_j is used for those cases that $\beta_j > 1$. In general, the set of all positive discrepancies for L_j's, $(\mu_1^+, \mu_2^+, \ldots, \mu_j^+) = M^+$ and likewise for $H_j(\nu_1^+, \nu_2^+, \ldots, \nu_j^+) = N^+$. ($M^-$ and N^- have similar definitions).

These constraints are reflected in the objective function in a manner different than previous constraints. Instead of minimizing both discrepancies, only one is minimized. In the case of L_j only μ_j^- is included, since, if μ_j^- is driven to zero, then $R_j(i)\beta_j - \mu_j^+ = L_j$, implying that $R_j(i)\beta_j > L_j$, the desired condition. Similarly, for H_j only ν_j^+ is in the objective function because minimizing ν_j^+ results in $R_j(i)\beta_j < H_j$.

These added constraints also have a physical interpretation relative to the evaluation of the system. No activity can be eliminated from a system (i.e., $\beta_j = 0$) since L_j must be greater than 0. In general, however, L_j represents the lower bound on the degree of reduction feasible for the current rate of usage for an activity. For example, $L_j = .25$ implies that the usage rate for activity j can be made, at most, four times faster, being reduced to 25% of its current rate. Similarly, H_j represents the upper bound on the degree to which an activity's rate can be increased (slowed down). It must be emphasized that these limits are only rough estimates, not exact values.

In order to reduce the number of alternatives one should minimize the number of modifications indicated per evalua-

tion iteration. Since modifications are characterized by the production of β_i's not equal to 1, a secondary objective of Stage 2 is to produce as few $\beta_i \neq 1$ solutions as possible. This is accomplished by including the constraint equation

$$\beta_i + \epsilon_i^- - \epsilon_i^+ = 1 \qquad (6)$$

while minimizing both ϵ_i^+ and ϵ_i^-

Constraints of this type provide a default value of 1 for the multiple goal programming procedures in the case where an activity is neither critically inefficient or excessive. (Note: $(\epsilon_1^-, \epsilon_2^+, \ldots, \epsilon_j^+) = E^+$.)

As a result of these added constraints, the actual formulation of the MGP problem used in Stage 2 is given in formulation [C] below:

[C] Minimize $P_D^+ \cdot D^+ + P_D^- \cdot D^- + P_M^- \cdot M^- + P_N^+ \cdot N^+ + P_E^+ \cdot E^+ + P_E^- \cdot E^-$

s.t. $R \cdot \beta + D^- - D^+ = G$

$\beta + M^- - M^+ = L$

$\beta + N^- - N^+ = H$

$\beta + E^- - E^+ = I$

where

R is the matrix of service rates

β is the array of scaling factors

G, L, H, and I are the arrays of goals for the user criteria and the respective β constraints

D^*, M^*, N^* and E^* are the arrays of positive and negative discrepancies from the respective goals

P_D^*, P_M^*, P_N^*, and P_E^* are the arrays of penalties for the associated discrepancies.

The solution variables for the MGP problem are the β's. They identify those activities that must be altered in order to improve user based or system based performance. If the value for a $\beta_i = 1$ then the service characteristics of activity j were adequate to satisfy all the user's criteria. If a $\beta_i < 1$, this implies that the current service rate of activity j is insufficient to meet the system's needs. The new service rate for the activity should be $R_j(\cdot) = (\beta_i) * (R_j(\cdot))$. If a $\beta_i > 1$ then the current service rate of activity j is faster than necessary and there exists the possibility of excess capacity. The new unit rate should be $R_j(\cdot) = (\beta_i) * (R_j(\cdot))$.

Assuming that an activity follows the characterization in Figure 3, then the analyst has three avenues of action when a $\beta_i \neq 1$. First, he can analyze the queue dispatching discipline in order to increase queue throughput (or possibly replace it with a simpler one if $\beta_i > 1$). Second, he can alter the service rate characteristics of the servers, e.g., slower hardware devices. And third, he can increase (decrease) the degree of parallelism among servers, for example, by adding (removing) a second channel, controller, etc.

LIAISON WITH STAGE 1

The critical factors in the Stage 2 evaluation are the values for the $R_j(i)$'s needed by the MGP formulation. These values are calculated in Stage 1 and are the statistical measure produced vis-a-vis the simulation. The specific model to be evaluated is the result of the heuristic procedures constituting Stage 3. The liaison is based upon the assumption that an information system can be viewed as a collection of resource allocation and task management activities and user oriented services. This view is supported by the literature, e.g., Madnick[28] and Zurcher and Randall.[29] IPSS also views the modeling of an information system in a similar manner, and thus, facilitates the development of the formal liaison with the MGP user evaluation.

The view of the system taken in Stage 2 (as illustrated in Figure 2) has an analog in IPSS. Its basic modeling concept is that of a service as shown in Figure 5. The service is classified in IPSS as a procedural facility and is capable of representing any information system activity including request (job) scheduling, task management, resource allocation, secondary storage I/O processing and application software. Since services are allocatable facilities in IPSS they have associated with them both queueing and utilization statistics. Furthermore, service behavior can be predicated on the quantity and characteristics of other IPSS hardware and software facilities. Therefore, the statistics associated with service facilities have the appropriate structure to service as the R_j's needed in Stage 2.

To complete the formal liaison between the two stages a second feature is employed. This is the Task facility. Through its use, the service facility statistics can be automatically segregated into service statistics by user. In this manner, the statistic $R_j(i)$, required by Stage 2, is collected. Thus, the Stage 2 users (indexed by i) and the activities (indexed by j) are, respectively, an IPSS model's Task and Service facilities. An $R_j(i)$ is the sum of queueing and utilization statistics gathered for Service i when executing Task j.

Stage 1 must also be adaptive to model changes dictated via Stage 3. Again, the IPSS model synthesis philosophy and language constructs permit the desired adaptiveness. This is possible for reasons too detailed to discuss in this paper. A complete description of IPSS is available in the document titled "The Information Processing System Simulator (IPSS): Language Syntax and Semantics."[30] Briefly, however, possible modifications to an existing and executing model without requiring complete reformulation include

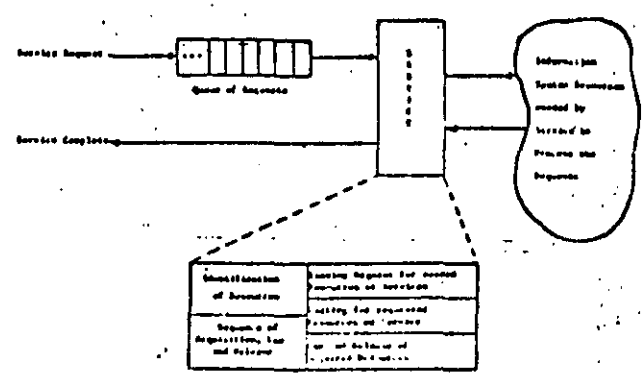


Figure 5—Functions of IPSS service entities

changes to:

1. timing and space characteristics associated with secondary storage hardware and storage media,
2. the secondary storage I/O configuration,
3. user usage patterns and service requirements,
4. file organization methods and space management policies,
5. the queueing disciplines associated with job scheduling, resource allocation and task management, and memory management policies.

IPSS supplies the Stage 1 processing with a capability of being self-adaptive with respect to Stage 3 outputs. Currently, the methodology employs modeler assistance in Stage 3. Future research will be directed at providing more sophisticated heuristics for Stage 3 in order to provide a truly self-contained iterative methodology for the multiple criteria evaluation of information systems.

STAGE 3 ANALYSIS

The functions of Stage 3 of the methodology are to determine whether the current configuration is satisfactory and to formulate a new model in light of the data provided from Stage 1 and Stage 2. Figure 6 shows the information flow to Stage 3. New models reflect the performance goals of both the user and the system. Heuristics using Sutherland's* definition of a heuristic are employed in Stage 3.

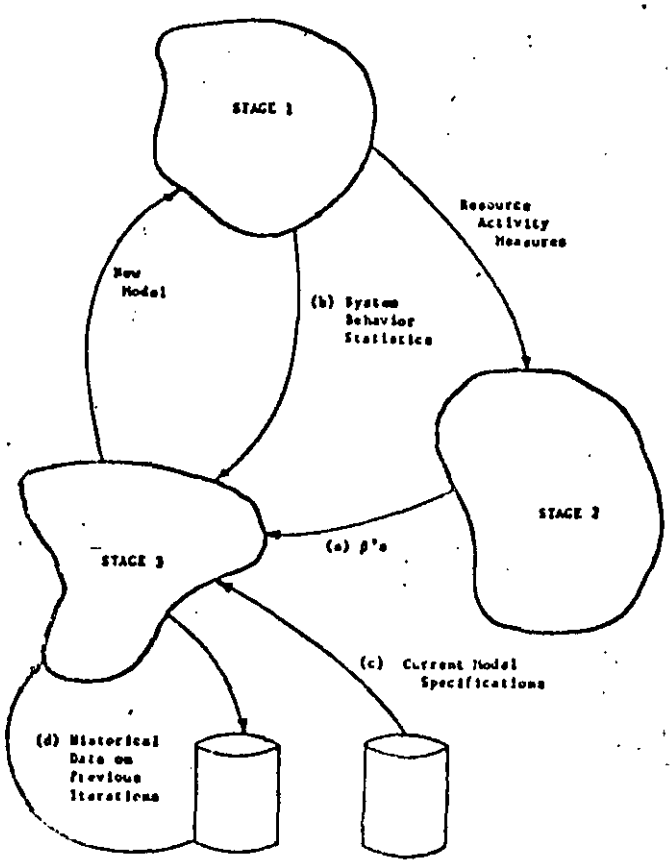
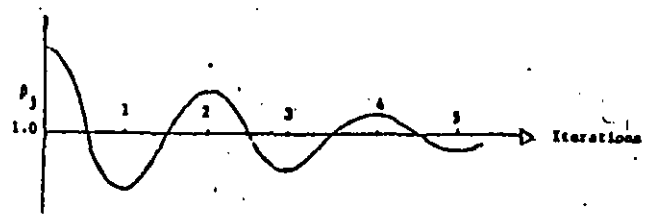


Figure 6—Information flow to stage 3

The problems encountered are complex and unstructured. Determining if the current design is acceptable requires a mixture of objective and subjective reasoning. It would be a rare situation if all the user goals and system constraints were satisfied simultaneously. Generally, an extremely wide spectrum of acceptable performance levels and alternative designs are possible, at each iteration, to satisfy both user and system criteria. Trade-offs will dominate the decision processes. Many factors effecting suitable designs may not be included in the formulations and procedures of the first two stages. For example, there may be external political, organizational or economic considerations that are not directly related to the performance of the system, but may be a major factor in the final decision. The methodology does assume, however, that the heuristic procedures have access to this external criteria.

When it has been determined that another iteration is desirable, it is assumed that the heuristics will examine current and past designs. Whatever the heuristic employed, ideally its objective is to produce a sequence of models whose β_i characteristics (for all β's) behave as follows.



The emphasis of the current research is to provide insight into the decision process for improving the performance of information systems. Stage 3 is this decision process. It is aided by input from four sources within the methodology. These sources are: (a) the Stage 2 outputs, specifically the β_i's identified to improve user and system goal performance, (b) system behavior statistics from Stage 1, (c) the current model, and (d) historical data from prior iterations. It should be emphasized that at this juncture in the development of the methodology no formal heuristic procedures have been implemented. It is one of the purposes of this research, however, to investigate the appropriateness and success of various heuristic decision rules. Rules of thumb such as those proposed by Buzen[21] are possible avenues to be investigated.

AN EXAMPLE

In order to validate the procedures developed in this methodology and the liaison between Stage 1 and Stage 2, a test case was developed. This example was modeled and executed in IPSS to satisfy the Stage 1 requirements.

* A heuristic is "a disciplined trial-and-error process, . . . an exercise in successive improvement, where we may learn from both success and failure and where the criteria for success and failure may vary with what we have previously learned" (p. 183, Reference [4]).

TABLE I—User Goal Evaluation

Goal	Over-Achievement	Under-Achievement
$USER_1$	26.6	0.0
$USER_2$	0.0	51.5
$USER_3$	0.0	258.5

Several iterations were applied, demonstrating the evaluative capabilities of the methodology. The following is a description of the problem, the corresponding model and the results of the first two iterations.

The example is a model of an on-line document retrieval system. There are three files associated with the system: an author/title index (A/T), a system document identification file (ID), and the document file itself (DOC). They are structurally related such that an entry in the A/T file points to one or more entries in the ID file and each ID file entry in turn is associated with only one DOC entry.

The model is designed so that a unique activity is associated with the accessing of each file: Activity 1 with the A/T file, Activity 2 with the ID file and Activity 3 with the DOC file. Each activity performs similar functions: obtaining and releasing devices, reading records, performing I/O techniques, but to different files.

It is assumed that the system supports three user classes, each with a different demand on the retrieval system and each characterized by a different combination of activities. The purpose of the first user class, $USER_1$, is to retrieve a document for a particular author, thus utilizing all three activities. Those in the second user class, $USER_2$, want to determine the existence/non-existence of an entry in the system for a given author, and therefore, need to use only Activity 1. The final user class, $USER_3$, already has the address of the ID entry and wants to retrieve the associated DOC entry requiring only Activity 2 and Activity 3.

In order to complete the formulation of the performance evaluation problem for this methodology, assumptions concerning the performance of the system were made. A summary of these assumptions for the user goals and β constraints and the penalties associated with the corresponding discrepancies in accordance to the requirements of formulation [C] is shown below.

$$G=(\ 150.0,\ 100.0,\ 400.0)$$
$$L=(\ .2,\ .2,\ .2)$$
$$H=(\ 5.0,\ 5.0,\ 5.0)$$
$$I=(\ 1.0,\ 1.0,\ 1.0)$$
$$P_D^+=(1000.0,\ 10.0,\ 1000.0)$$
$$P_D^-=(\ 1.0,\ 0.0,\ 0.0)$$
$$P_H^+=P_H^-=P_E^+=P_E^-=(\ 1.0,\ 1.0,\ 1.0)$$

The model constructed in IPSS assumed a simple configuration of one processor and one bank of IBM 2314 type direct access devices. Under a given loading (which is not a controllable variable in this methodology) the resulting performance statistics, the values of matrix R, are shown below.

$Q_1(1)=0.0$ $Q_2(1)=0.0$ $Q_3(1)=11.4$
$S_1(1)=37.2$ $S_2(1)=37.1$ $S_3(1)=90.9$
$R_1(1)=37.2$ $R_2(1)=37.1$ $R_3(1)=102.3$

$Q_1(2)=12.2$ $Q_2(3)=0.0$ $Q_3(3)=8.9$
$S_1(2)=36.3$ $S_2(3)=39.1$ $S_3(3)=93.5$
$R_1(2)=48.5$ $R_2(3)=39.1$ $R_3(3)=102.4$

This performance information, coupled with the goal assumptions, was input to Stage 2. The evaluation of the current configuration's performance with respect to the set of user's goals is shown in Table I. It indicates that the goals of user classes 2 and 3 were satisfied with a good margin of slack, (which is not penalized in this example) while the goal of the first user class was not satisfied (over-achievement implying non-satisfaction).

In the second phase of Stage 2, the β's are allowed to be manipulated until they satisfy the user goal constraints and best suffice the system guideline constraints. The result is the identification of these activities whose performance can be, and need to be, improved with respect to one or both of the criteria. The values for the β's as calculated were

$$\beta_1=1.0 \qquad \beta_2=1.0 \qquad \beta_3=.74$$

These are interpreted as indicating that both Activity 1 and Activity 2 were adequate to meet the user demands put to them. Activity 3, however, was found to be insufficient to satisfy the requirements of user classes 1 and 3. The modification indicated is to reduce the present rate of usage for Activity 3 by at least 1/4 in order to satisfy the user goals, in particular, the first user class goal.

The determination of whether to cease the design loop by accepting this performance or to continue by modifying the existing model is made in Stage 3. Given the stated goal/penalty structure, it was assumed that the over-achievement of $USER_1$ goal was at an unacceptable level and the design process must continue if possible. By examining the queueing and service time statistics for the first iteration, one can eliminate some of the modification possibilities. The result of Stage 3 analysis was a decision to replace the IBM 2314 type device with a faster one, i.e., an IBM 3330 type device.

The original model of this example system was dynamically altered to reflect this modification. Under the same loading as before, the following performance statistics were accumulated.

$Q_1(1)=0.0$ $Q_2(1)=0.4$ $Q_3(1)=3.8$
$S_1(1)=11.9$ $S_2(1)=11.3$ $S_3(1)=27.6$
$R_1(1)=11.9$ $R_2(1)=11.7$ $R_3(1)=31.4$

$Q_1(2)=3.8$ $Q_2(3)=0.0$ $Q_3(3)=0.0$
$S_1(2)=11.1$ $S_2(3)=12.0$ $S_3(3)=29.1$
$R_1(2)=14.9$ $R_2(3)=12.0$ $R_3(3)=29.1$

Stage 2 analysis showed that now all three user class goals were satisfied (i.e., not over-achieved). The calculation of the β's, however, indicated that while Activities 1 and 2 were still adequate ($\beta_1 = \beta_2 = 1.0$), Activity 3 now had the possibility of excess capacity ($\beta_3 = 4.0$). Although a slower and probably less expensive device for Activity 3 would be more appropriate, we had found in the first iteration that such a device was not able to satisfy all the user goals. Therefore, in future iterations, Stage 3 procedures had to examine more subtle methods of improving performance.

CONCLUSION

Modern information systems do not exist as entities unto themselves, but must interact with their environment, i.e., their users. The loading and mix of the users effect the performance of the system resources and likewise, the service characteristics of the system resources effect the satisfaction of user goals. In order to design such systems, one must satisfy a large set of users demanding a conflicting set of performance goals while operating within efficiency and minimum cost constraints. Thus, performance evaluation of information systems is a multiple criteria problem. Concurrently satisfying both of these sets of criteria is the goal of this methodology. Current available techniques, however, only address one side of the problem, either the user or the system. The methodology described in this paper establishes a formal liaison between the evaluation of user goals as a function of system behavior and the analysis of system resource performance as a function of user demand, thereby, facilitating multi-criteria evaluation.

The methodology is iterative, comprising three separate but integrated stages. The first stage models and evaluates system behavior. The particular technique employed in the first stage is IPSS and it is able to collect the necessary statistic, $R_i(i)$. The second stage evaluates the user based criteria and provides evaluative insight into performance improvement. Solution of the MGP formulation in [C] produces a set of β's, the variables of Stage 2 which indicate inefficiencies and/or excesses in the current model. And finally, the third stage heuristically determines the current model's acceptability and need for modifications.

The evaluative procedures developed for this methodology have been shown to be valid in practice. Furthermore, this methodology provides an excellent basis for continued research into areas such as:

a. investigation into the causal relationships between user demand and system activity,
b. sensitivity analysis of these relationships,
c. investigation into suitable heuristics for Stage 3, either testing existing heuristics or development of new ones, and
d. development of heuristic/modification rules to close the design loop into an automatic self-modifying process.

REFERENCES

1. Boehm, B., and T. E. Bell, "Issues in Computer Performance Evaluation: Some Consensus, Some Divergence," *PER*, Vol. 4, No. 3, 1975, pp. 4-39.
2. Gaver, D. P., and G. Hunfeld, "Multitype Multiprogramming: Probability Models and Numerical Procedures," *Proc. of CPMME*, 1976, pp. 38-43.
3. Buzen, J. P., "Computer Algorithms for Closed Queueing Networks with Exponential Servers," *CACM*, Vol. 16, No. 9, 1973, pp. 527-531.
4. Neilson, J. E., "An Analytic Performance Model of a Multiprogrammed Batch Time-Shared Computer," *Proc. of CPMME*, 1976, pp. 59-70.
5. Conger, C. R., "The Simulation and Evaluation of Information Retrieval Systems," Report 352-R-17, April 1965.
6. Roehrkasse, R. C., and D. Smith, "Simulation of Operating Systems," Tech. Report GITIS-70-11, 1970, Georgia Inst. of Tech.
7. Abate, J., H. Dubner and S. B. Weinberg, "Queueing Analysis of the IBM 2314 Disk Storage Facility," *JACM*, Vol. 15, No. 4, 1968, pp. 577-589.
8. Nahouril, E., "Direct Access Device Simulation," *IBM Systems Journal*, Vol. 13, No. 1, 1973, pp. 19-31.
9. Sherman, S. W., and R. C. Bric, "I/O Buffer Performance in a Virtual Memory System," *Symposium on the Simulation of Computer Systems*, 1976, pp. 24-35.
10. Hellerman, H. R., and H. J. Smith, "Throughput Analysis of Some Idealized Input, Output and Computer Overlap Configurations," *Computing Surveys*, Vol. 2, No. 2, 1970, pp. 111-118.
11. Kleinrock, L., and R. R. Muntz, "Processor-Sharing Queueing Models of Mixed Scheduling Disciplines for Time-Shared Systems," *JACM*, Vol. 19, No. 3, 1972, pp. 464-482.
12. Agrawala, A. K., and R. L. Larsen, "Experience with the Central Server Model on a Lightly Loaded System," *Symposium on the Simulation of Computer Systems IV*, 1976, pp. 102-109.
13. Lewis, P. A. W., and G. C. Shedler, "A Cyclic-Queue Model of System Overhead in Multiprogrammed Computer Systems," *JACM*, Vol. 18, No. 2, 1971, pp. 199-220.
14. Gonzalez, G., "Using Covariance Analysis as an Aid to Interpret the Results of a Performance Measurement," *Proc. of CPMME*, 1976, pp. 179-186.
15. Cantrell, H. N., and A. L. Ellison, "Multiprogramming System Performance Measurement and Analysis," *AFIPS Conf. Proc.*, Vol. 22, 1968, pp. 213-221.
16. Reeves, T. E., and U. W. Pooch, "A Multiple Subsystem Simulation of Processor Scheduling," *Symposium on the Simulation of Computer Systems III*, 1975, pp. 129-135.
17. Norland, K. E., and W. C. Bulgren, "A Simulation Model of GECOS III," *Proc. of ACM*, 1971, pp. 596-612.
18. Lum, V. Y., Ling, H., and Senko, M. E., "Analysis of a Complex Data Management Access Method by Simulation Modeling," *FJCC*, 1970, pp. 211-222.
19. Lindsay, D. S., "A Hardware Monitor Study of a CDC KRONOS System," Proc. of CPMME, 1976, pp. 179-186.
20. Hall, W. A., "A Simulation Model to Aid in the Design and Tuning of Hierarchical Databases," *Winter Simulation Conference*, 1974, pp. 277-284.
21. Buzen, J. P., "Fundamental Laws of Computer Performance," in *Proc. of Int'l. Symp. on Computer Performance Modeling, Measurement and Evaluation* (CPMME), 1976, pp. 200-210.
22. Charnes, A., and W. W. Cooper, *Management Models and Industrial Applications of Linear Programming*, New York, John Wiley & Sons, Inc., 1961.
23. Ijiri, Y., *Management Goals and Accounting for Control*, Chicago, Rand McNally, 1965.
24. Charnes, A., et al., "A Goal Programming Model for Media Planning," *Management Science*, Vol. 14, No. 8, April 1968, pp. 423-430.
25. Lee, S. M., *Goal Programming for Decision Analysis*, Philadelphia, Auerbach, 1972.
26. MacCrimmon, K. R., "An Overview of Multiple Objective Decision

Making," *Multiple Criteria Decision Making*, eds., J. L. Cochrane and M. Zeleny, Columbia, S. C., 1973, pp. 18-44.

27. Chandler, J. S., and T. G. DeLutis, "A Methodology for the Performance Evaluation of Information Systems Under Multiple Criteria," *Proc. of Computer Measurement Group*, 1976, pp. 221-230.
28. Madnick, S., and J. Donovan, *Operating Systems*, N. Y., McGraw-Hill, 1974.
29. Zurcher, F. W., and B. Randall, "Iterative Multi-Level Modelling: A Methodology for Computer System Design," *IFIP 68*, pp. 867-871.
30. DeLutis, T. G., "The Information Processing System Simulator (IPSS): Language Syntax and Semantics," unpublished research report (Grant No. G-36622).
31. Sutherland, J. W., *Systems: Analysis, Administration, and Architecture*, Van Nostrand Reinhold, New York, 1975.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

DISEÑO DE BASES DE DATOS

ARTICULO - DATA ARCHITECTURE AND DATA MODEL CONSIDERATIONS

EXPOSITOR:
Ing. Daniel Ríos Zertuche

MAYO, 1985

Data architecture and data model considerations*

by EDGAR H. SIBLEY and LARRY KERSCHBERG
University of Maryland
College Park, Maryland

ABSTRACT

The Data Base Management System is now a well established part of information systems technology, but the many architectures and their plethora of data models are confusing to both the practitioner and researcher. In the past, attempts have been made to compare and contrast some of these systems, but the greatest difficulty arises in seeking a common basis. This paper attempts to show how a generalized data system (GDS), represented by two different models, could form such a basis; it then proposes that data policy definitions can restrict the GDS to a specialized model, such as a relational or DBTG-like model. Finally, it proposes that this concept forms a better basis for data structure design of specific system applications.

INTRODUCTION

The seventies has seen the acceptance of the database management system (DBMS). Commercial systems and research efforts have proliferated, and the subject has become a major conference topic. However, the potential user is still left with most of the questions that first appeared: Which is the best system? Am I locking myself into one technique or implementation method?

There have been attempts at explaining similarities and differences in the basic classes of systems,[1,2] debate on the effectiveness of different data models,[3] and description of the selection and acquisition process,[4] but confusion remains.

Possibly the reason for difficulty is:

1. The topic is complex. DBMS exist, but they are so different that they defy simple comparison. They also run the gamut of size and sophistication.
2. They differ in methodology of data modelling, retaining, and querying, as well as their internal storage.
3. Testing is expensive: some representative system must be implemented on several DBMS for comparison, or difficult simulations[5] are needed.

Further problems arise in large scale database research and there is need for a common basis[6] as a formalism for describing such systems. Methods of defining the functionality of DBMS include set theory and graph theory constructs.[7] This paper attempts to define such a common basis, and show how it can be used to compare models.

DEVELOPMENT OF A FRAMEWORK

There are at least three distinct levels in an information system: the information and its structure, the data model, and the storage structuring. Obviously, no short paper can cover all three, and we will concentrate on the data model. However, consideration must be given to the information system/data model interface to set the stage for ways to define a good data model with its need to reflect the way data are interrelated, manipulated, and protected.

A *data model* is a system in which a schema may be defined; the DDLC's definition language[8] is principally a mechanism for defining the names and attributes of data elements, groupings, and relationships, while the definition of policy (integrity, security, efficiency, etc.) is almost an afterthought.

The information system—Data model interface

There is an important interface between th. organization view of information and the data model constructed to represent it. This interface is being investigated by researchers who are attempting to define a process for producing a good data base design given a set of user needs or aspirations. Reference 9 is a survey of current techniques.

Complete knowledge of the information system and its data usage characterizes the company. Operating policy, however, summarizes the internal constraints of the organization, and the way that the functional subsystems interact; one author[10] refers to these as operative and directive

* The authors wish to express their gratitude to the U.S. Army Computer Systems Command who, through grant number DAAG29-76-G-0300 (Title: Problems in the Translation and Standardization of Relational and Network Type Data Base), provided partial support and technical advice.

information structures. Several researchers[11,12] have recently advocated the collection of transactions as a basis for designing logical data structures.

Our goal, however, is to develop a framework in which to study data models, and to incorporate important parameters of the interface into the data model: Data Utilization and Operating Policy. By incorporating these parameters into the data model framework we expect to examine some classes and:

- Explain subtle differences,
- Explore declarative versus procedural aspects, and
- Characterize their "semantics."

Data architecture—A level concept

A recent paper[13] proposes that there are four abstraction levels for data machines and models. These, in increasing abstraction, are:

1. *The Defined and Populated Database:* a fully operational data system, with database defined via some definition language.
2. *The Database System:* it involves no data, but represents a specific system, with its description.
3. *The Data Model:* the data system prior to its application. The class(es) of data structures that may be supported by the system have been fixed, but not used.
4. *Data Model Theory:* a conceptual or generalized data base management system generator, assumed to be able to support all classes of data models.

These levels form a progression: From one to four is abstraction—from four to one is utilization; each level naturally subsumes or subsets the previous one.

As an example, Level 3 may be a Relational Model;[14] i.e., it can support a relational data system, but no other. Then Level 2 might be the implementation of a payroll data definition; i.e., a definition of those items (and their attributes) with procedures making up a relational payroll system. At Level 1, we see sets of payroll tuples; i.e., entries for specific people.

We shall use this concept as a basis for the paper. However, there are special operations performed in going from one level to another, defined as follows:

- Level 4 to 3: Data Policy Definition.
  In this step, the management and information system designers state the major constraints on the operational system. The resulting data model (or database machine) at Level 3 is restricted: only some classes of data structures are now allowed; some types of operation are restricted, allowing privacy or security (policy) decisions to be stated; some actions are performed automatically, allowing validation and integrity (policy) to be stated.
- Level 3 to 2: Data Operation Definition.
  Here, the administration is working with a restricted system in which data structure, some efficiency, and specific policy considerations may be stated: e.g., Data Policy Definition specified a relational system with validation at input, now Data Operation Definition defines a database of 2-tuples involving social security number and name, where the former is a nine digit element. This also involves definition of the procedures for building, maintaining, manipulating and retrieving data.
- Level 2 to 1: Data Population and Utilization.
  Finally the data must be loaded and used.

The data model generator—generalized data system

Level 4 data architecture or system can be viewed as either a theory or a machine: i.e., as either an abstract description of a method, or as a machine implementation of that method. If the concepts of Level 4 can be expressed in set theory, then the "machine" could be either a theoretical or working set processor. In this paper, we discuss two candidates for Level 4 machines, and then show how each may be restricted to Level 3 machines. It is, however, of tantamount importance that these machines be truly general, and this exercise is an attempt to show the need and generality, as well as to illustrate the parts and use of such machines. Obviously, if the Level 4 machine is sufficiently general, it will *cover* all possible data machines at Level 3 (at least relational, hierarchic, and network models). It may be considered a meta-data model or generalized data system (GDS). Furthermore, the use of such a system provides a framework for data model comparison.

The two models discussed here are:

i. The Functional Model[7] and
ii. The Set Theoretic and Extended Set Processor[16-19] modified to allow data policy and data operation definition.

THE FUNCTIONAL MODEL OF DATA

Here we consider the Functional Model of Data as a GDS. First, Level 4 structure is presented, then data policy constraints are shown to add form and structure to the model. The constraints are semantic, and allow the Functional Model to be viewed in restricted cases as either relational or DBTG (network) data models.

Level 4 structure

Level 4 is a meta-data level where the Functional Mode of data is viewed as a directed graph: its nodes represent sets and its arcs represent total functions. Nodes are either entity sets or value sets. *Entity Sets* may have any number of incoming or outgoing arcs; *Value Sets* may have only incoming arcs, because "values" are the ultimate logical

representation of information, so no arcs leave value set nodes. A typical Level 4 Functional Model graph is shown in Figure 1.

The definitional facilities for the Level 4 Functional Model consists of three creation and naming operations for value sets, entity sets, and functional specification of an entity set (i.e., specification of functions whose domain is the entity set).

There are also operations for deleting value sets, entity sets, and functions. The deletion operations have the following side-effects:

- Deletion of an entity set also implies deletion of those functions incident on it (both incoming and outgoing);
- Deletion of a value set also implies deletion of those functions incident on it;
- Deletion of a function does not affect its domain and range sets, but some may become isolated nodes and may no longer be relevant.

Data policy definition

Data policy decisions are of the following types:

- The methodology to be used to obtain the data structures.
- The representation of elements in the nodes (i.e., sets) of the Functional Model graph.
- The logical access mechanisms to be supported at Level 3.

While both information and management policy ramifications may be stated in a declarative fashion (CODASYL DDL,[8]), management policies are often enforced by transaction-driven "triggers."[20]

To refine these ideas we first address the ramifications of data policy. The most important decision is the choice of data model methodology; this will determine the richness and complexity of the allowable data structures. The methodology adopted in the Functional Model is semantic predication analysis,[21] a process developed to analyze the semantic structure of sentences.

A predication represents a whole sentence: e.g., an assertion, a command, or a question; it may be decomposed into zero, one, or two arguments and a predicate. Arguments may themselves be predications. "Downgraded predications" may qualify arguments (the semantic equivalent of adjectival clauses) or may modify predicates (the semantic equivalent of adverbial clauses). The lowest semantic level consists of semantic features which serve as atomic semantic description units. Downgraded predications play the role of semantic features of the arguments or predicates that they qualify or modify.

Here we assume that a specification of data interrelationships is available. The information analyst's role is to obtain the corresponding predication structures and map these to the Functional Model (an abstraction process). Consider the statement:

"Companies supply parts to departments in some volume"

The predication structure is shown in Figure 2, where the main predication structure "companies supply parts" is represented by the predication PN_1 with arguments A_1 (COMPANIES), A_2 (PARTS) and predicate P_1 (SUPPLY). The arrow under SUPPLY denotes the direction of the relationship represented by PN_1; it corresponds to the active voice of P_1. PN_3 and PN_3 are downgraded modifying predications representing the indirect object and adverbial

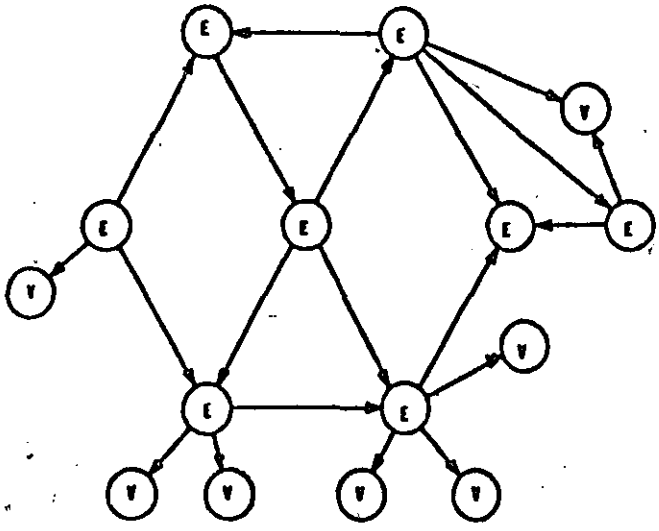


Figure 1—Functional model graph at level 4

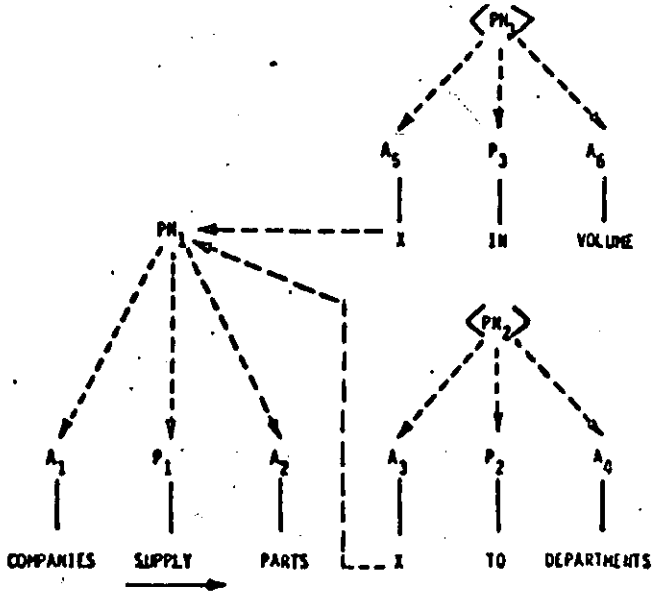


Figure 2—The predication structure for the sentence: "Companies supply parts to departments in some volume"

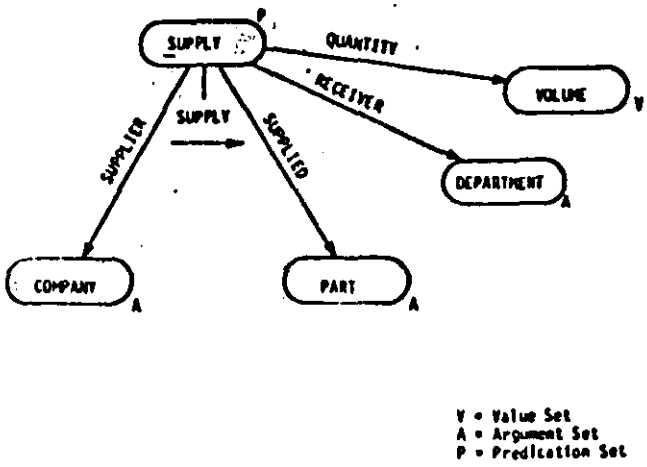


Figure 3—The functional model data structure

information, respectively. The X's refer to P_1, which their corresponding predications modify. Semantic features are not present in this example, but correspond to descriptions of the arguments (COMPANIES, PARTS, DEPARTMENTS and VOLUME). For example, DEPARTMENTS might be characterized by NAME, ADDRESS, and NUMBER.

The choice of the abstraction used to map predication structures to Functional Model data structures is part of data policy. As an example, the model might be restricted as follows:

- Semantic features map to functions whose range sets are value sets.
- Arguments corresponding to "real-world" entities map to named argument sets.
- Predications map to named predication sets, and the arcs pointing to arguments become named functions. Also the predicate and its arrow are attached to the predication set.
- Downgraded predications are represented by functions whose domain is the main predication set and range is either an argument set or a value set.

Figure 3 depicts the Functional Model data structure based on the predication structure of Figure 2 and the above abstraction rules.

Thus the choice of the methodology used to model the organization is one aspect of data policy which induces structure in passing from Level 4 to Level 3. At Level 3 the Functional Model has entity sets classified as either predication sets or argument sets (denoted P and A, respectively) and value sets remain unchanged. Functions perform two roles: a function may provide information about a predication structure or it may represent a semantic feature. The data structures supported by the predication analysis and abstraction process are depicted in Figure 4. All these data structures are possible for the Functional Model, but most applications use only [illegible]

- Functions emanating from a predication set cannot have predication sets as ranges,

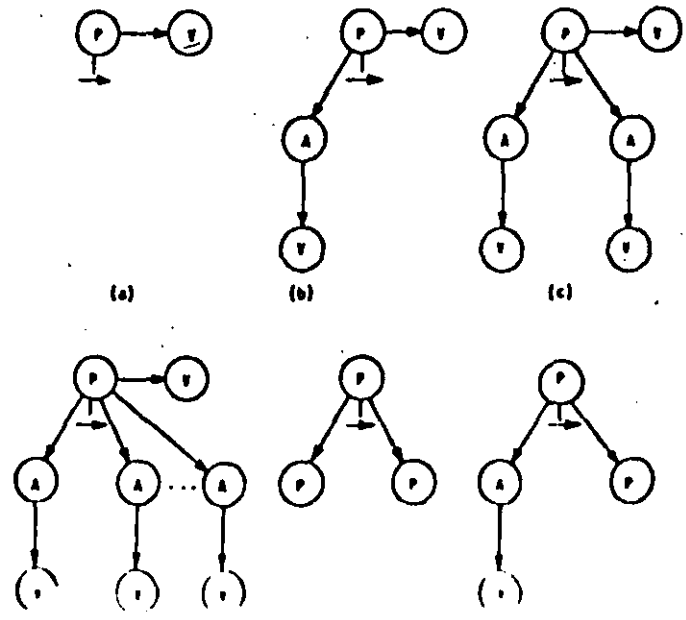
then only cases c and d are admissible. This is precisely the case in the Entity-Relationship model,[22] where predication and argument sets are called relationship sets and entity sets, respectively. In the Functional Model arcs all represent total functions, whereas in the Entity-Relationship model arcs are either 1:N mappings (for entity sets to relationship sets) or functions (for entity sets to value sets).

So far, we have only considered the methodology for obtaining data structures. Although the set types play an important role, the function types are important in expressing logical access mechanisms. Consider a binary relation α from set A to set B, and its representation, R_α, as the set of ordered pairs

$$R_\alpha = \{(a, b) \mid a \in A \;\&\; b \in B \;\&\; a \alpha b\}$$

Obviously, there exists an "inclusion" function, i, which assigns an element (a, b) of R_α to its corresponding element (a, b) of AxB, the Cartesian product of A and B. In addition, there exist functions $f: R_\alpha \rightarrow A$ and $g: R_\alpha \rightarrow B$ such that the diagram of Figure 5 "commutes."[23] In terms of the Functional Model predication structures, the binary relation α corresponds to a predication structure consisting of a predication node (labelled "α"), argument nodes (A and B) and arcs (f and g) (see Figure 5). The actual representation of the elements in the predication set is by means of the set R_α.

The functionality types of f and g are important in modelling the semantics of α: i.e., whether α is a relation, a partial function, or a total function. There are sixteen possible configurations for the predication structure, because f and g may be one-to-one, onto, both of these, or none of these. The most important configurations are



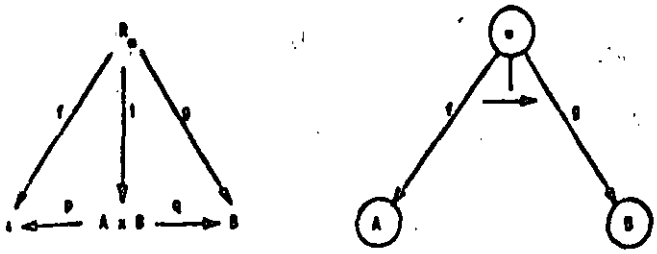


Figure 5—Representation of a binary relation and predication structure

summarized in the following:

Fact:

Let α be a binary relation from A to B with functional specification f and g.

Then:

1. If neither f nor g are one-to-one (1:1), then α is a relation;
2. If f is 1:1, then α is a partial function;
3. If f is 1:1 and onto, then α is a total function;
4. If f is 1:1 and onto, while g is 1:1, then α is a 1:1 function;
5. If f is 1:1 and onto, while g is onto, then α is an onto function;
6. If f and g are both 1:1 and onto, then α is also 1:1 and onto.

The functionality type of f determines whether α is a relation, a partial function, or a total function. If α is a function, then the functionality of g determines whether α is one-to-one, onto, a one-to-one correspondence, or none of these.

In terms of predication structures, a predication set may thus represent a relation or a function, which implies that, in the latter case, α may be represented by an arc. This is indeed true, but it must be considered a Data Policy decision.

One-to-one functions play an important role in accessing a particular element of a set. If the function f in Figure 5 is one-to-one, a particular $a \in A$ will participate in (at most) one ordered pair $(a,b) \in R_\alpha$, so that the second element of the ordered pair may be obtained by evaluating the function g. Thus, any one-to-one function outgoing from a set is a candidate (key) for accessing the elements of the set.

The notion of logical access can be extended to predication sets involving more than two argument sets and various value sets (as in Figures 3 and 4). In this case, the predication node represents an n-ary relationship, where each element may be viewed as an n-tuple of entities and values. For n-ary predication sets, a composite key is precisely the concept in Reference 22.

Finally we illustrate a management policy constraint whose predication structure has another predication node as an argument (e.g., Figure 4(f)). The operational constraint "A company must supply at least three parts to some department during a quarter to remain a valid supplier," could be modelled by a predication structure consisting of a "must" predication set with argument sets COMPANY and SUPPLY (a Figure 3 predication set), and semantic features (functions to value sets) TOTAL-PARTS and QUARTER. Elements of the "must" set might be updated by program on the occurrence of a SUPPLY transaction, but the validation would probably take place at the end of each operating quarter.

The relational model of data

The data policy constraints on the Functional Model predication structures which transform it into the Relational Model are:

- Value sets are *domains*;
- Argument sets and predication sets are *relations* whose elements are represented by *tuples* of values from domains;
- Functions whose range sets are value sets become the *attribute names* of their respective relations;
- Functions from predication sets to argument sets are replaced by the attribute name corresponding to the key (perhaps the composite key) of the argument set.

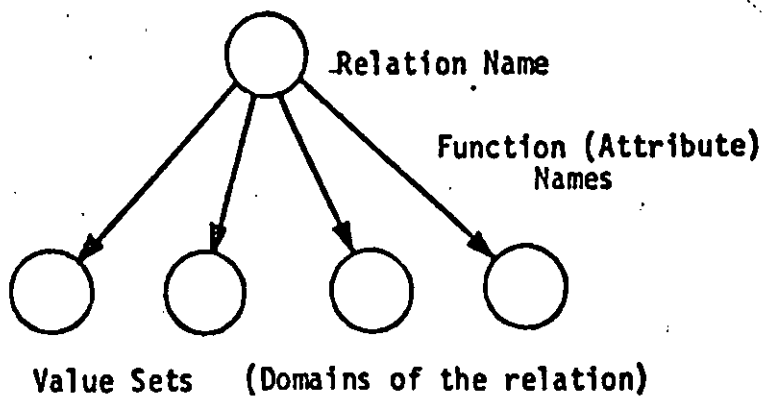
The consequence of these restrictions is to allow only nodes which represent Level 4 entity sets as relations: represented by tuples. The arcs can now only point away from nodes, and their functions are the names of the value sets of each element of the tuple: see Figure 6.

The DBTG data model

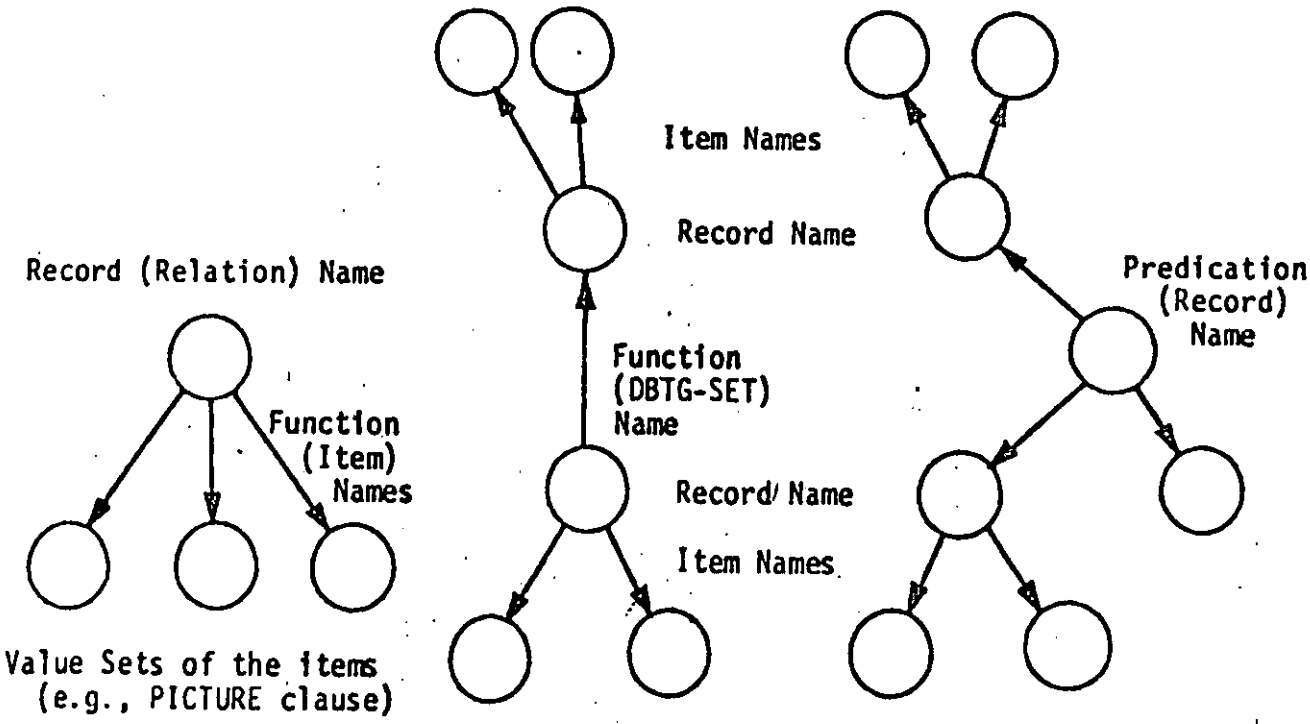
The Functional Model can also be transformed into DBTG type Data Structures by data policy definitions. First, the record will be simplified by ignoring repeating groups—in this case a record is a tuple, and may be represented in the same way as the relational data structure. The DBTG-set, in its simplest form, is a representation of a functional predication (i.e., a predication which is a function). The arrow of the function name is in the opposite direction to the arrow in the equivalent "Bachman" Diagram.[24] For a more complicated DBTG-set structure, there is a predication (a record) between two other relations (also records). This represents an intermediate or link record between two others; these structures have been discussed as linking records between two relations in Reference 25 and termed "associations" in Reference 26.

Whether the model transforms a function name into a DBTG-set or follows the Predication (Record) name model is a data policy decision: the same results may be obtained provided that one record is functionally dependent on the other—but not if the relation is N to M.

The insertion property of a DBTG-set is either AUTOMATIC or MANUAL. Which set has which property is defined at Level 3, but the ability to define these properties of a function is a Data Policy decision, which delimits the Level 4 to 3 structure and defines the DBTG Model. Similarly, the deletion properties (MANDATORY and OP-



a) Relational Data Structures



b) DBTG-like Data Structures

Figure 6—Level 3 data structures in the functional model

TIONAL) are Level 4 to 3 data policy decisions which define the operation of a DBTG model. These two properties are therefore *declaratives* associated with the arcs, similar to the functionality types in a functional model, but obviously having stronger effect. The enforcement of this policy is, of course, procedural.

Comparisons

It is useful to compare and contrast relational and DBTG data policy operations. Relational systems like System R[11] and INGRES[28] use the theory of normal forms to provide good data structures and then utilize these simple structures

through data-name linkages and data operations like JOIN and PROJECT; they apply other data policy such as consistency, integrity, and validation through system modification of the user statement (INGRES), which is immediately initiated, or through system triggers (SYSTEM R) which are transaction driven, but may be delayed. These two implementations differ in the fact that INGRES specifies policy in declaratives with procedural enforcement, while SYSTEM R is procedural both in declaration and enforcement.

The DBTG data policies are both declarative and procedural: some, such as AUTOMATIC, are declarative and apply a primitive function on operation implying a "semantics" at storage of a record; others, the Data Base Procedures, are procedural and are invoked by some trigger: e.g., on update (validation), privacy and security checking.

THE SET THEORETIC DATA MODEL

The set theoretic processor owes its existence to the concept of an extended set. The *extended set* consists of elements which themselves may be conventional sets, sequences, ordered sets, atoms, or even extended sets, but each element is identified by a position-identifier (numeric or mnemonic). The extended set is enclosed in square brackets [ ] while the ordered sets or sequences are in angle brackets ⟨ ⟩. Thus if X is an extended set consisting of elements Y and Z in positions 1 and NEXT, we have:

$$X=[\langle 1,Y\rangle, \langle NEXT,Z\rangle]$$

If we use braces { } to enclose sets, then if Y is the set of the first three integers and Z is the four-tuple consisting of "0,1,0,2" (in order), then:

$$Y=\{1,2,3\}=[\langle \#,1\rangle,\langle \#,2\rangle,\langle \#,3\rangle]$$

and

$$Z=\langle 0,1,0,2\rangle=[\langle 1,0\rangle,\langle 2,1\rangle,\langle 3,0\rangle,\langle 4,2\rangle].$$

Reference 16 shows that the extended set is a normal extension of set theory and that predicate calculus operations can be defined on the extended set. All normal set operations (union, intersection, difference, etc.) may also be defined, except that operations are position dependent.

Level 4 structure

The model consists of the following objects or elements: Atoms, which represent a number or character string; Sets, composed of any object; Ordered sets or sequences, composed of any object; Extended sets, composed of any object. Of course, no object may contain itself. The other model objects may all be represented as extended sets, position identifiers (which are atoms), and atoms. Commas are used to delimit objects in a sequence or set.

The basic language of the Level 4 data model consists of: predicate calculus expressions; algebraic expressions; assignment (storage) statements; retrieval expressions, with predicates to limit the response; and macros which simplify operations (e.g., Average, Sum).

In order to illustrate the operations of the extended set processor (XSP) at Level 4, a series of operations is given in Figure 7. The first ten operations are all of the "storage by assignment" type: they store ten extended sets, with synonyms (names) ME, YOU, etc. The VALUE function invokes a "copy" operation, which does not necessarily duplicate the string: the result may be represented internally by pointers. The set AUTHORS is therefore made up of the two extended sets ME and YOU.

The assignment operation for X defines a new extended set: it is a simple set containing the names of the two authors of this paper. This particular operation extracts (from the set AUTHORS) the values which have position indicator NAME. Furthermore, the summation operation (SUM) counts the elements (two) of this newly defined set X. The keyword LIST in the find instruction of Figure 7 is used to denote a set of 0 to "N" replications. The names and symbols which have not yet been described, such as PHONE, NAME, AUTHOR-TYPE and PICTURE, have no semantic meaning at Level 4. Later, some will have semantics in defining policy, but *here they are merely symbols*. At Level 4 the XSP is unrestricted. It will store, maintain, retrieve, or manipulate the whole or any part of any extended set. However, the XSP must have operations which can be triggered by events or conditions; these operations, applied in going from Level 4 to Level 3, are the Data Policy definitions.

Data policy definition

There are two types of XSP system applied constraints: static and dynamic. The *static constraint* is one that must always be satisfied: e.g., the data structure class must be hierarchic, or the access to some parts of the database are password protected. Such constraints imply a system action whenever violation occurs, and these may be implemented as special system actions ("compiled into the system") or as interpreted actions, with well defined error response. Most current systems "compile" these constraints (i.e., they "do not support other models of data" or "allow the following types of data protection . . ."). The *dynamic constraint* is one that is satisfied at specific times or for special operations; e.g., the validation of an element is only to be performed at input, or security is to be checked against type of user and type of operation for every access. These constraints are essentially interpreted.

The relational model of data

An example of the definition of allowable data structure classes for a relational model is given in the restrictions of Figure 8. This definition states that all sets are made up of ordered sets which contain "attribute-value" pairs (e.g., the elements ME and YOU in Figure 7 are ordered sets of three pairs). Moreover, the position identifiers shall be

```
ME = [<NAME,SIBLEY>,<SS#,017|32|7992>,<PHONE,(301)262-7138>]

YOU = [<PHONE,(301)937-7726>,<NAME,KERSCHBERG>,<SS#,294|36|4321>]

IT = [<TITLE,DATA ARCHITECTURE ETC.>,<AUTHOR,(VALUE(ME),VALUE(YOU))>]

AUTHOR-TYPE = [<NAME,NAME-TYPE>,<PHONE,PHONE-TYPE>,<SS#,SS#-TYPE>]

NAME-TYPE = (PICTURE X(25 MAX))

PHONE-TYPE = (PICTURE '('999')'999'-'9999)

SS#-TYPE = (PICTURE 999'/'99'/'9999)

ARTICLE-TYPE = [<1,TITLE-TYPE>],

TITLE-TYPE = (PICTURE X (UNRESTRICTED))

AUTHORS = (VALUE(ME),VALUE(YOU))

PRINT (NAME-TYPE)

X = (NAME | SET = AUTHORS)

Y = SUM(X)

Z = SUM ((AUTHOR|SET=IT))

X1 = (NAME|SET=AUTHORS AND SS# NOT='017|32|7992')

AUTHORSHIP-TYPE = [<AUTHOR,LIST (AUTHOR-TYPE)>,<ARTICLE,ARTICLE-TYPE>]

AUTHORSHIP = (VALUE(IT))
```

Figure 7—Some operations of an XSP at Level 4

POLICY RESTRICTIONS OF XSP.

OBJECTS: ATOM, ORDERED-SET, SET.

MEMBERSHIP: MEMBER (ORDERED-SET) IS <POSITION-ID, ATOM>,
MEMBER (SET) IS <#, ORDERED-SET>.

CONSTRAINT: ALL POSITION-ID (ORDERED-SET) IS MEMBER (POSITION-ID (ORDERED-SET-DEFINITION)).

LEVEL 3: DEFINITION WITHIN RESTRICTION.

SET OBJECT. AUTHORS.

ORDERED-SET OBJECT. AUTHOR-TYPE.

ATOM OBJECT. NAME-TYPE, PHONE-TYPE, SS#-TYPE.

TIMING.

APPLY PHONE-TYPE, SS#-TYPE ON INPUT.

APPLY NAME-TYPE ON OUTPUT, INPUT.

APPLY AUTHOR-TYPE ALWAYS.

An Invalid DEFINITION would be:

SET OBJECT. AUTHORSHIP.

ORDERED-SET OBJECT. AUTHORSHIP-TYPE.

... (See Figure 4.1: this is not a first Normal Form definition)

Figure 8— Policy statements and level 3 definition for an XSP: Example 1: Relational model

found within a definition. Thus, if we consider the Level 3 definitions of Figure 8: AUTHOR-TYPE defines the position identifiers (NAME, PHONE, SS#), then ME and YOU conform to this type of ordered set. Moreover, the set AUTHORS in Figure 7 now conforms to the definition AUTHORS in Figure 8.

It is now obvious that Figure 7 contains definitions of extended sets which can *apply* (or be applied) with semantics at Level 3. We class the elements by statements in Figure 8 which show the definition that AUTHORS contains elements (tuples) which comply with the (Figure 7) AUTHOR-TYPE definition, while the (validation) criteria of the atoms SS#-TYPE, etc., are applied on input, with NAME-TYPE checking also on output. Moreover, because some extended set operations might allow generation of invalid sets during valid operations, the AUTHOR-TYPE criterion is applied on all operations (this may be a duplicative statement, depending on the overall constraint . . . ALL POSITION-ID . . . etc., being universally applied, or "compiled into the system").

If we gave XSP, the definition for AUTHORSHIP-TYPE, an error must occur, because the definition in Figure 7 allows non-atomic elements in the ordered set (due to LIST).

The DBTG data model

In dealing with the definition of a DBTG-like structure, one is faced with some prior decisions. Using the definition of Reference 25, a DBTG-set consisting of an owner record (A_i) and three member records (B_{ij}, B_{ij}', B_{ij}''), in that order, will be represented as

$$\langle A_i, \langle B_{ij}, B_{ij}', B_{ij}'' \rangle \rangle$$

The restrictions imposed in an earlier section on DBTG-records is applied here also: no repeating groups are allowed. Thus the definition of a record in Figure 9 follows that of an ordered set in Figure 8. The definitions therefore are special only in their inclusion of membership in DBTG-sets.

Data Policy is represented by the ability to have AUTOMATIC operation of DBTG-set inclusion on storing a predefined record.

Comparisons

It has been suggested that the implementation of a relational structure in a DBTG system is a matter of:

1. Only allowing system owned sets (DBTG termed this a "SINGULAR SET");
2. Removing concepts of database procedures, inclusion and deletion properties (AUTOMATIC, MANDATORY, etc);
3. Making keys unique (using CALC with a DUPLICATES NOT ALLOWED clause);
4. Allowing new macros like JOIN and PROJECTION on (mathematical) sets of records.

It will be seen that the definitions do not truly reflect the first requirement, and that the second can be considered a triggered or system applied procedure (the result of an operation depending on the functional properties of the DBTG-set, etc.). Thus the restrictions are not correct. By further refining the models, it should be possible to determine true similarities and differences. However, it is first necessary to add the operations and their mappings—a non-trivial task.

```
POLICY RESTRICTIONS OF XSP.

    OBJECTS:      ATOM, RECORD, DBTG-SET, SYSTEM-SET.
    MEMBERSHIP:   MEMBER (RECORD) IS <POSITION-ID, ATOM>,
                  MEMBER (DBTG-SET) IS <1, RECORD> OR
                  <2, LIST (RECORD)> ,
                  MEMBER (SYSTEM-SET) IS DBTG-SET.
    CONSTRAINT:   ALL POSITION-ID (RECORD) IS MEMBER
                  (POSITION-ID (RECORD-DEFINITION)),
                  ...
```

```
LEVEL 3:  DEFINITION WITHIN RESTRICTION.
    SYSTEM-SET OBJECT.  AUTHORSHIP
    DBTG-SET OBJECT.  AUTHORSHIP-TYPE.
    RECORD OBJECT.  ARTICLE-TYPE, AUTHOR-TYPE.
    ATOM OBJECT.  TITLE-TYPE, NAME-TYPE, PHONE-TYPE, SS#-TYPE.

TIMING.
    APPLY AUTOMATIC TO ARTICLE-TYPE ON STORE.
```

Figure 9—Policy statements and Level 3 definition for an XSP: Example 2: DBTG model

CONCLUSIONS

Work with two GDS (Functional and Extended Set Models) leads us to the following conclusions:

1. The model of an enterprise information structure may be defined independently of the GDS used for its implementation.
2. If the GDS is "universal" it may store any information structure (both data syntax and semantics) in terms of its primitives.
3. The specialization of the GDS to a data model like that supported by most current research and commercial DBMS involves Data Policy definition. Using this, a GDS may be restricted to perform as one or more specific data models.
4. The process of mapping from an information structure within a GDS to a data structure within a traditional

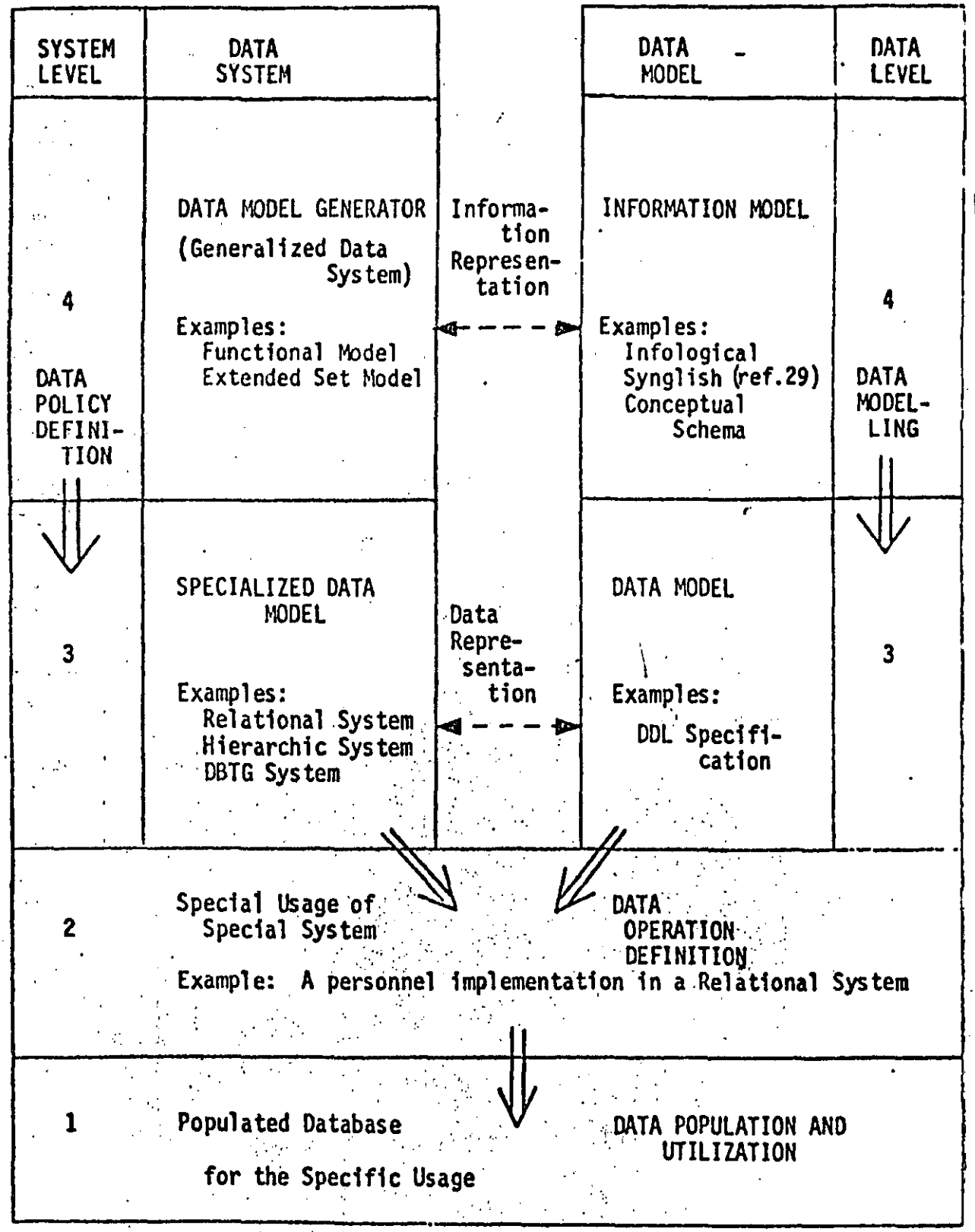


Figure 10—The parallelism of data policy and modelling

DBMS follows the restrictions of the Data Policy definition. Consequently there must be a correspondence between Data Policy restriction and *data modelling* (i.e., passing from information structure to data structure). This process is diagrammed in Figure 10.

Most information analysts already have a specialized data model (e.g., DBTG systems) in mind when constructing their information model. Thus, they take the Data Modelling route in Figure 10. We suggest that the correct (in general) process is to express conceptual inform

tures in the GDS and to "tune" the information structures to data structures by means of Data Policy decisions. In other words, it is advantageous to represent information at Level 4 so that all semantics of the information are retained.

Data Policy definitions impose added structure (with restrictions) on allowable data structures. Tradeoffs will then make it easier to consider the losses in representation of information structures in the supported data structures.

REFERENCES

1. Sibley, E. H., Guest Editor: "Special Issue on Data Base Management Systems," *ACM Computing Surveys*, Vol. 18, No. 1, March 1976, pp. 151
2. Kerschberg, L., A. Klug and D. Tsichritzis, "A Taxonomy of Data Models," *Proceedings Second International Conference on Very Large Data Bases*, Brussels, September 1976.
3. Rustin, R., Editor: *ACM SIGMOD 1974 Workshop on Data Description, Access, and Control*, "Data-Structure-Set versus Relational," May 1974, pp. 144.
4. CODASYL Systems Committee: "The Selection and Acquisition of Data Base Management Systems," Published by ACM, New York and IAG, Amsterdam, March 1976, pp. 252.
5. Reiter, A., "Data Models for Secondary Storage Representations," *Proceedings 1st International Conference on Very Large Data Bases*, ACM, Sept. 1975, pp. 87-119.
6. Hardgrave, W. T., and E. H. Sibley; "Database Research: Some Comments on Future Directions," *SIGMOD FDT 7*, pp. 3-4, 1975.
7. Kerschberg, L. and J. E. S. Pacheco, "A Functional Data Base Model," Computer Science Monograph, Pontificia Universidade Catolica do Rio de Janeiro, February, 1976, also available as Technical Report 13, Dept. of Information Systems Management, University of Maryland, 1976.
8. CODASYL Data Description Language Committee, "Data Description Language-Journal of Development," National Bureau of Standards Handbook 113, Washington, 1973, pp. 136.
9. Novak, D. O. and J. P. Fry, "The State of the Art of Database Design," *Proceedings Fifth Texas Conference on Computing Systems*, Austin, 1976, pp. 30-38.
10. Langfors, B., "Theoretical Aspects of Information Systems for Management," *Proceedings IFIP Congress 74*, pp. 937-945.
11. Rund, D. Sheppard, "Data Base Design Methodology Parts I and II," AUERBACH Publishers Inc., 1976.
12. Kahn, B. K., "A Method for Describing the Information Required by the Data Base Design Process," *Proceedings International ACM-SIGMOD Conference on Management of Data*, Washington, D.C., 1976, pp. 53-64.
13. Rothnie, J. B. and W. T. Hardgrave, "Data Model Theory: A Beginning" *Proceedings Fifth Texas Conference on Computing Systems*, Austin, 1976.
14. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, 13, June 1970, pp. 377-387.
15. Childs, D. L., "Description of a Set-theoretic Data Structure," *AFIPS Conf. Proc.*, Vol. 33, Part 1, AFIPS Press, Montvale, N.J., 1968, pp. 557-564.
16. Childs, D. L., "Feasibility of a Set-theoretic Data Structure: A General Structure Based on a Reconstructed Definition of Relation," *IFIP Congress 1968*, North Holland, Amsterdam, 1968, pp. 420-430.
17. Childs, D. L., "Extended Set Theory: A Formalism for the Design, Implementation, and Operation of Information Systems," Volume IV, *Current Trends on Programming Methodology*, edited by R. T. Yeh, Prentice-Hall,
18. Hardgrave, W. T., "A Technique For Implementing a Set Processor," *Proc. ACM Conference on Data: Abstraction, Definition, and Structure*, SIG-PLAN Notices, March 1976.
19. Hardgrave, W. T., "Set Processing: A Tool for Data Management," *Proc. ACM/NBS Fifteenth Annual Technical Symposium*, June 1976.
20. Eswaran, K., "Aspects of a Trigger Subsystem in an Integrated Database System," *Proceedings Second International Conference on Software Engineering*, San Francisco, 1976, pp. 243-250.
21. Leech, G., *Semantics*, Penguin Books Ltd., Middlesex, England, 1974.
22. Chen, P., "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Transactions on Database Systems*, Vol. 1, No. 1, March 1976, pp. 9-36.
23. MacLane, S. and G. Birkhoff, *Algebra*, Macmillan Co., 1968.
24. Bachman, C. W., "Data Structure Diagrams," *Data Base*, ACM SIGBDP Newsletter No. 1, 2, Summer 1969.
25. Sibley, E. H., "On the Equivalences of Data Based Systems," *ACM SIGMOD 1974 Workshop on Data Description, Access, and Control*, May 1974, pp. 43-76.
26. Schmid, H. A. and J. R. Swenson, "On the Semantics of the Relational Data Model," *Proceedings International ACM-SIGMOD Conference on Management of Data*, 1975.
27. Astrahan, M. M. et al., "System R: A Relational Approach to Data Base Management," *ACM Transactions on Database Systems*, Vol. 1, No. 2, 1976.
28. Stonebraker, M., "High Level Integrity Assurance in Relational Data Base Management Systems," *Proceedings of the 1975 ACM-SIGMOD Workshop*.
29. Kerschberg, L., E. A. Ozkarahan, and J. E. S. Pacheco, "A Synthetic English Query Language for a Relational Associative Processor," *Proceedings Second International Conference on Software Engineering*, San Francisco, 1976, pp. 505-519.

DISEÑO DE BASES DE DATOS

ARTICULO
A MULTI-LEVEL PROCEDURE FOR DESIGN OF FILE ORGANIZATIONS

EXPOSITOR:
Ing. Daniel Ríos Zertuche

MAYO, 1985

A multi-level procedure for design of file organizations

by EIVIND AURDAL and ARNE SØLVBERG
The University of Trondheim
Trondheim, Norway

ABSTRACT

This paper describes a multi-level procedure for design of file organizations. Necessary description of the application systems is discussed and a design procedure outlined. The main levels of the design procedure are:

- normalization of messages,
- synthesis of a logical model of the file organization,
- making a "best possible" physical realization of the logical model using a particular DBMS.

The final solution is evaluated through a performance analysis.

INTRODUCTION

The design of large data bases involves a wide range of problems, from the application system specification to the choice of hardware. The work reported here is restricted primarily to the problem of designing a "best possible" file organization given the application systems retrieval requirements, and given a particular database management system (DBMS) for the implementation. A multi-level approach to the design of file organizations is proposed.

Using a multi-level approach to the design of file organizations the following advantages can be achieved:

- transparency of the design procedure,
- elimination of non-effective solutions at an early stage,
- hardware/software selection can be postponed until the appropriate design level is reached.

The multi-level design procedure that is proposed here consists of

- specification of the information processing problem,
- transformation of the specified information structures into a logical model of the file organization,
- modification of the logical model to fit a particular DBMS,
- physical implementation of the modified model using a particular DBMS,
- evaluation of the final solution using a performance analysis.

The various levels of the design procedure are described with respect to the decisions which have to be made on each level, the necessary specifications of the application problem, and the DBMS specifications.

THE INFORMATION SYSTEM SPECIFICATIONS

We shall concentrate on that part of the information system specifications that are relevant to data base design. The specifications must model that part of the real world which we are interested in, the object system. The object system consists of a finite number of objects. An *object* is a unique part of the real world. It is something we are interested in, something we want information about. Objects may be concrete as well as abstract entities, like persons, enterprises, orders etc. Two kinds of features of an object are of interest: the properties of the objects and the relationships between the objects.

Objects of a system can be partitioned into object classes. An object class defines a set of objects which, for our purpose, are supposed to have so many common properties and relationships that we assign one common type of identifier to each object in the set.

Relationships between objects can be described by four different types of binary relations:

- 1:1 *one-to-one relation* describes a relationship between an object of one class and an object of the same or a different object class.
- 1:n *one-to-many relation* describes a relationship between one object of one object class and many objects of the same or a different object class.
- n:1 *many-to-one relation* is the inverse of the 1:n relation described above.
- n:n *many-to-many relation* exists if both the relation and its inverse are 1:n related.

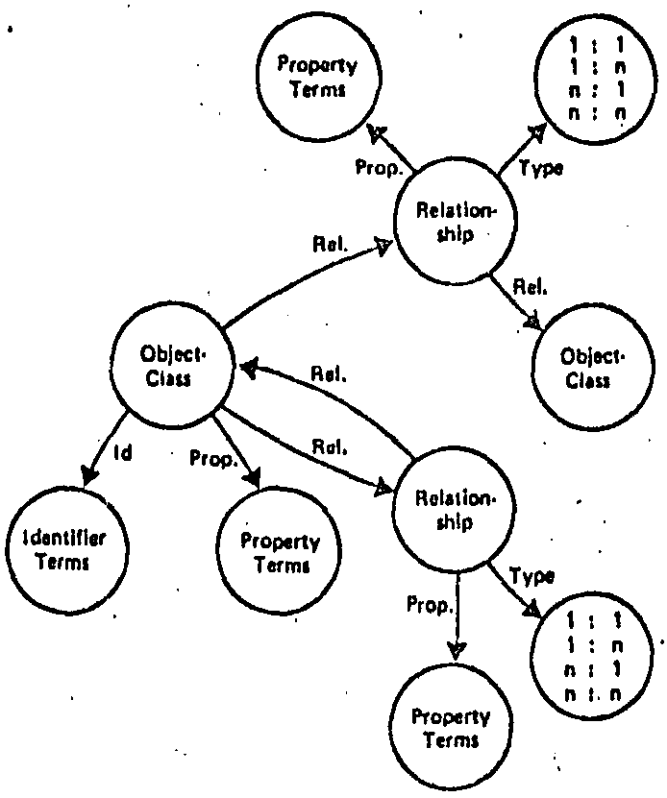


Figure 1—Features of an object

The different features of an object are illustrated in Figure 1. The figure shows how objects are related to other objects, and how identifier-items and property-items are attached to objects. The object description is illustrated with an example in Figure 2. The example shows that a customer may have a number of orders. The customer is identified by "customer no." and described by the property items "name" and "address." One order is identified by "order no" and has the property items "total sum" and "order spec."

In an information processing system, information about real world objects are stored in permanent files in a DBMS. Information about objects is exchanged between different parts of the information processing system by exchanging messages about those objects between processes. A convenient way of specifying this information flow may be

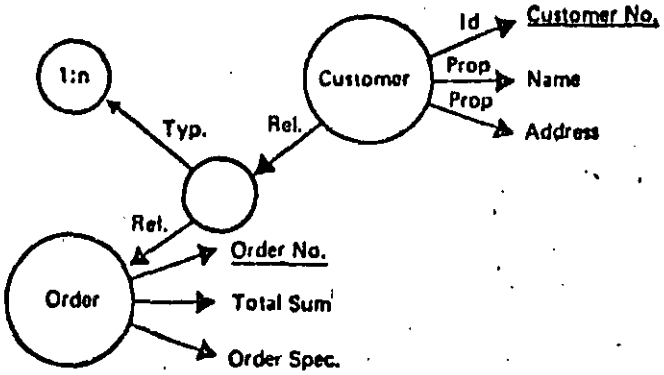


Figure 2

obtained using the principle of hierarchical systems partitioning. This will lead to a level-by-level more detailed specification of the application system, where the terminal elements will be subprocesses and logical files.

We shall illustrate our proposal for a multi-level file-design procedure by a case, which will be a simplified model of a warehouse (Figure 3). The system handles customers' orders for goods, it controls quantity in stock, and it produces refill orders for the vendors.

Suppose that a further detailing of the ORDER MANAGEMENT system results in the subsystem structure of Figure 4. The logical files "STOCK FILE" and "CUSTOMER FILE" represent permanent information, while the logical files "ORDERS," "ORDER ACCEPTED," "ORDER REFUSED" and "ORDER REGISTERED" represent message exchange between processes.

The elements of the logical files can be described by message types.

If we suppose that one customer may give several orders, and that one order may consist of a number of orderlines, the CUSTOMER FILE elements can be specified by the message type:

MT(CUSTOMER FILE):
customer no, customer name, customer address,
(order no, total price,
(order line no, article no, article name,
line price, number ordered))

where identifiers are underlined and brackets ( ) represent repeating groups.

Each of the permanent files have to be specified in this way. The permanent files are the basis for the design of the

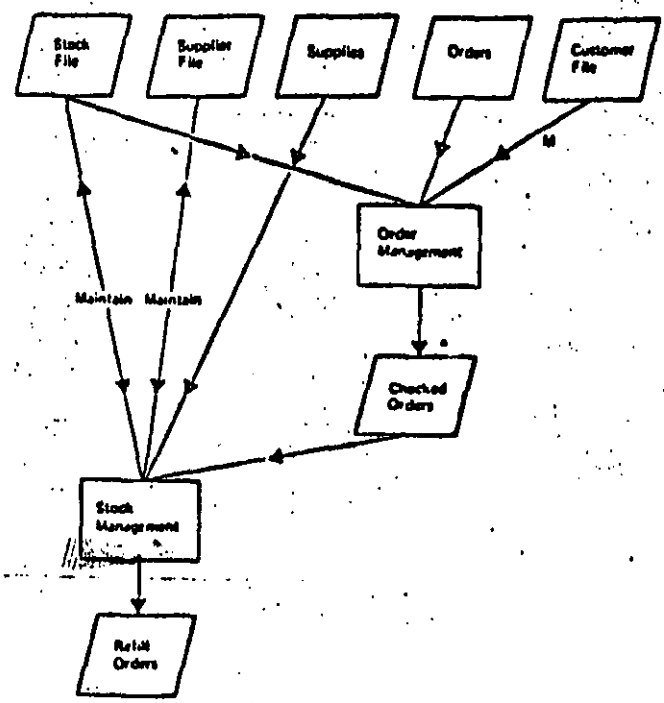


Figure 3

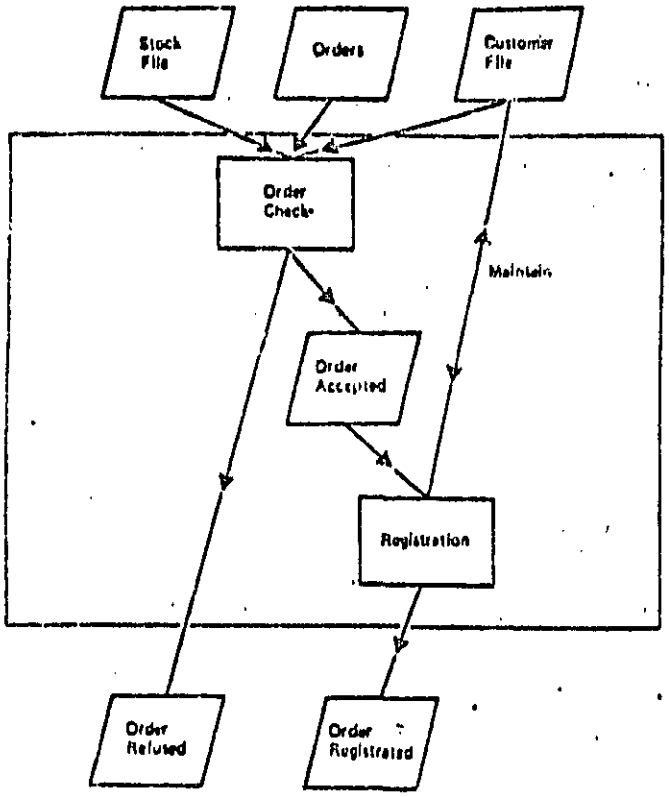


Figure 4—Order-handling system

file organization. Therefore, one has to describe the activity between them and the information system processes. This can be done with special types of processes, called *retrieval processes*. Each information system process may contain a number of retrieval processes.

Each of the retrieval processes must be one of the following four file operations:

READ,	the retrieval process reads element(s) from a permanent file.
UPDATE,	the retrieval process changes one or more property terms of an element of a permanent file.
WRITE,	the retrieval process stores new element(s) into a permanent file.
DELETE,	the retrieval process removes element(s) from a permanent file.

A retrieval process must have exactly specified search keys. The expected result must also be specified. This specification may be a description of which parts of the permanent file shall be read, or it may be a description of the message(s) which shall be stored.

The system of Figure 4 consists of two subsystems, ORDER CHECK and REGISTRATION. The subsystem ORDER CHECK handles orders from the customers. When an order arrives, the CUSTOMER FILE is checked to see if the customer is already registered. If not, some action has to be taken, for instance, to refuse the order, or to produce a customer number for this customer. Before the order can be accepted, an order number has to be produced, and the quantities in stock of the specified articles have to be checked. This subsystem will therefore contain two retrieval processes, one for checking the customer file, and one for checking the stock file. The keys are denoted by K (=key) and the expected result by O (=output).

S1: Check customer-file
Operation: READ
K: —customer name
O: —customer no.
S2: Check stock-file
Operation: READ
K: —article no.
O: —quantity

The system REGISTRATION stores new orders and possible new customers. The registration of new orders can be illustrated as follows:

S3: Order registration
Operation: WRITE
K: —customer no.
—order no.
O: —order no.
—total price
—order-line no.
—article no.
—article name
—line price
—number ordered.

} for each order-line

If all the retrieval processes defined by the application system are described in this way, then the application programs can be considered to consist of retrieval processes working towards a central data base containing permanent files (Figure 5). Through the specification of the application problem, we have now developed a basis for the design of a first logical model of the file organization.

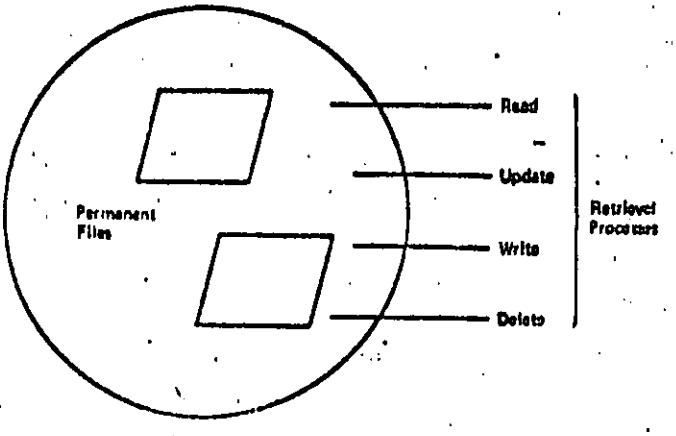


Figure 5—The basis for designing a logical data base model

DESIGNING A LOGICAL DATA BASE MODEL

A logical model of the file organization must satisfy the following requirements:

- It must be entirely based upon the information analysis (i.e., the a priori knowledge of the application problem).
- It must satisfy the requirements specified in the information analysis, i.e., the requirements of information structure and the requirements of retrieval.
- It must involve a "best possible" independence between the designed data structures and the specified application programs.
- It must be easy to realize using any particular DBMS.

The design of the logical model is done in three steps:

(1) Decomposition of the message types
(2) Synthesis of the decomposed messages into a model which satisfies the information requirements and finally
(3) A modification of the model in order to satisfy the retrieval requirements.

The decomposition procedure is similar to the normalization procedure proposed by Codd,[1] which results in an elementary file system.[2] The normalization of message types is a three-step procedure:

1. Elimination of repeating groups, hierarchical structures, or network structures,
2. Elimination of non-full dependence on the primary key,
3. Elimination of transitive dependence between the property terms.

The decomposition will be illustrated with an example. The permanent file CUSTOMER FILE, was described earlier in the following way:

$M1_0$: *customer no*, customer name, customer address, (order no, total price, (orderline no, article no, article name, line price, number ordered))

In this case a repeating group is subordinate to another repeating group, and the first step is to eliminate the first repeating group. The message type is therefore split into the message types:

$M1_1$: *customer no*, customer name, customer address
$M1_2$: *customer no, order no*, total price, (orderline no, article no, article name, line price, number ordered)

The description of an order is only dependent on "order no," not "customer no." The message type $M1_2$ is therefore split into:

$M1_{21}$: *customer no, order no*
$M1_{22}$: *Order no*, total price, (orderline no, article no, article name, line price, number ordered)

An elimination of the last repeating group results in:

$M1_{221}$: *order no*, total price
$M1_{222}$: *order no, orderline no*, article no, article name, line price, number ordered

In message $M1_{222}$ there is a transitive dependence between the property terms, "article name" is dependent on "article no," not the identifier of the message. The message type is therefore split into:

$M1_{2221}$: *order no, orderline no*, article no, line price, number ordered
$M1_{2222}$: *article no*, article name

Using this kind of normalization procedure, the message type $M1_0$ has been decomposed into the five message types $M1_1$, $M1_{21}$, $M1_{221}$, $M1_{2221}$ and $M1_{2222}$. The decomposition was entirely based upon specification of the application systems requirements, i.e., information about objects and message types. Normalization is a reversible process. That means that the original message types can be reconstructed, and therefore no information has been lost.

In the example of Figure 3, the system contains, in addition to the CUSTOMER FILE, also the permanent files STOCK FILE and SUPPLIER FILE. Suppose that the permanent file STOCK FILE contains messages which describe the various articles in stock, and that each article may have several substituting articles and several suppliers. A possible decomposition might be:

$M2_1$: *article no*, article name, quantity, price
$M2_2$: *article no, article no*
$M2_3$: *article no, supplier no*

Suppose the SUPPLIER FILE contains messages which describe the various suppliers and that one supplier may deliver several articles. That supplier may also have several refill orders registered. A possible decomposition might be:

$M3_1$: *supplier no*, supplier name, supplier address
$M3_2$: *supplier no, article no*
$M3_3$: *supplier no, refill order no*
$M3_4$: *refill order no*, number wanted, article no

Some of the message types are binary relations which represent relationships between objects. For example: $M1_{21}$: *customer no, order no*, represents a binary relation between the objects of the object classes CUSTOMER and ORDER. Other messages may describe properties of an object, for example $M1_1$: *customer no*, customer name, customer address.

Examining the message types, one may determine the

relationships between the various messages. This will be illustrated with examples:

(1) The message type
$M1_{m1}$: *customer no, order no,*
describes an 1:n relationship between messages identified by "customer no" and messages identified by "order no."

(2) In the message type
$M3_4$: *refill order no*, number wanted, article no.
the secondary key, "article no," is used as a property term. This indicates an 1:1 relationship between messages defined by "refill order no" and "article no."

The result of such an examination is illustrated in Figure 6. One may observe that there are both a 1:n relationship and a n:1 relationship between "article no" and "supplier no." This implies the existence of a n:n relationship.

The logical model may be represented in a convenient way, using a data structure diagram. The data structure diagram has two basic elements: boxes and arrows. Each box represents a class of records (or a file) and each arrow represents a meaningful relationship between records. Such a diagram may easily be drawn using the normalized message types. Message types may be represented as boxes, and a 1:n relationship is an arrow between boxes. A n:n relationship is represented as a coupling (or a relational file) between boxes.

A data structure diagram based upon the normalized message types of the permanent files CUSTOMER FILE, STOCK FILE and SUPPLIER FILE is shown in Figure 7. This is a logical model which satisfies the requirements of the information structure.

	Customer No.	Order No.	Order No. Orderline No.	Article No.	Refill Order No.	Supplier No.
Customer No.		1:n				
Order No.			1:n			
Order No. Orderline No.				n:1		
Article No.				1:n		1:n
Refill Order No.				n:1		
Supplier No.				1:n	1:n	

Figure 6—Relationships between identifiers of the normalized messages

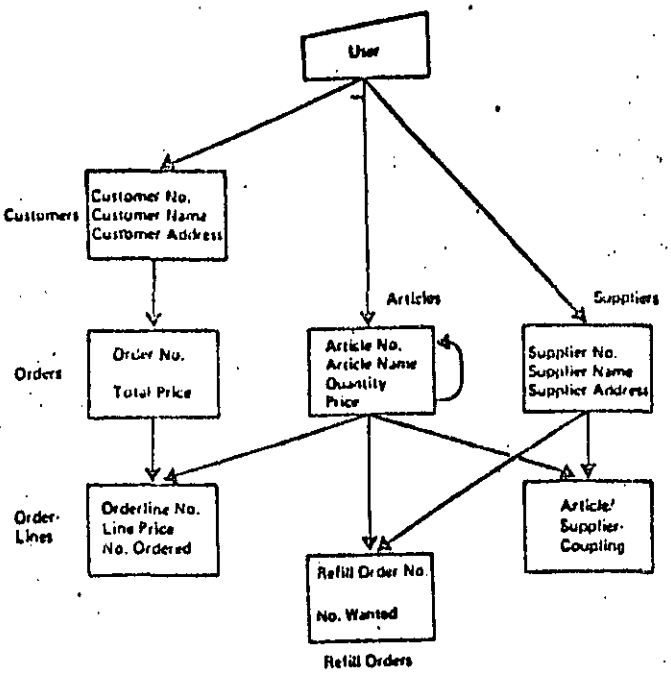


Figure 7—Data structure diagram of the normalized information structure

The next step of the design procedure is aimed at satisfying the retrieval requirements. Suppose that both "customer no" and "customer name" are used as keys by different retrieval processes, and that the response time requirements exclude a sequential processing of the CUSTOMER FILE. A modification of the data structure diagram is then necessary (Figure 8).

A search file consisting of "customer name" is established and 1:n-related to the customer file. Usually, the retrieval requirements may be satisfied establishing search files in a way similar to the one illustrated in Figure 8.

Suppose further examinations of the retrieval processes show that the items "article no.," "article name," "supplier no" and "supplier name" are retrieval keys. Necessary modifications of the logical model may result in a data structure diagram as shown in Figure 9.

A logical model has now been developed in three design steps. The first step, decomposition of messages, was based on message description and object description. The synthesis of the decomposed messages was based upon relation-

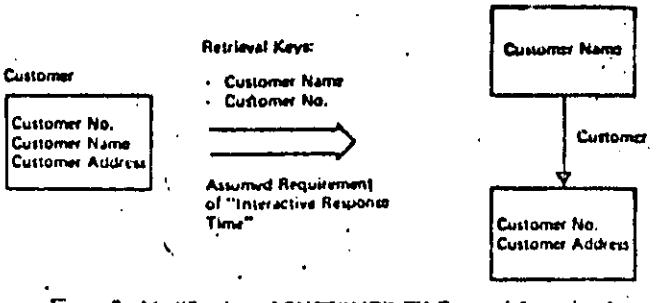


Figure 8—Modification of CUSTOMER FILE to satisfy retrieval requirements

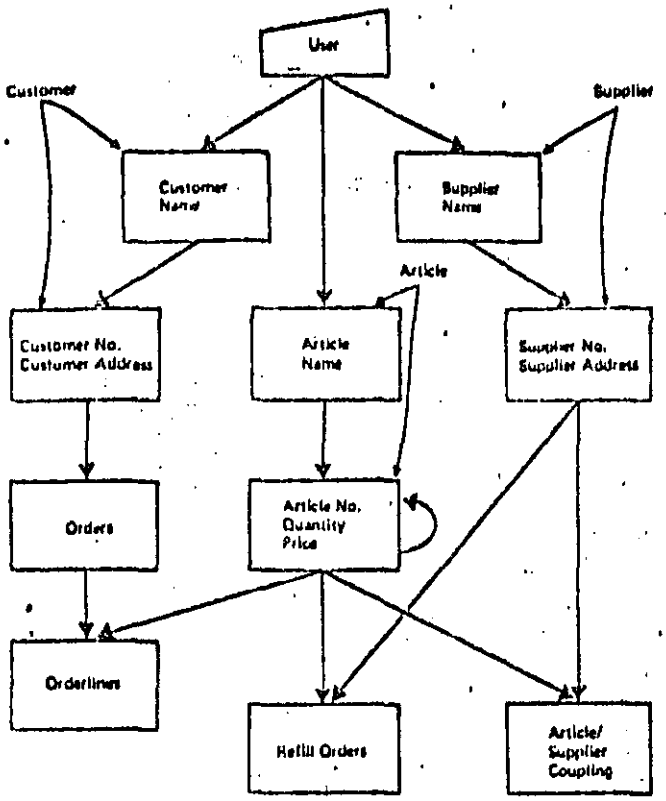


Figure 9—Logical model which satisfies the retrieval requirements

ships between objects, and the resulting model was modified, using information about the retrieval processes and the response time requirements. Thus, a logical model of the file organization has been developed, without making any decisions about any particular DBMS. The entire design process is reversible. Thus, the original information structure can be reconstructed, and the requirements stated in the information analysis is therefore satisfied.

FITTING OF THE LOGICAL MODEL TO A PHYSICAL MODEL

The next step in the design procedure is to select a particular DBMS, and modify the logical model in order to give a "best possible" physical solution. In this paper the efforts will primarily be devoted to DBTG-like DBMS-software products, but most of the problems will nevertheless be of a general character.

The available DBMS may have certain restrictions, like:

- the DBMS does not allow network-structures, only tree-structures,
- the DBMS allows direct access to only a limited number of hierarchical levels,
- the DBMS allows only a certain number of hierarchical levels
- combinations of the restrictions mentioned above.

Therefore, the first modification of the logical model is to satisfy the DBMS restrictions. Substitutions of lists by inverted lists will usually solve these problems.

The next step is to make certain adjustments of the logical model in order to:

- minimize the number of block accesses,
- minimize the data volume,
- minimize the transport of data between memory and the data files.

Unfortunately, these factors are conflicting, and an operation which takes all the factors into consideration must be carried out. One may also notice that the less complex a file organization is, the more it simplifies such operations as initial loading, reorganization and report generating.

The logical model describes the record design and the choice of access paths. Any adjustment of the logical model must therefore either modify the record design or the choice of access paths. A modification of the record design which has to satisfy the information structure requirements must be a choice between storing of duplicate data item values or the use of references. The modification of the access paths is a choice between the use of lists or inverted lists. Duplicate storing of data will be illustrated as shown in Figure 10.

The present version of the logical model describes the customers as illustrated in Figure 10(a). Suppose many retrieval processes are frequently using "customer no" as key and want information about "customer name." Such a situation will result in a large number of accesses from the member records to the owner records. An alternative version of the record layout is illustrated in Figure 10(b). The item "customer name" is stored in both the owner records and the member records. Such a solution will result in a decrease in the number of accesses, an increase in the file size, and an increase in the data read and written. In order to find the best solution one has to estimate the decrease in access costs versus the increase of input/output and storage costs.

Consequently, at this level of the design procedure, there is a need for an analysis tool which may be used in making such estimates. The analysis tool must satisfy the following

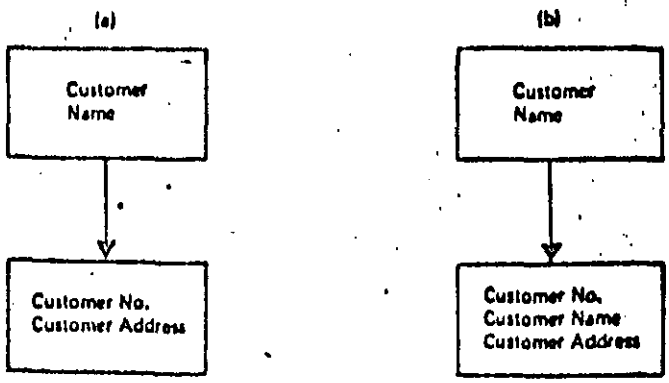


Figure 10—Alternative layouts of the customer file

conditions:

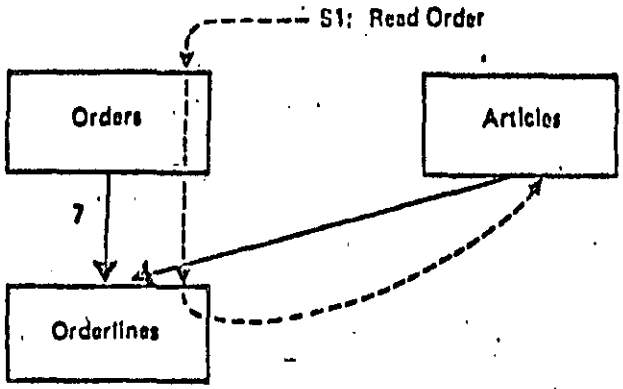
- It must be possible to perform an analysis without making any decisions about physical realization.
- the analysis must be based on a description of the logical model and the retrieval processes.
- the analysis must form a basis for making decisions of the type mentioned above.

The logical model describes the file organization, and the retrieval processes describe the activity against the file organization. Consequently, it is possible to describe the activity between the different record types and the activity between records within the same record type. It will be shown later that such a description is useful in making decisions about logical restructuring or about physical realization.

A part of the order management system described in the previous sections is shown in Figure 11. Suppose the following retrieval process exists:

S1: Read order
Operation: READ
K: —order no
O: —order no
—total price
—orderline no
—article no
—article name
—line price
} for each orderline

The process uses "order no." as key, and its access path is illustrated with a dotted line in Figure 11. One may observe that the process results in transitions between the record types ORDERS and ORDERLINES and between ORDERLINES and ARTICLES. These transitions are called retrievals of the first kind. An order will usually consist of more than one orderline, and there will be transitions between the records in the record type ORDERLINES. These transitions are called retrievals of the second kind. Suppose the average number of orderlines per order is seven. The retrieval process S1 then will result in one retrieval of the first kind between user and ORDERS, the same between ORDERS and ORDERLINES, six retrievals of the second kind within the record class ORDERLINES and finally seven retrievals of the first kind between ORDERLINES and ARTICLES.



S1: Read Order

K : • Order No.
O : • Order No.
• Total Price
• Orderline No.
• Article No.
• Line Price
• Number Ordered
} For Each Orderline

Figure 11—Illustration of the access path of a retrieval process

In order to describe the activity in the data base, one has to know the frequencies of the retrieval processes, and the scattering factors which are a result of transitions between the different record types. A transition from an owner record type to a member record type results in a scattering factor equal to the average number of member records per owner record. It should be obvious that transitions from user to a record type and transitions from a member record type to an owner record type result in a scattering factor equal to one.

The activity in the data base may be represented in a matrix (see Figure 12):

$\{\omega_{ij}\}$, $i=0,1,\cdots,n$ and $j=1,2,\cdots,n$

where

n	is the number of record types
ω_{0j}	is the number of retrievals of the first kind from user to record type no. j.
ω_{ij}	$i=1,2,\cdots,n$, $j=1,2,\cdots,n$ and $i \neq j$ is the number of retrievals of the first kind from record type no. i to record type no. j.
ω_{jj}	is the number of retrievals of the second kind within record type no. j.
$\omega_{Tj}=$	$\sum_{i=0}^{n} \omega_{ij}$, $j=1,2,\cdots,n$ is the total number of retrievals of record type no. j.

Suppose the retrieval process S1 is initiated 100 times a day. The example described in Figure 11, then results in the retrieval matrix shown in Figure 13.

The behavior of the retrieval processes will be described with a few more examples. Figure 14 shows the logical model designed in an earlier section. The numbers within the boxes are the numbers of the specific record types and the numbers on the arrows are the scattering factors for transitions from an owner record to the member records. The access paths of four retrieval processes are drawn as dotted lines. The retrieval process S1 has been described earlier, and a description of the remaining processes shown in Figure 14, may be as follows:

S2: Read customer name
Operation: READ
K: —customer no
O: —customer name

From \ To	1	2	---	j	---	n
0	ω_{01}	ω_{02}				ω_{0n}
1	ω_{11}	ω_{12}				
2						
i				ω_{ij}		
n						ω_{nn}
	$\omega_{T.1}$	$\omega_{T.2}$				$\omega_{T.n}$

n is the number of record classes

ω_{0j} is the number of retrievals of the first kind from user to record class no. j.

ω_{ij} $i = 1, 2, \cdots, n, \quad j = 1, 2, \cdots, n \quad \text{and } i \neq j$ is the number of retrievals of the first kind from record class no. i to record class no. j.

ω_{jj} is the number of retrievals of the second kind within record class no. j.

$\omega_{T.j} = \sum_{i=0}^{n} \omega_{ij}, \; j = 1, 2, \cdots, n$ is the total number of retrievals of record class no. j.

Figure 12—Retrieval matrix

The process results in one retrieval of the first kind between USER and CUSTOMERS and the same between CUSTOMERS and CUSTOMERNAMES.

S3: Update quantity
Operation: UPDATE
K: —article no
O: —quantity

The process updates the number of articles in stock, and results in one retrieval of the first kind from USER to ARTICLES.

S4: Remove refill order
Operation: DELETE
K: —refill order no
O: —texstring (deletion OK/not OK)

The process removes a refill order and deletes the relationships to ARTICLES and SUPPLIERS. The result is one retrieval of the first kind from USER to REFILL ORDERS, one from REFILL ORDERS to ARTICLES and one from REFILL ORDERS to SUPPLIERS. Suppose that, in addition to the retrieval processes already mentioned, there are other retrieval processes which complete the user's communications with the data base. A possible retrieval matrix may be the one in Figure 15.

One may observe that $\omega_{46}=3990$ and $\omega_{64}=0$. This indicates that "article no" and "article name" also should be stored in ORDERLINES. The decrease in the number of retrievals has to be paid for with an increase of the data volume.

Such a modification of the logical model results in the data structure diagram shown in Figure 16. Further examinations of the retrieval matrix may lead to other modifica-

From \ To	Orders	Orderlines	Articles
User	100		
Orders		100	
Orderlines		600	700
Articles			
Total	Σ	Σ	Σ

Figure 13—The contributions from the retrieval process S1 to a retrieval matrix

tions of the logical model and thus the final solution will be fitted to the selected DBMS in a "best possible" way.

PHYSICAL IMPLEMENTATION

The final logical model describes the final record design and the final access paths. The next step in the design

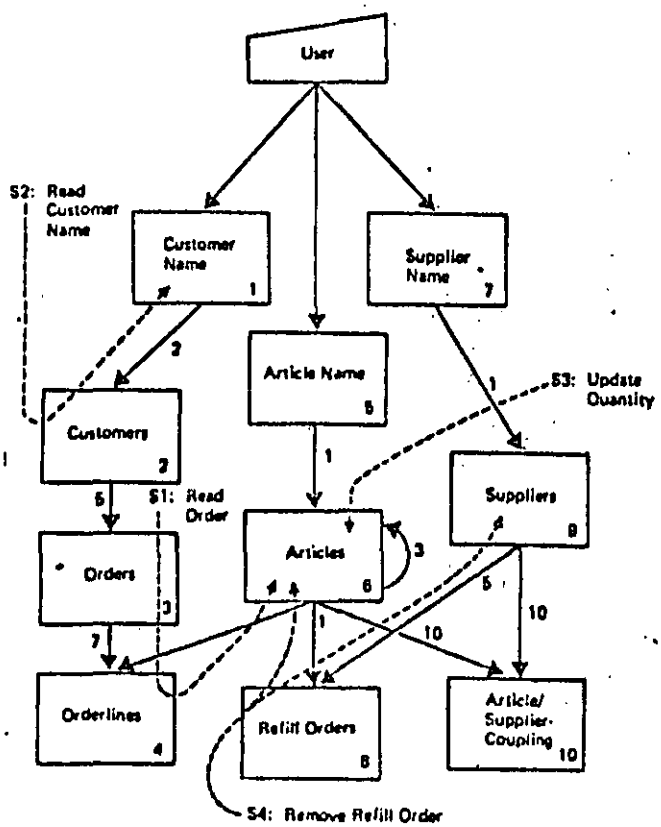


Figure 14

From \ To	1	2	3	4	6	6	7	8	9	10
0	150	105	570		155	495	5	20	11	
1		150								
2	105	150								
3		380		570						
4				3420		3990				
5						155				
6					1635	75		45		130
7									5	
8						20			55	
9							31			1
10						10			1300	1177

Figure 15—Complete retrieval matrix

procedure is a physical implementation of the logical model. This process consists of four main steps:

- choice of storage structures for the record types,
- physical location of the records,
- implementation of the relationships between the records,
- determination of the file dimensions.

The physical implementation is dependent on the particular DBMS and consequently it is not possible to formulate any general rules to solve the problem. Therefore, in this paper,

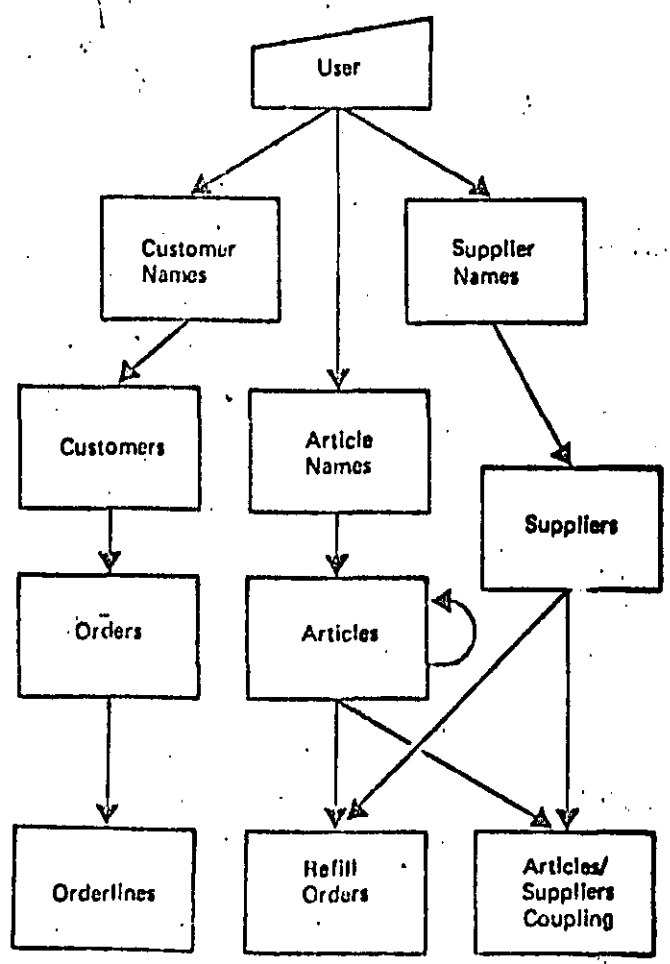


Figure 16

it will be shown how the problems can be solved using a DBTG-DBMS.

Choice of storage structure

A DBTG-DBMS allows the following types of storage structures:

- a direct structure (DIRECT)
- a randomized structure (CALC)
- an index sequential structure (INDEX)
- a sequential structure (VIA)

Usually, there will be no problem in making a choice between these structures. For example a choice between direct access and sequential access is determined by the response time requirements. If direct access is necessary, one should use a direct structured file if possible. If not, an index sequential or randomized structure has to be used. A randomized structure is usually cheaper than an index sequential structure. Therefore, if there is no sequential searching a randomized structure should be used. Of course, these rules are not of general validity, but they may be used in most cases.

Physical location

The physical location may, to some extent, be controlled by the following statements:

(i) $\left\{WITHIN \text{ area-name-1} \left[\left\{\begin{matrix}\text{integer-4}\\ \text{data-base-data-name-4}\end{matrix}\right\} \left\{\begin{matrix}THRU\\ THROUGH\end{matrix}\right\} \left\{\begin{matrix}\text{integer-5}\\ \text{data-base-data-name-5}\end{matrix}\right\}\right]\right\} \ldots$

The statement may be used to specify in what physical data file(s) the various record types are to be placed.

(ii) *LOCATION* MODE IS *VIA* set-name-1 SET
If a sequential record type has more than one owner record type, this statement may be used to place the member records close to one of the owner records.

(iii) *INTERVAL* IS integer-3 PAGES
This statement may be used to place member records close to their owner record, for example in the same physical block.

The retrieval matrix describes the activity between records and it may therefore be a useful tool in determining the physical locations of the records. High activity between certain records implies that they should have physical locations close together.

Implementation of relationships

The implementation of relationships between records (i.e., set types) are done via list structures in a DBTG-DBMS.

A set may be organized as a one-way or two-way list using the statement

MODE IS *CHAIN* [LINKED *PRIOR*]

The choice of list structure has to be based on the description of the retrieval processes. High activity from particular member records to their owner record indicates the use of a pointer from the member records to their owner record. This can be realized using the statement

[LINKED TO *OWNER*]

Determination of the file dimensions

The final step is to determine the dimensions of the data files, i.e., determine the parameters:

- file volumes
- volumes of overflow areas
- volumes of index tables
- load percents
- block sizes

A "best possible" choice of values for the parameters mentioned above is dependent on the particular DBMS and the particular hardware used. Consequently, such decisions should be made in connection with a performance analysis. This will be further discussed in the next section.

PERFORMANCE ANALYSIS

An evaluation of the final solution should be done using a performance analysis. The most convenient criterion of a good solution is low total cost per time. Various types of costs which affect the choice of file organization are:

- storage costs
- access costs
- data transfer costs
- sorting costs
- reorganization costs

In addition there will be costs for using cpu and memory.

In order to get a better picture of the performance analysis procedure, a simplified case, the one-identifier case, will be considered. In the one-identifier case the entire file organization consists of a single file with records consisting of one identifier item and one or more property items.

In order to develop a cost function, it is necessary to have information about

- the retrieval activity
- the storage structure and the dimensions of the file
- the accounting routine of the particular machine system

The first step is to decide on a storage structure. For a particular storage structure one may decide on block size and load percent. Those decisions will usually be results of local optimizations, depending on the particular computer

system that is used. In addition, it is necessary to estimate the volume of internal housekeeping data in the file. Thus, the file may be described by the following parameters:

- number of records
- record size
- block size
- load percent
- size of overflow-area
- volume of housekeeping data

Based on this description, one may estimate the average retrieval length (measured in block accesses). How this can be done for the most common used storage structures is described in Reference 3. Estimates of the average use of memory and CPU for each retrieval process must also be made. The retrieval processes are described with the following parameters:

- frequencies
- use of memory
- use of CPU
- retrieval lengths

The parameters mentioned above are used as parameters in the accounting routine of a particular machine system, and a price may be computed.

It is then possible to adjust the file parameters, for example the block size, and look at how these adjustments influence the total price. Thus, a minimization of the cost function may be done in an iterative way.

Searching in the multi-identifier case means that logical conjunctions between the separate identifier items have to be done. The most common ways to implement conjunctions are multilists or inverted lists.

Figure 17(a) illustrates a multilist. T1 and T2 are terms which are parts of the identifier. A multilist consists of key-

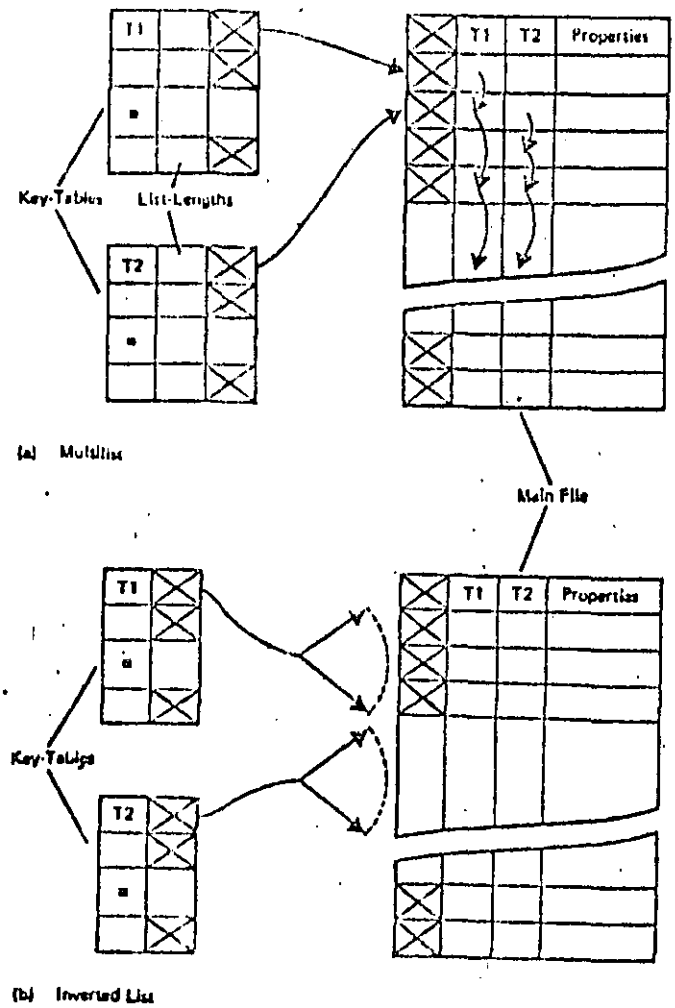


Figure 17—Multilist and inverted list

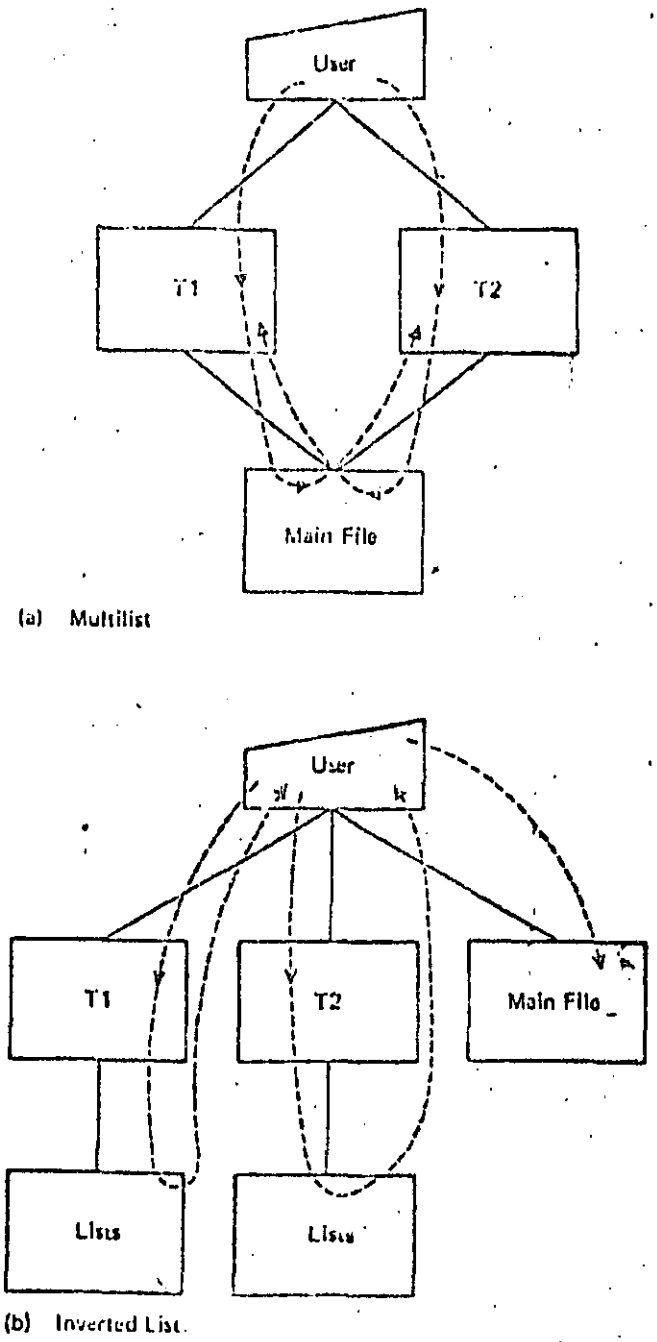


Figure 18—Multilist and inverted list in a data structure diagram

tables and a main file. A key-table consists of one record for each key value; the property parts of the records contain a list address and a list length. The search method is to find the list addresses in the key-tables and then search through the shortest list. The retrieval length will be the number of accesses to the key tables plus the number of accesses to the main file.

Figure 17(b) illustrates an inverted list. An inverted list consists of key tables and a main file. The result of searching in a key table is a record of variable length which contains all the addresses of the records in the main file having specified key value. The search method is to find the record addresses in the key-tables; then logical functions are computed in memory, and the remaining record addresses are found in the main file. The retrieval length will then be the number of accesses to the key tables plus the number of accesses to the main file.

These list structures consist of separate one-identifier-case problems—searching in key-tables and searching in the main file. Therefore, it is possible to develop a cost function using the same parameters as for the one-identifier case.

Fig 18 describes the multilist and the inverted list in a data structure diagram. The access paths are drawn with dotted lines. From data structures of the kinds shown in Figure 18, it is possible to construct any complex data structure, and any complex data structure can be decomposed into simple list structures. It should then be obvious that a performance analysis of a multi-identifier case may be performed using the same parameters as for the one-identifier case. One should, however, notice that the access paths of the retrieval processes have to be described, and that the retrieval length, in the multi-identifier case, may be written as:

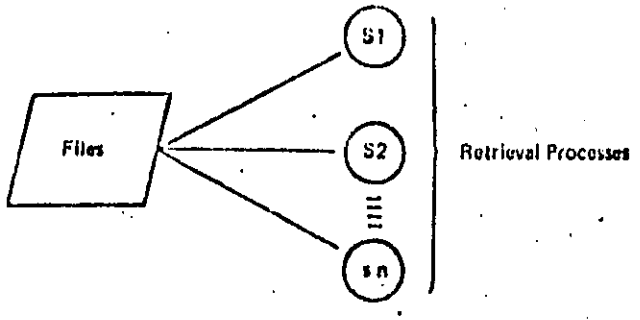
$$a = a_1 + a_2$$

where

a_1 = the number of block accesses caused by the retrieval keys

a_2 = the number of block accesses caused by the establishing relationships

Figure 19 gives a summary of the necessary parameters.



(i) Retrieval Process Parameters:
- Frequencies
- Retrieval Lengths
- Use of CORE
- Use of CPU

(ii) File Parameters:
- Number of Records
- Record Volume
- Block Size
- Load Per cent
- Volume of Administration Data

(iii) Parameters Which Describe a Particular Accounting Routine

Figure 19—Necessary parameters to carry out a performance analysis

CONCLUSION

In the previous sections a multi-level design procedure for design of file organizations has been developed. The design procedure was based on an information analysis of the application system. The information flow was described using the principle of hierarchical systems partitioning, and necessary information was extracted from the information analysis:

- description of objects and relationships between objects,
- description of permanent files, i.e., message types,
- description of retrieval processes.

A logical model was now developed in three steps:

- the message types of the permanent files were normalized,
- a synthesis of the normalized message types into a logical model was made,
- the model was modified in order to satisfy the retrieval requirements.

So far no decisions about any DBMS have been made. The logical model was now modified to fit a particular DBMS in a "best possible" way, i.e., modifying the record layouts and altering the access paths. The record layout was modified making a selection between duplicate storing of data or the use of a reference, and the access paths were altered by making selections between lists and inverted lists.

Therefore, a physical realization in four steps was done, i.e.:

- choice of storage structure,
- physical location of the records,
- implementation of the relationships between the records,
- determination of the file dimension.

The file dimensioning was done in connection with a performance analysis.

The design procedure outlined in this paper still needs a detailed description of the various levels, and further work has to be done in this area.

An experience during the work already done was the strong need for computer based tools for documenting information structures etc., and for analysis of DBMS-schema performance properties. The basis for such a computer-based tool is an automatic documentation of the information analysis. Such documentation systems exist today, for example, CASCADE, which with a few minor adjustments would be well fitted for this purpose. A transformation of the application system into a logical data base model will be the next step. It seems that the method outlined in this paper may be a useful basis for such a transformation algorithm.

Concerning the modification of the logical model, an analysis tool has to be developed. With the information documented through the analysis of the application system it seems possible to develop an analysis tool based upon activity between the records.

Finally, the physical realization using a particular DBMS has to be done. It seems that this may be done in connection with a computerized performance analysis.

Considering the obvious possibilities of automating the design procedure, there seems to be good hope of achieving good results in this area in the future.

REFERENCES

1. Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial," 1971 ACM Sigfidet Workshop, *Data Description, Access and Control*, edited by E. F. Codd and A. L. Dean.
2. Langefors, B., *Theoretical Analysis of Information Systems*, Third Edition, Studentlitteratur, Lund, 1971.
3. Bratbergsengen, Hofstad, Wibe: "Filsystemer og databaser."

DISEÑO DE BASES DE DATOS

A R T I C U L O

THE TEMPORAL DIMENSION IN INFORMATION MODELING

EXPOSITOR:
Ing. Daniel Ríos Zertuche

MAYO, 1985

...Models in Data Base Management Systems, G.M. Nijssen,(ed.)...
...Publishing Company, 1977

22

THE TEMPORAL DIMENSION

IN

INFORMATION MODELING

Janis A. Bubenko jr*

Mathematical Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights,N.Y.,10598

ABSTRACT

The problem of dealing with time-varying associations or relationships in conceptual information modeling is examined. A conceptual framework where time is treated in an unrestricted fashion is introduced. The binary and n-ary relational modeling approches are then discussed from this point of view. Also the paper comments on some approaches which include time as a basic concept in their frameworks. It is concluded that, when designing a conceptual schema, a time-unrestricted design level should precede the specification of a finite, time-restricted conceptual schema.

1 INTRODUCTION

According to the ANSI/X3/SPARC study group (AX3-75A) the 'conceptual schema' describes a limited (restricted) conceptual model of (parts of) the real-world (RW). This model is maintained for all applications. While several models for the conceptual schema have been suggested, very few have paid attention to the problem *how* to systematically design such a schema. In this paper we will not directly and explicitly address the process of designing and defining a conceptual schema. Instead, we will discuss modeling at the conceptual level and pay particular attention to the problem of dealing with *time-varying associations*. This is a highly relevant topic in modeling all RW systems as no system can be considered as static. By *temporal dimension* we denote the

* On leave from the Royal Institute of Technology/University of Stockholm,Sweden.

92

Conclusion: A *conclusion* is an assertion which can be proven (deduced) true on the basis of information about existing events and defined deduction rules. Consequently, every conclusion is associated with at least one time point or interval - the time when the conclusion was drawn.

Time: Relationships and associations in the real world do frequently change . So do the set of participating relevant entities. It is necessary that these properties are given explicit consideration in the abstract model. The concept of *time* is therefore fundamental in the realm of conceptual models. This has been earlier recognised by some researchers in the area of information modeling (see for example (LAN-66A), (SUN-73A), (BEN-76A) and (FAL-75A)) and also by a few authors in the field of artificial intelligence (for example (KAN-75A)). In our abstract modeling framework time plays two kinds of roles: *extrinsic* and *intrinsic* (cf also (YOU-58A)). The extrinsic time is the time *when* a particular assertion is made or conclusion is drawn. Whether this time relationship is explicitly recognized or not *it always exists*. Intrinsic time plays a role as part of the definition of an association type, i.e. it constitutes parts of its 'meaning'. An association type may or may not contain intrinsic time components. Consider the following examples: we draw a conclusion at time t_1 : THE QUANTITY ON HAND OF ARTICLE XYZ IS 41. Here t_1 is extrinsic. Next, consider the assertion at t_2 about the above conclusion, $t_2 > t_1$: AT t_1 THE QUANTITY ON HAND OF ARTICLE XYZ IS 41. Observe that the latter is true for all extrinsic $t_2 > t_1$ and that t_1 has now 'moved to an intrinsic role'. Consequently, t_2 will also move to an intrinsic role if we assert at $t_3 > t_2$ the assertion about the first conclusion etc. In light of this we see that information about events can be seen as assertions abot associations (describing the event), where the event occurrence or observation time has moved to an intrinsic role. In our abstract model of some application we will normally have both associations assertions about which do or do not depend on the extrinsic assertion time. Those which do not depend on time represent 'stable facts' or observations in our model where an intrinsic time relation implicitly or explicitly always is present.

An abstract model is designed by integrating known and anticipated information requirements from system owners and 'local' users (BUB-76A). The design process implies abstraction of the object system and classification of entities, events and associations into classes and types. Description of the AM also includes statement of axioms concerning dependencies ,consistency and completeness of the above mentioned components of the model. It is desirable that the description of the abstract model is non-procedural.

The 'state' of an abstract model (or abstract state (BIL-76A)) at time t is by many authors considered to consist of those entities which at t 'exist' in the AM and those associations which at t are *true*. The IM:s and the DM:s are then designed to maintain a correct current abstract state. This 'storage and current time-slice oriented' view leads, already at the abstract level, to insertion/deletion /updating problems and restricts the model's ability to evolve as new information requirements in the object system arise. Our general notion of an abstract model can be informally described as

> *An abstract model is a non-decreasing set of assertions about OS-events, a set of deduction rules and the set of conclusions that can be drawn about entity existence and associations at all system relevant times (historic, current or future).*

time aspect of conceptual information models of RW systems and we will discuss how time is treated, or *can* be treated, in various conceptual modeling approaches.

Dealing with dynamic systems is a common problem in engineering and science. The behavior of systems is conceptually modeled by sets of equations where time plays a fundamental role as an independant variable. Numerical treatment of these equations requires, however, that we transform the model to a discrete one which only considers a finite set of points on the time axis. We thus study the *state of the system at various points in time* and we may, for computational reasons, keep a 'history' of a set of model states at different time points in the limited computational storage.

Modeling of administrative systems has similarities with mathematical modeling of physical systems. The differences are that in administrative systems we have to deal with large and strongly varying sets of entities and that several aspects of the behavior of these systems are difficult to approximate by traditional mathematical equations. However, in both cases we wish to maintain a conceptual model of some system of interest. While mathematicians have approached the modeling problem from the existing mathematical framework, 'information scientists' have approached conceptual modeling of organisations and administrative systems from a 'computational framework' which initially was 'inspired' and restricted by existing computing and storage machines and concepts (for example sequential tape processing machines). Even today our computational resources are not unlimited. This fact has clearly influenced the development of conceptual modeling concepts for data base applications. The ANSI/X3/SPARC report (AX3-75A), for instance, deals in the *conceptual realm* with concepts such as *conceptual fields, conceptual groups, conceptual records and plexes* (not necessarily materialized). This should indicate the study group's view of a conceptual model as a finite and discrete data machine. While there are a few exceptions, most other conceptual modeling approaches have adopted this view.

In this paper we will therefore study how the temporal dimension can be incorporated at the conceptual level and survey how some earlier approaches have dealt with this problem. At the end we will discuss some consequences of *not* considering the temporal dimension at the conceptual modeling level. In our discussions we will,for illustration, use an extremely simple example which can be seen as a part of an inventory management application case. In spite of its simplicity, the case is general enough to illuminate problems of dealing with time at the conceptual level. First,however, we will outline a framework which includes the time dimension and which also facilitates a comparison of various modeling approaches.

2 A BASIC CONCEPTUAL FRAMEWORK

In this section a brief overview of our basic framework and its main realms will be given. A more detailed description is in preparation (BUB-76C).

The framework considers the following realms:

The *real-world* realm(RW),which includes the concept of *object system* (OS).

The *conceptual model* realm, which contains notions of the *abstract* model (AM) and the *information* model (IM).

The *datalogical* realm,which includes notions of *data structure* (DM) and *storage structure* modeling (SM).

The framework also includes descriptions of these models and the design processes (mappings) from one realm (level) to the next 'lower' realm. A particular method of information or data modeling is more or less explicitly concerned with all these realms,descriptions and processes. Most methods, however, focus their attention to techniques for definition and representation of information models or data structure models. It is obvious, also,that the use of a particular technique strongly influences the way the mapping/design processes above are carried out, i.e. the way models at different levels are developed.

The suggested framework rests, to a considerable extent, on an integration of notions and concepts for information modeling suggested earlier by the author (see for example (BUB-73C), (BUB-76A)) and others (see reference list). The most important of them is, we believe, the consideration of the *time dimension* in abstract information modelling suggested by Langefors (LAN-66A). The conceptual separation of *abstract* models and *information* models - or information modeling languages - as suggested in (BIL-76A) - has also helped to clarify certain conceptual issues. The time dimension in particular, as will be shown, facilitates a non-procedural and less restricted treatment and description of models at the conceptual abstract and information modeling levels.

The author would like to stress that the suggested framework does not claim to constitute a complete 'tool-box' for information modeling. The main purpose of presenting a crude outline of it in this paper is that it facilitates a discussion of certain modeling issues.

The real-world realm

The *object system* of an information system and its data base is the part of the real-world which is relevant to a particular application problem. Thus, the process of defining the relevant OS is essentially a *system analysis and design* activity which includes

- study of the actual organisation's goals, objectives and policies
- study and/or design of operations, flows(material or information) and decision processes
- study of the organisations interaction with its environment
- determination of information needs and requirements
- study of the information system's 'socio-technical' properties and consequences
- specification of goals,objectives and requirements concerning the information system including its data base (including a statement of design criteria) etc.

Several techniques and methods for carrying out and documenting these activities have been published (see for example (CGR-74A) for an excellent survey and (LAN-74C) for a general framework). It is, however, beyond the scope of this paper to more in detail consider this realm.

For instance, a conclusion drawn at time t_1 may not be possible to draw at time t_2, where $t_2 > t_1$, but the fact that it could be drawn at t_1 may still be of importance to the problem. We will attempt to elaborate this notion in connection with an example in the next section of this paper. It is desirable that the description of an abstract model is non-procedural.

The information model

In the abstract realm we have focused our attention to the *substance* of the model, i.e. which entities, associations and events to consider and their relationships and dependencies. In agreement with (BII.-76A),(CHN-75A) and (SEN-73A) we consider the information model realm as an emphasis on *representation* and user-referencing. In this stage we have to decide how the user should refer to the various components of the model. The only way humans can do this is by the use of *names*. We have here to decide how to refer to individual articles, warehouses, persons, colours etc. We also need user-informative names for concept classes, association types, roles etc. Furthermore, it is often the case that the same entity can be referenced by several different reference expressions composed of external names. We do, in this paper, not take any position for a particular formal language for this purpose but conclude that the reference issues have to be solved at this level. We stress again that, at this level, data-structuring and accessing problems are not considered.

The distinction between the abstract realm and the information realm may not be easy to make in a practical design situation because the only way users can reference components of a model is by the use of names which stand for (are 1:1-related to) 'known' concepts in our discourse. As names also must be considered as entities, the referencing problems will normally also influence our abstract model design. For instance, the decision to refer to a person by his name,birth-date and birth-time *or* by his social security number may imply two different abstract models and different set-ups of concepts. We do, however, maintain the position that the design and definition of an *abstract* model can essentially be done without particular consideration to referencing and naming problems, i.e. without the introduction of sets of external, user-informative names.

Ultimately, defining an information model we have to decide by which types of information objects or functions to represent event and conclusion types of our model. An information object is an entity which is fully based on and describable by external names. Information objects which represent events are denoted *statements*. A statement, once issued, never becomes false as it represents the occurrence of an event. A conclusion, drawn at time t, can be 'materialized' by an information object. It can be seen as a 'theoretical observation' (derivation) of our model and as such considered as an event representable by a statement. As for events, we may distinguish between external and internal statements. The problem of 'conclusion materialization' is a central issue discussing the temporal dimension. We will return to it in section 3.

According to our view, new statements informing about external events in the object system are, conceptually 'inserted' in the model at time points , ...t_{i-1}, t_i , t_{i+1}, ... etc. Let us denote by

$S(t)$ the finite set of external statements in an information model at time t .

$A(t \mid t_0)$ the set of conclusions that can be drawn at t_0 (on the basis of $S(t_0)$ and given inference rules) concerning relevant associations $A_i(t)$ of our model for all relevant times

The conceptual realm

The conceptual realm is seen to include two subrealms,which are outlined below.

The abstract model realm

The organisation's (or physical system's - in an engineering type application) data base will ultimately carry information, represented by data, about an *abstract model* of the organisation (resp. the physical system). The following are our basic notions of this realm:

- *Entity:* an undefined concept, but which has been sufficiently well informally and intuitively described by others (for example (CYL-62A)). Initially, our real-world and object system discourse consists of an unclassified, varying set of objects (concrete or abstract). These correspond to a set E of entities in our abstract model.

- *Property:* an undefined concept, but we consider properties as a subset of E - our discourse of entities.

- *Concept class:* a concept class C_i is a subset of E, such that all entities of C_i have one property in common. This is the *defining property* of that concept class. Concept classes may be non-disjoint.

- *Association:* an association is a well defined relationship between a finite number of entities, where each entity plays a well defined *role* (possibly multiple roles). This relationship can, conceptually, hold for time intervals or only for discrete points in time. Associations with the same meaning, but concerning different sets of entities, constitute an *association type*. It is defined by $A_i(r_1:C_{i1};r_2:C_{i2}; \ldots ;r_n C_{in})$, where A_i denotes the association type, r_j the roles and C_{ik} denotes concept classes. In some simple cases the roles are obvious and can be left out.

- *Event:* an event is an observation or decision in the object system at time t of the *beginning*, *ending* or *occurrence* of some association defined in our model. It is understood that an event is always related to a time point or interval - its occurrence or observation time. Furthermore, we can distinguish between *external* events and *internal* events. Both are parts of our model but external events 'lie on the boundary' of or system and can only be observed while information about internal events can be theoretically (sometimes approximately) derived and possibly also measured or observed in the object system. As we can see, the distinction between them depends on where we have drawn the abstract model's boundary. Events can be triggered by other external events outside the control of the information system or by external events in combination with information about the actual state of the model.

- *Assertion:* At any time t we can *assert* that a particular association holds, i.e. is *true*. Thus, every assertion is related to its *assertion time* t. The expression $A_i(t)(r_1:e_{i1}; \ldots ;r_n:e_{in})$, where $e_{ij} \in C_{ij}$, is either *true* or *false* (or possibly undefined, f ex. if some of its components e_{ij} do not exist at t). Basically, and practically, we are interested mainly in true assertions.

97

If $A(t \mid t_0)$ is assumed not to change in the interval $t_i \leq t_0 < t_{i+1}$, then we have designed a 'stepwise changing' model of our object system. This seems to be the normal case for most administrative type information system models. In this case, the time points t_i represent 'state changes' and $A(t \mid t_i)$ describes the current system state at time points t_i. We will observe, however, that several 'system variables' cannot 'exactly' be modeled in this way, for instance the moving average sales quantity for an inventory item changes from t_i to $(t_i + dt)$ even without having new statements introduced in this interval.

The datalogical realm

This realm is seen to focus on implementation, operational and efficiency problems of an information model. It corresponds to the *string*, *encoding* and *physical device* levels as defined in (SEN-73A). The types of design decisions that have to be made suggests two subrealms : one where one essentially deals with data structuring and one where storage allocation and lay-out problems are in focus.

The data-structure model realm

The conceptual realm is essentially un-concerned with procedural, processing and efficiency issues of the information system or the data base. The information model's conceptual view on information can therefore be - somewhat missleadingly - said to presume a 'machine' with infinite memory and infinite processing speeds.

When designing a data model (often based on and restricted by some available DBMS (Data Base Management System) - a *data - machine* - and its capabilities) these 'practical' issues (finite memory and finite speeds) must be considered. This implies - first of all - that we from a practical point of view will not be able to keep track of all 'historical statements' (for example 'transactions') about events signalling entity associations in the OS. Normally decisions are made to either shorten or aggregate the 'history' or to discard the statements and to maintain only the 'current' (read: as far as the system knows - *the last known) state* (set of conclusions) of the AM. When a statement, signalling an event in the OS, arrives ,this gives rise to - possibly complicated - data insertion, deletion and updating operations in the data model in order to maintain the models 'state' consistent and 'up-to-date'.

Thus, *a data model is conceptually based on the notion of a 'machine' with finite storage and finite operating speeds.*

Major decisions, designing a data model, concern

- How to represent statements and conclusions by data structures?
- How to group (cluster) and link data for efficient access and processing?
- Which statements and conclusions to store explicitly in the finite memory and which to declare as derivable - by a machine with finite speeds- when they are referenced?
- How 'actual' or consistent to maintain the explicitly stored information about statements and conclusions, which conditions to apply for triggering, state updating etc?
- e

Of course, other issues, such as data protection, user convenience and ease-of-use, system availability and maintainability must also be considered. While the data structure part of a data model can be described in non-procedural terms ,the complete description of a data model normally includes procedures for its operation and maintenance.

The storage model

The storage model is concerned with implementation and storage allocation details of a particular data model and has to consider ,in further detail, efficiency and operational problems. A deeper discussion of storage models is beyond the scope of this paper.

3 DISCUSSIONS AROUND AN EXAMPLE

In this section an example will be provided which both partly illustrates our framework at the abstract modeling level and serves as a basis for examining the temporal aspect of information modeling.

Suppose we are concerned with inventory management of *articles* which each are available at different *warehouses*. We are interested in keeping track of available *quantities-on-hand* (qoh) both 'per article' and 'per article per warehouse'. Events in the object system, which affect this information, are vendor *shipments* and *deliveries* to customers. Vendors and customers as such are outside our sphere of interest. They lie outside our system's 'boundary', we assume. Let us introduce the following concept classes (or entity sets):

A article types, or articles for short
W warehouses
Q quantities

which correspond to property concepts 'being an article', 'being a warehouse' and 'being a quantity'. The following association types reflect our information requirements, we assume

AQOH(A,Q)
AWQOH(A,W,Q)

where AQOH denotes 'quantity on hand of article' and AWQOH denotes 'quantity on hand of article in warehouse'. Assertions concerning both associations are, according to our framework, associated with an extrinsic time reference. The roles played by members of the concept classes A,W and Q in these associations are obvious ,They are therefore left out. Members of a concept class will be denoted by small letters e.g. $a \in A$, $w \in W$, $q \in Q$. An assertion AQOH(t)/ '*is true* ' we can conclude or observe that there at time t exist q units in inventory of article a

In this model we also introduce two event types which concern observation of *shipment events* (SHIP) and *delivery events* (DEL) at time points t_i

$o{:}SHIP(t_i)(A,W,Q)$
$o{:}DEL(t_i)(A,W,Q)$

Graphically, our simple conceptual model can be represented as in figure 1. As we will present several model alternatives for this sample problem we denote this by alternative A.

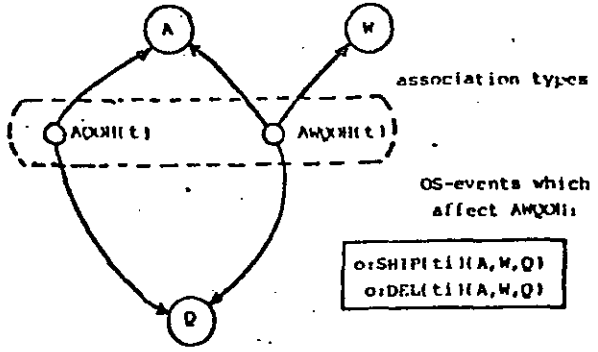


Figure 1
Model alternative A

We observe that AQOH and AWQOH change at time points t_i dictated by the occurrence of OS-events. In this model,information about the SHIP and DEL events directly affect the associations of type AWQOH and indirectly AQOH. The following equation and condition must hold for all times, warehouses and articles in inventory:

$$(\forall t)(\forall a)(\forall w)[\ AWQOH(t)(a,w,.) = [\ \Sigma_{a\le i\le t}\ o{:}SHIP(t_i)(a,w,.) - \Sigma_{a\le i\le t}\ o{:}DEL(t_i)(a,w,.)\ ] \ge 0\ ]$$

Here we use the notation AWQOH(t)(a,w,.) to denote [q : AWQOH(t)(a,w,q)]. This condition puts a restriction on feasible DEL-events : at no time can we deliver more than we have in inventory at a paticular warehouse. It also illustrates the case when an event of type DEL is dependent on the current AWQOH-information status and the occurrence of some other external event, e.g. a 'customer order'. The customer order event and information about quantity on hand will, if the order-quantity can be satisfied, cause a DEL-event.

We also observe that the association AQOH can be derived from AWQOH:

$$(\forall t)(\forall a)[\ AQOH(t)(a,.) = \Sigma_{w/a}\ AWQOH(t)(a,w,.)\ ]$$

where w/a denotes those warehouses which have a particular article a in storage. Thus, according to the conceptual framework outlined in section 2 the quantity on hand can be considered as a single-valued discontinuous function of *time*, *article* and *warehouse*. Also, we may *conceptually* refer to this quantity at *any point in time* (and not only at state changes,for instance).

Using this conceptual framework it is also convenient to define other types of *derivable* associations, for instance

$$CHANGE\text{-}RATE(t)(a,w,.) = (AWQOH(t\text{-}10)(a,w,.) - AWQOH(t)(a,w,.))/10$$

Clearly,other expressions are possible, for example

$$AWQOH(t)(a,.,q)$$

which denotes a set of warehouses, possibly empty, which at time t had q units of article type a on-hand in inventory. We are now viewing warehouses as a multivalued function of t,q and a.

In those modeling methods for the conceptual level, which do not explicitly recognise the temporal dimension, information about the associations of types AWQOH and AQOH is normally treated as a finite set of *stored conclusions* which reflect the *last known* true assertions about these associations. The time dimension is not a fundamental concept of those approaches. Consistency rules require in this case that

$$(\forall a)(\ AQOH(a,.) = \Sigma_{w/a} AWQOH(a,w,.) \ge 0\ )$$

Thus, the 'meaning' of the associations AQOH and AWQOH is now changed to *as far as the system knows ... the current quantity on hand*. Each time an OS-event of type SHIP or DEL occurs, these current quantities must be somehow 'updated' (for example by first adjusting AWQOH and then,for consistency, *deriving or* updating AQOH). This view is illustrated by our model alternative B in figure 2.

If we examine what we have done,by discarding the time dimension, we observe in this case that we have introduced *two new classes of entities* -we will call them *information objects* - which serve as means to store knowledge about the current status of two types of associations. 'Knowledge' is represented by associating each information object with adequate entities in the concept classes A, W and Q. These associations are *binary* and they can be given meaningful

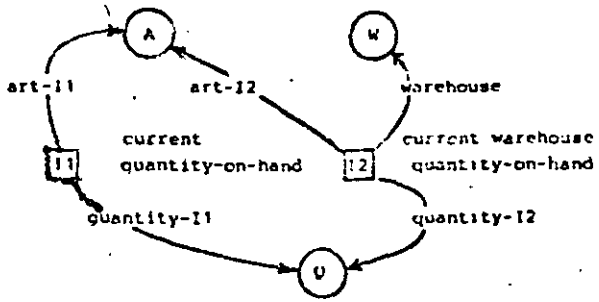


Figure 2
Model alternative B

names which describe their roles. Our 'transactions'- or statements about past SHIP and DEL events - are not part of this model alternative as we have made the design decision that the transactions in themselves are of no particular interest to our problem. Their only purpose is to 'update' the information objects. In figure 2 the information objects classes are represented by square boxes to mark our notion of them as entities. We note that, according to our assumptions, the following correspondences hold for I1,I2 and their associations in model alternative B

I1 → A is (1:1)
I2 → A × W is (1:1)
I1 → Q and I2 → Q are (M:1)

Choosing the model alternative B as adequate for our application problem we have tacitly also made the following asumptions

(a) the quantity on hand functions are constant between SHIP and DEL events

(b) there is no known requirement to be able to respond to queries where *time* is a vital parameter or queries which concern other system states as the current one,for instance:
1 quantity-on-hand two weeks ago
2 the average quantity on hand at (any) time t
3 quantity on hand trends
4 the difference in quantity on hand at times t_1 and t_2

Observe that we would not have any problems in formulating answers to these queries given our conceptual model alternative A. While most information or data modeling approaches would consider the model alternative B as 'conceptual' (or 'abstract' or 'logical') it is obvious that the decisions ma[illegible] and the 'conclusion materializations' introduced makes it unable to respond to queries give[illegible]) above.

It should, however, be obvious that we cannot -for practical reasons - realize (implement) information models with the full temporal generality of alternative A. For instance, we cannot in practice state requirements to be able to obtain responses to *on-line* queries concerning quantities on hand for any historical time point. So- at the interface between the information modeling level and the datalogical level decisions must be made which information about associations to support by representing it conceptually by a finite set of information objects. Note that we are at this interface *not* adressing datalogical design problems as we are *not* concerned with which information objects to *store* and which to *derive* or how to efficiently *access* and represent them in storage. For instance,information objects of type I1 in figure 2 are derivable from an associated set of objects of type I2 and the decision whether to always keep an updated version of I1 in storage or derive it when referenced can be postponed to the datalogical design phase or even to later phases. At this interface level we are rather concerned with which information objects to include in our model and how to deal with the time dimension. This is,in fact,the problem to design a *finite* model, processable by a finite machine, on the basis of an *infinite* model according to alternative A.

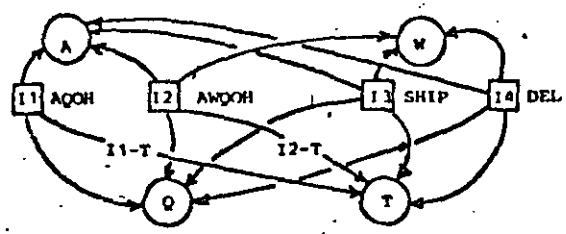


Figure 3
Model alternative C

In several applications, however, the information requirements are such that it is not sufficient to -conceptually- keep and maintain in storage only the current state of the associations. The ways to solve this problem depend on the model designer's insight and knowledge about potential user information requirements. One 'safe', but not always economical, way is to represent by stored information objects *all* statements (transactions) about external OS-events which are transmitted to the model. Time is an obvious parameter in these statements. In this way a *complete history* is maintained and responses can be generated to all conceivable queries which are derivable from the initial statements. Figure 3 (model alternative C) illustrates such a solution,where I3 and I4 are information object classes representing statements about shipment- respectively delivery events. Time is introduced as an infinite set of time points in class T. For simplicity the associations in alternative C need not be named to illustrate our point.

The number of elements in I3 and I4 is clearly *finite* and they form a binary-relational structure. Languages for navigating and retrieval in a *binary* structure,for instance the non-procedural FORAL (SEN-75C), are applicable to I3 and I4. We may also consider the I3 and I4 as 4-tuples with a well defined meaning constituting a subset of A × W × Q × T. This means t[illegible]erations of *relational algebra or calculus* (COD-73B) can be applied to I3 and I4. Now,[illegible]situation

concerning I1 and I2 is slightly more complex. Whether we may consider I1 and I2 as finite sets of information objects depends on our definition of them. If we define I1 and I2 as *quantities-on-hand at time points when a change in quantity on hand for that particular inventory point occurred*, then the number of elements in I1 and I2 is less or equal to the sum of I3 and I4. If we, however, wish to represent by I1 and I2 the quantities on hand at *any time* t ∈ T then, clearly, their number of elements is infinite as T is infinite. Algebraic operations or binary navigations on I1 and I2 can now only be performed on *time restricted* subsets of them. If we by I1 and I2 wish to represent the *current* quantity-on-hand, then time (t='now') is implicit,the sets I1 and I2 are finite and we leave out the associations I1-T and I2-T in figure 3 . Whether we decide to 'keep' subsets of I3 and I4 as part of our model depends on anticipated information requirements.

We have above, in connection with model alternatives B and C, only discussed *some* possible ways to represent the (infinite) alternative A by finite, model alternatives. Another alternative would be, if the requirements are such, to define function procedures for the 'quantities-on-hand' which had the arguments (A,T) and (A,W,T) respectively.

4 DEALING WITH THE TEMPORAL DIMENSION IN THE N-ARY AND BINARY RELATIONAL FRAMEWORKS

It might be interesting to discuss how the modeling problem according to the previous section could be dealt with within the conceptual frameworks of the n-ary relational respectively the binary relational approaches. We will start with the binary approach.

The binary approach

The binary approach is based on two fundamental concepts : the *entity* and the *binary association*. Modeling approaches based on these concepts have been suggested by Abrial (ABL-74A), Bracchi (BRA-76A), Bubenko et al (BUB-74A),(BUB-76A), Lindgreen (LID-74A) and Senko(SEN-75C). A possible 'schema' for the binary relational (or binary logical associative) approach is shown in figure 4 . We observe that AQOH and AWQOH can be considered as 'entities' or 'information objects' only for a finite set of time points. Thus, navigating and selecting entities in the binary model can be carried out if AWQOH and AQOH are somehow restricted for a finite set of time points.

The concept of *derivation of entities on the basis of other entities* has not been explicitly paid attention to in the literature on binary models. In principle, however, this is equivalent to derivation of new relations (i.e. sets of n-tuples) on the basis of other relations in the n-ary relational approach.

Therefore we could define AWQOH and AQOH as sets of entities derivable from (and existence dependent on) the finite set of SHIP and DEL entities. These entities will carry information

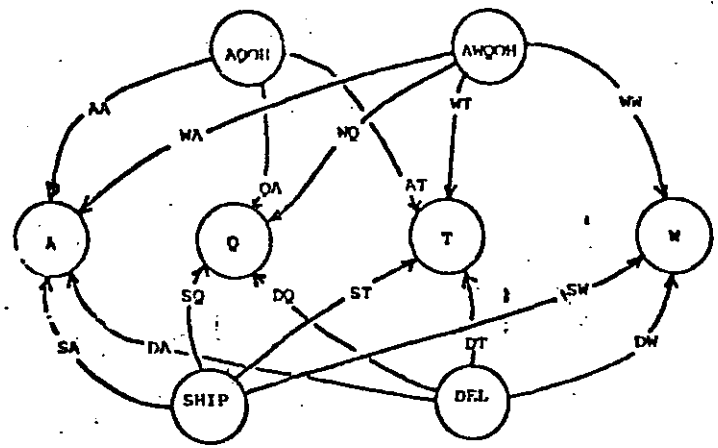


Figure 4
A binary relational schema. AA,WA,QA, ... etc are binary relation names

about quantities on hand at time points t ∈ T. To our knowledge there is no complete and generaly accepted notation developed for expressing derivability,consistency and integrity relationships in binary models. Possibly can some non-procedural, query-oriented languages be extended and generalized in this direction (for example FORAL (SEN-75C)).

Using a calculus-like notation we could formulate the derivation of AWQOH-entities (for all time points) as shown below. We will denote by small letters a member of an entity class (for example awqoh ∈ AWQOH) and use association names as *selectors* (for example WA(awqoh) will select one element a ∈ A and DQ(del) will select a quantity (delivered) , q ∈ Q , if WA resp. DQ are functional).

$$(\forall t)(\forall awqoh)(WT(awqoh)=t \wedge WQ(awqoh) = SUM(SQ(ship:ship\in SHIP(awqoh,t))) - SUM(DQ(del:del\in DEL(awqoh,t))))$$

Here SHIP(awqoh,t) and DEL(awqoh,t) denote time- and association restricted subsets of SHIP and DEL entities as follows

$$SHIP(awqoh,t) = [ship:SA(ship)=WA(awqoh) \wedge SW(ship)=WW(awqoh) \wedge ST(ship) \leq t]$$

$$DEL(awqoh,t) = [del:DA(del)=WA(awqoh) \wedge DW(del)=WW(awqoh) \wedge DT(del) \leq t]$$

The function SUM has here an obvious interpretation. Clearly, SHIP(awqoh,t) and DEL(awqoh,t) denote subsets of SHIP resp DEL which are associated with the same warehouse and article as the 'awqoh'-entity and which occured on or before time t. In the above example we have not paid attention to existential dependencies and other consistency/integrity constraints.

It is also possible to formulate the same derivation rule using a FORAL-inspired notation (SEN-75C).

```
DEFINE Q OF AWQOH
        (WHERE A = P1 AND W = P2 AND T = P3 )
= SUM Q OF SHIP
        (WHERE A = P1 AND W = P2 AND T ≤ P3 )
- SUM Q OF DEL
        (WHERE A = P1 AND W = P2 AND T ≤ P3)
```

In this definition P1 ε A , P2 ε W and P3 ε T are parameters. Using this derivation rule, a set of AWQOH-entities can now be created which are consistent with an existing set of SHIP and DEL entities.

The n-ary relational approach

A formalism for this approach was presented in 1970 by Codd (COD-70A) and it has subsequently been used by several others (see for example (ADI-76A),(BEN-76A),(CHN-75A),(HAL-76A)). Our previous discussion immediately suggests the following data model (relational schema)

```
SHIP(A,W,QSHIP,TS)
DEL(A,W,QDEL,TD)
AQOH(A,QOH,T)
AWQOH(A,W,QOH,T)
```

In this simple case, when all binary associations are functional, transformation to a n-ary model needs no normalisation and is straight forward. SHIP and DEL are clearly relations in the 'traditional' relational data base approach sense. SHIP and DEL define a finite set of 4-tuples, where TS and TD are domains of shipment resp. delivery time points. Relational algebra or calculus can be applied to these relations. The relations AQOH and AWQOH are not relations in the finite, traditional sense if we define their interpretation as 'quantity on hand at time T' where T is a variable. In this case they may be considered as an infinite number of tuples (considering all possible values on the time axis). The problem how to deal with this 'infinite' situation has not been addressed in the n-ary relational data base literature.

The concept of *deriving* one relation from a set of other relations is, however, well known and either relational algebra or calculus are candidate tools for this purpose (sometimes so called 'library' functions may be needed to augment their expressive power). It is also well known that a data model- or data submodel - may contain definitions of relations, which are derivations from (possibly derivations from,...etc) other relations (see for example (DAT-75A), p.115). This is equivalent to the concept of *user views* in n-ary relational systems under implementation (for example SYSTEM R (AST-76A)). Thus,we might consider AQOH and AWQOH as *views* with the understanding that they are materialized and treatable as finite sets of n-tuples only for a finite set of points in time.

Suggesting the concept of a 'time-view', we could define AWQOH in our data model definition as (using a mixture of SEQUEL (AST-76A) and DSL-ALPHA (see (DAT-75A)) -like syntax):

```
DEFINE TIME-VIEW AWQOH (A,W,Q,T)
RANGE SHIP S
RANGE DEL D
(∀T)(∃S)(∃D) ( AWQOH.A = S.A ∧ AWQOH.W = S.W
        ∧ AWQOH.A = D.A ∧ AWQOH.W = D.W
        ∧ S.TS ≤ AWQOH.T ∧ D.TD ≤ AWQOH.T
        ∧ AWQOH.QOH = TOTAL(S.QSHIP) - TOTAL(D.QDEL) )
```

TOTAL(,) is here an assumed library function. In order to have a view which can be operated upon by relational algebra or calculus the user now could define *time-restricted views* on top of this view, for example defining the current (T=NOW) quantity on hand as a *restriction* of AWQOH :

```
DEFINE VIEW CURRENT-AWQOH (A,W,Q)
SELECT AWQOH [ T = NOW ]
```

In the above definitions, the user is not concerned with *how* an implemented sytem maintains the defined views and restrictions. To our knowledge,however, there is to day no relational system in existence which supports the above 'viewing' of information.

Considering an experimental n-ary DBMS, SYSTEM R, it is eventually possible that a view concerning the *current* quantities on hand at warehouses (CURRENT-AWQOH) could be defined by the use of SEQUEL-like operations (such as SELECT,COMPUTE,TOTAL) on tuples in the relations SHIP and DEL, as follows

```
DEFINE VIEW CURRENT-AWQOH AS:
SELECT A,W,Q
FROM S IN SHIP AND D IN DEL
COMPUTE Q = Q1 - Q2
     COMPUTE Q1 =
     SELECT TOTAL(QSHIP)
     FROM SHIP
     WHERE A=S.A AND W=S.W;
     COMPUTE Q2 =
     SELECT TOTAL(QDEL)
     FROM DEL
     WHERE A=D.A AND W=D.W;
```

It is understood that this view is somehow automatically maintained - by the DBMS - consistent with existing tuples in the SHIP and DEL relations. In the SYSTEM R, the user could also himself control the maintenance of a relation,which is derivable or dependent on other relations, by the use of the TRIGGER-mechanism. For instance, in order to maintain a consistent set of stored tuples in the relation CURRENT-AWQOH, the user could define two TRIGGERS as follows (we describe them in a SEQUEL like syntax):

```
DEFINE TRIGGER XSHIP
ON INSERTION OF SHIP :
     (UPDATE CURRENT-AWQOH
     SET QOH = QOH + SHIP.QSHIP
     WHERE A = SHIP.A
     AND W = SHIP.W )

DEFINE TRIGGER XDEL
ON INSERTION OF DEL :
     ( UPDATE CURRENT-AWQOH
     SET QOH = QOH - DEL.QDEL
     WHERE A = DEL.A
     AND W = DEL.W )
```

In the above solution a problem arises when an inserted SHIP or DEL tuple has an <a,w> - pair, which does not match any 'existing' key in the relation CURRENT-AWQOH. This must be solved by introducing additional consistency and dependency checking and maintenance procedures. Note also that the designer-user now has full control over the degree of consistency in the data base. If he, for instance, is satisfied with with a 'correct' set of tuples in CURRENT-AWQOH only at 9 PM every day, then a triggering condition could be specified which activated a derivation process shortly before that time. This, however, would change the meaning of the corresponding relations.

Conclusions

Summarizing our discussion on modeling time-varying associations in the n-ary and binary-relational framework we have tried to demonstrate that neither of these approaches were originally intended for an abstract modeling level outlined in section 2. Both approaches view the information model for a real-world case as a *finite*,varying collection of *n*-tuples respectively a finite,varying collection of entities and binary associations. Time can,of course, be incorporated in these models by introducing time-domains in relations respectively time-entities and associations in binary models. We have also demonstrated that the notation and operations defined for models of this kind probably can be extended for use as definitional tools at the abstract, time unrestricted modeling level.

5 TREATMENT OF THE TEMPORAL DIMENSION IN SOME EXISTING INFORMATION MODELING APPROACHES

We have claimed that the majority of modeling approaches pay no explicit attention to the temporal dimension. The information model of a particular application is seen as a finite ,varying set of information objects normally reflecting the current (last observed) state of a model of some real-world system. The state is changed by inserting,deleting and modifying information objects. Sometimes,in these approaches, rules are discussed how to define and maintain a set of information objects consistent and complete. There are,however, a few exceptions to this time-restricted and storage oriented approach. These will now be briefly discussed.

Young and Kent

Young and Kent (YOU-58A) have included the concept of time in abstract formulation of data processing problems. Time is,however, not used for defining information relationships but essentially for statement of time (delay) requirements concerning data processing and document production operations and for specification of document dates as functions of the times when the document is produced.

Langefors and Sundgren

The first approach which paid explicit attention to the problem of time in information models was presented in about 1965 by Langefors (LAN-66A). This approach has later been followed up by Sundgren (SUN-73A). Additional discussions can be found in (LAN-74B),(LAN-74C) and (SUN-74A).

The basic building block in this, so called *infological* approach, is the *elementary message* (*e-message*), a 3-tuple <o,a,t> which informs about an *elementary situation* (e-situation) in the object system. In an e-message, 'o' is a reference to a non-empty set of objects (entities), 'a' is a reference to a relationship type *or* to a property/value pair,and 't' is a reference to a point in time or a time period when this relationship or property holds or is measured (observed). Messages which inform about objects of the same classes and state properties respectively relationships of

the same 'kind' are said to carry information of the same kind. They are said to conform to the same *elementary concept* (e-concept). There exist relationships between e-messages of a particular infological model in the sense that some e-messages can be produced from other e-messages, given a set of production rules. These relationships are called *precedence relationships*. Basic subsystems of an infological data base are considered the *schema*, the *nucleus* and the *filter*. The schema defines the particular object system model by enumerating all e-message types (e-concepts) and it also contains a set of message derivation rules. The nucleus is the set of explicitly stored e-messages and the filter function serves to protect the data base from false, meaningless or inconsistent messages. The nucleus may be *redundant* in the sense that some of its messages may be defined as derivable from other nucleus messages.

While time constitutes a fundamental part of the e-message, the references cited above (LAN-... and SUN-...) lack a complete notation and clear examples showing the author's basic intentions how to treat and reference the temporal aspect of information modeling. Our interpretation of their intentions follows.

The proposition *any message in an information system will be obtained either by observation in the object system or through the execution of a process which then takes other e-messages as input* (LAN-74C), indicates that their infological model is basically seen as a finite collection of e-messages. Also Sundgren's discussion of 'target sets' (p. 95 in (SUN-73A)) and associated examples reflect an, essentially, finite view of infological models. The contents of an 'infological data base', M(t), is at time t a set of known e-messages (SUN-73A). Some messages are explicitly stored in the nucleus while others are 'virtual' and derived when desired. Insertion and deletion of e-messages may only concern the nucleus. Each message in the infological data base is conceptually seen as having several *time-versions*. A new time version of an e-message is created when an e-situation in the object system occurs which concerns the particular <o,a> - pair. As some messages are derivable, it follows that an e-situation may, conceptually, introduce new time versions for several messages. Thus, an infological data base contains a finite set of e-messages, where each e-message may have several time versions. Each version signals a change in the property or relationship of a set of objects <o>. Conceptually the set of e-messages,including their time versions, is 'ever growing'. According to (SUN-73A), the number of time versions per e-message which the user considers to be of interest may be an important design parameter for the datalogical design phase.

Discussion

Our conceptual framework is to some extent inspired by Langefor's approach of dealing with time. Therefore similarities can be found between both conceptual views. Some differences are discussed below.

The concept of time versions appears quite natural when applied to *attributive* type e-messages, i.e. messages <o,a,t>, where 'a' is an attribute-type/attribute-value reference. We then can think of time versions $<o_1,a_1,t_1>$, $<o_2,a_2,t_2>$, ... etc. which constitute full or partial history of an object o_1 with respect to attribute (property) 'a'. An example of this situation is the e-concept <<article>,quantity-on-hand>, where we may think of e-message versions $<a,q_1,t_1>$, $<a,q_2,t_2>$, ... etc. where $t_2 > t_1$. Clearly, this set of versions describes the 'q-variation' for a particular article 'a'.

The situation becomes less clear when we talk about *relational* type e-messages, i.e. messages where 'a' denotes a relationship type. In this case, what changes - from time to time - is the set of objects referenced by 'o' and it is conceptually possible to think of the 'time versions' as being associated with every proper subset of objects of 'o'. The concept of 'saving the n latest time versions of a particular e-message type '(SUN-73A), p.159) does not seem particularily clear in this case, unless one also specifies for which subset of objects in 'o' this 'saving' has to be done. An example of this situation is the e-concept <<article,project,supplier>,SUPPLIES>, where SUPPLIES denotes a relation with well known interpretation from the n-ary relational world. If we have an e-message

$$<<a1,p1,s1>,SUPPLIES,t_1>$$

and *subsume* for it a new message at $t_2 > t_1$

$$<<a1,p2,s1>,SUPPLIES,t_2>$$

then this new message can be seen as a new relationship from the view-point of participating object subsets <a1>,<s1> or <a1,s1>. At the same time the set of SUPPLY-relationships where s1 participates is changed. Another consequence of this time-version approach is that in query references like $<o,<A_n,T>,t>$, if t is not the time of a historical time version of this message, then an adequate time version with a time reference 'nearest' to t must be located. Thus, a distinction is introduced between the contents of a data base and retrieved information.

Falkenberg

Falkenberg's *information sphere* (FAL-75A) consists of a set of associations, which each exist for limited time periods. An association is a semantically relevant set of objects $<g_1,g_2, \ldots g_n>$, where each object plays a specific *role* in the framework of the association. An association has a *beginning* and an *end* which are considered as *events* on the time axis. Specific representations (without regard to physical representations) of associations are denoted *significations*. *Building up and maintaining a data system means the subsequent input of all significations of all events happening in the information sphere.* In (FAL-75A) then a set of *language operators* are defined by which answers to queries about time-dependent relationships or associations can be obtained. The signification concept is also applied to query formulation.

In a later paper (FAL-76A) the time dimension is not given special attention. This paper,however, discusses semantic rules of 'data systems' *and* mentions concepts such as association dependencies (of the consistency type) and deduction of new associations.

Falkenberg's conceptual view on the temporal dimension is on several issues similar to our conceptual framework examplified earlier. The main difference seems to be that Falkenberg considers the conceptual information model of some application as a finite, stored and varying collection of non-redundant 'facts' about events (associations). Consequences or conclusions

from this set of facts seem not constitute a part of the model but are seen as derived responses to queries formulated with a relatively large set of operators, some of which are 'time-relational'.

Benci et al.

In their paper (BEN-76A) Benci et al. discuss *concepts for the design of conceptual schemas*. Their assumption is that the real-world can be completely described with *objects* (entities), *properties* and *associations* between these objects. The n-ary relational formalism is considered adequate to describe a conceptual data base. The existence of an object or an association is represented in the conceptual structure as (the existence of) a n-tuple of a relation. A conceptual data base is seen as a set of indexed relations $B^t = \{ R_1^t, R_2^t, \ldots, R_n^t \}$. A conceptual data base B^{t+1} is obtained by applying to B^t operations, which conform to defined s.c. *evolution rules*.

The representation of time (time points or periods) in the conceptual structure is simply done by introducing, when found necessary, data types, e.g. one or more time domains, in the relations representing a set of objects or a set of associations. Of course, the exact meaning of these time-domains in the actual context has to be defined. The authors conclude that the introduction of time in a DDL raises no specific problems.

Discussing consistency in an evolving data base, the authors observe that the consistency of a data base does not have to be *absolute* at all times. This means that, in a practical DB-implementation case, we may permit that certain associations are not reflecting the current correct values according to initial statements about OS-events. In order to save computing power, these associations are periodically or conditionally "updated" and not "immediately" following some event which affects them.

Thus, in the approach of Benci et al., time is incorporated in the conceptual model when found necessary (or natural) as data items (domains) in relations and a data base, conceptually seen as a finite collection of n-tuples of assorted degrees and interpretations, is maintained "partly" consistent (at time points) by updating operations triggered periodically or conditionally. From an information *modelling point* of view, this approach to definition of a conceptual schema brings it closer to the datalogical design level than to the abstract, time-unrestricted view outlined above.

6 CONCLUDING REMARKS

Suggested information modeling methods in the literature can, with respect to handling of the temporal dimension be grouped in two categories

1 Methods which more or less ignore the time aspect. A particular information model is here seen as a finite collection of information objects (statements,'facts'), which represent the current state, i.e. the set of true assertions made the last time information about some object-system event was inserted. Most modeling approaches, including the binary and n-ary relational, follow this underlying conceptual view. Of course, time can be, in a restricted sense, included in models following these approaches.This is, however, done in an ad-hoc (arbitrary) way by the model-designer by introducing intrinsic time domains in relations or by introducing entity-time associations in binary models.

2 Methods which include the time dimension as a fundamental concept of their framework. Langefor's and Sundgren's approches fall into this category where an information model at the conceptual level is seen as an 'ever-growing' set of e-messages (may be considered as information objects) with different time versions. Also Falkenberg's approach, considering association beginning and ending events, belongs to this category. A problem with these approaches is, however, that none of them have elaborated and examplified a complete notation for dealing with the definition and referencing of time varying associations and their dependencies or relationships.

Another issue, which concerns time-unconstrained (infinite) information models is the design problem to *restrict* them to a finite and state oriented model. It is clearly not possible to implement a model with the full temporal generality and therefore decisions must be made which states to maintain or which 'time versions' of information objects to support. This discussion suggests that we should consider two *sublevels* of the abstract (conceptual) information modeling level: one where we consider the time dimension in its full generality - leading to an 'infinite' model, and one where restrictions are decided to make the model finite and transformable to an implementable datalogical model. A method or 'strategy' how to transform an infinite model to a finite one has, however, not been addressed in the literature.

Studying published works on information modeling clearly shows that the temporal dimension has "bothered" several researchers. However, after stating the importance of the temporal dimension, this issue is somehow dropped and the 'finite sublevel' of conceptual modeling is directly addressed. Summarizing our discussions, we believe that this 'short-cut' may have the following consequences.

1 A 'storage-oriented' view is enforced : instead of defining functions and time varying relationships the view is focused to storage and maintenance of function and relationship 'values'.

2 Decisions *how* to introduce time in the model, *which* time versions of information objects to maintain etc are made more arbitrarily than if a time-unrestricted model was used as a basis.

3 If time restrictions are introduced while creating a conceptual model then this makes it less convenient to formulate and integrate end user information requirements (designing a shared view based on different external shemata). Time-unrestricted ("infinite") models are, from a *problem* - oriented point of view, 'cleaner' and not concerned with processing (storage,updating,deletion) and efficiency issues. The end user can view the object system in a *true time perspective* and express information requirements and relationships referring to any time point and not referring to a finite,stored collection of messages,n-tuples or entities.

What we have pointed out above in no way eliminates the need and importance of 'finite' information models. It is only finite models which can be mapped to datalogical models and with we can manipulate with relational algebra or navegational operators. Also, processing and efficiency issues can only be discussed starting out from the finite modeling level.

While methods based on the binary or n-ary relational view are significant achievements (in the sense of *problem orientation*) in information modeling over the stored data structure oriented views, represented by the hierarchical or network data structure manipulating machines (for example (CYL-71A)), they were not aimed for the time-unrestricted abstract modeling level. These approaches and the 'theories' around them are based on a finite view and the time - dimension problem is not always solved simply by adding time domains or entities. A new conceptual view and framework is needed, which,however,superficially may use some of the notational formalism introduced by these approaches. It is believed that the framework outlined in section 2 offers considerable generality in dealing with time which should make it a candidate basis for high level information modeling method development. Finally, it must be stressed that we do in no way suggest that the time-unrestricted level is suitable level for the 'conceptual schema' as defined by (AX3-75A). We consider this level more as a 'design phase' and a basis which should logically precede the design of a finite, conceptual schema, which for processing and efficiency reasons must be 'strongly time-restricted' - at least for some years to come.

ACKNOWLEDGEMENT

The author wishes to thank Dr. Michael E. Senko at IBM Research for valuable criticism and suggestions.

REFERENCES

ABL-74A Abrial J.R.: *Data Semantics*, in (KIM-74A), pp. 1-60.

ACM-76A ACM Sigplan/Sigmod: *Proceedings of the Conference on Data...: ABSTRACTION, DEFINITION AND STRUCTURE*, Salt Lake City, Utah,March 22-24, 1976.

ADI-76A Adiba M., Leonard M. and Delobel C.: *A Unified Approach for Modeling Data in Logical Data Base Design*, in (IFP-76A), p. 634ff..

AST-76A Astrahan M.M. et al.: *SYSTEM R : a relational approach to data base management*, IBM Research, Report RJ 1738, feb ,1976.

AX3-75A ANSI/X3/SPARC (Standards Planning and RequirementsCommittee): *INTERIM REPORT fromthe Study Group on Data Base Management Systems*, FDT Bulletin of ACM, Vol 7,No 2, 1975.

BEN-76A Bencl E., Bodart P., Bogaert H. and Cabanes A.: *Concepts for the design of a conceptual schema*, in (IFP-76A), p. 379 ff.

BIL-76A Biller H. and Neuhold E. J.: *On the Semantics of Data Bases : The Semantics of Data Models*, Report (Institut fur Informatik, Universitat Stuttgart, D-7000, Stuttgart,Herdweg 51),1976.

BRA-76A Bracchi G. Paolini P. and Pelagatti G.: *Binary Logical Associations in Data Modeling*, in (IFP-76A), p. 255 ff..

BUB-74A Bubenko Jr J. A. and Berild S: *CADIS System 4: A tool for incremental dewription and analysis of systems*, Research Group CADIS,Univ. of Stockholm,Dept of Information Processing. Report TRITA-IBADB -3082,1974 (also presented at the BIFOA/GMD International Symposium on *Organisation Structure and the Structure of Information Systems*, GMD,St. Augustin, Fed. Republic of Germany, June 26-28, 1974)

BUB-76A Bubenko Jr. J. A., Berild S., Lindencrona-Ohlin E. and Nachmens S., *From Information Requirements to DBTG- Data Structures*, ACM Sigplan/Sigmod Conference ,March 22-24,1976,Salt Lake City, Utah,USA.

BUB-76C Bubenko Jr J.A.: *Information modeling methods: a survey and an approach toward a conceptual basis*, in preparation,to appear (1976).

CHN-75A Chen Peter Pin-Shan: *The Entity-Relationship Model -- Toward a Unified View of Data*. Report, Center for Information System Research,Sloan School of Management, M.I.T., Cambridge, Mass., 02139 (also presented in (KER-75A)).

CYL-62A CODASYL Development Committee: *An Information Algebra* , Communications of the ACM ,1962, pp. 190-204.

CYL-71A CODASYL Systems Committee : *Report on the CODASYL Data Base Task Group*, ACM ,April, 1971.

COD-70A Codd E.F.: *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM, Vol 13,No 6 ,June 1970, p. 377 ff.

COD-73B Codd E. F.: *Relational Completeness of Data Base Sublanguages*, in (RST-73A), P. 65 ff.

CGR-73A Couger J.D.: *Evolution of Business System Analysis Techniques*, ACM Computing Surveys, 1973.

CGR-74A Couger J.D. and Knapp R.W.: *System Analysis Techniques*, John Wiley/Sons, N.Y.,1974.

DAT-75A Date C.J.: *An introduction to database systems*, Addison Wesley, Reading, Mass., 1975.

FAL-75A Falkenberg E.: *Design and application of a natural language oriented data base language*, Adv. Course on Data Base Languages and Natural Language Processing, Freudenstadt, Black Forest, Fed Republic of Germany, Aug. 1975.

FAL-76A Falkenberg E.: *A Uniform Approach to Data Base Management*, in (IFP-76A), p. 196 ff..

HAL-76A Hall P., Owlett J. and Todd S.: *Relations and Entities*, in (IFP-76A), p. 430 ff..

IFP-75A IFIP TC 2 SPECIAL WORKING CONFERENCE: *A technical in-depth evaluation of the DDL*, Namur, Belgium, PREPRINTS, Jan. 13-17,1975.

IFP-76A IFIP-TC-2 Working Conference on *MODELING IN DATA BASE MANAGEMENT SYSTEMS* January 5-9, 1976,Freudenstadt (Black Forest),Fed.Republic of Germany (Preprints).

KAN-75A Kahn K.M.: *Mechanization of temporal knowledge*, Project MAC, Report MAC-TR-155, Sept., 1975.

KER-75A Kerr D.S. (ed.): *Proceedings of the International conference on VERY LARGE DATA BASES*, ACM,Framingham,Mass.,U.S.A.,Sept.,1975.

KIM-74A Kimbie J.W. and Koffeman K.L.: *DATA BASE MANAGEMENT (Proceedings of the IFIP Working Conference,Corsica, 1974)*, North Holland/American Elsevier, 1974.

LAN-66A Langefors B.: *Theoretical Analysis of Information Systems*, Fourth ed., Studentlitteratur/Auerbach, Lund, Sweden,1973.

LAN-74C Langefors B.: *Theoretical Aspects of Information Systems fo Management*, IFIP Congress 1974, pp. 937-945.

LAN-74B Langefors B.: *On Information Structure and Data Structure*, Univ of Stockholm,Dept of Adm Inf Processing,Report TRITA-IBADB-1019, Stockholm, 1974 (also in preprints of (IFP-75A))

LID-74A Lindgreen P.: *Basic operations on information as a basis for data base design*, IFIP 1974, pp. 993-997 (1974).

RST-73A Rustin R. (ed.): *DATA BASE SYSTEMS*, Prentice Hall,1973.

SEN-73A Senko M.E., Altman E.B., Astrahan M.M. and Fehder P.L.: *Data structures and accessingin data base systems*, IBM Systems Journal, No. 1, 1973, pp. 30-93.

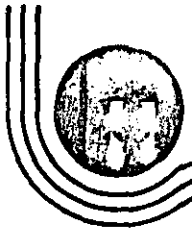
SEN-75C Senko M. E.: *The DDL in the Context of a Multilevel Structured Description : DIAM II with FORAL*, in (IFP-75A), pp.269 - 295.

SEN-76B Senko M. E.: *DIAM II: The Binary Infological Level and its Database Language - FORAL*, ACM Sigplan/Sigmod Conference, Salt Lake City, Utah, March 22-24, 1976.

SUN-73A Sundgren B.: *An Infological Approch to Data Bases*, Urval 7, SCB(Statistiska Centralbyran), Stockholm, 1973.

SUN-74A Sundgren B.: *Conceptual Foundations of the Infological Approach to Data Bases*, in (KIM-74A), pp. 61-94.

YOU-58A Young J.W. and Kent H.K.: *Abstract Formulation of Data Processing Problems*, in (CGR-74A), pp. 259- 274.



DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

DISEÑO DE BASES DE DATOS

FURTHER ANALYSIS OF THE ENTITY-RELATIONSHIP APPROACH TO DATABASE DESIGN

EXPOSITOR:
Ing. Daniel Ríos Zertuche

MAYO, 1985

Further Analysis of the Entity-Relationship Approach to Database Design

PETER A. NG

Abstract—The nondeterministic or deterministic entity-relationship model of a database is formally defined as a user's view of that database in terms of a collection of time-varying relations: the regular or weak entity relations, or the regular or weak relationship relations. Both nondeterministic and deterministic entity-relationship models have the same strength to characterize information concerning entities and relationships which exist in our minds. An improved table form of the relations is introduced to provide a clear and concise user's view of databases. The basic concept of the entity-relationship approach to the logical database design is provided, and is used to derive 3*NF* relations. Finally, a method of representing physically these relations, which are generated by the use of the entity-relationship approach to the logical database design, is presented. Thus, the entity-relationship approach to the logical and physical database design can also be realized.

Index Terms—Determinism and nondeterminism, entity-relationship approach, functional dependency, logical and physical database design, normal forms, relational, hierarchical, and network models.

I. Introduction

A DATA model, called the entity-relationship model, has been proposed in [2], [3]. And the recent development of its use for systems analysis and design has been summarized in [15].

In the entity-relationship model, one of the major characteristics is the existence of attributes, each mapped from an entity set or a relationship set into a value set or a Cartesian product of value sets. In this paper, we shall describe formally that a deterministic entity-relationship model is a model, in which each of the attributes is a many-to-one mapping. If an attribute is a many-to-many mapping, then it is a nondeterministic model. The nondeterministic and deterministic entity-relationship models are defined in terms of entity-relationship relations or relations, each described by a regular or weak entity relation, or a regular or weak relationship relation, according to the different levels of logical views of data [2], [14]. It can be shown that both nondeterministic and deterministic models have the same strength to characterize the same information about the real world.

The entity-relationship approach to logical database design, described in this paper, consists of three major phases: 1) defining the enterprise schema using the entity-relationship diagram that conforms to the description of a database application, 2) translating the enterprise schema into an enterprise-user schema, called the entity-relationship relations, such that the conversion of nondeterministic relations into deterministic relations can be done with ease, and 3) transforming the enterprise-user schema (or directly from the enterprise schema) into a user schema. The user schema can be expressed by a collection of regular or weak entity relations, or regular or weak relationship relations. These relations in an improved table form have a clear and concise description of the user views of the database. Indeed, the user schema also can be expressed by the network, hierarchical, or relational diagrams [3]. And the translation process from the entity-relationship diagram to the hierarchical diagram has recently been studied [13] to provide a unified approach to the logical design of the hierarchical model.

In defining the enterprise schema in terms of an entity-relationship diagram, one of the major concerns is that whether the diagram conforms to the description of application for an enterprise [11], [12]. The other concern is that whether the relations, obtained by translating from the entity-relationship diagram, are "equivalently" corresponding to 3*NF* relations of the relational model. Within the realm of these concerns, this paper outlines the major steps in logical database design using the entity-relationship approach.

In the course of constructing the physical representation of an entity-relationship model, a disciplined data-structure diagram [2] is obtained by translating from the entity-relationship diagram. It has been shown that this data-structure diagram can be used to recognize the multivalued dependence [11]. In addition to the entity-relationship diagram, this data-structure diagram and the user schema can be used as major components of an integrated design tool to assist in the process of the physical database design. Thus, the entity-relationship approach to the logical and physical database design can be realized.

In this paper, Section II introduces the nondeterministic and deterministic entity-relationship relations as an enterprise-user's view of data. Both nondeterministic and deterministic relations are shown to have the same strength to characterize information about the real world. Section III describes a user schema in terms of regular or weak entity relations, or regular or weak relationship relations. An improved table form for the relations is introduced. Section IV concludes that both deterministic and nondeterministic relations for the user schema have the same strength to characterize information concerning entities and relationships which exist in our minds. Section V describes an entity-relationship approach to the logical database design. It can be used to construct the rela-

Manuscript received July 31, 1979; revised August 22, 1980.
The author is with the Department of Computer Sciences, University of Missouri, Columbia, MO 65211.

0098-5589/81/0100-0085$00.75

tions as a user's view of data in the entity-relationship model such that their relations are equivalent to the 3NF relations in the relational model. Section VI describes a method of representing physically the entity-relationship diagram or its corresponding user schema.

II. Entity-Relationship Relations

In this section, we shall define formally the nondeterministic and deterministic entity-relationship models of a database as an enterprise-user's view of that database in terms of a collection of time-varying relations of assorted degrees. We shall show that both nondeterministic and deterministic entity-relationship models have the same strength to characterize the information concerning entities and relationships.

Definition 1: Given a collection of sets $A_1, A_2, \cdots, A_n$ (not necessarily distinct), R is a *relation* on these n sets if it is a set of ordered n-tuples $<a_1, a_2, \cdots, a_n>$ such that $a_1 \in A_1$, $a_2 \in A_2, \cdots, a_n \in A_n$. Sets $A_1, A_2, \cdots, A_n$ are the *domains* of R. The value n is the *degree* of R.

Definition 2: Let e denote an entity, which is an object that can be distinctly identified. *An entity set* E is defined as $E = \{e \mid p(e)\}$, where p is the aforementioned test predicate.

Definition 3: Let $E = \{E_i \mid 1 \leq i \leq n\}$ be a collection of entity sets. *A relationship set* R *over* E is defined as $R = \{<e_1, e_2, \cdots, e_n> \mid e_i \in E_i, 1 \leq i \leq n\}$. Each tuple of entities $<e_1, e_2, \cdots, e_n>$ is a *relationship*.

The *role* of an entity in a relationship is the function that it performs in the relationship. The ordering of entities in the definition of relationship can be dropped if the roles of entities in the relationship are explicitly stated as follows: $<r_1/e_1, r_2/e_2, \cdots, r_n/e_n>$, where r_i is the role of e_i in the relationship.

Example 1: Let e denote an entity which exists in our minds. Entities can be classified into different entity types; each entity type contains a set of entities, each satisfying a set of predefined common properties. The set of entities is called an entity set, such as EMPLOYEE, PROJECT, PARENTS, and so forth. Because of different predefined properties, the entity sets are of different types. Thus, we call an entity set, an entity type.

In general, there will be associations or relationships linking the basic entities together. For example, as shown in Fig. 1, a MARRIAGE is a relationship between two entities in the entity set PARENTS. Their roles are HUSBAND and WIFE. We shall call MARRIAGE a relationship set, or a relationship type. PROJ-EMP and PROJ-MANAGER are two different types of relationship sets over two entity sets PROJECT and EMPLOYEE.

Fig. 1 is an extension of the entity-relationship diagrammatic notation in [2]. The entity sets PARENTS and CHILDREN are shown as double rectangular boxes, each of which is connected by a directed edge from the entity set EMPLOYEE and the relationship set E (existence dependency) and EMP-CHILD, respectively. The diagram expresses the existence and identification dependencies of the entity set PARENTS on EMPLOYEE; it indicates that the existence of any entity in the entity set PARENTS depends on the corresponding entity in the entity set EMPLOYEE; that is, if an employee leaves the company, his parents may no longer be of interest. It also indicates that parents are identified by their own names and by the values of the primary key of the employees supporting them.

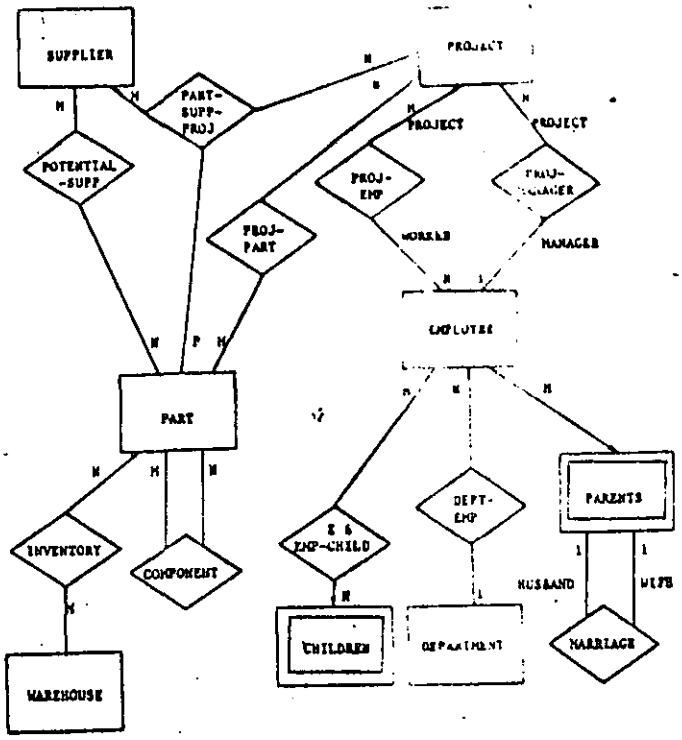


Fig. 1. An entity-relationship diagram for an enterprise schema.

If a relationship set E, represented in a diamond-shaped box, is introduced between the entity sets EMPLOYEE and CHILDREN, then it expresses the existence dependency of the entity set CHILDREN on EMPLOYEE, and the identification dependency of the entity set CHILDREN on EMPLOYEE does not exist. If the same diamond-shaped box has the names E and EMP-CHILD, then the relationship set EMP-CHILD of the entity sets EMPLOYEE and CHILDREN includes the existence dependency of the entity set CHILDREN on EMPLOYEE. If the same diamond-shaped box has the name EMP-CHILD, then the relationship set EMP-CHILD includes the existence and identification dependencies of the entity set CHILDREN on EMPLOYEE. Without this diamond-shaped box, it indicates the existence and identification dependencies of the entity set CHILDREN on EMPLOYEE.

Definition 4: A *value set* V is defined as $V = \{v \mid p(v)\}$, where p is the aforementioned test predicate.

Definition 5: Let $E = \{E_i \mid 1 \leq i \leq n\}$ be a collection of finite entity sets. Let $V = \prod_{j=1}^{m} V_j$, $1 \leq m$, be a Cartesian product of value sets. Let $R \subseteq \prod_{j=1}^{n} E_j$, $1 \leq n$. A *multivalued attribute* F is defined as $F \subseteq R \times V$. If $m = 1$, then V is a value set. If $n = 1$, then R is an entity set, and F is a mapping from an entity set into a value set or a Cartesian product of value sets. If $n > 1$, then R is a relationship set over E, and F is a mapping from a relationship set into a value set or a Cartesian product of value sets. In particular, if F is a function which maps R into V, in notation $F: R \rightarrow V$, then F is called a *single-valued attribute*.

Example 2: Consider the entity sets PROJECT and EMPLOYEE and their relationship set PROJ-EMP, shown in Fig. 2. The attribute STARTING-DATE is the date that an employee starts working for a particular project, and the attribute %-OF-TIME is the percentage of time that an employee is expected to spend on a particular project. The concept of attributes of a relationship is important in understanding the semantics of

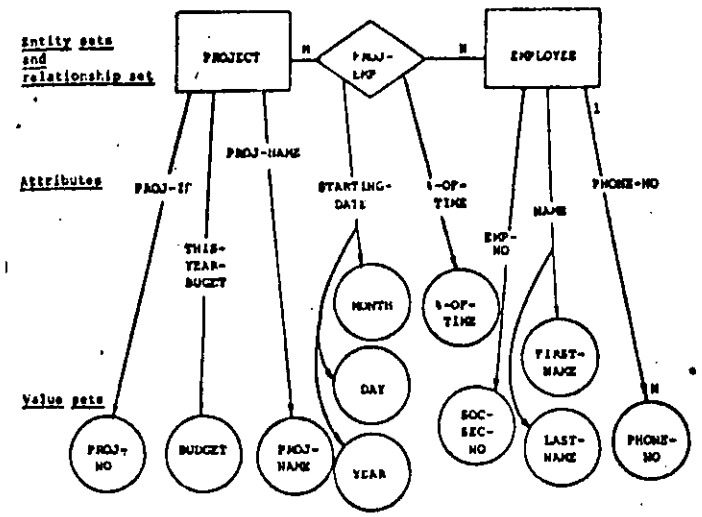


Fig. 2. Value sets of different types and attributes of entity sets and relationships.

data. This concept is similar to the relationship data in network (CODASYL) type database systems, and similar to the intersection data in hierarchical type (IMS type) database systems.

In some cases, an attribute may have more than one value for a given entity. For instance, the PHONE-NO of an employee x may have more than one value. In this case, we put $1:N$ beside the arrow to indicate that it is a multivalued attribute. If the home phone numbers and the office phone numbers of the employees are of interest, then HOME-PHONE-NO X OFFICE-PHONE-NO, which is a repeating group, may be taken into consideration. Both vector (the Cartesian product of value sets) and repeating group are data-aggregates of different types. For the latter, both HOME-PHONE-NO and OFFICE-PHONE-NO are referred to be individually addressable groups of data items. Thus, in the entity-relationship diagram, both HOME-PHONE-NO and OFFICE-PHONE-NO should be created. Each of these attributes maps the entity set EMPLOYEE into a value set PHONE-NO.

Definition 6: An entity-relationship diagram (ERD) is a diagram which consists of a collection of entity sets and relationship sets and their associations, which are in the upper conceptual domains, and the attributes and value sets which are needed to describe the properties of some of the entities and relationships which may be of interest, in the lower conceptual domains.

Example 3: Fig. 1 illustrates the associations of the entity and relationship sets. Fig. 2 illustrates the attributes and value sets needed to describe the properties of some of the entities and relationships which may be of interest to an enterprise. A diagram of Figs. 1 and 2 is an ERD.

Definition 7: A nondeterministic entity-relationship relation M over E (NERR) is defined as $M = (E, R, V, F)$, where

$E = \{E_i | 1 \leq i \leq n\}$ is a collection of countable entity sets,

$R \subseteq \prod_{i=1}^{n} E_i$ is a predefined relationship set over E; if $n = 1$, then $R = \phi$,

$V = \{V_i | 1 \leq i \leq m\}$ is a collection of countable value sets,

and

$$F = \left\{ F_i | F_i \subseteq \{E_\alpha \cup R\} \times \prod_{i=1}^{v_i} V_{ij},\ 1 \leq v_i \leq p \right\},$$

is a finite set of multivalued attributes.

Let E_i be a finite set of distinct entities. Let E_i' and E_i'' be subsets of E_i. If $R \subseteq E_i' \times E_i''$, then we can always write E_i into entity sets E_i' and E_i''. For this reason, without loss of generality, we can impose a restriction that if E is a singleton set, then $R = \phi$; that is, no relationship set is to be defined on the singleton set E.

We may restrict F into a set of functions F_i. Each F_i is a function which maps an entity set or a predefined relationship set R into a value set or a Cartesian product of value sets.

Definition 8: In M, if $F = \{F_i | F_i: \{E_\alpha \cup R\} \rightarrow \prod_{j=1}^{v_i} V_{ij},\ 1 \leq v_i \leq p\}$ is a finite set of attributes, then M is said to be *a deterministic entity-relationship relation M over E* (DERR).

Definition 9: A deterministic entity-relationship model (DERM) is defined as a collection of deterministic entity-relationship relations; otherwise, it is *a nondeterministic entity-relationship model* (NERM).

In the sequel, we shall use the term, entity-relationship relation, if the nondeterminism and determinism are not needed to clarify.

It also should be noted that the entity-relationship relation $M = (E, R, V, F)$ can be displayed in a table form, shown in Fig. 3, which demonstrates information about entities in an entity set or information about relationships in a relationship set. Each row of values is related to a relationship, which is indicated by a group of entities, each having a specific role and belonging to a specific entity set. Each column is related to a value set which, in turn, is related to an attribute. The ordering of rows (tuples) and columns is insignificant. The whole table shown in Fig. 3 is a deterministic entity-relationship relation, if each distinct relationship has one and only one value tuple.

In Fig. 3, that both the relation names and the relation scheme in the table are used for describing the structure of the relation $M = (E, R, V, F)$ is called its intention, which is static. That the relations in the table are used for denoting a set of tuples having the appropriate structure, is called its extension. In this paper, we shall use the term relation scheme to refer to an intention, that is to refer to a structural description of an entity-relationship relation, and the term relational schema to refer to a collection of intentions.

Example 4: The entity-relationship diagram shown in Fig. 2, can be organized in a collection of table forms, shown in Figs. 4-6. They contain information about entities in the entity sets, EMPLOYEE and PROJECT, and information about relationships between two entity sets, EMPLOYEE and PROJECT, in a relationship set PROJ-EMP. In Fig. 6, the attributes STARTING-DATE and %-OF-TIME are the attributes which map the relationships of the relationship set PROJ-EMP into the value sets MONTH X DAY X YEAR and %-OF-TIME respectively. They are neither the attributes of EMPLOYEE nor the attributes of PROJECT, since their meanings depend on both the employee and projects involved.

In the remaining section, we shall show that both NERM and

r.n	Name of other relations				Name of relation R					
r.s	RELATIONSHIP SET Name of relationship set				ATTRIBUTES					
	ro	r_1	r_2	...	r_n	F_1	F_2	F_3	...	$F_{p(R)}$
r.s	ENTITY SETS				VALUE SETS					
	E_1	E_2	...	E_n	V_1	V_2	V_{3_1}	V_{3_2}	...	V_{p_1} V_{p_n}
r	e_{11}	e_{21}	...	e_{n1}	x_1	y_1	a_1	b_1	...	d_1 z_1
	e_{12}	e_{22}	...	e_{n2}	x_2	y_2	a_2	b_2	...	d_2 z_2
	⋮	⋮		⋮	⋮	⋮	⋮	⋮		⋮ ⋮

Fig. 3. A table of an entity-relationship relation M. $r.n$, $r.s$, r, ro, and $F(N)$ denote relation names, relation scheme, relations, roles, and multivalued attribute F, respectively.

r.n	EMPLOYEE-RECORD				
r.s	ENTITY SET	ATTRIBUTES			
		EMP-ID	NAME		PHONE-NO(M)
		VALUE SETS			
	EMPLOYEE	SOC-SEC-NO	FIRST-NAME	LAST-NAME	PHONE-NO
r	e_1	001-7850	PETER	MO	445-3456
	e_1	001-7850	PETER	MO	082-4540
	e_2	070-3135	IDA	LIN	441-3103
	⋮	⋮	⋮	⋮	⋮

Fig. 4. A nondeterministic entity-relationship relation, EMPLOYEE-RECORD: information about entities in an entity set, EMPLOYEE.

r.n	PROJECT-INFORMATION			
r.s	ENTITY SET	ATTRIBUTES		
		PROJ-ID	THIS-YEAR-BUDGET	PROJ-NAME
		VALUE SETS		
	PROJECT	PROJ-NO	BUDGET	PROJ-NAME
s	p_1	M381457	200K	N-DAM
	p_2	A485329	40K	T-ROUTE
	p_3	A486471	3K	K-BRIDGE
	⋮	⋮	⋮	⋮

Fig. 5. A deterministic entity-relationship relation PROJECT-INFORMATION.

r.n	EMPLOYEE-RECORD		PROJECT-INFORMATION	EMPLOYEE-ON-PROJECT			
r.s	RELATIONSHIP SET PROJ-EMP			ATTRIBUTES			
	TO	WORKER	PROJECT	STARTING-DATE (M)			%-OF-TIME (M)
r.s	ENTITY SETS			VALUE SETS			
	EMPLOYEE		PROJECT	MONTH	DAY	YEAR	%-OF-TIME
s	e_1		p_1	12	3	1978	20%
	e_1		p_2	3	1	1979	80%
	e_2		p_2	3	1	1979	20%
	e_2		p_3	3	15	1979	100%
	⋮		⋮	⋮	⋮	⋮	⋮

Fig. 6. A nondeterministic entity-relationship relation, EMPLOYEE-ON-PROJECT: information about relationships in the relationship set PROJ-P.

DERM have the same strength to characterize the information about entities of a given entity set or the information about relationships of a given relationship set. Given any of these entities and relationships, the associated information can be retrieved in an exact form.

Definition 10: Let E be an entity set. Let $R \subseteq \prod_{i=1}^{n} E_i$ be a relationship set over a collection of entity sets E_i, $1 \leq i \leq n$. Let $F = \{F_i | F_i \subseteq (E \cup R) \times \prod_{j=1}^{v_i} V_{ij}, 1 \leq i \leq m, 1 \leq v_i\}$ be a set of attributes. Let $K = \{F_1, F_2, \cdots, F_n | 1 \leq n\}$ be a subset of F.

Then, for any e in E, *the operator K of an entity e* is defined as

$$\begin{aligned} K(e) &= \{F_1, F_2, \cdots, F_n\}(e) \\ &= (F_1(e), F_2(e), \cdots, F_n(e)) \\ &= \left\{(x_1, x_2, \cdots, x_n) | x_i \in F_i(e) \text{ and } x_i \in \prod_{j=1}^{v_i} V_{ij}, 1 \leq i \leq n\right\}. \end{aligned}$$

Let r be a relationship $\langle e_1, e_2, \cdots, e_n \rangle$ in R. *The operator K of a relationship r* is defined as

$$\begin{aligned} K(r) &= \{F_1, F_2, \cdots, F_n\}(r) \\ &= (F_1(r), F_2(r), \cdots, F_n(r)) \\ &= \left\{(x_1, x_2, \cdots, x_n) | x_i \in F_i(r) \text{ and } x_i \in \prod_{j=1}^{v_i} V_{ij}, 1 \leq i \leq n\right\}. \end{aligned}$$

Definition 11: Let $M = (E, R, V, F)$ be an entity-relationship relation. Let x be a tuple $(x_1, x_2, \cdots, x_n)$. We say that the tuple x, an information of an entity e in the entity set E (or an information of a relationship r in the relationship set R), can be *derived* from M if $F = \{F_1, F_2, \cdots, F_n\}$ is the set of attributes of M such that $x \in F(e)$ (or $x \in F(r)$) is in M.

Definition 12: Consider two entity-relationship relations $M = (E, R, V, F)$ and $M' = (E', R', V', F')$, where $E = \{E_i | 1 \leq i \leq n\}$ and $E' = \{E'_i | 1 \leq i \leq m\}$. Let $S \subseteq E \cap E'$. (If R and R' are not empty, then let $R \subseteq S \times \prod_{i=1}^{q} E_j$, where $1 \leq q \leq n$, $E_j \in E$, and $E_j \notin S$, and let $R' \subseteq S \times \prod_{j=1}^{p} E'_j$, where $1 \leq p \leq m$, $E'_j \in E'$, and $E'_j \notin S$.)

We say that M *preserves* M' *under* S if and only if any information $(x_1, x_2, \cdots, x_k)$ of s in S, that can be derived from M', can also be derived from M, where $1 \leq k \leq |F|$, $1 \leq k \leq |F'|$, and $x_i \in \prod_{j=1}^{v_i} V_{ij}$ and $x_i \in \prod_{j=1}^{v'_i} V'_{ij}$, $V_{ij} \in V$, and $V'_{ij} \in V'$.

We say that M *strongly preserves* M' *under* S if and only if M preserves M' under S and M' also preserves M under S.

Let $D = \{M | M = (E, R, V, F)$ is an entity-relationship relation$\}$ and $D' = \{M' | M' = (E', R', V', F')$ is an entity-relationship relation$\}$ be the entity-relationship models. Let $S = \{S'$ | for every $M = (E, \phi, V, F)$ in D and $M' = (E', \phi, V', F')$ in D', $S' \subseteq E \cap E'$, or for every $M = (E, \prod_{i=1}^{p} E_i, V, F)$ and $M' = (E', \prod_{i=1}^{q} E'_i, V', F')$, $S' \subseteq \{E_i | 1 \leq i \leq p\} \cap \{E'_i | 1 \leq i \leq q\}\}$. We say that *the model D preserves the model D' under S* if and only if, for every entity-relationship relation M' in D', there corresponds some entity-relationship relation M in D such that M preserves M' under S'.

We also say that *the model D strongly preserves the model D' under S* if and only if D preserves D' under S, and D' preserves D under S.

It should be noted that, in models D and D', S is a collection

of subsets of entity sets of relations in D and D'; or a collection of subsets of involved entity sets in the relationship sets in D and D'.

Theorem 1: Let $M = (E, R, V, F)$ be a NERR. There corresponds a DERR $M' = (E', R', V, F')$ such that M' strongly preserves M under $E \cup R$. The converse also holds.

Proof: For every $F_i \subseteq R \times \Pi_{j=1}^{v_i} V_{i_j}$ in M, create arbitrarily an entity set E_i' such that E_i' and $\Pi_{j=1}^{v_i} V_{i_j}$ is one-to-one corresponding. Then, let $R' \subseteq R \times \Pi_{j=1}^{v_i} V_{i_j}$. Construct $F' \subseteq R' \times \Pi_{j=1}^{v_i} V_{i_j}$ as follows. Whenever $r \in R$, $F(r) = \{F_1, F_2, \cdots, F_m\}(r) = \{(x_1, x_2, \cdots, x_m) | x_i \in \Pi_{j=1}^{v_i} V_{i_j}, 1 \leq i \leq m\}$ is in M, then for each $(x_1, x_2, \cdots, x_m) \in F(r)$, we have $<r, x_1, x_2, \cdots, x_m> \in R'$, $F'(<r, x_1, x_2, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots, x_m>) = x_i$ is defined by $F_i' \subseteq R' \times \Pi_{j=1}^{v_i} V_{i_j}$ for $1 \leq i \leq m$, and $F'(<r, x_1, x_2, \cdots, x_m>) = \{F_1', F_2', \cdots, F_m'\}(<r, x_1, x_2, \cdots, x_m>) = (x_1, x_2, \cdots, x_m)$ is in M'. Clearly, $F_i': R' \to \Pi_{j=1}^{v_i} V_{i_j}$ and M' strongly preserves M under R. From the definitions, the converse also holds.

Corollary 1: The NERM and DERM have the same strength to characterize the same information about entities of a collection of entity sets or the same information about relationships of a collection of relationship sets.

Proof: It follows from the definitions and Theorem 1.

In the entity-relationship approach to the logical database design, one of the major steps [2] is to identify the properties of entities and relationships, which are of interest to the enterprise. That is, we wish to identify attributes and value types for the entities and relationships. In order to reduce the amount of redundancy in the stored data and to minimize the problem of inconsistency in the stored data, the functional attributes are desired. However, in the process of logical database design, it becomes difficult to design a database if we are allowed to use only functional attributes. By Theorem 1, we can use relational attributes and value types to identify the properties of entities and relationships, and then we transform these relational attributes into functional attributes.

III. Entity and Relationship Relations

In this section, we shall identify an entity-relationship relation into one of the four types: a regular or weak entity relation, or a regular or weak relationship relation. They are different from the entity-relationship relation in the sense that they identify the entities or relationships of the entity-relationship relation by values, which are elements of a Cartesian product of value sets or elements of a value set.

In Section II, we have defined the operator K of an entity e and the operator K of a relationship r. We extend the operator K into a set operator $\{K_1, K_2, \cdots, K_t\}$; each K_i contains a set of attributes F_{i_m}.

Definition 13: Let E be an entity set. Let $E_i = \{E_i | 1 \leq j \leq n_i\}$ be a collection of entity sets, where $1 \leq i \leq t$. Let $R \subseteq \Pi_{j=1}^{n_i} E_{i_j}$ be a relationship set. Let $R \subseteq \Pi_{j=1}^{t}\{E \cup \cdot_j\}$. Let $F_i = \{F_{i_m} | F_{i_m} \subseteq \{E \cup R_i\} \times \Pi_{j=1}^{v_{i_m}} V_{i_{m_j}}\}$, $1 \leq i \leq t$. Let $K_i = \{F_{i_1}, F_{i_2}, \cdots, F_{i_m}\}$, $1 \leq i \leq t$. Let $K = \{K_1, K_2, \cdots, K_t\}$.

Then, for any $<r_1, r_2, \cdots, r_t> \in R$, where $r_i \in E \cup R_i$, *the operator K of a relationship* $<r_1, r_2, \cdots, r_t>$ is defined recursively as follows:

$$K_i(r_i) = \{F_{i_1}, F_{i_2}, \cdots, F_{i_m}\}(r_i), \quad \text{for every } 1 \leq i \leq t, \tag{1}$$

$$K_i(r_j) \quad \text{is undefined for } 1 \leq i \neq j \leq t, \tag{2}$$

and

$$\begin{aligned} K(<r_1, r_2, \cdots, r_t>) &= \{K_1, K_2, \cdots, K_t\} \\ &\quad \cdot (<r_1, r_2, \cdots, r_t>) \\ &= (K_1(<r_1, r_2, \cdots, r_t>), \\ &\quad K_2(<r_1, r_2, \cdots, r_t>), \cdots, \\ &\quad K_t(<r_1, r_2, \cdots, r_t>)) \\ &= (K_1(r_1), K_2(r_2), \cdots, K_t(r_t)). \end{aligned} \tag{3}$$

Definition 14: Let $M = (E, R, V, F)$ be an entity-relationship relation, where $E = \{E_i | 1 \leq i \leq n\}$; $R \subseteq \Pi_{i=1}^{n} E_i$, $1 < n$, and if $n = 1$, the $R = \phi$, and $F = \{F_i | F_i \subseteq \{E_\alpha \cup R\} \times \Pi_{j=1}^{v_i} V_{i_j}, 1 \leq v_i \leq p\}$. Then K, a subset of F, is a *superkey* of M if, for every r and r' in $E_\alpha \cup R$, $r \neq r'$ if and only if $K(r) \neq K(r')$, and both $K(r)$ and $K(r')$ are singleton sets.

In particular, if E is an entity set (i.e., R is empty), then $F = \{F_i | F_i \subseteq E \times \Pi_{j=1}^{v_i} V_{i_j}, 1 \leq v_i \leq p\}$. Then K, a subset of F, is a *superkey* of M if, for every e and e' in E, $e \neq e'$ if and only if $K(e) \neq K(e')$, and both $K(e)$ and $K(e')$ are singleton sets.

The set K is a *key* of M if it is a minimal superkey. K is the *primary key* of M if the minimal superkey K is selected as key of M.

Clearly, an entity-relationship relation M can have many keys. A superkey of M is any set of attributes in M that contains a key of M. Every key is also a superkey. Furthermore, if M is a deterministic entity-relationship relation, then K, a subset of F, is a superkey of M if, for every r and r' in $E_\alpha \cup R$, $r \neq r'$ if and only if $K(r) \neq K(r')$, and both $K(r)$ and $K(r')$ are singleton sets.

Based on the methods of identifying entities of relationships, the entity-relationship relations can be identified into four types. Without loss of generality, let K and F be disjoint sets of attributes.

Definition 15: Let $E = \{e_i | 1 \leq i \leq \alpha\}$ be a countable entity set. *A nondeterministic regular entity relation* (NRER) *M over E* is defined as $M_E = (K, R, V, F)$ where $R = \phi$, $V = \{V_i | 1 \leq i \leq m\}$ is a collection of value sets, $K = \{F_1', F_2', \cdots, F_n' | F_i' \subseteq E \times \Pi_{j=1}^{v_i} V_{i_j}, F_i'$ is not in $F, 1 \leq i \leq n\}$ is the primary key, and $F = \{F_i |$ for every $1 \leq i \leq k, F_i \subseteq K(E) \times \Pi_{j=1}^{v_i} V_{i_j}, 1 \leq v_i\}$ is the set of the attributes.

Definition 16: In M_E, if $F = \{F_i |$ for every $1 \leq i \leq k, F_i: K(E) \to \Pi_{j=1}^{v_i} V_{i_j}, 1 \leq v_i\}$ is the set of (functional) attributes, then M_E is said to be *a deterministic regular entity relation* (DRER) *M over E*.

Definition 17: Let $E' = \{E_i | 1 \leq i \leq n\}$ be a collection of countable entity sets. Let $E \notin E'$ be a countable entity set. Let K_i be the primary keys of the involved entities E_i. *A nondeterministic weak entity relation* (NWER) *M over E and E'* is defined as $M_{E,E'} = (K, R, V, F)$ where $R \subseteq \Pi_{i=1}^{n} E_i \times E$ is a relationship set over E' and E, $V = \{V_i | 1 \leq i \leq m\}$ is a collection of value sets, $K = \{K_1, K_2, \cdots, K_n, F_1', F_2', \cdots, F_k' | 1 \leq i \leq n, 1 \leq \alpha \leq k, K_i \subseteq E_i \times \Pi_{j=1}^{v_i} V_{i_j}$, where $1 \leq v_i$ and

$V_{i_j} = \Pi_{s=1}^{p} V'_s$, $1 \leq p$, for some $V'_s \in V$, and $F'_\alpha \subseteq E \leq \Pi_{j=1}^{v_\alpha} V_{\alpha_j}$, $V_{\alpha_j} \in V$} is the primary key, and $F = \{F_l$ for $1 \leq l \leq q$, $F_l \subseteq K$ $\Pi_{j=1}^{v_l} V_{l_j}$, $1 \leq v_l\}$ is the set of attributes.

Definition 18: In $M_{E,E'}$, if $F = \{F_l$ for $1 \leq l \leq q$, F_l: $K(R) \rightarrow \Pi_{j=1}^{v_l} V_{l_j}$, $1 \leq v_l\}$ is the set of attributes, then $M_{E,E'}$ is said to be *a deterministic weak entity relation* (DWER) M *over* E *and* E'.

Definition 19: Let $E = \{E_l | 1 \leq l \leq n\}$ be a collection of countable entity sets. Let K_l be the primary keys of the involved entities in the entity set E_l. Then *a nondeterministic regular relationship relation* (NRRR) M *over* E is defined as $M_E = (K, R, V, F)$ where $R \subseteq \Pi_{j=1}^{n} E_j$ is a relationship set over E, $V = \{V_l | 1 \leq l \leq v\}$ is a set of value sets, $K = \{K_1, K_2, \cdots, K_n | 1 < n$, $1 \leq l \leq n$, $K_l \subseteq E_l \times \Pi_{j=1}^{v_l} V_{l_j}$, where $1 \leq v_l$ and $V_{i_j} = \Pi_{s=1}^{p} V'_s$, $1 \leq p$ and for some $V'_s \in V\}$ is the primary key, and $F = \{F_l$ for $1 \leq l \leq k$, $F_l \subseteq K(R) \times \Pi_{j=1}^{v_l} V_{l_j}$, $1 \leq v_l\}$ is the set of attributes.

Definition 20: In M_E, if $F = \{F_l$ for $1 \leq l \leq k, F_l: K(R) \rightarrow \Pi_{j=1}^{v_l} V_{l_j}$, $1 \leq v_l\}$ is the set of attributes, then M_E is said to be *a deterministic regular relationship relation* (DRRR) M *over* E.

Definition 21: Let $E = \{E_l | 1 \leq l \leq n\}$ and $E' = \{E'_l | 1 \leq l \leq m\}$ be disjoint collections of entity sets. Let K'_l be the primary keys of the involved entities in E'_l. Let K_l be the primary keys of the involved entities in E_l. Then *a nondeterministic weak relationship relation* (NWRR) M *over* E *and* E' is defined as $M_{E,E'} = (K, R, V, F)$, where $R \subseteq \Pi_{j=1}^{m} E'_j \times \Pi_{l=1}^{n} E_l$ is a relationship set over E and E', $V = \{V_l | 1 \leq l \leq v\}$ is a collection of value sets, $K = \{K'_1, K'_2, \cdots, K'_m, K_1, K_2, \cdots, K_n | 1 \leq m$, $1 < n$, for $1 \leq \alpha \leq m$, $K'_\alpha \subseteq E'_\alpha \times \Pi_{j=1}^{v_\alpha} V_{\alpha_j}$, where $1 \leq v_\alpha$ and $V_{\alpha_j} = \Pi_{q=1}^{p} V'_q$, $1 \leq p$ and $V'_q \in V$, and for $1 \leq s \leq n$, $K_s \subseteq E_s \times \Pi_{j=1}^{v_s} V_{s_j}$, where $1 \leq v_s$ and $V_{s_j} = \Pi_{q=1}^{p} V''_q$, $1 \leq p$ and $V''_q \in V\}$ is the primary key, and $F = \{F_l$ for $1 \leq l \leq k$, $F_l \subseteq K(R) \times \Pi_{j=1}^{v_l} V_{l_j}$, $1 \leq v_l\}$ is the set of attributes.

Definition 22: In $M_{E,E'}$, if $F = \{F_l$ for $1 \leq l \leq k$, F_l: $K(R) \rightarrow \Pi_{j=1}^{v_l} V_{l_j}$, $1 \leq v_l\}$ is the set of attributes, then $M_{E,E'}$ is said to be *a deterministic weak relationship relation* (DWRR) M *over* E *and* E'.

Example 5: Figs. 7-13 demonstrate the table forms of the regular entity, weak entity, regular relationship, and weak relationship relations. This design of table forms does provide to the users a clarity, simplicity, and unity of the concepts and the structure of the entity-relationship relations.

In Fig. 9, the involved entities in the regular relationship relation are represented by their primary keys, EMP-ID and PROJ-ID of their regular entity relations, EMPLOYEE-RECORD, and PROJECT-INFORMATION, respectively. The role names provide the semantic meaning for the values in the corresponding columns. In the regular entity relation CHILDREN-RECORD, shown in Fig. 11, the dependents are identified by their children identification, CHILD-ID; NAME and AGE are attributes of the dependents. But, the appearance of EMPLOYEE-RECORD affiliated with the CHILDREN-RECORD indicates that the existence of the entity set CHILDREN depends on EMPLOYEE. However, since this existence relationship of the entity sets EMPLOYEE and CHILDREN without any identification purpose, it must be represented in a table form given in Fig. 10, and does not have any attribute and value. In the relation

R.n	EMPLOYEE-RECORD			
R.s	PRIMARY KEY	ATTRIBUTES		
	EMP-ID	NAME		PHONE-NO (M)
	VALUE SETS			
	SOC-SEC-NO	FIRST-NAME	LAST-NAME	PHONE-NO
R	001-7850 001-7850 078-3135	PETER PETER IDA	MC MC LIN	445-3456 882-4560 443-3183

Fig. 7. A nondeterministic regular entity relation EMPLOYEE-RECORD.

R.n	PROJECT-INFORMATION		
R.s	PRIMARY KEY	ATTRIBUTES	
	PROJ-ID	THIS-YEAR-BUDGET	PROJ-NAME
	VALUE SETS		
	PROJ-NO	BUDGET	PROJ-NAME
R	M301457 M405375 A486471	200K 40K 3K	H-DAM T-ROUTE K-BRIDGE

Fig. 8. A deterministic regular entity relation PROJECT-INFORMATION.

R.n	EMPLOYEE-RECORD	PROJECT-INFORMATION	EMPLOYEE-ON-PROJECT			
R.s	PRIMARY KEY		ATTRIBUTES			
	EMP-ID	PROJ-ID	START-ING-DATE (M)		% OF-TIME (M)	
	ro WORKER	PROJECT				
	VALUE SETS					
	SOC-SEC-NO	PROJ-NO	MONTH	DAY	YEAR	% OF-TIME
R	001-7850 001-7850 001-7850 078-3135	M301457 M301357 A485375 A485375	12 3 3 1	3 1 1 15	1978 1979 1979 1979	20% 80% 20% 100%

Fig. 9. A nondeterministic regular relationship relation, EMPLOYEE-ON-PROJECT.

R.n	EMPLOYEE-RECORD	CHILDREN-RECORD	EXISTENCE-DEPENDENCY-EMP-CHILD
R.s	PRIMARY KEY		ATTRIBUTE
	EMP-ID	CHILD-ID	NONE
	ro SUPPORTER	DEPENDENT	
	VALUE SETS		
	SOC-SEC-NO	SOC-SEC-NO	NONE
R	001-7850 001-7850 078-3158	100-2137 100-0313 203-9068	

Fig. 10. A deterministic regular relationship relation, EXISTENCE-DEPENDENCY-EMP-CHILD.

PARENTS-RECORD, shown in Fig. 12, the parents are identified by their names and by the values of the primary key of the employees supporting them. Therefore, the appearance of the EMPLOYEE-RECORD affiliated with the PARENTS-RECORD indicates the existence and identification dependencies of the entity set PARENTS on another entity set EMPLOYEE. PHONE-NO is the only attribute of the parents. Fig. 13 is a weak relationship relation, in which, YEAR-OF-MARRIAGE is an attribute of the relationship set MARRIAGE over the entity set PARENTS. Each of the relationships is identified by the NAME of FATHER and NAME of MOTHER and their children's EMP-ID supporting

r.n		CHILDREN-RECORD		
r.a	PRIMARY KEY	PRIMARY KEY	ATTRIBUTES	
	of EMPLOYEE-RECORD	CHILDREN-RECORD	NAME	AGE
	EMP-ID	CHILD-ID		
	TO SUPPORTER	[illegible]		
	VALUE SETS			
		SOC-SEC-NO	FIRST-NAME	NO-OF-YEAR
r		100-2137 101-0303 202-9088	ERIC EVAN LENIE	9 1 11

Fig. 11. A deterministic regular entity relation, CHILDREN-RECORD. The table also shows the existence dependency of CHILDREN on EMPLOYEE.

r.n		PARENTS-RECORD			
r.a	PRIMARY KEY			ATTRIBUTE	
	of EMPLOYEE-RECORD	PARENTS-RECORD		PHONE-NO	
	EMP-ID	NAME			
	TO SUPPORTER	PARENTS			
	VALUE SETS				
	SOC-SEC-NO	FIRST-NAME	LAST-NAME	PHONE-NO	
r	001-7850 001-7850 078-3135 078-3135	BILL JANE JOHN LINE	NG LIN LEE TER	345-7870 345-7870 445-6260 871-5643	

Fig. 12. A deterministic weak entity relation, PARENTS-RECORD. The table also shows the existence and identification dependencies of PARENTS on EMPLOYEE.

r.n		PARENT-RECORD	PARENTS-RECORD	PARENTS-MARRIAGE-RECORD		
r.a	PRIMARY KEY			ATTRIBUTE		
	of EMPLOYEE-RECORD	PARENTS-RECORD	PARENTS-RECORD	DATE-OF-MARRIAGE		
	EMP-ID	NAME	NAME			
	TO SUPPORTER	FATHER	MOTHER			
	VALUE SETS					
	SOC-SEC-NO	FIRST-NAME	LAST-NAME	FIRST-NAME	LAST-NAME	MONTH	DAY	YEAR
r	001-7850 078-3135 125-7411	BILL JOHN ED	NG LEE KEN	JANE LINE LINE	LIN TER TER	5 3 12	25 2 30	1969 1943 1960

Fig. 13. A deterministic weak relationship relation, PARENTS-MARRIAGE-RECORD. The table also shows the existence and identification dependencies of PARENTS on EMPLOYEE.

them. The appearance of EMPLOYEE-RECORD affiliated with the PARENTS-RECORD also indicates the existence and identification dependencies of the entity set PARENTS on another entity set EMPLOYEE.

In conclusion, there corresponds a relation (of table form) for every entity set or relationship set in the entity-relationship diagram. For any entity set E', the identification of the entities in E' is dependent on another entity set E if and only if the relation for the entity set E' is a weak entity relation in which the primary key is the combination of the primary key in the relation for E and an attribute (or possibly a set of attributes) in the relation for E'. Furthermore, if there is a relationship set R over the entity set E' and others, then the relation for the relationship set R is a weak relationship relation, and vice versa. Otherwise, the relation for any entity set or any relationship set is a regular entity relation or a regular relationship relation, respectively.

Definition 23: A deterministic entity-relationship model D is defined as $D = \{M |$ for every M, M is one of the deterministic types: a regular or weak entity relation, or a regular or weak relationship relation$\}$. In D, if one of the relations is nondeterministic, then D is said to be *a nondeterministic entity-relationship model*.

Definition 24: Let $M = (K, R, V, F)$ be one of the regular or weak relation, or regular or weak relationship relation. Let x be a tuple $(x_1, x_2, \cdots, x_n)$. We say that the tuple x, an information of a given k, which is a value of the primary key K, can be *derived* from M if $F = \{F_1, F_2, \cdots, F_n\}$ is the set of attributes of M such that $x \in F(k)$ is in M.

Note that k is either a value in V_i, for some V_i in V, or a tuple in $\prod_{i=1}^{n} V_i$, for V_i in V, and k represents uniquely an entity or a relationship, respectively.

Definition 25: Consider two relatations: $M = (K, R, V, F)$ and $M' = (K', R', V', F')$ which are one of the types: a regular or weak entity relation, or a regular or weak relationship relation. Let $S \subseteq K \cap K'$. We say that *M preserves M' under S* if and only if any information $(x_1, x_2, \cdots, x_k)$ of s, which is a value of S, that can be derived from M', can also be derived from M, where $1 \leqslant k \leqslant |F|$ and $1 \leqslant k \leqslant |F'|$, and $x_i \in \prod_{j=1}^{v_i} V_{i_j}$ and $x_i \in \prod_{j=1}^{v_i} V'_{i_j}$, V_{i_j} in V and V'_{i_j} in V'. We say that *M strongly preserves M' under S* if and only if M preserves M' under S, and M' preserves M under S.

Similarly, we say that the model D preserves the model D' under $S = \{S' | S' \subseteq K \cap K'\}$, if and only if for every relation $M' = (K', R', V', F')$ in D', there corresponds some relation $M = (K, R, V, F)$ in D such that M preserves M' under S' in S.

We also say that *the model D strongly preserves the model D' under S* if and only if D preserves D' under S and D' preserves D under S.

It should be noted that, in the relations M and M', S is a subset of K, where K is described by a group of attributes that are used to identify uniquely entities in the entity set, or relationships in the relationship set. s is an attribute value, which is an element of a Cartesian product of value sets or simply a value set. For the model D and D', S is a collection of attribute sets S'; each attribute set S' is a subset of K' and K.

IV. Relationship Between Entity (Relationship) Relation and Entity-Relationship Relation

In this section, we shall show that the entity-relationship relations can be translated exactly into relations of the user schema. We shall demonstrate the fact that each tuple of any entity-relationship relation can be uniquely identified by some primary keys. Basically, any entity key is a group of attributes such that the mapping from the entity set onto the corresponding group of value sets is one-to-one. If we cannot find such a one-to-one mapping on available data, or if simplicity in identifying entities is desired, we may define an artifical attribute and a value set so that such mapping is possibly defined. In the case where several keys exist, we usually choose a semantically meaningful key as the entity primary key.

In certain cases, the entities in an entity set cannot be uniquely identified by the values of their own attributes; then we must use a relationship(s) to identify them. Theoretically, many kinds of relationships may be used to identify entities. The method of identification of entities by relationships with other entities can be applied recursively until the entities which can be identified by their own attributes' values are reached. Therefore, we have two forms of entity relations. If relationships are used for identifying the entities, we shall call it a weak entity relation. If relationships are not used for identifying the entities, we shall call it a regular entity relation.

Since a relationship is identified by the involved entities, the primary key of a relationship can be represented by the primary keys of the involved entities. We also have two forms of relationship relations. If all entities in the relationship are identified by their own attributes' values, we shall call it a regular relationship relation. If some entities in the relationship are identified by other relationships, we shall call it a weak relationship relation. Thus, we state the following.

Theorem 2: For any entity-relationship relation $M = (E, R, V, F)$, there corresponds a regular or weak entity relation, or a regular or weak relationship relation $M' = (K, R', V, F)$ such that M' strongly preserves M under $K(E \cup R)$, the identification of entities or relationships. The converse is true.

Proof: Let $M_{E,E'} = (K, R, V, F)$ be a weak relationship relation over E and E'. Then, we can construct an entity-relationship relation $M' = (E'', R', V, F')$ as follows. Let t_1 be a tuple $<e'_1, e'_2, \cdots, e'_m, e_1, e_2, \cdots, e_n>$ in $E' \times E$. Let t_2 be a tuple $(z_1, z_2, \cdots, z_m, x_1, x_2, \cdots, x_n)$ in $\Pi_{i=1}^{m} \Pi_{j=1}^{v_i} V_{i_j} \times \Pi_{r=1}^{n} \Pi_{s=1}^{v_r} V_{r_s}$, where V_{i_j} and V_{r_s} are in V. For each $K(t_1) = t_2$, we have $E'' = \{\overline{E}_i \mid 1 \leq i \leq n\} \cup \{\overline{E}'_j \mid 1 \leq j \leq m\}$ such that $z_j \in \overline{E}'_j$, for $1 \leq j \leq m$ and $x_i \in \overline{E}_i$, for $1 \leq i \leq n$, and $t_2 \in R' \subseteq \Pi_{j=1}^{m} \overline{E}'_j \times \Pi_{i=1}^{n} \overline{E}_i$. $F' = \{P_i \mid P_i \subseteq R' \times \Pi_{j=1}^{v_i} V_{i_j}, 1 \leq i \leq k, 1 \leq v_i$ such that for every i, $F_i \in F$, $y_i \in F_i(K(t_1))$, where $K(t_1) = t_2$ is defined by $F_i \subseteq K(R) \times \Pi_{j=1}^{v_i} V_{i_j}$ if and only if $y_i \in P_i(t_2)$ is defined by $P_i\} \cup \{P'_i \mid 1 \leq i \leq n, P'_i \subseteq R' \times \Pi_{j=1}^{v_i} V_{i_j}, 1 \leq v_i$ such that $P'_i(t_2) = x_i \in \Pi_{j=1}^{v_i} V_{i_j}$ if and only if $x_i \in K_i(e_i)$ for $e_i \in E_i$ and $K(t_1) = t_2\}$. For $1 \leq i \leq n$, $K_i = \{F_{i_j} \mid 1 \leq j \leq \alpha\}$, $P'_i = \{P'_{i_j} \mid 1 \leq j \leq \alpha$, $P'_{i_j} \subseteq R' \times \Pi_{s=1}^{p} V'_s$, such that $P'_i(t_2) = x_i = (x_{i_1}, x_{i_2}, \cdots, x_{i_{j-1}}, x_{i_j}, x_{i_{j+1}}, \cdots, x_{i_\alpha}) \in \Pi_{j=1}^{v_i} V_{i_j}$, where $V_{i_j} = \Pi_{s=1}^{p} V'_s$ if and only if $x_i \in K(e_i) = \{F_{i_1}, F_{i_2}, \cdots, F_{i_{j-1}}, F_{i_j}, F_{i_{j+1}}, \cdots, F_{i_\alpha}\}(e_i)$, and $K(t_1) = t_2\}$. Clearly, M' is an entity-relationship relation such that M' strongly preserves M under $K(R)$. Similarly, it can be shown in the same manner that the other types of relations can be transformed into entity-relationship relations, without loss of information. And so is the converse.

Corollary 2: For any entity-relationship model D, there corresponds D', which contains a collection of relations M, each described by one of the types: a regular or weak entity relation, or a regular or weak relationship relation such that D' strongly preserves D under the collection of primary keys $K(E \cup R)$. The converse if true.

Proof: It follows from Theorem 2.

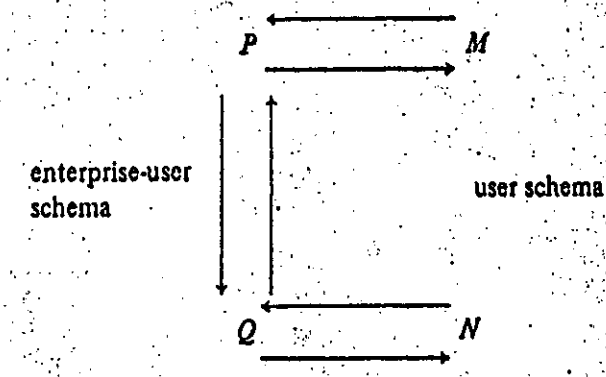
Corollary 3: For any NERR $M = (E, R, V, F)$, there corresponds a DRER, DWER, DRRR, or DWRR $M' = (K, R', V, F')$ such that M' strongly preserves M under $K(E \cup R)$. The converse is true.

Proof: In the proof of Theorem 2, it can be shown that the set of attributes is indeed a set of functions.

Corollary 4: For any NRER, NWER, NRRR, or NWRR $M = (K, R, V, F)$, there corresponds a DERR $M' = (E', R', V', F')$ such that M' strongly preserves M under $K(E \cup R)$ or $E \cup R$. The converse is true.

Proof: In the proof of Theorem 2, it can be shown that the set of attributes that we have constructed for the NERR are functions.

At this point, we can summarize our results, which we have accomplished, in terms of the following graph. Let M and N, respectively, be a nondeterministic and a deterministic relation, each described by one of the types: a regular or weak relation, or a regular or weak relationship relation. Let P and Q be a NERR and DERR, respectively. Let $M \rightarrow N$ denote that M can be transformed into N such that N strongly preserves M under $K(E \cup R)$, where K is used to identify uniquely entities in the entity set E or relationships in the relationship set R.



V. The Entity-Relationship Models Versus Other Data Models

In [2], [3], [13], the use of an entity-relationship diagram is fully explained, along with rules and examples for translation into hierarchical or network structures. In this section, we shall modify the major steps in logical database design using the entity-relationship approach in [3], [14]. The modified steps in logical database design can be used to derive entity-relationship relations that correspond equivalently to $3NF$ relations of the relational model. Hence, the use of entity-relationship diagram for translating into $3NF$ relations supported by the relational database system can be realized.

In the relational model, to illustrate transitive dependency, consider a relation scheme S concerning information about employees, departments where they work, and the nature of the work within the department [9].

$S(E\#, EN, JC, D\#, M\#, CT)$ where
$E\#$ = employee identification number,
EN = employee name,
JC = employee job code,
$D\#$ = department number of employee,
$M\#$ = identification number of department manager, and
CT = contract type (government or nongovernment).

In the description of the database application, the functional relationships among application attributes are given as follows.

Each employee is given only one job code and is assigned to only one department. Each department has its own manager and is involved in work on either government or nongovernment contracts, not both. The nontrivial functional dependencies in S are as shown in Fig. 14 (the nondependencies are implied). Clearly, S is not in $3NF$.

Using the process of further normalization, the relation scheme S can be converted to a relational schema containing $S1(\underline{E\#}, EN, JC, D\#)$ and $S2(\underline{D\#}, M\#, CT)$, which are in $3NF$. In addition, no essential information has been lost, since at any time, the original relations can be recovered by taking the natural join $S1$ and $S2$ on $D\#$.

Using the entity-relationship approach to the logical database design, without further considerations, we might define the enterprise schema for S, in terms of an entity-relationship diagram for EMP (employee) which is given in Fig. 15, and then translate this enterprise schema into a user schema, described by the regular entity relation EMP, which is given in Fig. 16. Clearly, all the undesirable anomalies, that can be created by insertion, updating, and deletion of an tuple in the relation scheme S, can also occur in the regular entity relation EMP. To eliminate these problems, the relation scheme S must be put into $3NF$. That is, the relation scheme S must satisfy the property: every nonkey attribute is fully functionally and nontransitively dependent on the primary key. In the regular entity relation, EMP of the ERD for EMP, the attributes which map from $E\#$ into their own value sets, are functional. The determinism of entity-relationship relations, described by the types: the regular or weak entity relations, or the regular or weak relationship relations, do not guarantee that the entity-relationship relations can overcome all anomalies with respect to storage operations. However, in Fig. 15, the entity-relationship diagram for EMP does not capture all the given semantics, namely, the given description of the database application in the real world. For instance, it does not include the fact that "each department has its own manager and is involved in work on either government or nongovernment contracts, not both." In conformity to the given description of the database application, a number of directed edges can be added deliberately to the value sets in the entity-relationship diagram for EMP. However, the obtained diagram given in Fig. 17 is no longer an entity-relationship diagram.

In order to conform to the semantics "each department has its own manager and is involved in work on either government or nongovernment contracts, not both," and to preserve the property of an entity-relationship diagram, a relationship set DEPT-MANAGER must be created. The relationship set is a one-to-one mapping defined on two entity sets DEPT and EMP. The relationship set has a functional attribute INVOLVED-PROJ-TYPE, which maps relationships of the relationship set DEPT-MANAGER into a value set CT (contract type). In Fig. 18, the entity-relationship diagram does conform to the functional dependencies, given in Fig. 14. It should be noted that the relationship set DEPT-EMP is a one-to-many mapping defined on two entity sets DEPT and EMP; this construct justifies the fact that DEPT is functionally dependent on EMP. The attribute INVOLVED-PROJ-TYPE maps functionally relationships of the relationship set DEPT-EMP into the value set CT.

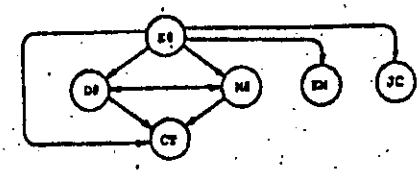


Fig. 14. Example of several transitive dependencies.

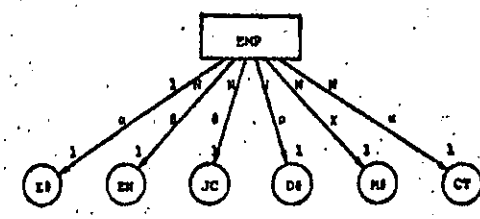


Fig. 15. An entity-relationship diagram for EMP, where EMP-ID is the primary key. $\alpha, \beta, \theta, \rho, \chi$, and κ denote EMP-ID, EMP-NAME, JOBCODE, HIS-DEPT-NO, HIS-DEPT-MANAGER, and HIS-WORKING-PROJ-TYPE, respectively.

r.n	EMP		SKILL	EMP-SKILL
r.s	PRIMARY KEY			ATTRIBUTE
	EMP-ID		SKILL-NO	SKILL-LEVEL
	FO	EMPLOYEE	SKILL	
	tn	N	N	
	VALUE SETS			
	EI		SI	SL
r	1		005	A
	1		015	C
	2		010	A
	2		015	C
	2		005	F
	3		015	A

Fig. 16. A regular entity relation EMP.

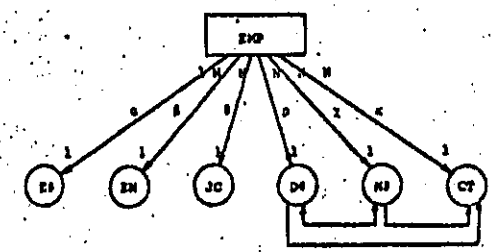


Fig. 17. A diagram conformable to the functional dependences given in Fig. 14.

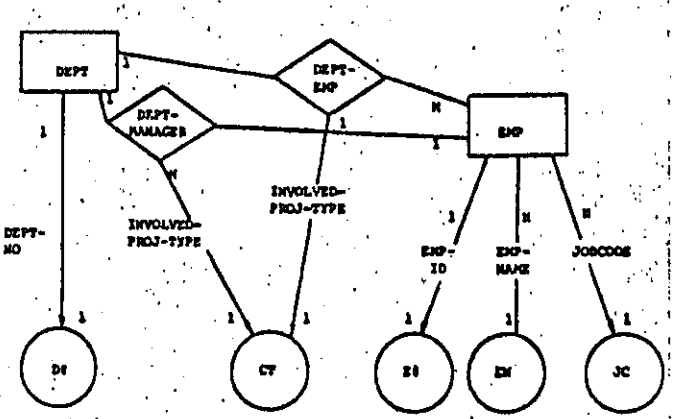


Fig. 18. An entity-relationship diagram concerning information about EMP and DEPT.

The construct also conforms to the fact that CT is functionally dependent on DEPT and is transitively dependent on EMP. However, the attribute INVOLVED-PROJ-TYPE, which maps functionally relationships in the relationship set DEPT-EMP

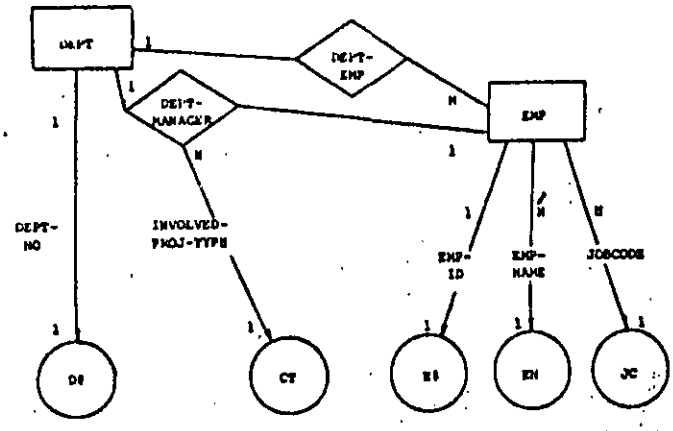
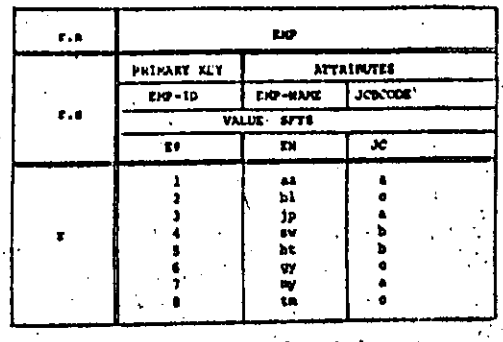


Fig. 19. An entity-relationship diagram concerning information about EMP and DEPT.

into the value set *CT*, is redundant because of the existence of the attribute INVOLVED-PROJ-TYPE which also maps functionally relationships in the relationship set DEPT-MANAGER. The existence of the attribute INVOLVED-PROJ-TYPE mapping relationships in the relationship set DEPT-MANAGER into the value set *CT* implies that *CT* is transitively dependent on EMP (because *CT* is functionally dependent on DEPT, and DEPT is functionally dependent on EMP) [4], [5]. The elimination of the redundant attribute INVOLVED-PROJ-TYPE yields the entity-relationship diagram as shown in Fig. 19, which conforms to the given semantics of the real world. Figs. 20–22 illustrate the translation from the entity-relationship diagram given in Fig. 19 into a user schema.

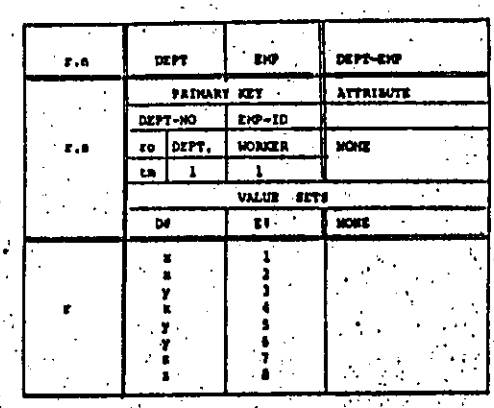
Several problems will arise from this construct.

1) *In Eliminating the Redundant Relationship Sets:* Without loss of generality, let the entity set EMPLOYEE be partitioned into two entity sets EMP′ and MEMP, containing nonmanagerial and managerial employees, respectively. Then, as shown in Fig. 23, create three relationship sets DEPT-EMP, DEPT-MANAGER, and EMP-MANAGER defined on the entity sets DEPT and EMP′, the entity sets DEPT and MEMP, and the entity sets EMP′ and MEMP, respectively. This construct justifies the facts that DEPT is functionally dependent on EMP′ (employee), that DEPT is functionally dependent on MEMP (manager), and vice versa, and that MEMP is functionally dependent on EMP′. That is, this entity-relationship diagram conforms to the given semantics of the real world: "Each employee is assigned to only one department. Each department has its own manager . . . ." However, the existence of the relationship set EMP-MANAGER defined on the entity sets EMP′ and MEMP is redundant, because the functional dependencies of DEPT on EMP′ and of MEMP on DEPT imply the functional dependency of MEMP on EMP′. Furthermore, the diagram shown in Fig. 23 is equivalent to the entity-relationship diagram shown in Fig. 24, which is obtained by combining the entity sets MEMP and EMP′ into EMP. The elimination of the relationship set EMP-MANAGER from both diagrams does not delete any semantic information from the original diagrams. Thus, in the entity-relationship diagram shown in Fig. 19, the functional dependency of MANAGER on EMP (nonmanagerial employee) is implicitly defined. Thus, the claim that the existence of the attribute INVOLVED-PROJ-TYPE mapping relationships in the relationship set DEPT-MANAGER into the value set *CT* implies that *CT* is transitively dependent on EMP, is valid.



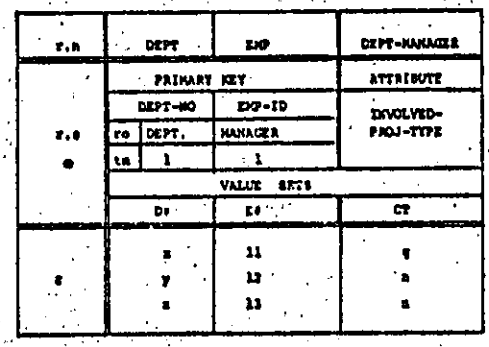
F.n	EMP		
	PRIMARY KEY	ATTRIBUTES	
F.n	EMP-ID	EMP-NAME	JOBCODE
	VALUE SETS		
	E#	EN	JC
F	1	aa	a
	2	bl	c
	3	jp	a
	4	sw	b
	5	bt	b
	6	yy	c
	7	my	a
	8	tn	c

Fig. 20. A regular entity relation EMP.



F.n	DEPT		EMP	DEPT-EMP
	PRIMARY KEY			ATTRIBUTE
	DEPT-NO		EMP-ID	
F.n	to	DEPT.	WORKER	NONE
	tn	1	1	
	VALUE SETS			
	D#		E#	NONE
F	x		1	
	x		2	
	y		3	
	x		4	
	y		5	
	y		6	
	z		7	
	z		8	

Fig. 21. A regular relationship relation DEPT-EMP. *tm* denotes type of mapping.



F.n	DEPT		EMP	DEPT-MANAGER
	PRIMARY KEY			ATTRIBUTE
	DEPT-NO		EMP-ID	INVOLVED-PROJ-TYPE
F.n	to	DEPT.	MANAGER	
	tn	1	1	
	VALUE SETS			
	D#		E#	CT
F	x		11	q
	y		12	n
	z		13	n

Fig. 22. A regular relationship relation DEPT-MANAGER.

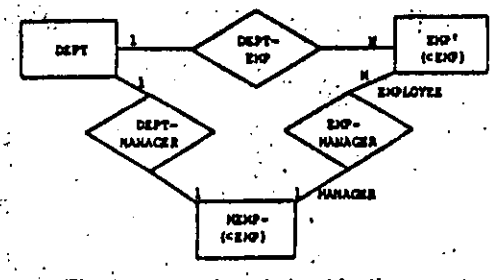


Fig. 23. An entity-relationship diagram.

It should be noted that, in Figs. 23 and 24, instead of the relationship set EMP-MANAGER, the elimination of the relationship set DEPT-EMP also does not delete any semantic information from the original diagrams. However, the elimination of the relationship set DEPT-MANAGER from the diagrams will destroy, in turn, the given semantic information: "each department has its own manager . . . ." In the reality, the implication of functional dependence from others is not quite obvious, especially, if a very large set of functional de-

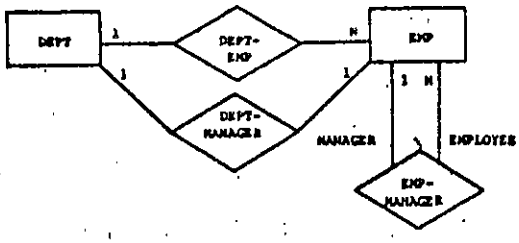


Fig. 24. An entity-relationship diagram.

pendencies is considered, and the criteria for determining which of the redundant relationship set is to be eliminated without modifying any given semantic information, is nontrivial.

In order to allow an additional illustration of full functional dependency, the description which conerns employees, their skills and skill levels [10] which is shown in Fig. 25, is added to the description of S which is given in Fig. 14. Fig. 25 represents the semantic information. "An employee may hold several skills (represented by the skill number $S\#$ and its skill name SN) and a particular skill may be held by several employees. A particular employee has different skill levels (represented by SL) for the skills he holds. A particular skill is held by several employees each of whom may have a different skill level for the skill in question." It is clear that SL is fully functionally dependent on the concatenated key $E\#.S\#$. As shown in Fig. 26, by creating the relationship set EMP-SKILL, the fully functional dependency of SL on $E\#.S\#$ can be expressed exactly and implicitly in the entity-relationship diagram. For the diagram shown in Fig. 26, the corresponding relations are given in Figs. 27 and 28.

2) *In Constructing the User Schema in Third Normal Form:* Consider the regular relationship relation DEPT-MANAGER. The corresponding relation scheme (in the relational model) is $S2(\underline{D\#}, \underline{M\#}, CT)$, in which only $D\#$ or $M\#$, but not both, is to be selected as the primary key because of the fact that the mapping between DEPT-NO/DEPARTMENT and EMP-ID/MANAGER is a one-to-one correspondence [10]. This is one of the major reasons that the attribute INVOLVED-PROJ-TYPE which maps relationships into the value set CT is kept away from the elimination. However, it is true that if both $D\#$ and $M\#$ are primary keys, the $S2$ is no longer a relation of third normal form, because CT is not fully functionally dependent on $(D\#, M\#)$. That means, the regular relationship relation DEPT-MANAGER is not similar to a $3NF$ relation, even if the mapping between DEPT-NO/DEPARTMENT and EMP-ID/MANAGER is one-to-many, many-to-one, or many-to-many.

Consider again the entity-relationship diagram given in Fig. 19, in which the attribute INVOLVED-PROJ-TYPE maps relationships in the relationship set DEPT-MANAGER into the value set CT. That means, each of the entity pairs of the entity sets DEPT and EMP/MANAGER is involved in work on either government or nongovernment contracts, not both. Since the relationships DEPT-MANAGER over the entity sets DEPT and EMP/MANAGER are one-to-one correspondence, in order to conform to the given semantic information, it suffices to define the attribute INVOLVED-PROJ-TYPE mapping the entity set DEPT into the value set CT, as shown in Fig. 29. One of the major differences of this connection of the value set CT to the entity set EMP from that in Figs. 18 and 19 is that the latter

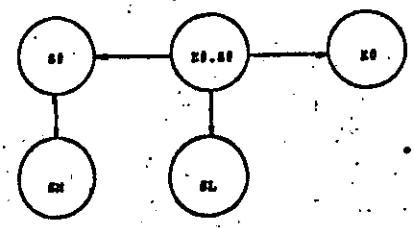


Fig. 25. A description concerning employees, their skills, and skill levels.

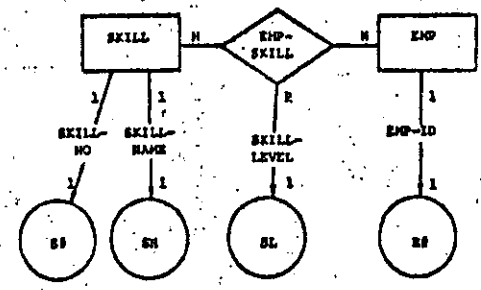


Fig. 26. An entity-relationship diagram.

r.n	SKILL	
r.a	PRIMARY KEY	ATTRIBUTE
	SKILL-NO	SKILL-NAME
	VALUE SETS	
	S#	SN
r	005 010 015.	writer printer typist

Fig. 27. A regular entity relation SKILL.

r.n	EMP					
r.a	PRIMARY KEY	ATTRIBUTES				
	EMP-ID	β	θ	ρ	γ	κ
	VALUE SETS					
	E#	EN	JC	DP	HP	CT
r	1	as	a	x	11	q
	2	bl	c	z	11	q
	3	jp	e	y	12	n
	4	sw	b	x	11	q
	5	hc	b	y	12	n
	6	qy	c	y	12	n
	7	my	a	z	13	n
	8	tm	c	z	13	n

Fig. 28. A regular relationship relation, EMP-SKILL.

connections do not satisfy the fully functional dependency, and, therefore, the problem that the regular relationship relation DEPT-MANAGER is not in $3NF$ could be avoided. Fig. 30 is a modified regular relationship relation DEPT-MANAGER, which is in $3NF$. As a matter of fact, it is a nontrivial problem of deciding what are the value sets and attributes for the entity or relationship sets from the given description of an application. For example, the value set CT and the attribute INVOLVED-PROJ-TYPE are assigned to the relationship set DEPT-MANAGER is indeed an erroneous representation.

Summarily, the user schema in the entity-relationship model, as shown in Figs. 20 and 21, Figs. 27 and 28, and Figs. 30 and 31 obviously are similar to $3NF$ relations in the relational model, but the former relations have clearer representation, and are obtained without using the normalization operations [4], [5], [9]. In the normalization of the relational model,

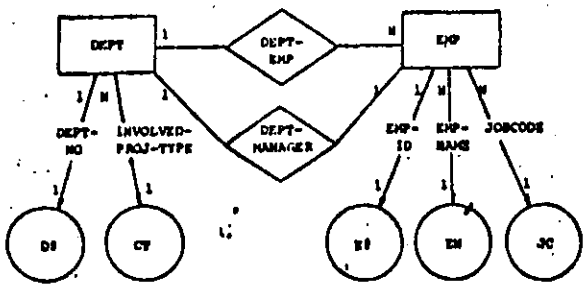


Fig. 29. An entity-relationship diagram concerning information about EMP and DEPT.

F.n	DEPT		EMP	DEPT-MANAGER
r.s	PRIMARY KEY			ATTRIBUTE
	DEPT-NO		EMP-ID	NONE
	r0	DEPARTMENT	MANAGER	
	tn	1	1	
	VALUE SETS			
	D#		E#	NONE
F	x y z		11 12 13	

Fig. 30. A modified regular relationship relation, DEPT-MANAGER.

r.n	DEPT	
r.s	PRIMARY KEY	ATTRIBUTE
	DEPT-NO	INVOLVED-PROJ-TYPE
	VALUE SETS	
	D#	CT
F	x y z	q n m

Fig. 31. A regular entity relation DEPT.

$S1(\underline{E\#}, EN, JC, D\#)$, $S2(\underline{D\#}, M\#, CT)$, $S3(\underline{E\#}, S\#, SL)$, and $S4(\underline{S\#}, SN)$, are the fewest possible number of 3*NF* relations [4], [5], [9]. Perhaps, the severe setback for the entity-relationship approach is the tradeoff between the clearer semantic representation and the fewest possible number of relations.

Consider again the diagram given in Fig. 16. For the entity set EMP, implicitly, the value set $D\#$ is functionally dependent on $E\#$ (the primary key), but not vice versa. It is easy to show that the nonkey value set CT is transitively dependent on the primary key value set $E\#$. However, such a transitive dependency can be eliminated by introducing a relationship set DEPT-EMP, as shown in Figs. 18, 19, and 29. For each of the entity sets EMP and DEPT and the relationship sets DEPT-EMP and DEPT-MANAGER, no transitive dependency among value sets is present. Therefore, the use of the entity-relationship approach is eliminating the transitive dependency among value sets is more effective than the normalization process for the relational model. Furthermore, by the definition of attribute, the entity-relationship diagram always provides the fact that e nonkey value set is fully functionally dependent on the primary key's value set(s).

Now, we shall outline the major steps in the logical database design using the entity-relationship approach. Given a description of an application for an enterprise, we intend to find a "good" schema that describes the structure of the corresponding database. Unfortunately, the entity-relationship approach, which we shall outline as follows, is a heuristic approach. In addition, we are only considering the deterministic case, for the sake of simplicity.

The entity-relationship approach to the logical database design consists of the following major steps.

1) *Given a description of an application.* The description of a database application for an enterprise can be formulated as a set of functional relationships among application attributes. Whether the set of functional relationships among application attributes is complete, nonredundant, and minimal [4], [5] is of no concern.

2) *Identify entity sets.* From the given description of an application, identify all possible entity sets which are of interest to the enterprise.

3) *Identify relationship sets.* Relationships may exist among the entities, and can be classified by the relationship sets. In general, there are m-way relationships defined on entity sets, where m is greater than or equal to 1. Conformable to the given description of an application, identify all possible relationship sets and specify their types of mapping (e.g., one-to-one, one-to-many, or many-to-many). Eliminate the redundant relationship sets.

4) *Draw an entity-relationship diagram with entity and relationship sets.*

5) *Identify value sets and attributes.* From the given description of an application, we may decide what are the value sets and attributes for the entity or relationship sets. The functional attributes, mapping from the entity or relationship sets into the value sets or the Cartesian product of the value sets must conform to some functional dependencies among application attributes. At the end of this step, an entity-relationship diagram with the entity and relationship sets, the value sets, and the attributes are obtained.

6) *Identify primary keys for entity sets.*

7) *Check the entity-relationship diagram in conformity to the description of the application.* More specifically, does the entity-relationship diagram include all possible functional relationships among the value sets? Is the transitive dependency among application attributes "properly" represented in the entity-relationship diagram? Is every nonkey attribute value set(s) nontransitively or fully dependent on the primary key value set(s)? If the answer of any of these questions is inaffirmative, then go back to step 2).

8) *Translate the entity-relationship diagram into entity-relationship relations.* These relations are described by any one of the deterministic types: the regular or weak entity relations, or the regular or weak relationship relations.

For any entity-relationship diagram, it also can be translated into data-structure diagrams supported by the CODASYL (network) type database system or hierarchical structures supported by a hierarchical database system such as IBM's IMS [2], [3].

Although it can be translated into the relational schema supported by the relational database system, it would be easier to obtain the relational schema by transforming directly from the entity-relationship relations, described by the regular or weak entity relations, or the regular or weak relationship relations.

It should be noted that if one wants to change from one

nondeterministic relation to a deterministic relation, one would probably have to change the enterprise-user schema and then reconstruct the user schema, since the enterprise-user schema using the entity-relationship relations and the enterprise schema using the entity-relationship diagram are independent of the primary keys, the unique identification of entities, and relationships. The changes of the enterprise-user schema and enterprise schema are one-to-one corresponding, and, thus, the conformity between the enterprise schema and the description of the real world can be checked.

9) *Design record formats.*

From our discussion, clearly, we can state the following theorem.

Theorem 3: For any entity-relationship relation M, described by one of the types: the regular or weak entity relation, or the regular or weak relationship relation such that none of the nonkey attribute value set (or Cartesian product of value sets), which is fully dependent on the primary key value set(s), is transitively dependent on the primary key value set(s), then M can be transformed into a $3NF$ relation (in the relational model) that strongly preserves M under the primary keys.

VI. Physical Representation of Logical Structures

In this section, we will discuss physical representation of logical structures. In the previous sections, we are concerned with the way the database administrator or those system analysts who see the entire database and the application programmer views of data. Neither the entity-relationship diagram nor the corresponding relations reflects the way the data are stored physically. In the sequel, we will be concerned with how it is laid out on the storage units. That is, for a given overall view of the data in terms of the schemas, how do we represent this overall logical organization physically.

Given an entity-relationship diagram, it can be translated into a data-structure diagram [5], [6] by the following rules.

1) For each of the entity (relationship) sets in the entity-relationship diagram, there corresponds an entity (relationship) record type, called an owner (member) record type, in the data-structure diagram.

2) For $1:N$ binary relationships of a relationship set C defined on two entity sets A and B, in the entity-relationship diagram, two directed links, one with double arrows and the other with a single arrow, are drawn from the entity record types A and B, respectively, to the relationship record type C.

For $M:N$ binary relationships, two directed links with double arrows are drawn.

Likewise, for k-ary ($k \geq 3$) relationships, the same rules apply.

3) If the existence and identification of an entity set B are dependent on another entity set A in the entity-relationship diagram, then a path dependency from the entity record type A to the entity record type B is created in the data-structure diagram. The path dependency can be represented by a directed line with single or double arrows pointed to B depending upon whether the mapping from the entity set A into the entity set B is of one or many associations, respectively.

If only the existence dependency of B on A occurs, then the path dependency can be represented by a bidirected line, which can be obtained by adding a single arrow (pointed to A) on the other end of the directed line.

Fig. 32 is a data-structure diagram which corresponds partially to the entity-relationship diagram given in Fig. 1. The structure demonstrates the $M:N$ relationship set E and EMP-CHILD on the entity sets EMPLOYEE and CHILDREN, and the $M:N$ relationship set E and I (the existence and identification dependencies) on the entity set EMPLOYEE and PARENTS.

Clearly, the entity relationship model can be used as a tool in the structured design of databases using the network models [6]. The user first draws an entity-relationship diagram and then may simply translate it into a data-structure diagram using the rules specified above.

In the remaining section, we will use the entity-relationship diagram shown in Fig. 1 to demonstrate the design of the record formats, and the use of pointers for representing associations between records or segments and associations between data.

In the course of constructing the physical representation of the entity-relationship diagram, the corresponding data-structure diagram and the corresponding relations can be used as major components of an integrated design tool to assist in the physical database design process. However, it suffices to use the entity-relationship diagram and/or the corresponding relations which are described by one of the types given in the previous section as a design tool in designing the physical database organization.

The data-structure diagram such as that in Fig. 32, can be represented physically in a variety of ways. Figs. 33-35 illustrate one of these ways, which involves parent, child, twin, and cousin pointers. Fig. 34 is a physical representation of the data-structure diagram of the record types PROJ-EMP, EMPLOYEE, and PROJECT. In the relationship record type PROJ-TYPE, two parent pointers are used from each relationship record to that record's parents, which are of entity record types EMPLOYEE and PROJECT. From these two parent pointers, the primary keys of each of the relationships can be accessed directly from the entity record record types EMPLOYEE and PROJECT.

For the existence and identification dependencies of the entity set PARENTS on the entity set EMPLOYEE, as shown in Fig. 34, a relationship record type is used to represent physically the existence dependency and a cousin pointer from each instance of the entity record type PARENTS to the instance of the subordinate entity record type is used to represent the identification dependency. That is, each instance of the entity record type PARENTS is identified by the combination of the primary keys, one from the EMPLOYEE and the other from itself, PARENTS, as shown in Fig. 12. If only the existence dependency occurs, then the cousin pointer will not be used.

In Fig. 35, all the instances of the relationship record type MARRIAGE contain two physical parent pointers. However, both parent pointers of each of the instances are pointing to the different instances of the same entity record type PARENTS, since the relationship set MARRIAGE is defined on a singleton set, the entity set PARENTS. Again, each of the relationships is identified by the combination of the primary keys, one from

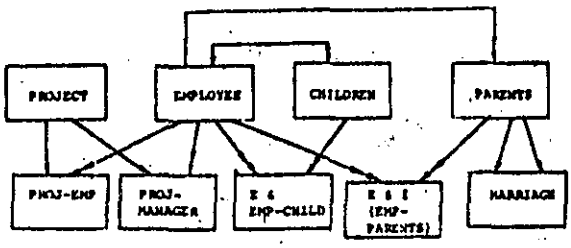
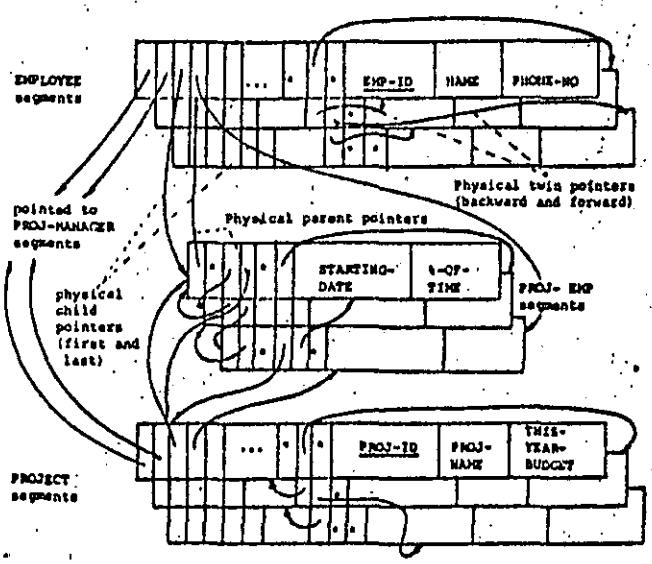


Fig. 32. A logical data-structure which corresponds partially to the entity-relationship diagram given in Fig. 1.



33. A physical representation of the data-structure diagram of the record types PROJ-EMP, EMPLOYEE, and PROJECT.

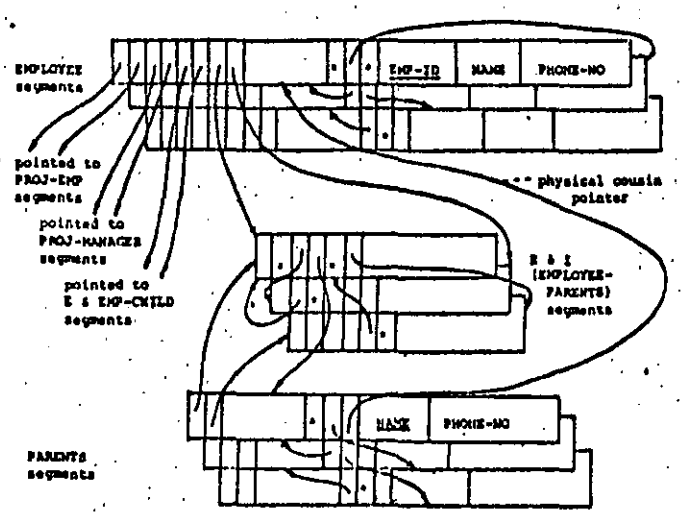


Fig. 34. A physical representation of the data-structure diagram of the record types *E* and *I* (EMPLOYEE-PARENTS), EMPLOYEE, and PARENTS.

the entity set EMPLOYEE and the other from itself, PARENTS, as shown in Fig. 13, the weak relationship relation PARENTS-MARRIAGE-RECORD.

VII. Conclusion

In this paper, we have defined formally the entity-relationship model in terms of the entity-relationship diagram as the enterprise schema, the entity relationship relations as the enterprise-user schema, and four types of relations as the user schema. We have described the entity-relationship approach to logical database design. The fact that the nondeterministic

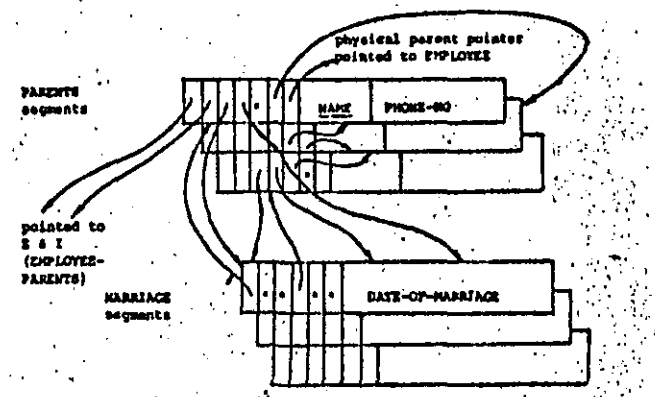


Fig. 35. A physical representation of the data-structure diagram of the record types MARRIAGE and PARENTS.

and deterministic have the same strength to characterize information about the real world enhances the entity-relationship approach because the database designer is not constrained by the functional attributes to identify the properties of entities and their relationships.

The entity-relationship approach which provides an easy to understand yet comprehensive methodology for the logical database design independent of storage and efficiency considerations is to add to the logical database design process, an intermediate stage between information about entities in the real world and the user schema, the final product of the process. The product of this intermediate stage is the enterprise schema, the description of an enterprise view of data in the real world in terms of entity-relationship diagrams. In this paper, we have explored that the entity-relationship diagram can be used as an effective tool in describing exactly and concisely the properties of entities and their relationships, in eliminating the redundant relationships among the entities and in constructing the user schema in third normal form without using the process of normalization. In addition, we have presented an improved table form of the relations as the user schema to provide a clear, and concise, and better organized user's view of databases. It is easy to understand and to modify.

Finally, we have demonstrated the process of physical representation of logical structures. The construction of the soundness and completeness of an entity-relationship approach to the logical database design requires further research. It seems desirable that a theoretical study of properties of the model be conducted in the framework of formalism.

Acknowledgment

The author is grateful to P. P. S. Chen and R. T. Yeh for their comments on an earlier version of this paper.

References

[1] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, pp. 377-387, June 1970.

[2] P. P. S. Chen, "The entity-relationship model: Towards a unified view of data," *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9-36, 1976.

[3] P. P. S. Chen, "The entity-relationship approach to logical database design," Q.E.D. Information Science, Inc., Wellesley, MA, 1977.

[4] P. A. Bernstein, "Synthesizing third normal form relations from functional dependencies," *ACM Trans. Database Syst.*, vol. 1, no. 4, pp. 277-298, 1976.

[5] C. Beeri and P. A. Bernstein, "Computational problems related to the design of normal form relational schemas," *ACM Trans. Database Syst.*, vol. 4, no. 1, pp. 30-59, 1979.
[6] C. J. Date, *An Introduction to Database Systems.* Reading, MA: Addison-Wesley, 1975.
[7] R. Fagin, "The decomposition versus synthetic approach to relational database design," in *Proc. 3rd Int. Conf. VLDB*, Tokyo, Japan, October 1977, pp. 441-446.
[8] C. Delobel and R. G. Casey, "Decomposition of a database and theory of boolean switching functions," *IBM J. Res. Develop.*, vol. 17, pp. 374-386, Sept. 1972.
[9] E. F. Codd, "Further normalization of the database relational model," in *Data Base Systems, Courant Computer Science Symposia Series*, vol. 6. Englewood Cliffs, NJ: Prentice-Hall, 1972, pp. 33-64.
[10] M. Vetter, "Database design by applied data synthesis," in *Proc. 3rd Int. Conf. VLDB*, Tokyo, Japan, Oct. 1977, pp. 428-440.
[11] Y. E. Lien, "On the semantics of the entity-relationship model," in *Entity-Relationship Approach to Systems Analysis and Design*, P. P. S. Chen, Ed. Amsterdam, The Netherlands: North-Holland, 1980, pp. 155-168.
[12] M. A. Melkanoff and C. Zaniolo, "Decomposition of relations and synthesis of entity-relationship diagrams," in *Entity-Relationship Approach to Systems Analysis and Design*, P. P. S. Chen, Ed. Amsterdam, The Netherlands: North-Holland, 1980, pp. 277-294.
[13] H. Sakai, "A unified approach to the logical design of a hierarchical data model," *Entity-Relationship Approach to Systems Analysis and Design*, P. P. S. Chen, Ed. Amsterdam, The Netherlands: North-Holland, 1980, pp. 61-74.
[14] P. A. Ng and J. F. Paul, "A formal definition of entity-relationship model," *Entity-Relationship Approach to Systems Analysis and Design*, P. P. S. Chen, Ed. Amsterdam, The Netherlands: North-Holland, 1980, pp. 211-230.
[15] *Entity-Relationship Approach to Systems Analysis and Design* P. P. S. Chen, Ed. Amsterdam: The Netherlands: North-Holland, 1980.

Peter A. Ng received the B.Sc. degree in mathematics from St. Edward University, Austin, TX, in 1969, and the Ph.D. degree in computer sciences from the University of Texas, Austin, in 1974.

He served two years as Assistant Professor of Computer Sciences at Hunter College, City University of New York, New York, NY, from 1975 to 1976. In August 1976 he joined the Department of Computer Science, University of Missouri, Columbia, where he is presently an Associate Professor. His current major research interests are information systems, database management system, data communications, and several aspects of software engineering. He has published over 30 technical papers on these fields in international journals and conference proceedings. He is an industrial Consultant on these areas.

Dr. Ng is a member of the Association for Computing Machinery and the IEEE Computer Society.

Mapping Considerations in the Design of Schemas for the Relational Model

SABAH AL-FEDAGHI AND PETER SCHEUERMANN

Abstract—The typical design process for the relational database model develops the conceptual schema and each of the external schemas separately and independently from each other. This paper proposes a new design methodology that constructs the conceptual schema in such a way that overlappings among external schemas are reflected. If the overlappings of external schemas do not produce transitivity at the conceptual level, then with our design method, the relations in the external schemas can be realized as a join over independent components. Thus, a one-to-one function can be defined for the mapping between tuples in the external schemas to tuples in the conceptual schema. If transitivity is produced, then we show that no such function is possible and a new technique is introduced to handle this special case.

Index Terms—Conceptual schema, external schema, functional dependencies, independent components, interferences, logical database design, mapping functions, relational database model.

Manuscript received February 1, 1980; revised June 11, 1980. This work was supported in part by the National Science Foundation under Grant MCS77-03904.

The authors are with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201.

I. Introduction

IN THE RELATIONAL database model, a given external view V_k and the conceptual view V are described by sets of relations. The set of relations in the conceptual view constitutes the *conceptual schema* and the set of relations in a given external view constitutes an *external schema*. Given a relational model let $E = \{E_1, E_2, \cdots, E_n\}$ be the set of the external schemas and C be the conceptual schema of the model. The mapping problem covers several transformations between a given $E_k \in E$ and C. Let $E_k = \{e_1^k, e_2^k, \cdots, e_v^k\}$ and $C = \{c_1, c_2, \cdots, c_u\}$ where e_i^k, $1 \leq i \leq v$, is a relation in E_k and c_j, $1 \leq j \leq u$, is a relation in C. Several mappings are of interest.

1) *Relations Mapping:* $\alpha_1(e_i^k) = C' \subseteq C$. That is, α_1 constructs the relation $e_i^k \in E_k$ from the relations C' in C.

2) *Tuples Mapping:* $\alpha_2(t(e_i^k))$, where $t(e_i^k)$ is a tuple in e_i^k. The function α_2 maps a tuple of e_i^k to a tuple or a set of tuples in $C' \subseteq C$.

0098-5589/81/0100-0099$00.75

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

DISEÑO DE BASES DE DATOS

A R T I C U L O

BEHAVIORAL AND ORGANIZATIONAL CONSIDERATIONS IN THE DESIGN OF INFORMATION SYSTEMS AND PROCESSES FOR PLANNING AND DECISION SUPPORT

Expositor:
Ing. Daniel Ríos Zertuche

MAYO, 1985

Behavioral and Organizational Considerations in the Design of Information Systems and Processes for Planning and Decision Support

ANDREW P. SAGE, FELLOW, IEEE

Abstract—Determinants of performance of systems and processes for planning and decision support are discussed. This paper is directed at people who design such systems and processes, who use such systems and processes, and who manage organizations in which these may be used. The literature cited is associated with several areas including psychology, organizational behavior and design, information science, management science, computer science, and related disciplines. Performance determinants and design requirements for systems and processes for planning and decision support are especially stressed. A number of areas where additional research appears needed are mentioned, and some recommendations and interpretations are given concerning both contemporary efforts and needed future efforts.

I. Introduction

THAT there is much interest in planning and decisionmaking efforts to determine effective public and private sector policies is made evident by the number of recent texts and case studies devoted to these topics [2], [4], [13], [18], [20], [21], [44], [45], [48], [51], [80], [84]–[86], [89], [104], [105], [108], [134], [135], [139], [141], [150], [178], [179], [198], [212], [219]–[222], [237], [243], [283], [293], [318]–[320], [334], [359], [361], [363], [377], [394], [397], [398], [400], [412]. These in part, concern the numerous complexities associated with practical implementation of the results of systemic efforts for planning and decision support. Advances in digital computer technology coupled with advances in systems science, systems methodology and design, and systems management suggest extension of the information analysis and display capability provided by management information systems to include interpretation and aggregation of information and values such as to result in decision support systems (DSS) or planning and decision support systems. There is a growing literature in this area [5], [36], [39]–[41], [76], [86], [110], [133], [138], [224], [226], [227], [239], [240], [258], [309], [350], [356], [366], and this indicates much contemporary interest and activity.

There are a number of requirements for design success with respect to systems for planning and decision support. These involve a considerable number of disciplines. The result of not making appropriate use of pertinent contributions from a number of disciplines in the design of systems for planning and decision support is likely to be a system or process that is deficient in one or more important ways. The purpose of this effort is to discuss, from a systems engineering perspective, some of the many requirements for design success in this area.

It is possible to disaggregate planning and decisionmaking processes into a number of steps. In essence, they are purposeful futuristic efforts which involve the entire systems engineering process [301]–[305], [307], [308] and can, therefore, be described by any of a number of frameworks for systems engineering such as the three-or the seven-step framework which involves:

1) Formulation of the issue.
   a) problem definition (determination of needs, constraints, alterables)
   b) value system design (determination of objectives and objectives measures)
   c) system synthesis (identification of possible decisions or action alternatives and measures of the accomplishment of these);
2) Analysis of the issue.
   d) systems analysis and modeling (determination of the structure of the decision situation, the impacts of identified decisions or action alternatives and the sensitivity of these to possible change in conditions)
   e) optimization or refinement of alternatives (adjustment of parameters or activities such that each identified decision is the best possible in accordance with the value system);
3) Interpretation of the issue
   f) evaluation and decisionmaking (each possible decision alternative is evaluated, prioritized, and one or more alternatives are selected for implementation action)
   g) planning for action (commitment of resources are made and implementation is accomplished).

Janis and Mann [177] have identified a four-stage model of the decisionmaking process. Fig. 1 presents a slightly modified version of this decision process model. We note that it contains the same essential steps involved in the systems engineering process. Of particular interest are the questions asked at each step of the process. We will elaborate upon this model and other models of the decisionmaking process in our efforts to follow.

Comprehensive efforts involving decisionmaking will be complex because of the many disciplines and areas involved as well as because of the subject matter itself. Probably the formal study of decisionmaking first began with the rational economic man concepts of the 18th century mathematicians Cramer and Bernoulli who explained the St. Petersburg paradox. Since then there have been many workers from a large number of disciplines who have been concerned with various types of decisionmaking studies and the provision of assistance to enhance the understanding of rationale for plans and decisions as well as improvements in the efficiency, effectiveness, and equity of the resource allocations that constitute planning and decisionmaking.

Contemporary choicemaking issues in the public and private sector are complex, contain much uncertainty, and require inputs from many sectors for full understanding and resolution. Many writers have indicated bounded rationality limits in decisionmaking that would appear to make provision of information system

Manuscript received April 1, 1981; revised June 21, 1981. This work was supported in part by the U.S. Office of Naval Research under Contract N0014-80-C-0542.

A. P. Sage is with the Department of Engineering Science and Systems, University of Virginia, Charlottesville, VA 22901.

0018-9472/81/0900-0640$00.75 ©1981 IEEE

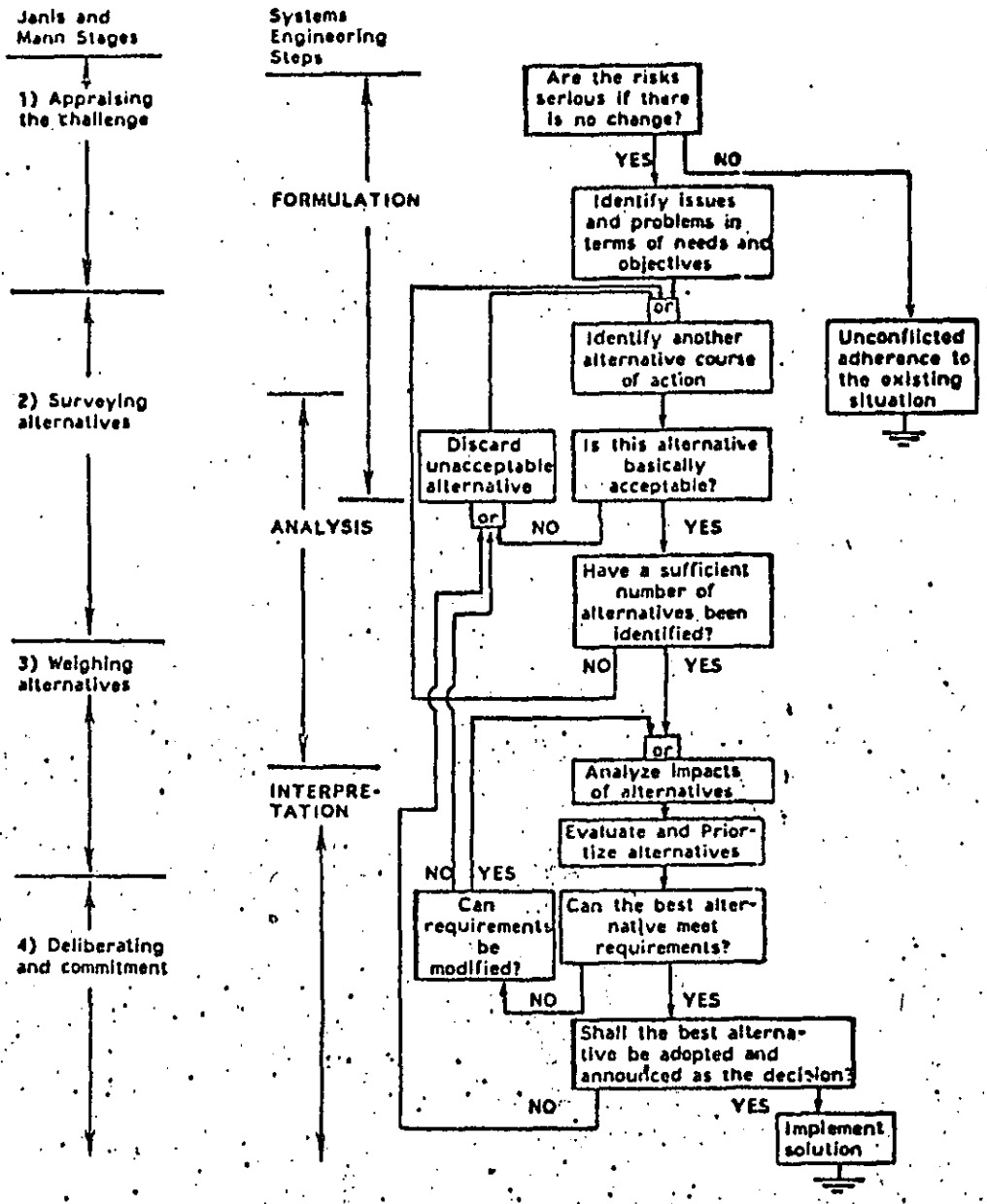


Fig. 1. Systems engineering interpretation of the decision process model of Janis and Mann.

adjuvants for choicemaking normatively very desirable. Such planning and decision support systems could, in principle, provide decisionmakers with rapid access to the information and knowledge needed to enhance decision quality. Unfortunately this promise of enhanced decision quality has not always been realized in practice. There are, doubtlessly, a number of causitive factors inhibiting the potential benefits possible from information systems for planning and decision support. Principal among these factors which, at present, pose fundamental limits to information system success appear to be

1) the need to insure substantive or input-output rationality, such that evaluations of plans and decisions are veridical;
2) the need to insure process rationality, such that the information system accommodates the capabilities of, and the constraints placed upon, the user;
3) the need to understand and cope with human cognitive limitations as they affect the formulation, analysis, and interpretation of decision situations and alternatives; and
4) the need to understand and integrate the normative or prescriptive components with the descriptive components of decision situations in order to evolve realistic adjuvants for the formulation, analysis, and interpretation of decision options.

This paper presents a survey, status report, and integration and interpretation of research from a diversity of areas that supports the design of information systems capable of coping with the needs and fundamental limits to improved judgment. We discuss and describe

1) the cognitive styles of decisionmakers;
2) individual human information processing in decision situations and biases in the acquisition, analysis, and interpretation of information;
3) decision rules for individual decision situations;
4) contingency task structural models of decision situations; and
5) decisionmaking frameworks, organizational settings, and information processing in group and organizational decision situations.

In a very real sense the structural models section, Section V, is the principal portion of this effort. It contains the basic decisionmaking paradigm, including action selection. The contingency task structure, which comprises the issue at hand, the environment into which the issue is imbedded, the decisionmaker, and decisionmaker experiential familiarity with the issue and environment, is the determinant of cognitive style and perfor-

mance objectives for the particular task at hand. These, in turn, influence selection of an information processing approach and selection of a decision rule.

The literature in this area is enormous. But there is the need for efforts to integrate it from the perspective of systems engineering design of information systems for planning and decision support. There are a number of recent surveys available that discuss one or a limited number of the topics important for the design of planning and decision support systems. These include the surveys of Barron [27], Benbasat and Taylor [35], Bettman [37], Craik [67], Dunnette [87], Einhorn, Kleinmuntz, and Kleinmuntz [95]; Einhorn and Hogarth [98], Ericsson and Simon [99], Hammond, McClelland, and Mumpower [142]; Hammond [143], Hogarth [159], Hogarth and Makridakis [161], Johnson and Huber [179], Kassin [190], Keen [193], Libby and Fishburn [214], Libby and Lewis [215], Mintzberg [248], Nisbett and Ross [264], Nutt [266], Robey and Taggart [292], Sage and White [304], Schneider and Shiffrin [313], Slovic and Lichtenstein [345], Slovic, Fishhoff, and Lichtenstein [348]; Svenson [365], and Zmud [415]. This work attempts a selective integration of this voluminous literature and extensions and interpretations of it from the perspective of ultimate potential usefulness for the design of information systems for planning and decision support. Generally, references are provided only to published literature of the last half decade with limited references to earlier seminal literature and reports. This was felt desirable in order to limit the reference list to an almost manageable size. Despite our attempt to make this report comprehensive, it doubtlessly fails to incorporate the important contributions of a number of authors. And there are doubtlessly unintentional misattributions and misinterpretations as well. For this apologies are offered and forgiveness requested.

II. Cognitive Styles

It is becoming increasingly clear that it is necessary to incorporate not only problem characteristics, but also problem solver or decisionmaker characteristics, into the design of information systems for planning and decision support. A deficiency in some past designs has been the neglect of the human decisionmakers' role and characteristics and their effects. Essentially all available evidence suggests that problem characteristics and user characteristics influence the planning and choice strategies adopted by the decisionmaker. This section discusses a number of cognitive style models from these perspectives.

Mason and Mitroff [238] have suggested that each person possesses a particular specific psychological cognitive style or "personality" and that each personality type utilizes information in different ways. In their research on (MIS) management information system design, they claim that an information system consists of a person of a certain psychological type who faces a problem in some organizational context for which needed evidence to arrive at a solution is made available through some mode of presentation.

There are five essential variables in the information system characterization of Mason and Mitroff. Each of these are disaggregated into subelements. Mason and Mitroff characterize the *psychological-type variable* according to the Jungian stereotypology. In this typology, people differ according to their preference for information acquisition and analysis and the preferred approach to information evaluation and interpretation. At extremes in the information acquisition dimension are sensing-oriented or sensation types who prefer detailed well-structured problems and who like precise routine tasks, and intuitive-oriented type people, who dislike precise routine structured tasks and perceive issues wholistically. At extremes in the information evaluation dimension are feeling-oriented people, who rely on emotions, situational ethics, and personal values in making decisions; and thinking-oriented individuals, who rely on impersonal logical arguments in reaching decisions.

Mason and Mitroff characterize the *problem variable* into structured and unstructured problems. These may be further divided into decisions under certainty, decisions under risk, and decisions under uncertainty. The *organizational context variable* is characterized as strategic planning, management control, and operational control. The *method-of-evidence-generation variable* involves five types of inquiry systems: the data-based Lockean inquiry system, the model-based Leibnitzian inquiry system, the multiple model-based Kantian inquiry system, the conflicting model-based Hegelian inquiry system, and the learning system based Singerian-Churchmanian inquiry system [254]. A fifth variable, *mode of presentation*, includes personalistic modes of presentation such as one-on-one contact as in drama and art and impersonalistic modes such as abstract analytical models and company reports. These latter four variables do not formally relate to cognitive styles, and some further comment on them is contained in other portions of our effort. A number of works by Mason and Mitroff and their colleagues discuss various aspects of this categorization [252]–[254]. Of interest in this regard is a work by Kilmann [200] which suggests the design of organizations with the Jungian personality characteristics of individuals.

Among the many other studies which have emphasized the need to incorporate decisionmaker characteristics into information system design is that of Doktor and Hamilton [78]. They studied the influence of cognitive style on the acceptance of management science recommendations and found a strong correlation between the decisionmaker's cognitive style and willingness to accept these recommendations. They found that differences in acceptance rates were due not only to differences in cognitive style but also to differences in this subject population. From this and many other investigations [34], [74], [77], [79], [101], [151], [166], [174], [229], [252], [253], [263], [267], [268], [311], [330], [369], it appears that appropriate consideration of the human behavioral variable of cognitive style is very necessary for successful design of decision support systems.

A number of studies such as those by Taylor [369], Craik [67], Payne [272], Schneider and Shiffrin [313], [327] and Simon [342], indicate, as we will discuss in later sections, that human decisionmakers attempt to bring order into their information processing activities when confronted with excess information or the lack of sufficient information. Many early studies assumed that static fixed patterns of dealing with information were "preferred" by the decisionmaker for the process of experiencing the world, and these were referred to as "cognitive style." Some early studies view cognitive style as a mode of functioning that is static and pervasive throughout a person's perceptive and intellectual activities. A number of intellectual processes are subsumed within the term cognitive style. These concern the way in which information is acquired or formulated, analyzed, and interpreted. Thus, cognitive style includes such human activities as information filtering and pattern recognition.

Zmud has indicated [414], [415] that those individual differences which influence information system success most strongly involve cognitive style, personality, and demographic/situational variables. Cognitive style refers to the process behavior that individuals exhibit in the formulation or acquisition, analysis, and interpretation of information or data of presumed value for decisionmaking. Doubtlessly cognitive style is somewhat influenced by such personality variables as dogmatism, introversion, extroversion, and tolerance for ambiguity. However, little appears known concerning these influences. Gough discusses personality and personality assessment in his chapter [87]; but it is rare to find, with some notable exceptions [249]–[251], [330], [332], [352], discussions of personality effects upon decisionmaking behavior in cognition studies. The demographic/situational variables involve personal characteristics such as intellectual ability, education, experience with and knowledge of specific contingency tasks, age, and the like. An important situational variable is the level of stress encountered by the decisionmaker in a

specific problem situation. The level of stress, which results in the adoption of a coping pattern, influences the decisionmaker's ability in acquisition and processing of the information necessary for decisionmaking. The subject of stress will be dealt with in some detail in Section V. Many variables are especially important for an information processing model of cognitive behavior. Some will be discussed in Section III. Our efforts in this section will be devoted primarily, therefore, to cognitive style concepts, especially the role of personality variables in the adoption of cognitive style.

There are a number of cognitive style models in addition to that of Mason and Mitroff; Bariff and Lusk [24], for example, have discussed three cognitive style characteristics relevant to information system design: cognitive complexity, field dependent/independent, and systematic/heuristic. The cognitive complexity characteristic involves three structural characteristics of thinking and perception: differentiation, the number of dimensions sought or extracted and assimilated from data discrimination; the fineness of the articulation process in which stimuli are assigned to the same or different categories; and integration, the number and completeness of interconnections among rules for combining information.

Benbasat and Taylor [35] note that much cognitive complexity research deals with interpersonal perception and has limited value for modeling activities of managers in processing information and making decisions. Mischel is especially perceptive in discussing the potential hazards of attributions and enduring categorizations of people into fixed slots on the basis of a few behavioral signs in his study of the interface between cognition and personality [251]. The assumptions that static characterizations are sufficiently informative to enable behavior predictions in specific settings are strongly challenged. An evaluation of the uses and limitations of static trait characterization of individuals is presented and the strong interacting role of context is emphasized. Mischel is especially concerned with "cognitive economics," that is to say the recognition that people are easily overloaded with an abundance of information and that simplified methods of acquisition and processing of information are, therefore, used. He is especially concerned also with growth of self-knowledge and rules for self-regulation with maturation, topics to be discussed in Section V. We concur with these views in that we believe that it is the individual's experience with the task at hand that is the primary determinant of cognitive style. Further we believe that it is an individual's information processing capacity under various levels of stress and in different contingency task structures that determines, in part, the quality of decisionmaking. These factors depend strongly upon experience. Thus we support the information processing view of Simon [337]–[344] that few characteristics of the human information processing system are invariant over the decisionmaker and the task. These characteristics are generally experiential and evolve over time in a dynamic fashion. They are not static and can not be treated as static and task invariant for a given individual.

In the Bariff and Lusk cognitive style model [24], individuals may be categorized according to whether they are tightly bound by external referrents in structuring cognitions, in which case they are called field dependent or low analytic; or whether they can make use of internal referrents as well as external referrents in structuring cognitions, in which case they are high analytic or field independent. In a field-dependent mode, perception is dominated by the overall organization of the field. There is limited ability to perceive discrete parts of a field, especially as distinct from a specific organized background. Field independent people have more analytical and structuring abilities in comparison to field dependent people in that they can disaggregate a whole into its component parts.

The systematic-heuristic categorization of Bariff and Lusk describes cognitive styles associated with people who either search information for causal relationships that promote algorithmic solutions, or who search information by trial and error hypothesis testing. Systematic individuals utilize abstract logical models and processes in their cognition efforts. Heuristic individuals utilize common sense, past experience, and intuitive "feel." Systematic individuals would be able to cope with well-structured problems without difficulty and would approach unstructured problems by attempting to seek underlying structural relations; whereas heuristic individuals would attempt to cope with unstructured problems without a conscious effort to seek structural identification.

Of particular importance with respect to cognitive styles are relationships between the environmental complexity of the contingency task structure and information processing characteristics. A number of authors have attempted experiments based on the hypothesis that the conceptual structure of the individual determines information processing characteristics. Conceptual structure is typically measured on a dimension of abstract versus concrete. Abstract individuals would be capable of using integratively more complex conceptual processes than concrete type individuals. Abstractness may be characterized by the ability to differentiate a greater variety of information and to discriminate and integrate information in complex ways. Abstract individuals would, therefore, be expected to base actions on more information and to develop more complex strategies for information evaluation than concrete individuals. This is somewhat similar to Piaget's account of evolving cognitive development,[1] in that the "formal" thinker is capable of abstract thought whereas the "concrete" thinker relies more on preceptual experience as a basis for thought and problem solution. While the work of Piaget appears to assume that cognitive capacity evolves over time, some research involving personality and cognitive style assumes that an individual's cognitive style is not task dependent and not subject to change as a function of contingency variables such as experience.

Among other efforts, Driver and Mock [83] developed decision-style theory, a set of four decision styles based upon the heuristic–analytic characterization of Huysman [172] to relate conceptual structure of decisionmakers to both the amount of information they tend to use and the degree of focusing that they exhibit in the use of information. A heuristic person will use intuition, past experience, concrete thought, and a wholistic approach to reach decisions. An analytic person will utilize abstract logical models and will search for causal relationships and underlying structure to evolve rationale for decisionmaking. The four decision styles are determined by the degree of focus in the use of information and the amount of information desired. A decisive person is one who wishes to see the minimum possible amount of information and who will likely identify a single workable decision. Decision speed obtained from short summary, often verbal reports, is a characteristic of the decisive person. A flexible person is one who utilizes minimum information but who will identify a number of potentially acceptable decisions. A hierarchic person is one who utilizes much information, often obtained in a thorough way from long involved precise reports to identify a single acceptable decision. An integrative person utilizes much information to identify a number of potentially acceptable decisions.

Vasarhelyi [389] has also examined the analytic–heuristic dimension. His experimental results indicate generally that analytic type people tend to use computers and other analytic tools more in planning than do heuristic types. Heuristic types use less information than the analytic types and are more concerned with the lack of flexibility in computers than analytic types. However, his study of correlations among various style-measuring instruments indicates that these are relatively low.

Driver and Mock also suggested a fifth style which they referred to as the complex style, which is characterized by a wide search and analysis of information. It is a mixture of the integra-

[1] See Section V of this paper.

TABLE I
FOUR MODELS OF COGNITIVE STYLE

BARIFF AND LISK [24]	DRIVER AND MOCK [83]
COGNITIVE COMPLEXITY • DIFFERENTIATION • DISCRIMINATION • INTEGRATION	DEGREE OF FOCUS IN USE OF INFORMATION • MULTIPLE SOLUTIONS IDENTIFIED • ONE SOLUTION IDENTIFIED
FIELD INDEPENDENT/DEPENDENT SYSTEMATIC/HEURISTIC	AMOUNT OF INFORMATION USED • MAXIMUM • MINIMUM
MCKENNEY AND KEEN [242]	MASON AND MITROFF [238]
INFORMATION ACQUISITION • RECEPTIVE • PRECEPTIVE	INFORMATION ACQUISITION • INTUITIVE • SENSING
INFORMATION EVALUATION AND INTERPRETATION • SYSTEMATIC • INTUITIVE	INFORMATION EVALUATION AND INTERPRETATION • THINKING • FEELING

tive and hierarchic types. Zmud [415] has performed some experimental studies of this decision style theory. His findings indicate that perceptual differences can indeed be observed for specific cognitive styles and among subjects with different educational and experiential backgrounds. However, his results also indicate that there is no apparent relationship between cognitive style perceptions and actual cognitive behavior despite consistent differences in perceptions of cognitive styles.

McKenney and Keen [242] have done extensive work on cognitive style measurements. These have become, in part, the basis for several definitive efforts [192], [193], [258] in decision support system design. They conceptualize cognitive style in two dimensions: information acquisition and information processing and evaluation. The information acquisition mode consists of receptive and preceptive behavior, both at the opposite extremes of a continuum. They claim that preceptive decisionmakers use concepts, or precepts, to filter data, to focus on patterns of information, and to look for deviations from or conformities with their expectations. Receptive people tend to focus on detail rather than patterns and derive implications from data by direct observation of it, rather than by fitting it to their own precepts.

With respect to information processing and evaluation, McKenney and Keen measured individuals on a scale, with the systematic thinker at one extreme and the intuitive thinker at the other extreme. They have shown, using a battery of pencil and paper tests, that systematic thinkers approach a problem by structuring it in terms of some method which would lead to a solution, whereas intuitive thinkers use trial and error, intuition, and previous experience to obtain solutions.

We have examined four cognitive style characterizations in this section. Table I summarizes the models of cognitive style that result from these efforts. We note the considerable similarity among these four constructs. There have been a number of studies of the measuring instruments involved in classifying people according to these cognitive styles. Many, such as the study by Vasarhelyi [389] mentioned previously, have found rather low correlations among test instruments. Zmud [413], [415] has indicated low correlation also among test scores on different instruments indicating cognitive styles. Chervany, Senn, and Dickson [57], [76] have expressed much concern and pessimism concerning the validity of much of the contemporary research in this area. They comment that the study of individual personality differences as predictors of human behavior and performance have been basically unsuccessful in that it has not been possible to predict performance on the basis of personality characteristics. Their comment and the comment of others that the characteristics of the task in which the individual involved is a prime determinant of human behavior appears unassailable. We will provide and discuss additional evidence supporting a dynamic cognitive style characterization that will incorporate the contingency task structure and the decisionmaker's task experience in several other sections of this paper. In particular we emphasize the strong need for consideration of the structure and the content of planning and decision situations in order to evolve contextually meaningful support.

III. Information Processing

Problem solving, judgment, and decisionmaking imply both thought and action. Hence decisionmaking can be defined as the processes of thought and action involving an irrevocable allocation of resources that culminates in choice behavior. In making a decision, more often than not, the decisionmaker is dealing with environments characterized by risks, hazards, uncertainty, complexity, changes over time, and conflict. Further, the quality of a decision depends upon how well the decisionmaker is able to acquire information, to analyze information, and to evaluate and interpret information such as to discriminate between relevant and irrelevant bits of data. Decision quality also depends upon how well the decisionmaker is able to cope with stress, which is invariably encountered in important decision circumstances. Effective management of these factors enables strategies by which the decisionmaker may arrive at a good problem solution, decision, or judgment.

A number of studies such as those by Barron [28], Bettman [37], Chorba and New [58], Delaney and Wallsten [73], Feather [107], Howell and Fleishman [165], O. Huber, [168], G. Huber [170], [171]; Ives, Hamilton, and Davis [174]; Libby and Lewis [215], Lucas and Nielson [228], MacCrimmon and Taylor [232], Montgomery and Svenson [256], Moskowitz, Schaefer, and Borcherding [259]; Payne [275], Simon [342], Tushman and Nadler [379], Tuggle and Gerwin [380], Wallsten [395], [396]; and Wright [402]–[406] discuss the vital role of human information processing in decisionmaking. Most contemporary researchers regard information processing as a crucial task for effective decisionmaking and state that the type of decision problem, the nature of the decision environment, and the current state of the decisionmaker combine to determine decision style and decision strategy for a specific task. The term information processing refers to the processing of verbal reports as well as quantitative data since verbal reports are data [99].

An information processing theory of problem solving, judgment, and decisionmaking is based on the assumption that individuals have an input mechanism for acquisition of information, an output mechanism for interpretation and choicemaking, internal processes for filtering and other analysis efforts associated with information, and memories for long- and short-term storage of information. There are a large number of ways of representing

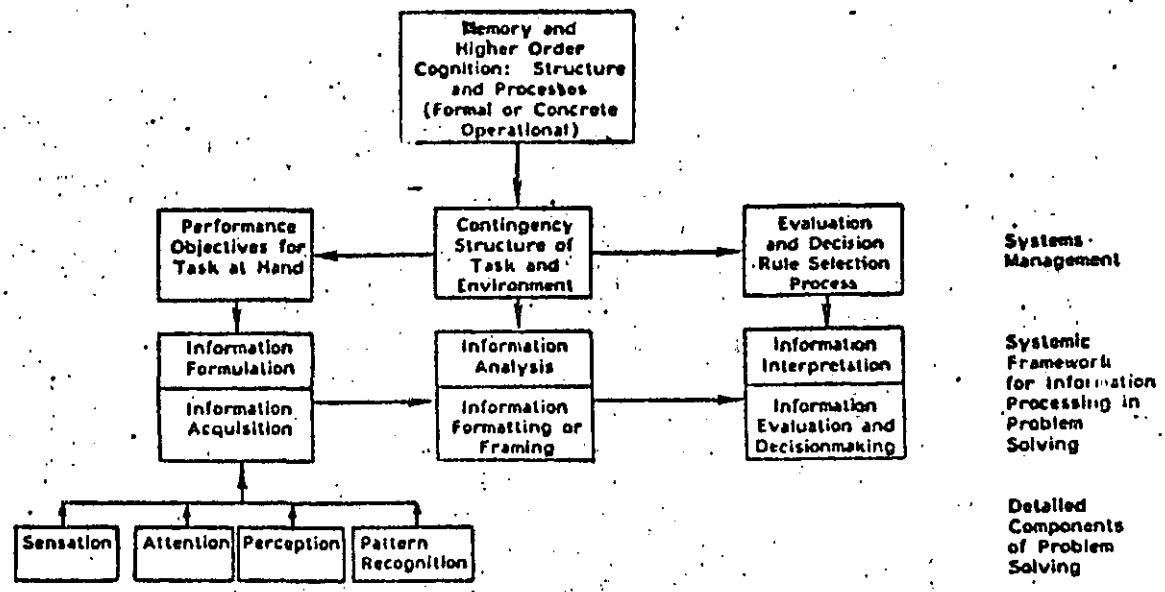


Fig. 2. A systems engineering conceptual model of human information processing.

human information processing. Many of these are described in texts in cognitive psychology such as Anderson [7], Posner [281], or Solso [354], and in works in consumer choice such as Bettman [37]. Much of the work in this area owes a great deal to Simon [334]–[344] who has developed information processing theories in psychology and in artificial intelligence.

Fig. 2 presents a conceptual model of a systems engineering framework [308] for human information processing. There are doubtlessly a number of components missing from this model. It does not show, for example, the essentially iterative nature of the process. Nevertheless we feel that it provides a useful point of departure and a structure for our efforts to follow.

The key functions, which determine how a specific problem or decision situation is cognized, depend upon an interaction of the memory and higher-order cognition of the problem solver with the environment through the contingency task structure. We will be very concerned with development of a conceptual model of higher-order cognition and the contingency task structure in Section V. It is appropriate to remark here that the various information analysis and interpretation processes of thinking, task performance objective identification, evaluation, and decision rule identification, are called "higher order" cognition. This is not because they are somehow more important than the so called "lower order" cognition efforts of information acquisition involving formulation: sensation, attention, perception, and pattern recognition; but because they occur later in time in the overall information processing effort.

It is important to note that information processing and decisionmaking efforts intimately involve memory. Memory [102] influences human judgment in a number of ways. It will influence the perception of the contingency task structure associated with an issue as well as the decision rules used for evaluation of alternatives. Two characteristics of human memory are of special importance for our efforts here. First, information will be encoded in more or less efficient and effective ways in terms of human abilities for recall. The coding process is dependent, also, upon the interpretation attached to information and this strongly influences event recall, perceptions, and associated cognitive biases. The literature concerned with memory and its components, and their relations and interaction with human perceptual experience and behavior is vast and speculative in nature. There have been many studies, both physiological and psychological, concerned with the identification of the memory "engram," which is hypothesized to be the fundamental unit of memory. We need not be especially concerned in this effort with the various physiological structures and processes associated with human memory; or with various related behavior therapies [109]; however the essentials are reviewed below briefly. A useful brief survey of the literature on memory is presented by Thomassen and Kempen in chapter 3, vol. II [244], by Fox [128], and by Radcliff [284].

Human memory constitutes two major components, short-term memory and long-term memory. Short-term memory plays a key role in immediate recall of actively rehearsed limited information [7], [354]. Unless conscious effort is put forth in recalling information from short-term memory, this cannot be done after a lapse of 30 to 60 s from initial presentation. Models of a working short-term memory involve a number of mechanisms, such as an articulatory rehearsal loop that has the capacity to retain short verbal sequences. This is just one mechanism by which short-term retention is possible. There are a number of other sensory registers. It is important to note that short-term memory is an integrated network of many mechanisms, and is associated, in use, with a number of skilled processes.

Shiffrin and Schneider [313], [327] incorporate concepts of attention, memory, and perceptual learning in their theory of short-term retention. They hypothesize short-term storage, the function of which is active control of thinking, reasoning, and general memory processes. According to Shiffrin & Schneider, short-term storage is an activated subset of long-term storage. Transfer of information from short-term storage to long-term storage is dependent on attentional limitations, interference from strong external and internal stimuli, extent of analysis of information, and formation of associations in long-term storage. There have been many studies involving concepts such as retrieval processes, memory trace identification, encoding processes, and recognition which we will not discuss as they appear of secondary importance to the goals of this particular effort. While five to seven unconnected items is believed to be the maximum amount of information that can be retained in short-term memory, long-term memory may contain a virtually limitless amount of information.

Thus we see an enormous difference between human abilities and computer abilities. Because of its large long-term memory and ability for quick search and recall, a human mind easily reasons wholistically. Wholistic reasoning, such as reasoning by analogy, is not at all easy, at this time at least, for a computer. Significant unaided computational effort would be difficult for a human since computation must be done in short-term memory. There exists the possibility that information stored in long-term memory is flawed because of cognitive-biases introduced by processing in short-term memory. A principal task of computer-aided support must be to augment human capabilities in need of

augmentation, while not diminishing abilities in those areas in which human abilities exceed those of the computer [81].

Our effort in the remainder of this section will be devoted to a description of the various processes which support information acquisition and information analysis. We will also discuss some of the cognitive biases that can result from "poor" information acquisition and information analysis. Information interpretation, which leads to alternative evaluation and decisionmaking, is an important and somewhat distinct part of the overall information processing model. It will be discussed in the next four sections from several perspectives.

The types of operations involved in information acquisition are sensation, attention, perception, and pattern recognition. Doubtlessly there are other valid ways of categorizing these operations [7], [37], [67], [100], [137], [148], [175], [281], [297], [298], [313], [327], [333] but the taxonomy used here is sufficient for our purposes. In sensation, information is acquired through the five major sense modalities, which are environmentally activated, in response to a specific array of stimulus energies. In a specific decisionmaking situation, the decisionmaker filters out bits of data believed to be irrelevant. The filtering process is based upon task characteristics, experience, motivation, as well as other features and demands of the specific decisionmaking situation. If such a filtering mechanism were not to exist, the decisionmaker would often encounter information overload which generally results in saturation and the inability to process sufficient information for the task at hand. Short-term and long-term memory components play key roles in the information acquisition process as the decisionmaker proceeds with efforts that culminate in choice. A response system couples the memory system to the sensory system and the environment. Thus it controls or activates the sensory modalities on the basis of the actions taken. Through the response system we close the information flow feedback loop. Bower, in volume 1 of Estes [109], has summarized principal components of the flow system. A model of the principal components of information flow might consist of: the response system, the sensory system, the memory system, and the central processor. The central processor coordinates memorizing, thinking, evaluation of information, and final decisionmaking.

Ultimately involved in retention processes is the notion of attention [7]. In order for information to be transferred from short-term memory to long-term memory, constant conscious attention, in terms of rehearsal, is required. Information entering short-term memory that is not attended to, through specific conscious processes, is lost. Processing of information demands attending to relevant bits of incoming data and transfer of the data into long-term memory for future retrieval for making a decision. Interferences of various types may interrupt attention and thus hinder transfer and retention of relevant stimuli into long-term memory.

Inherent in the processing of information acquisition, is the process of pattern recognition. This process generally involves two phases: extraction and identification. A given stimulus is "coded" in terms of its features. These extracted features of the object or stimulus describe the stimulus. The term "features" implies such characteristics as angles, lines, or edges. A stimulus may be received through any of the sense modalities. The meaning that this conveys to the decisionmaker, or the manner in which the decisionmaker perceives the stimulus, is dependent upon the patterns extracted from the stimulus. In the identification phase, the sensory—perceptual system classifies the stimulus object. The way in which this is often assumed to occur is by a weighted matching of the current feature list against a likely set of prototypes in long-term memory [7], [313], [327], [354] with the input being classified according to the name of the best matching prototype. The quality or extent of the sensory information extracted determines the accuracy of identification.

Thus pattern recognition processes involve memory and the other three components of information acquisition: sensation, or the initial experience of stimulation from the sensory modalities; attention, or the concentration of cognitive effort on sensory stimuli; and perception, or the use of higher-order cognition to interpret sensory stimuli.

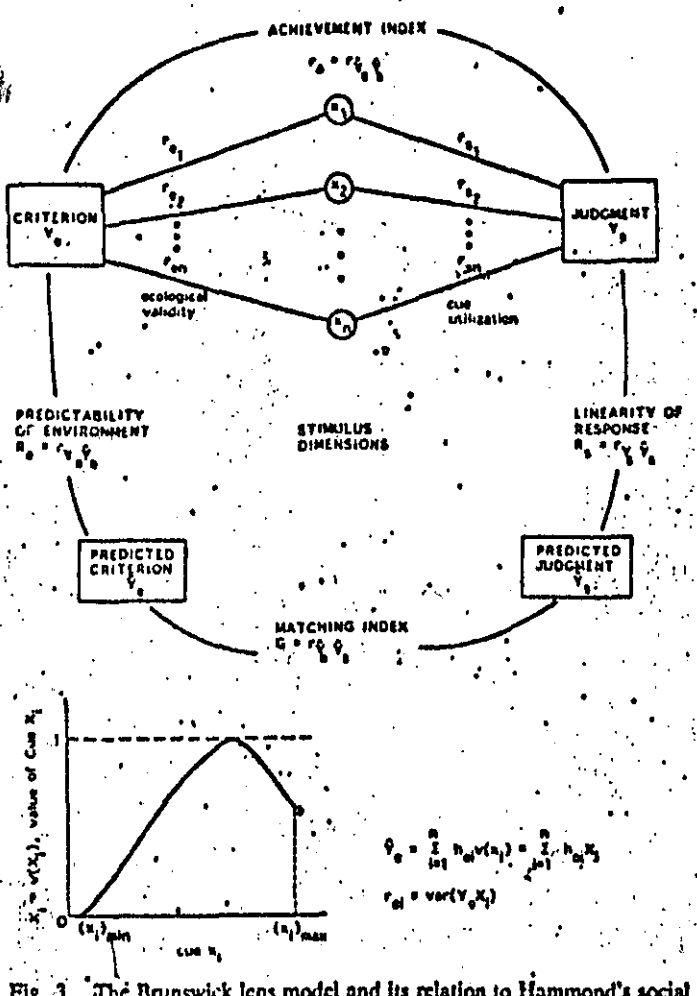


Fig. 3. The Brunswick lens model and its relation to Hammond's social judgment theory.

We have just described what might be regarded as a component or physiological model of information processing. In these stimulus response approaches, behavior is seen as being initiated by the onset of stimuli. A seeming deficiency in approaches of this sort is that there is little consideration of how information bits are aggregated to influence choice and how the decisionmaker goes about the process of information formulation or acquisition, analysis, and interpretation.

A lens model developed by Brunswick and his students is a notable exception to this. The Brunswick lens model is the basis for the policy capture or social judgement theory approach of Hammond and his colleagues [140]–[143]. The lens model, displayed in Fig. 3, assumes that people are guided by rational programs in their attempt to adapt to the environment. There is a criterion value Y_e, and the subjects response, judgment, or inference Y_s. The left side of Fig. 3 represents ecological cue validities which are the correlations r_{ei} between the cues and the criterion value. On the right or organismic side of Fig. 3, a subject will base a response, judgment, or inference Y_s, on the perceived ecological structure. By calculating the correlations r_{si}, that exist between the cues and the response or criterion evaluation, learning concerning the response system can be obtained.

We note that the value of the environmental criterion Y_e and the subject inference Y_s are directly comparable if linear combinations of the cues are assumed. We have, for n cues,

$$Y_e = \sum_{i=1}^{n} h_{ei} x_i + r_{e0} \qquad \hat{Y}_e = \sum_{i=1}^{n} h_{ei} x_i$$

$$Y_s = \sum_{i=1}^{n} h_{si} x_i + r_{s0} \qquad \hat{Y}_s = \sum_{i=1}^{n} h_{si} x_i$$

where h_{y_1} and h_{y_2} are optimum regression weights for the independent cues x, which provide measures of the importance weights of the cues, r_e and r_s are error terms due to inadequacy of the linear model, Y_e and Y_s are the true criterion value and subject response, and $\hat{Y}_e$ and $\hat{Y}_s$ are the predicted criterion value and subject response based on the observed cues. The many works of Hammond and his associates [2], [21], [45], [54], [140]–[143], [186], [187], [261], [290], [294], [295], [404] concerning social judgment theory make use of this lens model. The approach has been shown to be useful in a variety of areas such as policy formulation, negotiation, and conflict resolution. Recent efforts by Hoffman, Earle, and Slovic [154] have shown that the computer displays of social judgment theory: which show both task characteristics, in terms of cue values and corresponding criterion values; and response characteristics, in terms of individual cue values and associated subject responses and judgments; provide a very effective feedback mechanism which might enable people to effectively learn much about complex functional relationships and tasks. There are a number of studies of regression analysis approaches to determination of parameters for decision rules [260], [290]. Use of regression analysis is central to social judgment theory. Recent applications of the approach [261] have involved using simulation models to generate responses which are valuated by the decisionmaker.

Questions concerning the cognitive style used by the decisionmaking are, we believe, very important. Information analysis and information interpretation may be accomplished in a concrete operational mode of thought or in a formal operational mode. We will describe the essential features of these two higher-level cognition processes in Section V. The concrete operational thought process, which is typically applied in familiar situations which people perceive to be well structured, may involve efforts such as reasoning by analogy, or affect, or standard operating procedures. The formal operational thought process, typically applied in situations with which the problem solver is unfamiliar and inexperienced, may involve explicit use of quantitative or qualitative analytical thought.

In either of these modes or "styles" of thought or cognition, information acquisition, analysis, and interpretation may be quite flawed. Many recent studies emphasize the strong need for modeling problem solving behavior in a descriptive, or positive sense in order to detect possible flaws in information processing. Our discussions thus far in this section have been concerned with physiological models in which people have input and output mechanisms, a memory for information storage and retrieval, and a central processor for coordination and control. Here, we wish especially to underscore the need not only for physiological or stimulus–response models but especially for process tracing [72], [95–98], [255] models of information formulation, analysis, and interpretation as well as associated decisionmaking. Knowledge of the actual unaided process of problem solving or descriptive process tracing should serve as a useful guide to the design of information systems that avoid, or at least ameliorate the effects of, cognitive heuristics and biases. This involves requirements for a knowledge of the ways in which people apply strategies in order to reach judgments.

A large number of contemporary studies in cognitive psychology indicate that the attempts of people, including experts, to apply various intuitive strategies in order to acquire and analyze information for purposes such as prediction, forecasting, and planning are often flawed. Many studies have been conducted to describe and explain the way information is acquired and analyzed and the results of faulty acquisition and analysis. Generally the descriptive behavior of subjects in tasks involving information quisition and analysis is compared to the normative results that ould prevail if people followed an "optimal" procedure. There have been a number of recent discussions of cognitive biases from several perspectives [61], [62], [98], [142], [154], [156], [160], [161], [185], [234], [263], [304], [309], [346–349], [351], [352], [385], [386], [406–408]. The recent texts by Nisbett and Ross [264] and Hogarth [159] concerning the strategies and biases associated with judgment and choice are especially noteworthy. Among the cognitive biases that have been identified are several which affect information formulation or acquisition, information analysis, and interpretation. Among these biases, which are not independent, are the following.

1) Adjustment and Anchoring [345], [383]—Often a person finds that difficulty in problem solving is due not to the lack of data and information, but rather to the existence of excess data and information. In such situations, the person often resorts to heuristics which may reduce the mental efforts required to arrive at a solution. In using the anchoring and adjustment heuristic when confronted with a large amount of data, the person selects a particular datum, such as the mean, as an initial or starting point, or anchor, and then adjusts that value improperly in order to incorporate the rest of the data such as to result in flawed information analysis.

2) Availability [383], [385]—The decisionmaker uses only easily available information and ignores not easily available sources of significant information. An event is believed to occur frequently, that is with high probability, if it is easy to recall similar events.

3) Base Rate [25], [291], [386]—The likelihood of occurrence of two events is often compared by contrasting the number of times the two events occur and ignoring the rate of occurrence of each event. This bias often occurs when the decisionmaker has concrete experience with one event but only statistical or abstract information on the other. Generally abstract information will be ignored at the expense of concrete information. A base rate determined primarily from concrete information may be called a causal base rate whereas that determined from abstract information is an incidental base rate. When information updates occur, this individuating information often is given much more weight than it deserves. It is much easier for individuating information to override incidental base rates than causal base rates.

4) Conservatism [210], [259], [345]—The failure to revise estimates as much as they should be revised, based on receipt of new significant information, is known as conservatism. This is related to data saturation and regression effects biases.

5) Data Presentation Context [161]—The impact of summarized data, for example, may be much greater than that of the same data presented in detailed, nonsummarized form. Also different scales may be used to considerably change the impact of the same data.

6) Data Saturation—People often reach premature conclusions on the basis of too small a sample of information while ignoring the rest of the data that is received later on, or stopping acquisition of data prematurely.

7) Desire for Self-Fulfilling Prophecies—The decisionmaker values a certain outcome, interpretation, or conclusion and acquires and analyzes only information that supports this conclusion. This is another form of selective perception.

8) Ease of Recall [205], [382], [383]—Data which can easily be recalled or assessed will affect perception of the likelihood of similar events occurring again. People typically weigh easily recalled data more in decisionmaking than those data which cannot easily be recalled.

9) Expectations [161], [235]—People often remember and attach higher validity to information which confirms their previously held beliefs and expectations than they do to disconfirming information. Thus the presence of large amounts of information makes it easier for one to selectively ignore disconfirming information such as to reach any conclusion and thereby prove anything that one desires to prove.

10) Fact-Value Confusion—Strongly held values may often be regarded and presented as facts. That type of information is sought which confirms or lends credibility to one's views and values. Information which contradicts one's views or values is ignored. This is related to wishful thinking in that both are forms of selective perception.

11) Fundamental Attribution Error (Success/Failure Error) [263], [264]—The decisionmaker associates success with personal inherent ability and associates failure with poor luck in chance events.

This is related to availability and representativeness.

12) Gamblers Fallacy—The decisionmaker falsely assumes that unexpected occurrence of a "run" of some events enhances the probability of occurrence of an event that has not occurred.

13) Habit—Familiarity with a particular rule for solving a problem may result in reutilization of the same procedure and selection of the same alternative when confronted with a similar type of problem and similar information. We choose an alternative because it has previously been acceptable for a perceived similar purpose or because of superstition.

14) Hindsight [112–114], [116]—People are often unable to think objectively if they receive information that an outcome has occurred and they are told to ignore this information. With hindsight, outcomes that have occurred seem to have been inevitable. We see relationships much more easily in hindsight than in foresight and find it easy to change our predictions after the fact to correspond to what we know has occurred.

15) Illusion of Control [209], [210]—A good outcome in a chance situation may well have resulted from a poor decision. The decisionmaker may assume a feeling of control over events that is not reasonable.

16) Illusion of Correlation [115], [383]—A mistaken belief that two events covary when they do not covary is known as the illusion of correlation.

17) Law of Small Numbers (see Kahneman and Tversky [235])—People are insufficiently sensitive to quality of evidence. They often express greater confidence in predictions based on small samples of data with nondisconfirming evidence than in much larger samples with minor disconfirming evidence. Sample size and reliability often have little influence on confidence.

18) Order Effects [161], [184]—The order in which information is presented affects information retention in memory. Typically the first piece of information presented (primacy effect) and the last presented (recency effect) assume undue importance in the mind of the decisionmaker.

19) Outcome Irrelevant Learning System [96], [97]—Use of an inferior processing or decision rule can lead to poor results and the decisionmaker can believe that these are good because of inability to evaluate the impacts of the choices not selected and the hypotheses not tested.

20) Overconfidence [114], [183], [216]—People generally ascribe more credibility to data than is warranted and hence overestimate the probability of success merely due to the presence of an abundance of data. The greater the amount of data, the more confident the person is in the accuracy of the data.

21) Redundancy—The more redundancy in the data, the more confidence people often have in their predictions, although this overconfidence is usually unwarranted.

22) Reference Effect [30], [383]—People normally perceive and evaluate stimuli in accordance with their present and past experiential level for the stimuli. They sense a reference level in accordance with past experience. Thus reactions to stimuli such as a comment from an associate, are interpreted favorably or unfavorably in accordance with our previous expectations and experiences. A reference point defines an operating point in the space of outcomes. Changes in perceptions due to changes in the reference point are called reference effects. These changes may not be based upon proper, statistically relevant computations.

23) Regression Effects [183], [383]—The largest observed values of observations are used without regressing towards the mean to consider the effects of noisy measurements. In effect this ignores uncertainties.

24) Representativeness [382], [383]—When making inference from data too much weight is given to results of small samples. As sample size is increased, the results of small samples are taken to be representative of the larger population. The "laws" of representativeness differ considerably from the laws of probability and violations of the conjunction rule $P(A \cap B) \leq P(A)$ are often observed.

25) Selective Perceptions [161]—People often seek only information that confirms their views and values. They disregard or ignore disconfirming evidence. Issues are structured on the basis of personal experience and wishful thinking. There are many illustrations of selective perception. One is "reading between the lines" such as, for example, to deny antecedent statements and, as a consequence, accept "if you don't promote me, I won't perform well" as following inferentially from "I will perform well if you promote me."

26) Spurious Cues [161]—Often cues appear only by occurrence of a low probability event but they are accepted by the decisionmaker as commonly occurring.

27) Wishful Thinking—The preference of the decisionmaker for particular outcomes and particular decisions can lead the decisionmaker to choose an alternative that the decisionmaker would like to have associated with a desirable outcome. This implies a confounding of facts and values and is a form of selective perception.

Doubtlessly there are other information acquisition, analysis, and interpretation biases that we have not identified here. Any categorization into acquisition, analysis, and interpretation bias is somewhat arbitrary since iteration and feedback will often, in practice, not allow this separation. Also, many of the identified biases overlap in meaning and, therefore, are related to others. Some further discussion of cognitive biases will be presented in our discussion of the situation framing phase of prospect theory in Section III. Certainty, reflection, and isolation effects are three results of these biases that have particular prominence in prospect theory.

Of particular interest are circumstances under which these biases occur, their effects on activities such as decisionmaking, issue resolution, planning, and forecasting and assessment; and appropriate styles which might result in debiasing or amelioration of the effects of cognitive bias.

Many of the cognitive biases that have been found to exist have been found in the unfamiliar surroundings of the experimental laboratory, and generalization of this work to real world situations is a contemporary research area of much interest. However most of the laboratory experiments have concerned very simple if unfamiliar tasks. A number of studies have compared unaided expert performance with simple quantitative models for judgment and decisionmaking, such as those by Brehmer [47], Cohen [62], Dawes [70], [71], Goldsmith [132], Kleinmuntz and Kleinmuntz [204], and by several authors in Wallstein's recent definitive work concerning cognitive processes in choice and decision behavior [396]. While there is controversy [62], [263], [349], most studies have shown that simple quantitative models perform better in human judgment and decisionmaking tasks, including information processing, than wholistic expert performance in similar tasks. This would appear to have major implications and to sound major caveats for such areas as "expert forecasting." This caution is strongly emphasized in the works of Hogarth and Makridakis [161], Makridakis and Wheelright [235], and Armstrong [14]–[16]. This is a caution noted in but a few [18] of the contemporary works on forecasting and assessment.

There are a number of prescriptions which might be given to encourage avoidance of possible cognitive biases and to debias those that do occur [96], [98], [161], [184], [235], [355], [386]. Some suggestions to avoid cognitive bias follow.

1) Sample information from a broad data base and be especially careful to include data bases which might contain disconfirming information.

2) Include sample size, confidence intervals, and other measures of information validity in addition to mean values.

3) Encourage use of models and quantitative aids to improve upon information analysis through proper aggregation of acquired information.

4) Avoid the hindsight bias by providing access to information at critical past times.

5) Encourage decisionmakers to distinguish good and bad decisions from good and bad outcomes in order to avoid various forms of selective perception such as, for example, the illusion of control.

6) Encourage effective learning from experience. Encourage understanding of the decision situation and methods and rules used in practice to process information and make decisions such as to avoid outcome irrelevant learning systems.

7) Use structured frameworks based on logical reasoning [255], [376] in order to avoid confusing facts and values and wishful thinking and to assist in processing information updates.

8) Both qualitative and quantitative data should be collected, and all data should be regarded with "appropriate" emphasis. None of the data should be overweighted or underweighted in accordance with personal views, beliefs, or values only.

9) People should be reminded, from time to time, concerning what type or size of sample from which data are being gathered, so as to avoid the representativeness bias.

10) Information should be presented in several orderings so as to avoid recency and primacy order effects, and the data presentation context and data saturation biases.

Kahneman and Tversky [235] discuss a systemic procedure to enhance debiasing of information processing activities. A definitive discussion of debiasing methods for hindsight and overconfidence is presented by Fischhoff [185]. Lichtenstein and Fischhoff present a number of helpful guidelines to assist in training for calibration [217]. Clearly more efforts along these lines are needed. Studies to determine the extent to which learning feedback acquired through use of methods such as social judgment theory contributes to debiasing would be especially rewarding. This is especially the case since confidence in unaided judgment is learned and maintained through feedback even when there is very little or no justification for this confidence [94]. Typically outcomes which follow from decisions based on negative judgments are not observed. Reinforcements of self-fulfilling prophecy type judgments through positive outcome feedback only occur in spite of, rather than due to, judgment validity.

Research integrating the methods whereby people integrate or aggregate information and attribute causes [8]–[12], [142], [143], [186], [190], [199], [321], [364] with methods for the identification and amelioration of cognitive biases would be of interest and of much potential use also.

In a sense, the results of this section are disturbing in that they tend to support the "intellectual cripple" hypothesis of Slovic ([142], pg. 14) and imply that humans may well be little more than masters of the art of self deception. On the other hand there is strong evidence that humans are very strongly motivated to understand, to cope with, and to improve themselves and the environment in which they function. While there are a number of fundamental limitations to systemic efforts to assist in bettering the quality of human judgment, choice, and decisions [282], [307], there are also a number of desirable activities [16], [305], [385]. These can assist in increasing the relevance of systemic approaches such as those which result in information processing adjuvants for policy analysis, forecasting, planning, and other judgment and decision tasks in which information acquisition, analysis, and interpretation play a needed and vital role.

IV. Decision Rules

In order to select an alternative plan or course of action for ultimate implementation, the decisionmaker applies one or more decision rules which enable comparison prioritization, and ultimately, selection of a single policy alternative from among a set of choice alternatives. The purpose of a decision rule is to specify the most preferred alternative generally from a partial or total ordering or prioritization of alternatives. To utilize a decision rule we must have a set of alternatives, a set of objectives to be accomplished by the alternatives, a knowledge of the impacts of the alternatives, evaluation of these impacts, and associated preference information. Decision rules may be explicit or implicit in terms of the way in which they are used in the decision process.

We can assume, without loss of generality, that each single policy alternative may represent a complex portfolio of individual alternatives and that the set of choice alternatives contains mutually exclusive components. This formulation can always be accomplished but may result in a very large set of policy alternatives since n individual alternatives can be combined into 2^n possible portfolios of alternatives. Failure to consider a combination of alternatives may result in significant errors in decisionmaking unless each of the individual alternatives represents one component of a portfolio of all possible combinations of individual alternatives, or unless the individual alternatives are independent or mutually exclusive.

It is assumed at the interpretation step of the decision process that formulation and analysis have been accomplished such that there exists a decision situation structural model and the results of exercising the model. Thus objectives, relevant constraints, some bounds on the issue, possible policy alternatives, impacts of policy alternatives, etc. are assumed known. The choice of a decision rule will depend, in large measure, upon the decision situation structural model as reflected in the contingency task structure. We will discuss dynamic models for contingency task structures in our next section.

The above discussion may appear representative primarily of the judgment and decision process associated with the formal operational thought mode that we will elaborate upon in our next section. For purposes of clarity of exposition, we have presented an oversimplified view of how decision rules are used to aggregate information and evaluate alternatives. The sequence we have described implies comparison and evaluation of alternatives only after we have first accomplished formulation and analysis of the issue under consideration. As we have noted throughout our discussion, decisionmakers typically compare and evaluate alternatives while they are in the process of decision situation formulation and analysis. These partial comparisons and evaluations lead to searches for additional policy alternatives, additional analysis, etc. As we have also noted, the entire decision process typically occurs in a parallel-simultaneous-iterative fashion rather than an exclusively sequential series of steps in which formulation is followed by analysis which is followed by interpretation.

Individuals and decision environments vary so greatly that there are a great number of decision rules that will be needed to describe actual decision situations. Schoemaker [315] is among a number of authors [121], [255], [364], [365], [372] who have attempted classification schemes to allow categorization of various descriptive decision rule models. His first level categorization separates decision rules into holistic and nonholistic categories. In a holistic decision rule each alternative or portfolio of alternatives is evaluated and assigned a value or utility. After all alternatives have been evaluated, they are compared and alternative A is said to be preferred to alternative B if its evaluation has given it a greater utility such that $U(A) > U(B)$. In nonholistic decision rules individual alternatives or portfolios of alternatives are generally compared with one another in a sequential elimination process. This comparison may be against some standard across a few attributes within alternative pairs or across alternatives, with alternative attributes being compared one at a time.

Each of these categories appears to imply disaggregation into components of the event outcomes likely to follow from decisions. Our section on contingency task structure models will propose a dynamic evolving cognitive style model which admits of expert situational understanding that involves reasoning by analogy, intuitive affect, and other forms of nonverbal, almost unconscious perception. We elect to call this type of reasoning wholistic and add a third category to the classification scheme of Schoemaker.

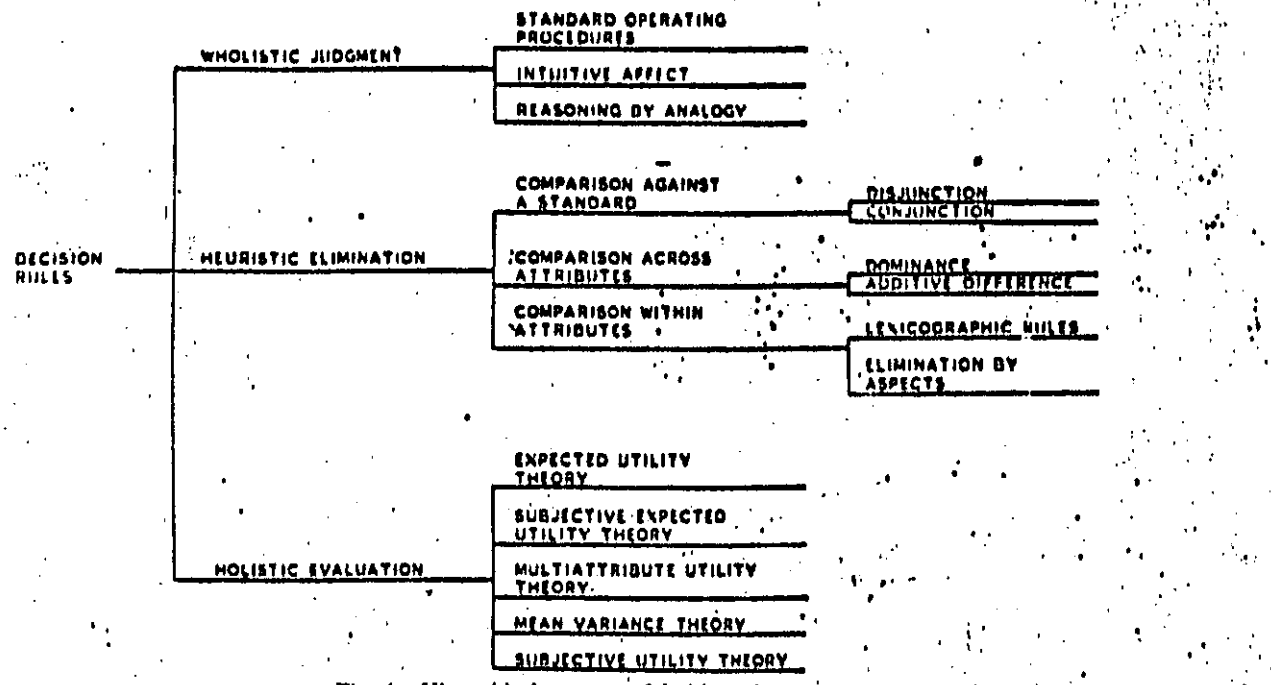


Fig. 4. Hierarchical structure of decision rules.

Consequently we envision three first level general categories of decision rules: holistic, heuristic, and wholistic. In a *holistic* decision rule there is an attempt to consider all aspects of a decision situation in evaluating choices by means of disaggregation of various choice components. In a *heuristic* decision rule, detailed complicated comparisons are not used. Rather, simplified approximations to holistic decision rules are used. In a *wholistic* decision rule, the evaluation and choice of alternatives is based upon use of previous experience, hopefully true expertise, with respect to similar decision situations. The selection of an alternative is based upon its perceived or presumed worth as a whole and without detailed conscious consideration of the individual aspects of each alternative. It is possible to define a number of decision rules and categorize them. The first level categories we have defined are not mutually exclusive. A number of decision rules doubtlessly can be categorized into more than one of these first-level decision categories. Figure 4 illustrates a possible inclusion structure for the decision rules we will describe here.

Expected Utility Theory: Our first decision rule is based on expected utility theory and is doubtlessly the most familiar decision rule to engineers. This rule derives from a "rational actor"[2] decision model [3], [4], [89], [103], [121], [134], [169], [192], [222], [256], [265], [285], [315], [359], [397] which is more fully discussed in Section VI.

The rational actor model is a normative model. Von Neuman and Morgenstern, who introduced the axioms of the model of rational man, stated the purpose of their work as

> to find mathematically complete principles which define "rational behavior"... a set of rules for each participant which tell him how to behave in every situation which may conceivably arise.

The idea of rationality originated in the economics literature where microeconomic models of the consumer and the firm assumed complete information and rationality. The rational person is assumed to have identified a set of well-defined objectives and goals and is assumed to be able to express preferences between different states of affairs according to the degree of satisfaction of attaining these objectives and goals. A rational person has identified available alternative courses of action and the possible consequences of each alternative. The rational person makes a consistent choice of alternative actions in order to maximize the expected degree of satisfaction associated with attaining identified objectives and goals.

A number of elements are assumed to exist in the rational actor model:

1) a set of policy alternatives A;
2) the set of possible consequences of choice or future states of nature or decision outcomes called S;
3) a utility function $U(s)$ that is defined for all elements s of S;
4) information as to which outcomes will occur if a particular policy alternative a in A is chosen; and
5) information as to the probability of occurrence of any particular outcome if an alternative $a \in A$ is chosen. $P_a(s)$ is the probability that $s \in S$ will occur if $a \in A$ is chosen.

There are a number of ways in which the axioms associated with the rational actor model may be stated. Each statement of the axioms allows proof of the fact that cardinal utility functions will exist and be unique only up to positive linear transformations. Further, the evaluation of expected utility allows choicemaking and prioritization of alternatives in accordance with the expected utility of each alternative. There are a number of textbook accounts of expected utility theory to which the interested reader of this review may turn for alternative sets of axioms and detailed accounts of the use of expected utility theory [51], [163], [196], [222], [285], [302], [315]. MacCrimmon and Larson interrelate the major axiom systems in expected utility theory [3] in a noteworthy contribution to understanding of the several systems that lead to (essentially) the same results for the rational actor model. A very readable introductory treatment of expected utility theory, relating descriptive psychological concerns with normative concerns, is presented by Vlek [244].

The rational actor model is often accepted as a normative model of how decisions should be made, at least in a substantive or "as if" fashion. It is often observed that the model is not an accurate description of either the substance or the process of actual unaided choicemaking behavior. Some of these observers use empirical evidence of the deviation of actual decisionmakers from either substantive rationality or process rationality. These observations are doubtlessly correct. The rational actor model is

[2] Technological or economic rationality would be a more appropriate term.

however, invaluable in that it can be often used as reference for comparison of actual behavior with ideal "aided" or normative behavior. Further, it provides a benchmark against which to compare simplified heuristics. Our efforts and discussions in this section concern primarily substantive behavior although we recognize the great difficulty, in practice, of separating substance from process.

Simon and his colleagues introduced the concept of bounded rationality and developed a satisficing model for individual choicemaking. It is worth noting that boundedly rational actors are basically rational subject to constraints on the formulation, analysis, and interpretation of information, and the substitution of achievement of a target level of return, or aspiration level, for selection of the best alternative. Typically people satisfice by adaptive adjustment [72] of aspirations such that, in repetitive decision situations, optimizing behavior is approached [270].

There is absolutely nothing in the formulation of the rational actor model which requires identification of *all* objectives, *all* possible alternatives, *all* possible impacts of alternatives, etc. The rational actor model is perfectly capable of being used to allow prioritization and selection of the best alternative by evaluating some impacts and with knowledge of some objectives, from among an incomplete set. It in no sense necessarily requires completeness in everything and the associated complexity that this would require. Actual decisionmaking behavior may not, however, even be boundedly rational but may employ such poor heuristics as to result in inferior choicemaking even to the extent of selecting inferior choices from among those in a bounded set.

There have been a number of experimental studies and field studies of the appropriateness of the expected utility model [3], [111], [117], [119], [125], [184]–[186], [236], [336]–[341], [385] as a descriptive model of substantive unaided behavior. Among the surveys which comment upon the experimental and field studies are [27], [98], [206], [348], [372]. Schoemaker [315] provides a very readable brief survey of some of this literature. While the evidence is mixed, most studies indicate that the expected utility decision rule simply does not function well in a *descriptive substantive* sense.

In its simplest form the expected utility of alternative a_i is computed from

$$E[U(a_i)] = \sum_{j=1}^{n} p[s_j(a_i)]U[s_j(a_i)] \qquad (1)$$

where the $s_j(a_i)$, $j = 1, 2, \ldots, n$, are the states which may result from alternative a_i and the $p[s_j(a_i)]$ are the associated probabilities. In the expected utility formulation the $p[s_j(a_i)] = p_j(a_i) = p_j$ are assumed to be objective probabilities and, of course, $\sum_{j=1}^{n} p_j = 1$. Generally these probabilities are not alternative invariant although notationally they are sometimes written as if they were independent of alternatives. The $U[s_j(a_i)]$ are the utilities or values [296] of the decisionmaker for the various outcome states. Johnson and Huber [179] survey a number of procedures that can be used to elicit utility functions. Most of the textbooks cited earlier also contain discussion of utility assessment procedures.

Subjective Expected Utility: Often it occurs that objective probabilities are, for any of a variety of reasons, unavailable in a given situation. The subjective expected utility model is obtained when subjective probabilities $f(p_j)$ are substituted for the p_j in (1). The $f(p_j)$ are generally elicited such that $\sum_{j=1}^{n} f(p_j) = 1$ and so the subjective probabilities behave in a way consistent with the laws probability. There are a number of discussions concerning probability elicitation [31], [223], [257], [355] that present appropriate procedures to enable determination of subjective probabilities from individuals and groups. Conventional approaches to elicitation of utility in expected utility theory may confound strength of preference felt for alternative event outcomes and attitude toward risk. Also the elicitation procedure can become cumbersome. Recent research has formally separated these factors [33] and shows much promise in enhancing understanding of attitude towards risk. In this approach the utility concept is devoid of risk. It takes on a meaning more like that in conventional microeconomics where it measures strength of preference for certain outcomes only. This research [33] could provide additional linkages and understanding between the expected utility and subjective expected utility concepts by providing for incorporation of risk aversion effects in a relatively simple way. A related approach to incorporation of risk aversion is described by Howard and decision analysts at the Stanford Research Institute [164] who have been responsible for a number of major application studies in this area. There have been a number of related approaches [65], [66], [121] and the subjects of risk and uncertainty are of much contemporary interest [6], [136], [153], [304].

A number of studies have indicated that the relation between subjective and objective probabilities is nonlinear and situation dependent. It is usually indicated that people often underestimate high probabilities and overestimate low ones. More recent research has indicated that this appears true only for favorable outcomes. Just the opposite appears true when the outcome is unfavorable. This appears to be a form of wishful thinking for low probability events and "everything bad happens to me" for high probabilities. What we will call subjective utility theory attempts to incorporate situation dependent nonlinearities that may exist between subjective and objective probabilities.

Multiattribute Outcomes: Often decision situations are sufficiently complex that it is difficult to evaluate, in a wholistic fashion, the utility of each outcome. Often it is possible to disaggregate the features on which utility is based into a number of components called attributes. An attribute tree is a hierarchical structure which, when quantified through elicitation of values of the outcomes on the lowest level attributes and relative weights of the attributes, can be used to determine the utility of event outcomes. The types of multiattribute utility models used have varied from very simple unit weight linear models to rather complex multiplicative models [106]. Dawes [71] documents the robust beauty of linear models of the form

$$U(s_i) = \sum_{j=1}^{m} h_j u_j(s_i), \quad \sum_{j=1}^{m} h_j = 1 \qquad (2)$$

where there are assumed to be m attributes, h_j is the weight of the jth attribute and $u_j(s_i)$ the value score on the jth attribute of outcome s_i. In much of the work in this area decisions under certainty are considered such that there is a one to one correspondence between alternative a_i and outcome s_i. Under decision-under-certainty conditions we can let $s_i = a_i$ in (2).

Multiattribute models have been very successfully used to predict the decision behavior in field settings or many professional groups. Hammond [140]–[142] and his colleagues have, as discussed in Section III, developed an approach known as social judgment theory in which the "policy" of the decisionmaker, equivalent in this circumstance to the weights h_j, are identified from wholistic prioritization of decision outcomes through use of regression analysis techniques. Ward Edwards and his colleagues [186], [301] and elsewhere, elicit weights from decisionmakers for the model of (2) in a useful straightforward procedure called simple multiple attribute ranking technique (SMART) that has seen a number of realistic applications. Results of the surveys of Armstrong [14], [15]; Fischer [111], Slovic and Lichtenstein [345], Slovic, Fischhoff, and Lichtenstein [348]; Shanteau [324], and others indicate that simple linear models [64] are very potent predictors of reliable judgment, especially under conditions of certainty in that one can replicate the substantive judgment of decisionmakers. This is the case even though the simple linear model may not do a very good job of modeling the decision process. "Boot strapping" is the name given to the task of substituting a decision rule for the decisionmaker. The studies in the cited references show that the elimination of human judgment

error made possible by boot strapping enables it to be superior to unaided human judgment. One can even misspecify weights and ignore attribute dependencies and still find that weighted linear models do quite well [71].

The fact that the weighted linear rule may be so good is a rather mixed blessing. In circumstances in which there is no requirement for knowledge of the underlying decision process, the substantive predictive ability of the linear additive model may make it quite useful. Situations such as evaluating credit card applicants or applicants for admissions to colleges are repetitive judgment and decision situations which fit into this category. Use of simple formal linear model may well, in situations such as these, lead to a more efficient as well as more effective and equitable selection process than one based on unaided human intuition [70], [71]. (Dawes in Shweder [332]). In unstructured or semistructured nonrepetitive decision situations it is much less clear that a decision rule that is not guaranteed to be faithful to the underlying decision process will be nearly as valuable as one that is in terms of enabling decisionmakers to make better decisions. Fischhoff, Goitein, and Shipira [119] provide a number of perceptive comments concerning this and the consequent need for a theory of errors to explicate the effects of poor decision situation structural models and parameters within the structure. A hoped-for achievement is a sensitivity-based analysis of deviations from optimality to determine, among other things, the role of experience in decisionmaking and those components and principles of decisionmaking which can be usefully and meaningfully learned from experience [47], [94]–[97], [115], [116].

Multiattribute utility models based on the expected utility theory of von Neumann and Morgenstern are considerably more complex than those of behavioral decision theory. Often there are efforts to determine existence of various attribute independence conditions such as to validate use of a linear model of the form of (2) or a multiplicative model of the form

$$1 + kU(s_i) = \prod_{j=1}^{m} \left[1 + k_j u_j(s_i)\right], \qquad \sum_{j=1}^{m} k_j = 1. \quad (3)$$

The foremost proponents of this approach are Keeney and Raiffa [196]. There are many contributions to this area and variations of the basic approach [23], [29], [75], [93], [127], [231], [277], [278], [300], [301], [302], [358], [399]. It is proposed exclusively as a normative approach and has been successfully used for a variety of applications including proposal evaluation [245], [310], siting power plants [197]; and budgeting and planning [52], [191].

Mean Variance—There are a number of models and associated decision rules based upon mean-variance (EV) models. Markowitz's portfolio theory, which is well summarized in Libby and Fishburn [214] and Baron [26], is based in part on the assumption of a quadratic utility function

$$U(s) = \alpha + \beta s + \gamma s^2 \quad (4)$$

where the same states are assumed invariant over all alternatives such that we have a quadratic programming problem in prioritizing alternatives where

$$E\{U(a_i)\} = \sum_{j=1}^{n} p_j(a_i)U(s_j)$$
$$= \alpha + \beta E(a_i) + \gamma E(a_i^2)$$
$$= \alpha + \beta\mu_i + \gamma(\sigma_i^2 + \mu_i^2).$$

Coombs [65], [66], [185] has also been concerned with portfolio theory and assumes an optimum risk level in the form of a single peaked risk preference function for every expected value level. Gambles of equal expected value are judged on the basis of lower variance in the Markowitz portfolio theory and on the basis of deviation from optimum risk level in Coombs' portfolio theory. Stochastic dominance concepts [124] are especially useful in dealing with problems in the mean-variance models of portfolio theory. Unfortunately as has been shown by a number of authors [124], the results from using mean-variance portfolio theory are not necessarily consistent with results obtained from expected utility theory. For example, if the outcomes of decision a_1 are \$10 with probability 0.5 and \$20 with probability 0.5 and the outcome of decision a_2 is \$10 with probability 1.0; then the EV rule ($\mu_{a1} = \$15$, $\sigma_{a1} = \$5$) ($\mu_{a2} = \$10$, $\sigma_{a2} = 0$) is indeterminate in that there is no pareto superior or dominance alternative in an EV sense. Yet any reasonable person would prefer alternative a_1 to alternative a_2.

Fishburn [123] has considered a variation of the mean-variance model which involves concepts based upon target level of return or aspiration level or reference level to define the risk of an alternative. The "risk" of alternative a is determined from the probability of receiving a return not to exceed x, denoted $F(x)$, by

$$R(a) = \int_{-\infty}^{t} (t - x)^{\alpha}\, dF(x) = \int_{-\infty}^{t} (t - x)^{\alpha} p(x)\, dx$$

where t is the target return, α is a nonnegative parameter that is used to indicate relative importance of deviations below target return. For $0 \leq \alpha < 1$ the decisionmakers primary concern is failure to achieve the target with little regard to the size of the deviation. For $\alpha > 1$ the decisionmaker is very concerned with sizeable deviations from target and relatively unconcerned with small deviations. In the former case the decisionmaker is risk seeking for losses and has a utility function that is convex for losses. In the latter case the decisionmaker is risk averse for losses and has a utility function that is concave for losses.

In this model the mean return from an alternative and its risk are the two attributes determining preference. This model thus appears much similar to the standard EV model in that $a_1 > a_2$ if and only if $\mu(a_1) \geq \mu(a_2)$ and $R(a_1) \leq R(a_2)$ with at least one inequality being valid. In the example just considered, the mean values are as given previously and the risks are

$$R(a_1) = \begin{cases} 0, & t \leq 10 \\ 0.5(t - 10)^{\alpha}, & 10 \leq t \leq 20 \\ 0.5(t - 10)^{\alpha} + 0.5(t - 20)^{\alpha}, & 20 \leq t \end{cases}$$

$$R(a_2) = \begin{cases} 0, & t \leq 10 \\ (t - 10)^{\alpha}, & 10 \leq t \end{cases}$$

Thus we see that the risk is the same, that is zero, if $t \leq 10$ and so we prefer a_1. The risk associated with a_1 is one half that associated with a_2 if the target return is between \$10 and \$20. The risk associated with a_1 is less than that associated with a_2 if $t \geq 20$. And so, since $\mu(a_1) > \mu(a_2)$, we prefer a_1 regardless of the target return. Generally, as in this case, Fishburn's below-target model will resolve ambiguities associated with the standard mean-variance model. The decisionmaker is free to specify α and t. Thus this represents a rather useful dominance type decision rule. Extensions of this rule to the case of multiattribute and multiple objective preferences would have considerable value.

Subjective Utility Theory: A number of researchers have proposed holistic decision rules based on the observation that people, in unaided situations, do not typically perceive (objective) probabilities such that the fundamental probability property $\sum_{j=1}^{n} p_j = 1$ is satisfied. There presently exists several decision situation models based upon a subjective utility theory in which probabilities do not sum to one. Among these are certainty equivalence theory due to Handa [144], subjectively weighted utility theory due to Karmarkar [188], [189]; and prospect theory due to Tversky and Kahneman [184], [385]. There have been several additional studies involving prospect theory including those of Thaler [371], and Hershey and Schoemaker [152], [153]. Some of the foundations for these subjective utility theory efforts can be found in the early work of Allais [3] who was among the

first to note that the normative expected utility approach of von Neumann and Morgenstern, and the subjective expected utility modifications, did not necessarily describe actual descriptive choice behavior. We believe that these studies are especially relevant to information system design and so summarize relevant features from these effects here.

In certainty equivalence theory five axioms are assumed. We will use the term prospect or prospect (s, P) to mean the opportunity to obtain outcome s with probability P. Simply stated, these are as follows.

1) Preferences are governed only by utilities and outcomes. One is indifferent between a nonsimple prospect and an actuarially identical simple prospect with a single event node.
2) Complete ordering of prospects is possible and transitivity of prospects exists.
3) Continuity exists such that if $(s_1, P_1) \succcurlyeq (s_2, P_2), \succcurlyeq (s_3, P_3)$ then there exists an α such that $(s_2, P_2) \sim [\alpha s_1, P_1) + (s_3 - \alpha s_3, P_3)$.
4) Independence exists such that if $(s_i, P_i) \sim (x_i, 1) \forall i$, then $(\underline{s}, \underline{P}) \sim (\Sigma x_i, 1)$ where $\underline{s}$ and $\underline{P}$ represent vectors of outcomes and probabilities s_i and P_i.
5) Enhanced prospects are preferred if and only if a basic prospect is preferred. Thus $(\beta s_1, P_1) \succcurlyeq (\beta s_2, P_2) \forall \beta \geq 0$ if and only if $(s_1, P_1) \succcurlyeq (s_2, P_2)$.

These axioms are sufficient to insure that the subjective utility function of alternative a_i, $CE(a_i) = CE[s(a_i), P(a_i)] = U(\underline{s}^i, \underline{P}^i)$, is linear in s_i and of the form

$$U(\underline{s}^i, \underline{p}^i) = \sum_{j=1}^{n} s_j^i w(P_j^i) = \underline{w}^T(\underline{P}^i)\underline{s}^i. \tag{5}$$

Axioms 1, 4, and 5 incorporate the major changes from the von Neumann–Morgenstern axioms. It appears unduly restrictive to require that the utility function be linear in the outcome and this is reason enough to warrant the development of a more robust theory.

Fishburn [125], however, has shown that certainty equivalence theory must reduce to the expected value model, $U(\underline{s}, \underline{P}) = \underline{P}^T\underline{s}$, $\Sigma_{j=1}^{n} w(P_j) = 1$. This occurs because of the requirement that one must be indifferent between a nonsimple prospect and an actuarially equivalent simple prospect. To insure this for the two outcome case, for the general actuarially equivalent two outcome prospects of Fig. 5, requires that $w(P) + w(1 - P) = 1$.[3] This certainty must be viewed as another limitation of this certainty equivalence theory and indicates the considerable care that must be exercised in modifying the basic utility theory axioms.

The subjective weighted utility model yields for the SWU of alternative a_i

$$SWU(a_i) = \sum_{j=1}^{n} w[P_j(a_i)]U[s_j(a_i)] \tag{6}$$

where the subjective weighted probabilities are

$$w[P_j(a)] = \frac{f[P_j(a)]}{\sum_{j=1}^{n} f[P_j(a)]}. \tag{7}$$

Although a variety of probability weighting functions are possible, Karmarkar [188], [189] proposes use of a log normal function

$$\ln\left(\frac{f}{1-f}\right) = a \ln\left(\frac{P}{1-P}\right) \tag{8}$$

[3] For the n outcome case we would have $\Sigma_{j=1}^{n} w(P_j) = 1$ and we see that the only general $w(P_j)$ that will insure this is $w(P_j) = P_j$.

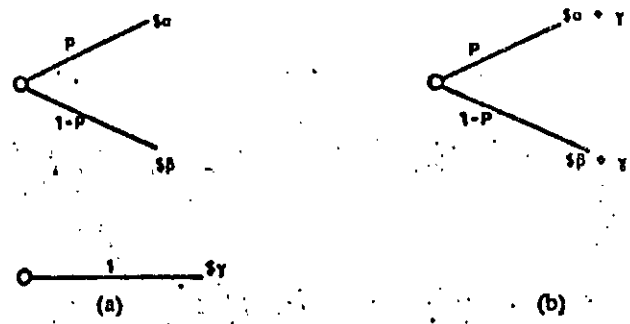


Fig. 5. Two actuarially equivalent prospects.

or

$$f(P) = \frac{P^a}{P^a + (1-P)^a} \tag{9}$$

where $0 \leq a \leq 1$. This transformation of probabilities is such that large probabilities are understated and small probabilities overstated. Karmarkar emphasizes that the probability weighting function does not represent a probability perception phenomenon but represents a bias in the way in which (objective) probabilities are descriptively incorporated into the evaluation, prioritization, and choice making process. In this model, the final weighted probabilities do sum to one in accordance with the conventional subjective expected utility theory. However, the expression for any normalized weight $w[P_j(a)]$ is actually a function of the value of all other probabilities as seen in (7). The effects of this confounding influence remain to be investigated.

The considerably more sophisticated prospect theory of Tversky and Kahneman [384], [385], contains a number of modifications to expected utility theory. Prospect theory consists of an editing phase involving framing of contingencies, alternatives and outcomes, followed by an evaluation phase. These modify subjective expected utility theory such as to enhance unaided descriptive realism of the theory

1) In the editing phase, the decision situation is recast into a number of simpler situations in order to make the evaluation task simpler for the choicemaker. The tasks in editing are very much dependent on the contingency situation at hand and offer possibilities for coding, combining, segregating, cancelling, and detection of dominance.
2) Value functions are devoid of risk attitude and are unique only up to positive ratio transformations.
3) Outcomes are expressed as positive or negative deviations from a reference or nominal outcome which is assigned a value of zero. Thus value changes represent changes in asset position. Positive and negative values are treated differently with the typical value function being an S-shaped curve that is convex below the reference point and concave above it. Displeasure with loss is typically greater than pleasure associated with the same gain.
4) Probability weights $w[P_j(a)]$ reflect an uncertain outcome contribution to the attractiveness of a prospect. As in SWU theory, high probabilities are underweighted and low ones overweighted. The following are among the properties of the probability weighting function.
   a) True at extremes $w(0) = 0, w(1) = 1$
   b) Subadditive at low P, $w(\alpha P) > \alpha w(P), 0 < \alpha$
   c) Overweighted for small p, $w(p) > p, p \ll 1$
   d) Underweighted for large P, $w(P) < P, P > 0$
   e) Subcertain $w(p) + w(1-p) < 1$
   f) Subproportional $w(\alpha P)/w(P) \leq w(\alpha\beta P)/w(\beta P), 0 < \alpha, \beta, \leq 1$.
5) The value of a prospect $(\underline{s}, \underline{P}) = (s_1, P_1) + (s_2, P_2)$ is given by
   a) $V(\underline{s}, \underline{P}) = v(s_2) + w(P_1)[v(s_1) - v(s_2)]$ (10)

for strictly positive prospects in which $P_1 + P_2 = 1$ and $s_1 > s_2 > 0$, or strictly negative prospects in which $P_1 + P_2 = 1, s_1 < s_2 < 0$.

b) $V(\underline{s}, \underline{P}) = w(P_1)v(S_1) + w(P_2)v(s_2)$ (11)

for regular prospects which are prospects that are neither strictly positive nor strictly negative in that either $P_1 + P_2 \neq 1$ and/or $v(s_1)$ and $v(s_2)$ are of opposite sign.

In no sense is prospect theory posed as a normative theory of how people should make decisions. The editing or framing of contingencies, alternative acts, and outcomes is similar to the formulation step of the systems process. It is in this forming phase that the contingency task structure and decision situation model are, in effect, formed. For example, in a population of one million people where black lung disease might kill two thousand people, possible forms are

Form 1—Alternative a_1 will save 500 people whereas if alternative a_2 is adopted there is a 0.25 probability of saving 2000 people and a 0.75 probability of not saving anyone.

Form 2—Alternative a_3 will result in death of 1500 people whereas alternative a_4 will result in a 0.25 probability that no one will die and a 0.75 probability that 2000 people will die.

These two forms are really the same, yet many people will interpret them differently. The editing or forming phase of prospect theory allows different interpretations and thus makes provision for different evaluation of results in terms of alternative formulations of the same issue.

Prospect theory is especially able to cope with *certainty effects* in which people overweight outcomes considered certain compared with those considered only highly probable; *reflection effects* in which preferences are reversed when two positively valued outcomes are replaced by two negatively valued outcomes; and *isolation effects* in which people disregard common outcome components shared by outcomes and focus only on components that distinguish alternatives. Kahneman and Tversky have established an axiomatic basis for prospect theory [184] for the two outcome case.

In a recent study involving prospect theory, Hershey and Schoemaker [152] question the generality of the reflection hypothesis of prospect theory which states that asymmetric preferences are found when comparing gain prospects with loss prospects. They introduce four types of reflectivity depending upon whether subjects choose positive prospect (s_1, P_1) or the noninferior prospect (s_2, P_2), and whether they choose negative prospect $(-s_1, P_1)$ or $(-s_2, P_2)$. Across-subject and within-subject reflectivity are examined in terms of whether subjects do or do not choose and do or do not switch from safe to risky prospects. They conclude that predictions of prospect theory concerning reflectivity depend upon the size of probabilities. For P large enough to insure underweighting of probabilities, it appears that the reflectivity hypothesis is quite valid. For smaller values of P, reflectivity is neither predicted nor excluded from the results of Hershey and Schoemaker.

In another study Hershey and Schoemaker [153] examine preferences for basic insurance-loss lotteries and show that risk taking is prevalent in the domain of losses. They suggest a utility function which is concave for low losses and convex for larger ones. They indicate a context effect in which various insurance formulations lead to more risk averse behavior than for statistically equivalent gambling formulations. Their conclusion that probabilities and outcomes may be of less guidance in influencing decision behavior as uncertainties concerning their magnitude increase, strengthens conjectures concerning the influence of context and perceptions of decision situation structural models upon decision results.

Thaler [371] examines a number of the tenets of prospect theory with generally very positive confirming results. Additional comments concerning the seminal prospect theory appear in a previous survey in these transactions [304] including the observation that a number of the results of prospect theory, which are seemingly at variance with expected utility theory, can be accommodated successfully using multiple attribute utility theory. Extensions of prospect theory to include multiple attribute preferences, large number of outcomes, sequential multistage decisionmaking, risk aversion coefficients, and subjective probability effects would do much to enable this significant development to be of even greater usefulness in explaining complex positive or descriptive decision behavior. This might well be of much normative use as well.

Heuristic Decision Rules

A number of decision rules do not involve comparisons in a true holistic fashion. Rather they involve comparisons of one alternative with another, generally within a restricted alternative set and attribute set. Within the heuristic class of decision rules we may distinguish those which compare alternatives against some standard by means of conjunctive or disjunctive comparisons, those which compare alternatives across attributes, and those which make comparisons within attributes. Generally, these rules can result, when improperly applied, in intransitive choices [289]. We will consider several rules from each subcategory. First we will discuss two noncompensatory rules [90] that are often used when there is an overabundance of data present.

Disjunctive—A disjunctive decision rule is one in which the decisionmaker identifies minimally acceptable value standards for each relevant attribute. Alternatives which pass the critical standard on one or more attributes are retained. Alternatives which fall below the critical standards on all attributes are eliminated. A single alternative is accepted when the critical standards are set such that all but one alternative fail to exceed any of the critical standards on any attributes. Unlike multiattribute utility theory (MAUT) rules, where poor performance on one attribute can be made up by good performance on other attributes such that the rule is compensatory, a disjunctive decision rule is noncompensatory. A compensatory approximation to a disjunctive decision rule for attributes s_i is

$$U = \sum_{i=1}^{m} \frac{1}{\left(1 + \frac{s_i}{c_i}\right)^{n_i}}, \quad n_i > 1 \tag{12}$$

where m represents the number of attributes and c_i is the critical value on the ith attribute. If U is greater than one, the alternative in question is retained.

Conjunctive—A conjunctive decision rule is one in which minimally acceptable value standards for each relevant attribute are identified. Alternatives are acceptable if they exceed all minimum standards. They are rejected if they fail to exceed any minimum standard. The critical values for disjunctive and conjunctive rules are generally different. A compensatory approximation to the noncompensatory conjunctive decision rule is

$$U = \prod_{i=1}^{m} \frac{1}{\left(1 + \frac{c_i}{s_i}\right)^{n_i}}, \quad n_i \gg 1. \tag{13}$$

An alternative is retained if the corresponding utility U is above a threshold which is set just slightly below 1. These approximations for the disjunctive and conjunctive rules become noncompensatory as n_i approaches infinity.

By iterating through the conjunctive acceptance and disjunctive rejection rule several times with adjustable critical values or

DISEÑO DE BASES DE DATOS

A R T I C U L O

INFOLOGICAL MODELS AND INFORMATION USER VIEWS

EXPOSITOR:
Ing. Daniel Ríos Zertuche

MAYO, 1985

Inform. Systems Vol. 5, pp. 17–32
© Pergamon Press Ltd., 1980. Printed in Great Britain

22

0306-4379/80/0301-0017/$02.00/0

INFOLOGICAL MODELS AND INFORMATION USER VIEWS

BÖRJE LANGEFORS
University of Stockholm, Department of Administrative, Information Processing,
Fack, S-1045 Stockholm, Sweden

(*Received 15 January 1979; in revised form 1 June 1979*)

Abstract—The study of data systems as information systems put into focus the role of data as representations of information in the sense of knowledge about some slice of the world. This information system-view—or infological view—made it clear that the data alone cannot "carry" information. They can only, at best, give rise to information in the minds of people and only in those people who hold a suitable frame-of-reference or world view, or "receiving structure", in their mind. Thus the infological perspective had to be widened successively from a concern merely with information representation, structuring, and exploitation to the study of social, socio-psychological and socio-linguistical aspects and of "object system", job design and other socio-technical issues.

The term "user view" is employed widely in recent data base work. The use of the term "user view" suggests such an "infological" perspective. However, a closer look at how the term is actually used indicates a much more delimited interpretation which focuses on representational aspects and processing. This is also made explicit, to some degree, by the usual formulation "user view of the data" rather than, for instance, "user view of the world".

In this paper a brief study is made of the two aspects of user views,

(i) the *infological/conceptual aspect*, which is concerned with how conception relates to data and information, and to reality, and

(ii) the "*datalogical*" *aspect*, which is concerned with the selection of data from a data base and the rearrangement of them to suit a "user view" of the data (as seen from the application programmer).

The infological aspect is illustrated through a discussion of some of my own earlier results which are here brought together. The datalogical aspect is exemplified by some quotations from the most recent data base literature, as well as by some earlier results of my own.

The term "user view" is frequently used in the datalogical sense whereas the infological or information-system-theoretical studies often have addressed questions that have to do with the infological/conceptual aspects of "user views", without employing that term. In this paper some aspects of the problem of infological/conceptual user views are treated with a view to gain understanding of how both aspects of user views affect the design and use of information systems and data bases. Illustrations are taken from the author's own earlier work, for three reasons: (i) they were most easily available, (ii) they are directly associated with the problem at hand, and (iii) they have earlier been scattered over several works and it was useful, for the purpose of the present discussion, to bring them together.

INFORMATION SYSTEMS THEORY (IS THEORY) VS TRADITIONAL DATA PROCESSING CONCEPTS

In traditional data processing an information item is usually assumed to be a number which is to appear at a certain part of an application program. However, when taking on to study the problem of how to design data processing from another perspective, as a function within an information system IS (Langefors[1–3]) it was found that it is necessary to look closer into how data can provide information to people and to organizations. It becomes immediately obvious that if data are what are processed by computers (or other means) then the information which people get from the data is something distinct from the data. If data may aid people in making decisions or performing actions, this must mean that the data *inform* people in the sense of making something known to them. It is clear then that the data must represent something that can also be expressed by natural language. Clearly then, in this sense, written sentences of natural language (as well as spoken or recorded sentences of any kind) are also data, that is, they are sets of signs representing knowledge or information.

It is interesting to note the perspective advanced above is similar to ideas treated by philosophers at least as far back as Aristotle, and also, more recently, by, e.g. Russel, Wittgenstein, Carnap and Bar-Hillel. It took some time, however, before these ideas were recognized as useful to the understanding of data processing systems. Also, recent development in cognitive pshychology and psycholinguistics is providing new insights into this area (Bar-Hille[4], Carnap[5], Piaget[6], Wittgenstein[7], Bower and Anderson[8], Lyons[9]).

The perspective of information systems theory and infology studies has been that to design the right data and data systems one has to begin with the questions about the information or knowledge that the data are to represent. This means that the information content of the data in the system has to be defined and described in a way that is independent of how the data are handled in the system. The information is described in terms of the views of the world that the information users hold. Information modelling, thus, is world modelling. One consequence of the data independent data description is that the data system (files, data base, programs) may be restructured without the need for changes of the information specification.

Recent data base work and structured-programming work has stressed the importance of describing the data

independently from the representational details. For instance in Codd[10], it is stated:

"The relational view (or model) of data.....provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes".

While this piece of text does not speak of the information conveyed by the data, the reference to "its natural structure" suggests a somewhat information-oriented perspective. However, the starting point here is the data and its handling by systems whereas the infological perspective is that of people and their world. Indeed, the information specified may not necessarily come to be handled by computers at all. It seems that the focus on data—rather than on information—has some consequences. Thus some of the basic concepts of the relational data base theory, such as normalization and join operations, are in fact based on arguments that are concerned with how the data are viewed as being stored and processed by the system, in spite of the expressed intention to consider the "natural structure only". Likewise, recent data base work on "user views" turn out to be mainly concerned with views on how the data are arranged for processing, as we shall see. As to the perspectives of structured programming we may quote from Jackson[23]:

"The problem environment is the real world...The customer file is a model, intelligible to the computer, of the set of customers;...."

This text also suggests an informational (or infological) perspective but, on the other hand, the phrase "intelligible to the computer" indicates that the thoughts quickly touch upon the processing of the data representing the information. As we shall see—also in this work—the view of the data and their structure is similar to the "user view of the data" as discussed in the data base works.

The intention behind this paper is to propose the widening of the conceptions of "user views of data" and the "natural structure of the data" that may be obtained by merging these conceptions with infological conceptions of information structure and the pre-knowledge of the various intended users.

We shall see that "user views" are not merely of importance to the way data are to be organized in the store. They are crucial to the possibility for the users to interprete the data. Thus, not only must the data be arranged and ordered to allow efficient processing of the data, according to some view. The data themselves will need to be changed in order to represent the same information to users with a different view. Moreover, some people may have such a view that they are quite unable to hold any information whatever data one might try to use. This, of course, is critical; it is a threat to the very assumption of large IS or large, shared data bases.

SOME BASIC INFOLOGICAL CONCEPTS AND TERMS

The basic concepts or models of information systems theory and infology are intimately associated with semantical questions and user views. Thus, before discussing some aspects of information user views we review some of these basic concepts.

Particular context of a single data term

Let's turn back to the question of how an elementary data item, such as a number, can represent information. It is clear that the traditional view, that the treatment of the data item by a certain program determines the information, cannot provide the full answer. If certain data (for instance a certain printed sentence) are to represent some knowledge of the world, this must be achieved through reference to phenomena in the world and not merely through its relation to any specific computer program. Thus, given a data item or term, for instance the number 17, how can it inform about the world?

Two kinds of informational questions arise in connection with any individual data term:

(A) What other data terms are needed in order to combine with the particular term so that together they may provide information?

(B) How may one retrieve any specific term as stored in a data system? (Retrieval may be requested by various application programs or by various users at computer terminals.) (Langefors[1-3]).

In order to discuss the two questions presented above we will introduce some terminology. We shall say that a group of data which is necessary and sufficient for the retrieval of a specific term "t" is a "*retrieval determinant*" (or key) for "t" and a group of data which is necessary and sufficient in order to supplement "t" to provide information is an "*information determinant*" or *i-determinant* for "t".

It is seen that some kind of dependency must exist between a data term and its retrieval determinant and, likewise, a dependency must exist between the term and its i-determinant.

Furthermore, we may conclude that the sources of these dependencies between the data are dependencies between the phenomena that the data refer to. These dependencies give rise to corresponding mathematical dependencies among the data (as such dependencies are studied in recent data base work).

Let us consider a specific data term, such as the number 17 or, more precisely, the printed sign "17". What does it tell us? It is natural to say that, as standing alone, it does not convey any information. On the other hand it does have meaning in the sense of referring to a concept, the concept of a number. Thus, we seem to require something more than meaning (in some sense) from a piece of information. Now, let us assume that we are told that the term "17" is a quantity on hand in a store. This attaches some more specific meaning to the term. Accordingly if the data term "17" is supplemented with the data term "Quantity-on-Hand, 17" some more precise specification of meaning is conveyed. But still we do not obtain information. Instead, if we are presented with the sentence "The quantity in store of articles of type A is 17 pieces" we feel having been informed. We might draw some conclusions from such a statement, provided we perceive what it will look like in the stock room or how we might proceed if we would want to

verify the sentence. Thus it adds to our *view of the world*. It seems then that a text string, thus a group of data terms, in order to convey information not only must have meaning, it must have a truth value also—it must state a fact and would be regarded as false if contradicted by real facts. More generally, it should present knowledge of a system state.

It appears now that the question whether some data convey information is either related to formal logic or asks about a view of the world. Furthermore, in logic there is sometimes made a distinction between a *declarative sentence* (which is intended to state a fact) and the *proposition* which is formulated or represented by the sentence. The proposition is "the cognitive content" of the sentence and "the thought of a fact" which may or may not prevail. It is clear, now, that the proposition (in the logical sense of above) is the *information to be conveyed* by the sentence and the sentence is a group of data used to represent that information (the proposition). However, even if a proposition is a paradigm case of an information unit, as we found, it may not be the only kind. Conceiveably, data might be used to represent information that is not of the clear-cut logical kind of a proposition. A view of the world may be something else than a proposition. This we do not pursue here; we only want to make the declaration that until further study has been done on this issue, we will use the concept of logical propositions merely as the simplest case of information units—but not necessarily the only possible type (Langefors[1, 3]). For this reason the term "message" (more recently i-message) for a piece of information or knowledge was chosen. The term "proposition" was seen as a special case of an "i-message". Clearly, the question of what is a piece of information would have been a lot easier if one simplified oneself to just a "proposition". Then, simply, it would have been possible to refer to textbooks in logic. But important practical IS design questions fall outside such a narrow framework so we have to be more general, for practical reasons. Perhaps it would be better to use the term "proposition" instead of "message" but then with a somewhat generalized meaning. Better still, perhaps, we could use "i-proposition" to indicate this generalization.

Elementary information units-e-messages

A message, like a proposition, may be a composition of other messages. And a view may be a composition of views. A message (or a proposition) which cannot be decomposed into several messages is referred to as an elementary message (*e-message*) (or an elementary proposition—or "e-proposition"—respectively). Correspondingly a data group (such as a sentence) may represent a compound message in which case it may be decomposed into *elementary sentences* (e-sentences) or *e-records*. An e-record or an e-sentence is data representing an e-message (according to some syntax/semantics). Obviously an e-message is "*irreducible*" in the sense described above.

We are now in a position where we could say that to find the i-determinant, that is, to determine the additional data terms needed as the supplement of any given term used (such as "17") in order to represent a piece of information, we have to identify (at least) an elementary sentence of natural language, containing the given term ("17"). However, this would require us to have the computer handling, and interpreting, natural language texts. This was clearly impractical in 1966 and it still is. For instance, natural language is highly ambiguous. Consequently, to study data one needs to have some formal data models and information models for e-messages and e-records (Langefors[3]). Because of the close relationship between e-messages and elementary propositions as well as between e-records and elementary sentences, one could turn to formal logic for help. This would need some caution, though, because it was found necessary not to be restricted, already by definition, to logical e-messages (propositions) only. Also, because it is necessary to handle information about information we cannot accept to be locked in within the bounds of first order formal predicate calculus and because the time dimension is also significant to an IS this is another reason for using a wider frame than traditional, formal logic. In spite of this, we may often find that formal logic can usefully be employed in many special cases.

Elementary sentences or e-records

Before proceeding let's look at some alternative sentences formulating the same e-message as the sentence proposed on p. 18, 2nd column. This should make it clear that some formal modelling is needed.

(1) "There remain 17 pieces of article type A for disposal"
(2) "Of article A there are 17 pieces on hand"
(3) "We are now left with 17 pieces"

We notice that the sentence (3) clearly presupposes some *particular* context (in addition to the "general context" or "frame-of-reference" that is always necessary).

From formal logic one may be led to the formal sentence, or formula,

(4) "Qty-on-Hand (A, 17)"

In (4) Qty-on-Hand is assumed to be the predicate "quantity-on-hand of a specified article type". Such formulations as (4) do not appear convenient enough for the general information system user. In addition the use of free variables and quantifiers of predicate logic appear unnecessarily complex for many data design situations (though they are sometimes hard to avoid). Also, as stressed above, it was not felt to be acceptable to be totally constrained to the use of predicate logic. Hence formal logic was rather seen just as a source of suggestions for a general information model. It did not appear to be feasible without some modification and amendment (Langefors[3]). It is worth noticing, furthermore, that the use of the common form of predicate logic statement, as in (4), calls for a particular syntax/semantics description for each individual predi-

The structure of e-messages

Based on arguments like those above it was concluded (Langefors [1, 3]) that an e-message, however otherwise formulated must *make known*, one way or the other:

(a) What object or entity it is intended to inform about (the "Subject" of the e-message)

(b) What it makes known about that entity, the characteristicum part (or property part) of the e-message

(c) A time during which the Subject holds the characteristic

(d) Because the above states the minimum content of a message it is important also to assign a name (identifier) to each e-message (so that other messages may be provided which add information about this message).

It is important to recognize that *how* the aspects (a), (b), (c), (d) are made known is left totally open. Thus any e-message may be structured in many ways as long as the three basic references (*Subject Reference*, *Characteristicum Reference*, and *Time Reference*) are established. A group of data that may be used to make known the e-message is an *e-record* or an *e-sentence*. Obviously many different e-sentences may be used to represent an e-message. *Remark*: The individual data terms or items, that make up an e-record may be put in distinct places in the "physical data structure". When we want to remind on this fact the term "e-entry" may be used, rather than "e-record" (borrowing "entry" form CODASYL).

If one looks at one of the example sentences above: "Quantity-on-hand of article type A is 17 pieces" it may be concluded that the object it informs about is "article type A" and the property or characteristic that it states about that object is "quantity-on-hand = 17 pieces". Furthermore, the time stated (implicitly) is "now". Thus we may specify, more formally:

(1) e-message identifier: em 1
Subject: Article-Type, A;
Characteristic: Quantity-on-Hand, 17 pieces;
Time: Present;

This structure may be presented in many other ways provided, of course, that the form used is made known through some sort of format statement. Some such forms may be, as shown below, Nos (2–4).

(2) Format: Identifier: Subject Reference; Characteristicum Reference; Time Reference;;
em 1:
Article-Type, A; Quantity-on-Hand, 17; Time, Present;;

or

(3) Format: Identifier: Characteristicum-Type (Subject Reference, Time Reference, Value);
em 1:
Quantity-on-Hand (Article-Type, A; Time, Present; 17 pieces);

(4) Format: Identifier: Characteristicum-Type (Object Class, Time-Type)
((Obj-id); (Time-Value); (Quantity));
em 1:
Quantity-on-Hand (Article-Type, Time, Value) (A, Present, 17 pieces);

It may be noted that each of the message-representations (or sentences) presented above are in a sense both complete and minimal). They are minimal (or irreducible) in the sense that if any of the terms is deleted (hence ignoring the associated reference or knowledge) then that which remains does not provide information. The reader should verify this for himself. In our particular illustrative case this means that no logical proposition is represented when any single term is deleted. The messages are complete, each of them, in the sense that they make known something that may be true or false and that thus is a proposition. For instance if the fact is that at present there are 18 pieces in inventory, of article type A, then the proposition is false. We may thus say that any of the message representations above (1, 2, 3 or 4) defines a complete minimum supplement to the single term "17" and, thus, makes known the information which "17" takes part in. In other words, any of the e-message representations above defines an i-determinant of the term "17".

It follows from the discussion that any of the above e-message representations (as printed) can be regarded as an e-record for the e-message em 1. Any other group of data which can be formally mapped into any e-message representation for em 1 may be used as an e-entry for representation of em 1 in a data system (provided the mapping procedure is defined).

The e-message representation No. (4), above, has a form which has separated the variable terms as the "value tuple" (or "tuple" for short): (A, Present, 17 pieces). It is seen that any e-message (and e-record) contains a relation instance as a proper part. Thus any e-message type defines a (data base) relation (once value domains have been chosen).

Illustration of some basic concepts

Infological studies call for a large number of concepts, as a result of analyzing the concepts of information into the aspects: representations, conception, and reality and into the type and instance levels as well as into distinct levels of aggregation and definition. All the resulting concepts have been found necessary. They are treated explicitly, almost one by one, in the systematic information analysis. They call for a corresponding, large number of terms. Of course, this requires some mental effort from the analyst before he can make use of the concepts, as the reader will have noticed. This is a reflection of the complexity of the problem of infomodelling. To keep down the number of entirely distinct terms a systematic set of prefixes has been introduced. It may be useful to illustrate the employment of the terms by means of an example, as an aid to the user's comprehension of the infological frame of reference employed here.

The natural-language sentence

M1: "The customer C ordered 5 pieces of article A today" may be interpreted as representing a real-life event or fact. If that is so the sentence M1 may be said to convey a piece of information which is the knowledge of this fact. The fact, in this case, is an event in the real world. The piece of information is referred to as a *message*. (In this specific case it is a *proposition* in the sense of logical texts.) A structure of data that may be used to *represent* a message is a *record*. The natural-language sentence M1 is thus a record, by definition. The terms record and sentence may, in fact, be used interchangeably, as a consequence of the definition. The sentence M1 may now be regarded as "referring to" the event in reality as well as to the information. In other words, M1 refers to the fact (reality) and to the message (information)—(as well as to the sentence). Any message may be represented by several distinct sentences or records.

The intended users of the record M1 may conceive of the fact informed about as a composition of two facts which may be referred to by the two sentences:

M2. The customer C ordered article A today.
M3. The quantity ordered in the event M1 was 5 pieces.

Then the fact is a combined fact or a *consolidated fact*. The corresponding message and sentence is then also regarded as a *compound message* and a *consolidated record*, respectively. If the users do not regard the facts represented by the sentences M2 and M3 as compound facts, these facts are regarded as elementary. Then the messages and the records (or sentences) are also elementary. Because we often need these distinct terms, some abbreviations were introduced. Thus the prefix "e-" is used to indicate "elementary" and "c-" is used to indicate "compound" or "consolidated". Thus M1 is a *e-sentence* (or *e-record*) used to represent a *e-message* which is the knowledge of a *e-fact*. The sentence M2 and M3 are *e-sentences* (or *e-records*) representing e-messages about *e-facts*. The e-message represented by the e-sentence M2 may be regarded as information about "customer C", but from another perspective it may instead be regarded as information about "article A" or, indeed, about "today" or about the group (Customer C, Article A, today). The analysis of the c-sentence M1 into the e-sentences M2 and M3 made it clear that the part "5 pieces" of the sentence M1 informs about the event represented by the rest of the sentence (or represented by M2) whereas the part "ordered article A today" informs about the customer C. (This analysis is, of course, essentially a kind of parsing of the natural-language sentence 1.)

The e-sentence M2 refers to "customer C" as its subject (=entity informed about). The part "customer C, today" i-determines the part "ordered article A", that is, it makes known what information the characteristic "ordered article A" belongs to (or participates in). Similarly, the e-sentence M3 has "event M2" as its subject-plus-time reference and as the i-determinant of the characteristic "the quantity ordered . . was 5 pieces".

When we want to refer broadly to both e-records and c-records we use the prefix "i-". Thus e-sentences, or e-records, and c-records are examples of i-records (or sentences). In connection with the term "sentence" it is not necessary to employ the prefix "i-". Analogously M1, M2, and M3 represent a collection of i-messages (or i-propositions). M1, M2, and M3 represent a collection of i-messages (or i-propositions). M1, M2, M3 were formed according to (a subset of) the *rules of formation* of natural language. Other rules of formation were used in

M1a: (Customer, C; Date, D; Order, Article, A) Quantity-ordered, 5 pcs::
M2a: (Customer, C; Date, D) Order, Article, A::
M3a: (Event, M2a) Quantity-Ordered, 5 pcs::

The formation rule here is such as to explicitly reflect the reference structure of the i-messages: the subject reference, followed by the time reference and (then) the property (or relationship) reference. In addition, the important change was introduced that the time reference was made absolute. This, of course, is necessary when the records are to be stored. The brackets are used to enclose the part which provides both subject and time reference. For example, M2 and M2a instatiate two e-record types and may also be seen as representations of two e-message type instances, both of which are the same piece of information. The e-message represented by M2 or M2a is an instance of an "elementary information-kind" or *e-concept* (in this case an elementary predicate). This e-concept may be described by the e-concept schema: (Customer, Date, Order = Article). We may represent the e-message by yet another e-record format which reflects the e-concept:

M2b: ({Customer, Date} Order = Article) (C, D, A)::

In this e-record format the e-concept schema appears in front of the "value tuple" and all e-messages "belonging to" the e-concept may be represented by a set of such tuples, each tuple being coupled to the e-concept schema. (Here the linkage between e-concepts and relations is seen.) In this connection it will be noticed that the i-determination of "ordered article A" by "customer C, today", mentioned above, gives rise to a (multivalued) dependency $(\bar{C}, \bar{D}) \rightarrow\!\!\rightarrow \bar{A}$, where $\bar{C}, \bar{D}, \bar{A}$ are domains such that $C \in \bar{C}$, $D \in \bar{D}$, $A \in \bar{A}$. The e-concept M2b is seen to give rise to a 4NF relation on its value domains. This is typical of e-concepts. If we introduce the additional e-records

M4a: (Customer, C; Date, D; Order, Article, A) Delivery-Address, Ad::
M5a: (Customer, C; Date, D) Amount-Due, Amt::

we find that M1a and M4a have identical subject parts and so have M2a and M5a. We may thus consider, during the data design, consolidating M1a with M4a and M2a with M5a:

M1-4u: (Customer, C; Date, D; Order, Article, A) Quant-Ord, 5; Deliv. Addr, Ad;;
M2-5u: (Customer, C; Date, D) Order, Article, A; Amount-Due, Amt;;

or

M1-4b: ((Customer, Date, Order = Article) Quant-Ord, Deliv-Addr) (Ç, D, A, 5, Ad)
M2-5b: ((Customer, Date) Order = Article, Amount-Due) (Ç, D A, Amt).

These c-records give rise to "c-concepts" (or consolidated i-concepts) as described by the left parantheses. By definition a collection of e-records (or c-records) is an e-file (c-file), provided that the e-records or c-records are of the same type and, hence, associate with the same e-concept or c-concept, respectively.

The c-record M1-4b is a representation of a c-message instance of the c-concept

((Customer, Date, Order = Article) Quant-Ord, Deliv-Addr)

obtained from the e-concepts

((Customer, Date, Order = Article) Quant-Ord)

and

((Customer, Date, Order = Article) Deliv-Addr)

through consolidation on the common subject part

((Customer, Date, Order = Article)

This consolidation gives rise to a join on (C, D, A) of the two relations (C, D, A, 5) and (C, D, A, Ad). Similarly a c-concept

((Customer, Date), Order-Article, Amount-Due)

is obtained through consolidation of the two e-concepts having (Customer, Date) as their subject parts. The consolidated (or joined tuples) (Ç, D, A, 5, Ad) and (Ç, D, A, Amt) belong to two relations that are still 4NF. This is partly due to the fact that the consolidated e-concepts all have identical subjects. Instead, if one would, e.g. consolidate all the e-concepts mentioned above the resulting relation would not even be 1NF.

INFORMATION, CONTEXT AND SEMANTIC BACKGROUND

Pieces of information, i-messages, are knowledge of particular facts. They must be constructed from references to aspects of these particular facts, as we have seen. But this could only be realized if the receiver of the information has already some background knowledge. Some aspects of this background knowledge are presented below. *Particular* information and *general, pre-existing* background knowledge appear as the two basic components crucial to all information use, communication and data processing.

The need for a general context or semantic background

We stated that any of the e-message representations above could be regarded as complete in a sense. Now we want to make this sense more specific. Let's do this for the e-message format No. (3). To receive the e-message (or to establish it in our mind) as represented by the format No. (3) (Quantity-on-Hand (Article-Type, A; Time, Present; 17 pieces)) we need to have the pre-knowledge of what "Quantity-on-Hand" means as well as what is meant by "Article-Type", by "A", by "Time", by "Present" and by "17 pieces". In addition we must also know the meaning of "Format", "Identifier", "Characteristicum-Type", "Subject Ref.", "Time Ref." and "Value" (Langefors[3, 11]).

It may now be seen that "completeness" of an e-message representation is taken to hold in the sense that those *particular* references are made known that could not be known beforehand. For instance, the particular knowledge that this e-message informs about "Quantity-on-Hand" could not be known in advance. Contrary to this, the general background knowledge of what quantity-on-hand means can and must be known in advance. It is "pre-knowledge" needed for the communication and the usage of the e-message. It is the "user view" for which the data design has to be made.

The result of our discussion so far can be summarized by the statement that the information that may be conveyed by a set of data, D, depends on the person receiving the data in that the semantic background or "receiving structure" S of the person is crucial to interpreting the data. In addition, the time available for the interpretation is also significant. All this can be expressed by a concise "conceptual formula" (Langefors[3]):

I = i(D, S, t)
D = data representing the intended info
S = the "receiving structure" (pre-knowledge) of the user
t = the time available to the user for interpreting the data D
I = the information conveyed by the data D
i = the information function

Another important insight gained from the discussion is:

Ia Data *are not* information and
Ib Data *do not contain* information
Ic But data *may only*, at best, *convey* information to such people the conceptional fram work of whom are both consistent with the data representation chosen *and* with the model of the part of reality being involved
Id If the amount of data is large compared to available time, t, no information may come to be conveyed.

A consequence for the information analysis is that in addition to identifying the information needed and defining a data representation of it, (D), one will also have to obtain some knowledge of the "users" infological model S. Alternatively, one will have to verify that the designed data (D) comply with the world view and the language of the intended user

It may be concluded that once the distinction information/data is recognized it is immediately found that user needs for information and user world views both are to become involved when data are to be designed. But this does not necessarily imply that the user view (S) will have to be represented in the data system, nor does it have to be formally defined. It is only necessary that the data are adapted to the views of the users and that it is properly verified that the users understand the data. If the users understand the data (which can be tested), then this means that their world view has been "involved" correctly in the data design. On the other hand, it may be of interest to represent part of S in their minds. In this way the need for information/data dictionaries and information management became apparent too.

We shall see later that the term "user view of data" is used in recent data base works in a way that is only remotely similar to the concept of "view of the world" or "semantic background" (S) as discussed in the older information systems theory and presented above. We shall try to identify the drawback that is associated with this.

The infological background, or user view, as a system model

It has been seen that the information conveyed by some data depends on the semantic or *infological* framework of the user. It has been stressed that such an infological background must be a system model of (part of) the system in which the fact to be informed about appears (Langefors[11]). Such an infological system model may be in the mind of the user but it may partly be represented by data available to the user on documents or on a computer terminal. This may make it essential to supplement the e-record (e-message representation), which is presented by the system, with the identifier S' of the relevant stored system model. An example may explain this (Langefors[11]) (see also Langefors and Samuelson[12], Fig. 1). For instance, it may not be enough for the receiver of the message to have a dictionary which tells what "Boiler 5" means. He may need a model of the system to see what effect the observed temperature may have and, thus, what decision is called for. On the other hand, any system model can always be constituted by a set of e-message representations.

It is easy to see that "understanding" some data (one e-record for instance) may mean different things. In the minimum case it will imply merely the imagining of the e-fact as the observer perceived it. This would mean that the user of the data would conceive of the part of the world that was perceived by the observer who originated the message. This seems to be the only "understanding" which is generally requested from e-records stored in an IS or data base. It will require a small piece (S1, say) of the semantic background S only. In more demanding cases "understanding" may mean, in addition, a capability of predicting some effect of the observed e-fact.† However, distinct users may be interested in distinct effects. This implies that they all need to possess the "view" S1 while they may then hold distinct additional views, S2 or S3 for instance. It would seem appropriate to regard the "view" S1, basic to all usage of certain data, as the minimum required user view for these data (corresponding to it is the "conceptual Schema" of ANSI[13]) whereas the additional models Si (i = 2, 3, 4 etc.) might rather be regarded as distinct ways of applying the same information (i(D, S1, t)). (They correspond to (or, rather, are associated with) the "external Schemas" of ANSI[13]). Now the common use of the term "user view", or "world model" is of the kinds illustrated above by S2, S3, etc., and thus they ought to be regarded as an *application procedure view* rather than as a user view. The distinct applications, associated with S2, S3, etc., will process the same data (use the same pieces of information) in distinct combinations and sequences. This does not necessarily imply that the data *mean* distinct things to the distinct users though these may infer distinct consequences from the same information, because they have distinct problems to solve.

The "view" S1 will typically be associated with objects and object-relations in the real world (the object system) as well as with names and frames of reference associated with these objects and relationships. But notice that this does not mean that all these objects and relationships must be separately modelled in the data base. Rather, whenever a user specifies an e-concept (or e-message type) to be represented in the system he introduces one or more object classes and a property or relation as well. This will thus appear among the data (D) and does not necessarily need to be further represented as data (D(S1) that represent a model S1.

Remark

It should be noticed that the insight that the interpretation of a record (for instance, a sentence) depends on the world view S of the "user" does *not* imply that the view S must be explicitly modelled in a schema, for instance. The only thing that it implies is that any record may only convey information to *some* users and, hence, it is important to ensure that the data (the record) are designed with proper attention given to the views, S, of the intended users. How to verify this is a problem that is open to further research. It should not be treated by tacit assumption.

Information dictionary and information management

An important consequence of the crucial role of the user view (S1 for instance) is that all those who are to share some common data must hold the same (basic) view (S1), at least approximately, whereas they may hold distinct application procedure views (corresponding to what is commonly referred to as "user views"). It follows that it is necessary to manage to have all data terms or names defined in a common, authorized information/data dictionary. This dictionary has to be verified by the relevant user groups and thus it must be made consistent with the relevant user view (such as S1 for

† To an "action oriented" social scientist, inferring the reasons or "motives" behind observed behavior may be the aim, rather than predicting effects.

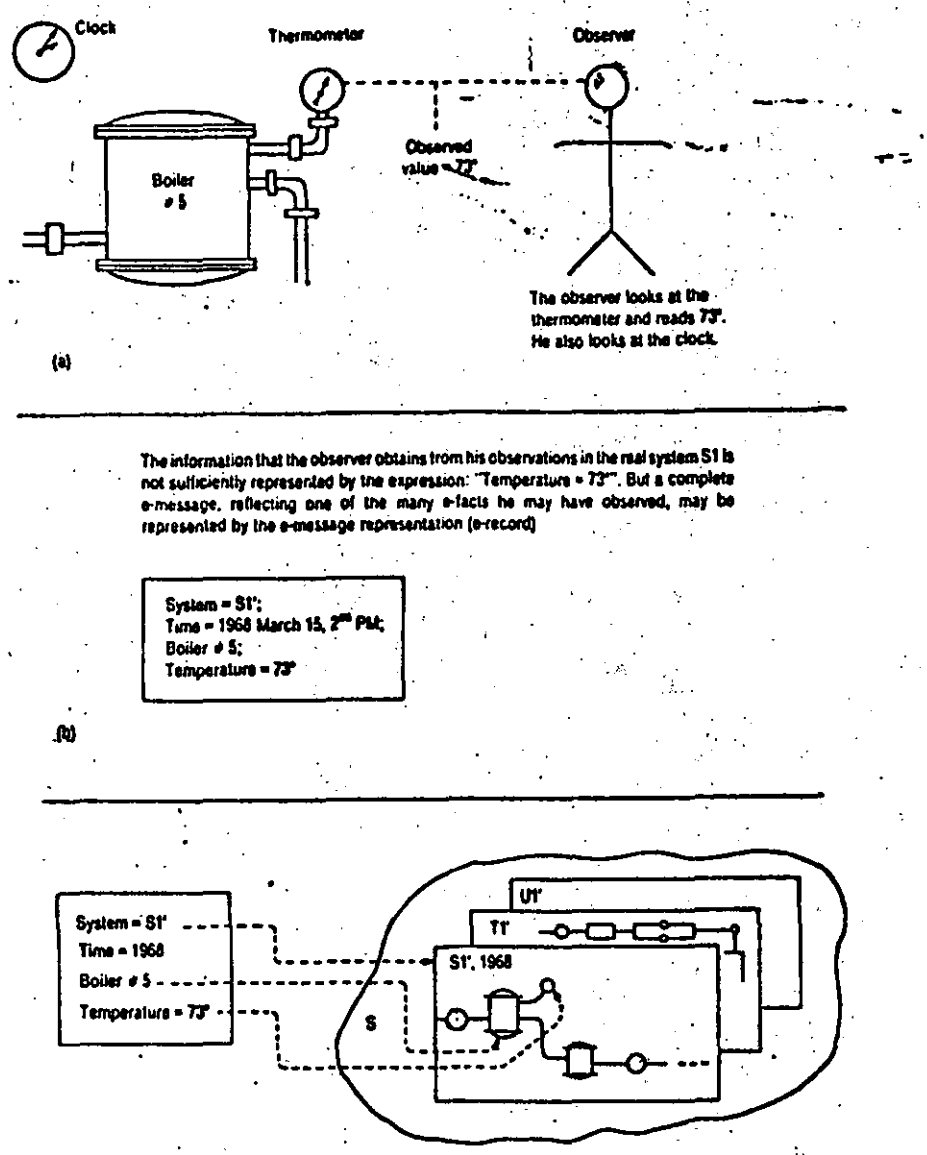


Fig. 1.

instance). Consequently, any time that some e-message type is to be documented as required information, all the names involved must conform to the dictionary. However, it will often happen that new names are needed, when new e-message types are introduced. It will then be necessary to have institutionalized a managerial procedure for how to introduce and authorize new names. These names will, typically, be names of object classes or property types. Thus it is seen to be necessary for any IS design process to have a well established information management procedure (Langefors[14]). This need for data management and information management functions has also been recognized, more recently, by (CODAYL[15]) and (ANSI-SPARC[13]). It is desirable to have the information management function supported by a computer (Langefors[14]). Of course, the information dictionary will have to contain entries not only for object classes and property types but also for e-concepts and other i-concepts. Furthermore, not only will the meaning of each name have to be documented but various qualitative and quantitative characteristics will be put down and serve as a basis for data and process design as well as for integrity constraints.

Purposive information background

We have discussed how the understanding of data, receiving the information, depends on the personal world views, S. This is a kind of cognitive background—what the data users know of the world. But the information received will be used to support decisions and actions. This, however, does not only involve factual information but also goal or purpose information. In management information system design one is interested in providing goal information, in addition to the factual information, in order to influence or control the decisions or actions. However, as shown in (Langefors[24]) decisions made by people are necessarily dependent on their personal "purposive inclination". Thus, human beings have internalized goals that they are usually unaware of and that

automatically influence their decisions. So decisions cannot be totally controlled by explicit goal information and the expected behavior of data users is thus not only dependent on their cognitive world view (S) but also on their "goal views", or intentions. This is testified, for instance by the fact that people normally make decisions without making explicit the underlying goals or aims and, even, without being aware of them.

The reference parts of an e-message and the infological background

We stressed that the reference parts of an e-message must *make known* what they refer to but that this may be done in various ways. We now may infer that how it may be made known will depend on the structure of the real system and on the infological system model (S) available to the users. We may illustrate this fact in connection with the earlier example

em1: Quantity-on-Hand (Article-Type, A; Time, Present; 17 pieces)

First, the time reference can obviously not make known the time it refers to if the e-message representation em1 would be stored without supplementary time reference data, in a data base.

Secondly, the Subject reference Article-Type, A; does not make known, by itself, the object it refers to. First we notice that we probably have to do with an object which is, itself, a collection of objects that are articles of the type A. Furthermore we guess that this collection is stored somewhere. Suppose this collection is stored in Store No. (1) and that such articles may also be stored in Store No. (2). Now we must conclude that the expression "Article-Type, A" does not, alone, make known that which was intended. It must be supplemented to "Article-Type, A; Store, No. 1;" and the infological model of the users must be compatible with this.

You may now ask whether the needed supplementing should be done by the analysts or by the users. Obviously, the analysts could only do this if they know that there is more than one store. Regardless of who does the specification, if it has been verified by the relevant users that the object reference chosen for the e-message formulation does identify the intended object, then the data can be designed. It is not required that the data system has a stored system model (if the data system is not an artificial intelligence system). The data system will associate data objects with the object identifier chosen and the users will associate this identifier with the real life object as defined by themselves.

SOME DIFFERENT ASPECTS OF USER VIEWS

The information that may be conveyed by data depends on the view of the world, held by the users of the data, as well as it depends on the language that these people use to express the views. This is one of the basic points-of-departure of infology and information systems theory but, also, it is a focal point of hermeneutics, the "science of understanding". We present some aspects provided by these fields of study. More data system-oriented studies such as data base theory and structured-programming theory speak of "user views of the data" and thereby consider how the same data are processed distinctly in distinct applications. The aim of this paper has been to compare—very briefly—these distinct perspectives of "views".

Abstract IS, real-world model and user view

The e-messages correspond to e-facts in the world. Hence the abstract IS (AIS), consisting of all e-messages, e-algorithms and e-concepts, forms a model of a selected part of reality. This was the view utilized in early IS-theory (Langefors[1, 3]) and it still is a basic infological view (Langefors and Sundgren[16]). This view stressed, as we have seen, that the abstract IS (or infological model) (AIS) must always be based on another world model, the semantical background (or collection of personal world views) S.

The abstract IS, AIS, will be assumed represented partly by data (D) and programs, in the typical IS design case. Contrariwise, the personal world view collection, S, will not be represented through data in the typical case. It is only necessary that it exists in the minds of the IS users or, more precisely, that relevant parts of it are invoked at relevant times.

The data base literature often ignores the term "information", in its meaning of knowledge as used here. Instead one speaks of the data base and of a "real-world model" assumed to be represented by the data base. It should be observed that the "real-world model" of the data base literature corresponds to our abstract IS (AIS) and not to the "semantic background system" S. Because the distinction AIS/S is never articulated in the DB literature it is usually quite confused as to what should be represented in the data base. As a consequence, it is often taken for granted that all of the real-world model (thus both AIS and S) ought to be put into the data base. This is obviously both undesirable and impossible. Behind any representation of a world view, such as S, there will always have to be another world view S'. Thus, regardless of how much is put inside the data base, there must always be a worldview outside it, available to the users.

Changing personal views, a concept of concepts

A special problem with the importance of user views for the interpretation of data is that personal views change continually. This is sometimes intended, sometimes not. In fact the IS itself may well contribute to this change. As a result, any IS will tend to loose some of its relevance over time.

One example of an intended change of view would be the possible effect that the efforts on reshaping the usage of the term "user view" might have upon some readers of this paper.

The phenomenon of concept formation and change is, of course, a profound psychological phenomenon. However, as shown in (Langefors[17]) it is possible to explain much of it based solely on simple data modelling

and assuming the saving of memory space to be the main criterion guiding the concept forming process.

To illustrate, we bring an extremely simple example. Suppose we have an IS which receives and stores information as represented by e-records such as these:

TOM has a head;
NAUTILUS has a hull;
TOM has a body;
BILL has a head;
HARRY has an arm;
TOM weighs 150 pounds;
BILL has 2 legs;
etc.

The IS compiles the e-records into internal data structures as depicted in Fig. 2. These structures will be referred to as data objects. A pattern analysis algorithm operates on these data objects and extracts structures that are found to be common to several data objects. Thus new data objects get defined and names are assigned to them. They will be referred to as *concepts*. From the data base associated with Fig. 2, a concept data object called Person is obtained from the algorithm, Fig. 3.

Notice that the concept object, Fig. 3, has one constant or closed, part named Person-Basic and one open part called Person-Form. The closed part, Person-Basic, contains a set of complete, or closed, property references (such as "arms, qty, 2;") whereas the open part contains names of property types but leaves the property values open (such as "weight, pounds ();"). The open concept data structure part allows real-life objects (called persons in this case) to be regarded as instances of a concept even though they have distinct values for those properties listed in the open part. (For instance, persons may have distinct weights, nevertheless they all have the common characteristics of having some weight.)

Of course, the open property values will usually be constrained to stay within specified bounds (for instance, persons must have weight values between zero and 400 pounds). Such defined bounds serve as integrity constraints for the data base.

It is interesting to see that already now we may recognize some properties of the concept data objects that have significance for IS design. Thus, the concept objects (such as in Fig. 3) have a degree of permanence which makes it possible for people to keep them avail-

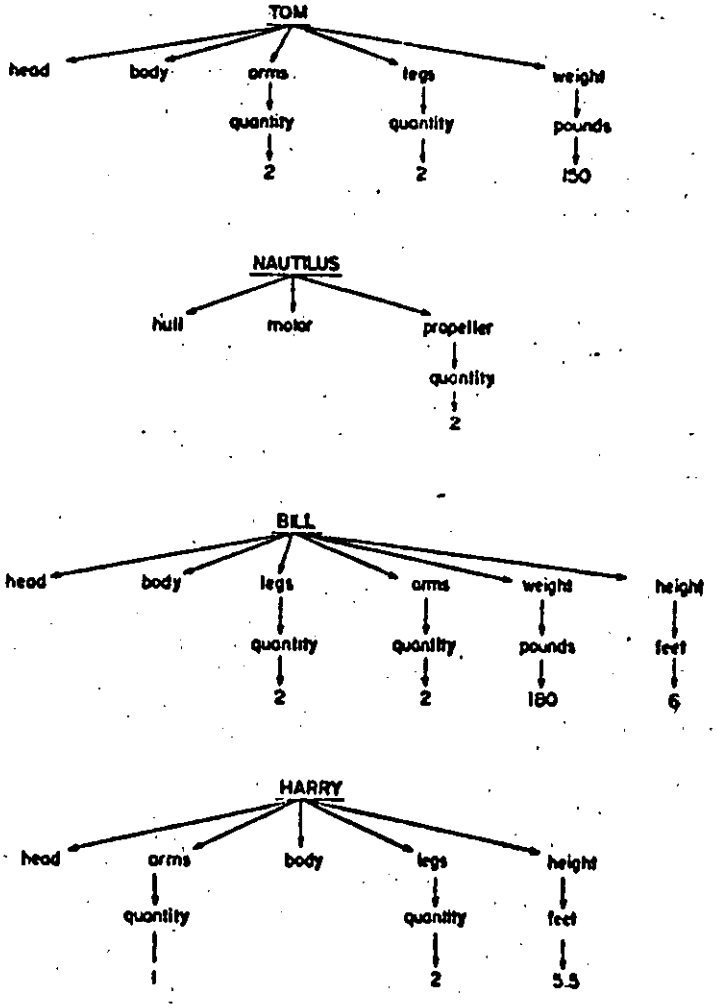


Fig. 2

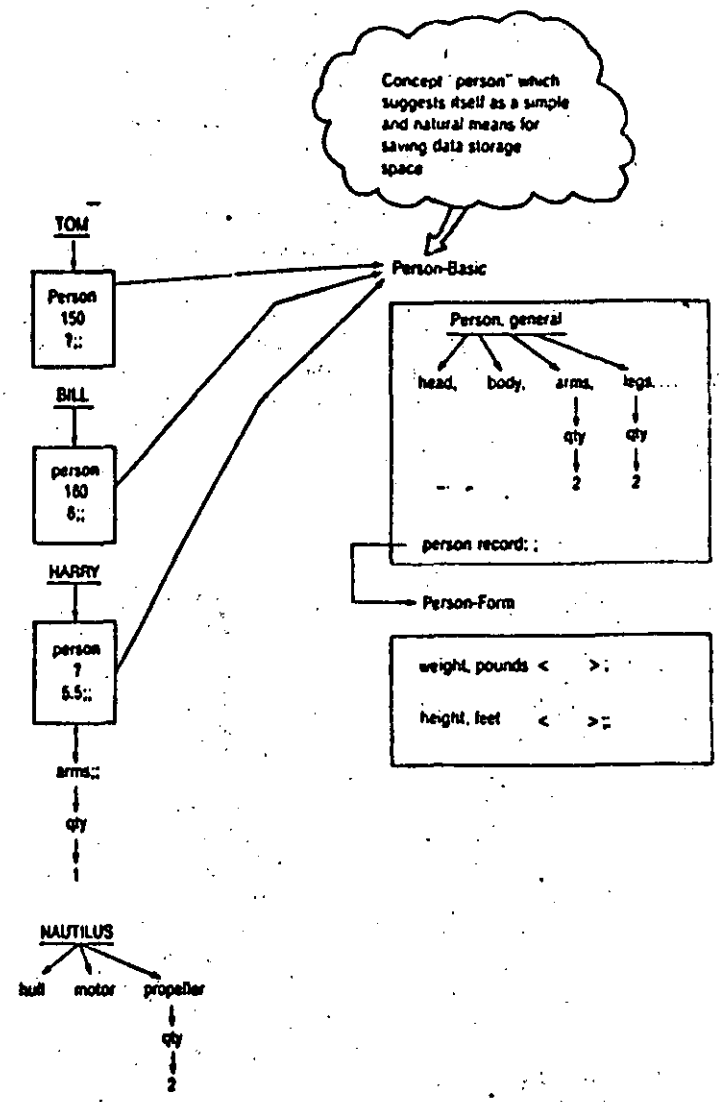


Fig. 3.

able. They are obviously closely associated with the "semantic background" or "user view" S as we discussed earlier. Instead, the open value positions represent information that cannot usually be available to people in advance. It is that information which calls for particular messages and, hence, needs data records (such as sentences) for its communication. Furthermore we now also see what *particular context* is needed with any single term before it can convey information. This particular context, we can now see, will have to identify the concept object that the value belongs to as well as the proper place of the value in that concept object. We may now also conclude that the open parts of concept objects correspond, roughly, to data record types in data bases. Finally, the names, or terms, appearing in a concept object must be names of other concept objects that are available to the information users, in their minds or through some external means.

The concept objects, clearly, form semantic systems or networks because the property references that they contain refer to other concept objects.

The introduction of the "concept-data, objects" may be justified solely by the saving of memory space (in human memory or in computer) that they bring about. For instance, the data object Tom of Fig. 2 which we may write as

Tom (head; body; arms, qty, 2; legs, qty, 2; weight, pounds, 150);; may be substituted by

Tom (person; 150;?);;

if it is assumed that the value terms following a concept name, belong to the open value slots of the open part of the concept object and "?" is used to denote a missing value specification (which thus leaves a value still open).

In general we will encounter data objects that are at most partly consistent with a concept object. The difference may be that the object has another value for a property than one specified in the closed part of the concept. For example, the data object Harry, in Fig. 2, specifies one arm only. Alternatively, the object may be lacking a property reference altogether or it may contain property references not contained in the concept object. For such object a reference to a concept may still be used if those (exception) property references that deviate

from the concept structure or content are added. For instance the object Harry in Fig. 2, may be written

Harry (person; ?; 5.5) arm, qty = 1

To decide whether a deviating data object should be regarded as associated with a concept (while supplemented with exception data) or not, one may assess the resulting data space requirement.

For any given set of data objects—any given data base—and any established concept object, an algorithm may identify all those data objects that may become reduced by referring to the concept (as well as those being increased). The total saving may then be computed. Now this may be repeated for a number of modified concept objects so as to find the concept object which is most efficient in reducing memory space. It is clear that this procedure will not have a unique result so that some strategy for choosing one solution out of a set of equally efficient solutions will have to be created.

Obviously, when a number of concept objects have been created one may apply the same procedure again to create concepts that reduce groups of concepts. Thus a hierarchic concept structure will emerge. The model presented may be seen as a possible, simplified model for how human views of the world are developed.

The concept of "concepts" based on memory saving only, appears to be something much more simplistic than "concepts" as commonly thought of. However, several known characteristics of human conceptions are reflected by the model, simple though it is. Hence it is of interest to look at some statements that may be derived from the model:

(1) The definition of a concept will not be unique, several concepts may have approximately equal efficiency.

(2) Whether a certain object should belong to the extension of a certain concept or not, is a question which may not have a unique answer—hence concept logic seems to be three-valued (true, false, not decidable). It is of interest to notice that this is similar to the answer possibilities of statistical decision theory.

(3) Concepts will change as the subject will experience new perceptions.

(4) Distinct persons may generate distinct conceptions even when they have the same experiences.

(5) There is no fixed set of properties that all instances must possess, rather, any instance must have enough many properties in common with the concept.

It seems likely that these five statements, suggested by the very simple model, would be still more relevant the more complexity is added to the model. Hence the statements appear to be generally true. All these conclusions are clearly consistent with experience and thus this experience may be explained by the simple assumption of a storage economy criterion. It seems also clear that this model gives a lot of guidance for the consideration of user views and user conceptions in IS design and operation. It is worth noticing, especially, the indication of concept variation with time that has to be reckoned with, according to the model. Any system containing automatic concept forming mechanisms will change its set of concepts continually, as new information is received.

A famous work which is concerned with some aspects of concept changes and, hence, "user views" and their importance is (Kuhn[18]). Kuhn uses the term "paradigm" in several ways, some of which are related to effects of varying infological/conceptual "user views".

Hermeneutics, infology and "user views"

Infology and IS design are concerned with how data may be related to the world and to information about the world. Hermeneutics is a field of inquiry which is explicitly devoted to how human beings can understand or interprete behavior or texts such as historical or law texts, for instance. Thus infology and hermeneutics clearly have common interests. Hermeneutical work is usually presented in a verbose and entangled prose. This often makes it difficult to know whether anything can be contributed to IS design and, indeed, it is natural to a scientist to question whether hermeneutical texts carry any precise meaning at all. But once the close relation between IS-analysis and hermeneutics is recognized, it becomes desirable to seriously look into hermeneutics. Now hermeneutics itself is perhaps more concerned with injecting meaning into a text (Sinngebung) than with extracting meaning out of it (Auslegung). This of course, implies that the question about precise meaning becomes somewhat irrelevant. Thus, in reading hermeneutical works one should try to see if it is possible to make sense out of them through explicit, constructive efforts of interpretation. In any case, it is clear that the question of user world views and data understanding may be regarded as a hermeneutical question. For this reason it appeared desirable to try to see if it was possible to attach concrete, infological meaning to selected sections of hermeneutical texts and to try to find out whether that field might be useful to IS design. This little study turned out to be reasonably successful (Langefors[19]).

Hermeneutics stresses the critical importance of the "pre-understanding" (Vorverständnis) held by the subject. A typical hermeneutical view is the one (by Gadamer) which claims that interpretation is not merely a looking for an objective content in the text (which may never have existed), nor is it merely a looking for the true intentions of the author (which he, himself, may not have been aware of). It is rather something "belonging to the current event of interpretation" and of the "being-in-the-world" of the interpreter. It is certainly not easy for a scientist to attain a Gadamer kind of "being-in-the-world" and to understand in that sense, but however that may be, one is clearly concerned here with "user views" of some kind.

For a more penetrating discussion let's pick up another piece of hermeneutical text and try to attach infological meaning to it (Langefors[19]). The text is from (Apel[20]) (my translation from German).

"A language-hermeneutical analysis assumes that the understandable human behavior responses contain, themselves, the property of understanding, being, as they are, language related intentional images. This

analysis has to conclude that the world knowledge (Weltdaten), against which the behavior is to be understood, must itself be understood against the understanding of this same behavior".

This piece of text may, perhaps, be watering the mouth of a recursion-loving computer scientist but to "unbundle" this spaghetti phraseology does not appear to be easy. But this is what should be tried.

To bring the problem within the grasp of infology we may start by assuming the simplest possible case when the human behavior to be understood is a single e-fact (an "elementary event" in this case) so that the understanding would give rise to an e-message ε. The observed behavior may now be regarded as the data ẽ representing the e-message ε. We may now put down the infological equation

$$\varepsilon = i(\tilde{e}, S, t)$$

which says that the information (understanding) ε will be obtained through a process i(ẽ, S, t) which works on the data ẽ, and employs the pre-understanding S during a period of time t. Now, once the knowledge ε has been generated, the pre-understanding by the subject may change to S' = S + ε. This symbolization is, of course, merely to be understood in an intuitive sense but it is not hard to imagine how it could even be modelled in a computer and thus be made very concrete and formal.

We are now led to the new equation

$$\varepsilon' = i(\tilde{e}, S + \varepsilon, t)$$

which indicates that the knowledge ε'—which is generated by a reiteration of the same process (i) from the same observation (ẽ) but with the new knowledge background (S + ε)—might be different from ε. This corresponds, possibly, to the text "...must itself be understood against the understanding of this same behavior". However, if we imagine a computer simulation of the equation ε = i(ẽ, S, t)—which we do in order to try to test our understanding in a concrete way—then we would think of S as being represented by a network data structure, S̃ (e.g. a semantic network), and the understanding process (i) would establish an integration of ẽ with S̃. Thus ε would be represented through S̃ + ẽ. Then it would be natural to a computer scientist to assume that the model would be such that the repetition of the process (i): ε' = i(ẽ, S̃ + ẽ, t) would change nothing at all; that is he might assume i(ẽ, S̃, t) = i(ẽ, S̃ + ẽ, t). However, it is easy to conceive of a change of the above model. In fact, the equation itself suggests a more complex model because the effect of time for interpretation (t) ought to be represented also. A modification which suggests itself is to assume S̃ to be stored on an auxiliary store and to be searched by the process (i) which brings relevant parts of it into main memory (corresponding to areas of higher awareness in the human memory). Thus, the time constraint (t) will restrict the time allowed for search in the backing store. In this modified model the part of S̃ + ẽ that resides in main memory may well be distinct from (and more powerful than) that which resulted in the first execution of (i). Correspondingly, ε' would be distinct from ε and would be associated with a semantic network, more powerful than the previous one with respect to the understanding of ẽ. Now, clearly, the modified model does *construct* sense to the piece of text by Apel that we are discussing, though it is doubtful whether this constructed meaning was intended by the author (Apel). Nevertheless, our hermenutical exercise appears to have been quite successful in generating some deeper understanding (as well as suggesting some interesting research of an artificial intelligence kind).

The phenomenon that we have just discussed is, incidentally, a simple example of what is sometimes referred to as the "*hermeneutical circle*".

The interpretation experiments made above indicate, through their positive results, that hermeneutical texts should be given some extensive studies by information analysts. The authors of such texts may not be aiming at a very concrete, engineering type of understanding. Nevertheless, such a concrete understanding was easily possible to reach in the above interpretation experiments. Furthermore, this became possible through explicit hermeneutical efforts of constructing meaning that did fit in this attempt. The outcome of the efforts was a deeper understanding of the process of interpretation. It seems reasonable to expect that when the small extracts of hermenutical texts, used in the experiments, turned out to be quite helpful, more useful knowledge should be obtained from more extensive studies. Because hermenutics is very much a matter of "views" of authors and their readers it appears to be potentially advantageous for more extensive studies of "user views" as related to the use of data in information systems.

Process-oriented views on data and user views on data

It was pointed out in (Langefors[3], Section 212.2) that it is advisable to

> "make use of the fact that the record is uniquely defined *to some extent by file and to some extent by the* actual process. Thus one should have a *basic record description* which is defined by the file and an additional structural description which adds.....to the basic record description. Only the latter would then have to be defined for the process".

This idea was also forwarded in [Olle[25]] and has much more recently been taken up in the data base management systems work. For instance the "basic record description" concept in (Langefors[3]) is very closely related (if not identical) to the "physical structure" or to the "storage schema" of the CODASYL DATA DESCRIPTION LANGUAGE COMMITTEE 1978 (Metaxides[21]) and other CODASYL reports during the 1970s. Likewise, there is a clear resemblance between the "additional, structural description", mentioned above, and "logical structure" or the "schema" (and the subschemas) of CODASYL. An alternative term to "subschema", proposed by (ANSI/SPARC[13]) is the "external schema" and the associated "external level" of the data base architecture. In the ANSI/SPARC terminology the "basic record description", mentioned above, corresponds to the "internal level". The "external

level" is said to be concerned with "individual user views" (Date[22]). It is to be noticed that one is, here, concerned with user views of the data base, not with user world views or infological user views though, unfortunately, this distinction is usually not made clear.

The common use of the term "user view" in the data base literature employs the idea that the data base is modelling part of the world and that an individual user sees only a subpart of this part of the world. Consequently, the user would only "see" a part of the data base such that of all the data about a real-world entity, contained in the data base, an individual user will only see a subset of the data base objects and only a subset of the properties of each of these objects. This may suggest the term "user view". However, it seems important to recognize that this is only a very special aspect of the user's view of the world. It ignores entirely the infological insight that the information—which the users may obtain from the data—is totally dependent on the general "semantic background" of the users, as we have seen. It takes for granted that if distinct people are concerned with distinct parts of the world then each of them would be served by obtaining those data from the data base that are associated (by somebody else) with those parts of the world.

The way the term "user view" is employed in recent data base literature may be further illustrated by presenting some quotations from Date[22]. "The external schema defines the user's view of the data base". The data orientation—rather than information orientation—of this conception of "user view" is further exhibited by the following two quotes from Date[22]. "Suppose that the user wants to see the same information as in the PARTLOC example, but in the form of a hierarchy rather than a relation" (p. 182). "For instance, in the conceptual schema we may have

DOMAIN WEIGHT *NUMERIC* (4)

whereas the corresponding domain in an external schema to be used in a PL/I application may be defined as

DOMAIN WEIGHT, *FIXED BINARY* (14)".

It is clear that the "user's view of the data base" which is thought of in these pieces of text—and several similar ones in data base literature—is very different from what the term "user view" would suggest from an infological perspective.

In Codd[10] there is another illustration of the data oriented way in which "views of the data" are in fact treated in data base works: "For application programs that do more than merely read the data, there are theoretical limitations which must be observed if the data base integrity (including consistency of all permitted views) is to be maintained". As we shall see, the "theoretical limitations" considered are merely those of a purely formal kind. No theoretical limitations of a psycho-linguistic-conceptual (or infological) kind are, at all, mentioned. Furthermore, it seems to be assumed that "all permitted views" could be consistent. This is a very strong statement to make. And our present discussion indicates that it is not tenable.

In the illustration Codd assumes that "the community schema" contains the relations R(A, B) and S(B, C), which at a certain moment have the tabulations:

R(A B)		S(B C)
sl		fu
tl	and	lv
...		...

and that a user requests the schema T(A, B, C) where T is the natural join of R with S on the common attribute B. The tabulation of T becomes:

T(A B C)
slu
slv
tlu
tlv
....

Now the assumption is made that the user wants to delete the triple (t, l, v) and it is pointed out that if he was allowed to do this T would change to a relation that is not the join of *any* two relations.

Clearly, there is an implied assumption here that the user is concerned with how the data are stored—or how they might be conceived of as being stored. This may be important to a user who is an application programmer. From an infological point-of-view the problem illustrated would be irrelevant (similarly to the above illustration from Date[22]. From the infological perspective the data (e-records) in the data base are thought of as *representing* information entities (elementary messages) that are instances of elementary information kinds (e-concepts). The user expects the data base to contain e-records of many distinct kinds and that he will obtain a proper selection of them on request. He does not require that the other data (e-records) appear as non-existent and he wants to disregard the way the data are aggregated (into n-ary relations or hierarchies or whatever). Thus, for instance the relations R(A, B) and S(B, C) might be representations of e-concepts and the (infological) user would expect the management of these to be unaffected by the request of some (other) user to be allowed to forget a certain tuple (such as t, l, v) or a combination of e-records. Furthermore, if the user wants the tuple (t, l, v) he wouldn't care whether this is a tuple from T or is combined from e-records from R and S. He probably also would not want to learn about the name T and the content of T. On the other hand, the problems discussed in the illustration are real problems at the stage when the "implementation" is about to be designed.

It may be of interest to look also on an example of how, in the structured-programming literature, the "problem environment and....our understanding of it" is regarded as defining the structure of the data while, then, a particular processing structure is defined (Jackson[23]). We quote:

"(i) consider the problem environment and record our

understanding of it by defining structures for the data to be processed;

(ii) form a program structure based on the data structures";

It appears that the understanding of the problem environment mentioned, is related to the data base text's concepts of "user views". Then, when it is suggested to form a program structure based on the data structure, this indicates that the author has already imposed some thoughts about a specific processing task upon his "data structures". But such a structure may not at all be inherent in the structure of the information (or the reality represented by the data). For instance records on store movements are from one point-of-view independent of each other; each represents an individual event of issuing or receiving a set of parts. When it is decided to sort the records into part-number order and to compute distinct summaries, this means to impose upon the data some structure which reflects what one wishes to do with the data. It is then not surprising that this imposed, process-oriented, structure of the data can be used to determine the structure of the program for this specific application. But, the same data may be processed for another problem after imposing quite another structure. Clearly, these structures are not inherent in the information represented by the data, or in the reality informed about. In fact, the typical examples of the use of the basic structural categories: sequence, iteration and selection reflect decisions on how to navigate across some data in order to solve some specific problem and they have little to do with the infological structure underlying the data themselves. Thus the view of the data and their relations to reality is rather distinct from the infological (or conceptual view). It seems, rather, closely related to the "user view of the data" as the term is employed in the data base literature. It appears that the structured-programming would become more systematic—more structured—if one would define the infological structures, independently of the intended processing first, before proceeding to the design of the processing-oriented data structures that one does not start with.

A more general and abstract process-oriented view of the information structure (thus not merely the data structure) was developed in Langefors[1, 3]. Information precedence relations were analyzed by identifying the information units (precedents) that may be used to derive specified information units. This precedence structure was used both in order to making the information needs analysis systematic and to finding the record and file consolidations and process groupings that would be efficient from the data transport (access & transfer) point-of-view. Thus data design and program design was also made systematic by information precedence analysis. Information precedence graphs, content graphs (component graphs) as well as lists and matrices were used as tools for describing the information precedence structure.

CONCLUSION

For some years the author's ideas of information systems or "large shared data bases" have been that all data in the data base should be available to all "legal" users. To this end all data should be adapted to what is usually called a "community view". According to recent data base writings, the data administrator is supposed to define this community view. This is to be done by means of the conceptual schema (that is the infological model). Then each particular user view is assumed to be serviced by a subset of the data, that is arranged with consideration to the processing pattern assumed to be associated with the particular user view. This data subset and arrangement is supposed to be declared by an external schema, presumed to model the particular user view. All data in the data base that are not within the scope of an external schema are assumed to be invisible—indeed non-existent—to the particular users associated with the particular external schema.

The infological (or conceptual) aspects of data and information—as well as of "user views"—that has been discussed in this article imply some fundamental disagreement with the current data base theory aspect (or "datalogical aspect") of user views, community view and external schema. Below four aspects are described:

(1) If the data to be used by a particular user have been modified in order to be consistent with some kind of "community view", they may be unintelligible to the users, no matter what data selection and rearrangement is brought about by the use of the external schema. In other words, *it may be impossible to make some data "shareable"*.

(2) The user's view of data depends on his view of the world. To such a view one or more sets of data may be adapted. The user may then want to use the same data for the solution of distinct tasks. *Based on the same view of the data* he may thus want to *process* them in distinct ways and, for that reason, he may want to arrange them differently. This may call for distinct externals schemas that, however, have to be based on the same user view. *The interpretation of some data must always be based on the same* (infological/conceptual) *user view—the view for which the data have been designed.* But these data may be rearranged for processing reasons and the application programmer may view the data from the processing point of view. Thus, the *programmer* may view the data according to how they are to be processed—rather than according to what they mean. But the data must always mean the same while distinct inferences are drawn from them in distinct applications.

(3) The idea of *one* community view, declared by *one* conceptual or infological schema has to be replaced by a system of conceptual (or infological) schemas. One or more of these schemas, infological/conceptual subschemas, may describe such information as has been possible to establish as "community" information. This cannot be decided by the "data base administrator". It must be determined through learning and negotiation among the relevant users. Some other infological/conceptual subschemas may describe information that can be shared by distinct user groups but that requires distinct frames-of-reference for distinct user groups. This assumes that the users have "distinct but reconcilable" infological views. The implication here is that distinct

users will require *distinct data* to obtain the same information (approximately). This is the case of "shareable information" but non-shareable data. Finally there will be infological/conceptual subschemas declaring user views that are irreconsilable and that, hence, correspond to non-shareable data.

(4) The hypothesis—forwarded above—that there will be user views that are incompatible, implies that the information systems or data bases will contain "islands" of non-shareable data. One consequence is that it will also be pointless to try to cater for formal consistency testing among distinct subsystems. Recognizing this fact (if it is a fact) will save a lot of useless analysis, formalization, and verification work as well as a lot of gathering of testing data. "Total data base integrity" will be meaningless and impossible, while "island-wise" integrety will still be important—and easier to achieve.

REFERENCES

[1] B. Langefors: *Some Approaches to the Theory of Information System.* BIT 1963.

[2] B. Langefors: Towards integration.....In *Vistas in Information Handling. Augmentation of Mans Intellect by Machine.* (Ed. by P. Howerton). Spartan Books, New York (1963).

[3] B. Langefors: *Theoretical Analysis of Information Systems.* Studentlitteratur, Lund, Sweden, 1973 Edn, with Auerbachs and Petrocelli/Charter, New York (1966).

[4] Y. Bar-Hillel: Language and information. *Selected Essays on their Theory and Application.* Addison-Wesley, Reading, Mass. (1964).

[5] R. Carnap: *Meaning and Necessity.* The University of Chicage Press (1970).

[6] J. Piaget: *The Origin of Intelligence in the Child.* Penguin Books, New York (1936).

[7] Wittgenstein: *Tractatus Logico—Philosophicus.* Routledge and Kegan Paul (1921). 2nd Impression 1963.

[8] Bower and Anderson: *Human Associative Memory.* Wiley, New York (1974).

[9] N. Lyons (Ed.): *New Horizons in Linguistics.* Penguin Books, New York (1970).

[10] E. F. Codd: Recent investigations in relational data base systems. *Information Processing 74 (IFIP Cong. 1974,* Stockholm). North Holland, Amsterdam (1974).

[11] B. Langefors: *Introduktion till Informations-behandling* (Introduction to Informatics, Swedish). Natur och Kultur, Stockholm Sweden, 1968.

[12] B. Langefors and K. Samuelson: *Information and Data in Systems.* Petrocelli/Charter, New York (1976).

[13] ANSI-SPARC: Study group on data base management systems. *Interim Rep. FDT (Bull. of ACM-SIGMOD)* 7, No. 2, 1975.

[14] B. Langefors: Some problems with recognizing and identifying similar info. Types in EDB systems. *IB-ADB* 1967, Dept. Adm. Data Proc. Institute of Technology, Stockholm (also in *Systemering* 70, pp. 83 FF. Studentlitteratur, Lund, Sweden).

[15] CODASYL: Systems Committee. Feature analysis of generalized data base management systems. *Tech. Rep.* (May 1971).

[16] B. Langefors and B. Sundgren: *Information Systems Architecture.* Petrocelli/Charter, New York (1975).

[17] B. Langefors: A concept of concepts. *ID-ADB* 1070, No. 29, Dept. Adm. Data Proc. Royal Institute of Technology, Stockholm, 1970.

[18] T. S. Kuhn: *The Structure of Scientific Revolutions,* The University of Chicage Press 2nd Edn. Enlarged (1970).

[19] B. Langefors: Hermeneutics, infology and information systems. *TRITA-IBADB* 1052, 1977. Royal Institute of Technology, Stockholm. Dept. Adm. Data Processing.

[20] K.-O. Apel: Szientifik, Hermeneutik, Ideologie-Kritik. In Man and World. *Int. Philosophical Rev.* 1. 37–63 (1968).

[21] A. Metaxides: Report of the CODASYL data description language committee. Special Issue of *Inform. Systems,* Vol. 3(4), 248 (1978).

[22] C. J. Date: *An Introduction to Data Base Systems* 2nd Edn. Addison-Wesley, Reading, Mass. (1977).

[23] M. A. Jackson: *Principles of Program Design.* Academic Press, London (1975).

[24] B. Langefors: *System för Företagsstyrning* (Systems for Management Control, Swedish). Studentlitteratur, Lund, Sweden, 1968.

[25] T. William Olle: *Data structures and storage structures.* In *File Organisation,* IFIP Administrative Data Processing Group (IAG) Swets & Zeitlinger, Amsterdam (1969).

DISEÑO DE BASES DE DATOS

BUSINESS SYSTEMS PLANNING

EXPOSITOR:
Ing. Daniel Ríos [illegible]

MAYO, 1985

Contents

Figures

Introduction

The Changing Environment

The rapidly changing economic environment and the constant need for businesses to adjust quickly to it make it necessary for executive management to have up-to-date information available at all times, so that through meaningful analyses and resource allocation tradeoffs they can manage their businesses more effectively. With organization-wide availability of information, strategies can be improved, decisions made more soundly, and operations performed more efficiently.

From Operation to Management Control

Data processing is in transition. Until comparatively recently, most businesses considered it a service function and used it to support single-operation units or locations. In the last two decades applications have been developed independently, with very little regard for the support they could give other functions and for the information they could supply for management control. Functional autonomy has been the rule. This has resulted in fractionalized and redundant data files and in the inaccessibility of data from the many operational applications installed in the various functions.

Today, however, businesses are recognizing data more and more as a resource that is as important as personnel, cash, facilities, or materials. They see the need to consolidate the key data files and make information available not just to individual functions or departments but throughout the business, in order for management to gain an overall view of the business and be able to make multifunctional decisions.

Many companies have recognized the need for company-wide information systems but have been unable to develop them for one or more of the following reasons:

- Failing to obtain executive commitment and involvement
- Establishing objectives and strategies that were not in line with their overall business objectives
- Attempting to implement information systems without first understanding the business from general management's viewpoint
- Setting out to implement totally new company-wide information systems rather than a comprehensive plan evolving from current systems
- Failing to put in place those information resource management functions required to adequately manage the information resources

A Justified Approach

In an article entitled "Blueprint for MIS,"[1] Dr. William Zani states, "Traditionally, management information systems have not really been designed at all. They have been spun off as by-products while improving existing systems within a company. No tool has proved so disappointing in use. I trace this disappointment to the fact that most management information systems have been developed in the "bottom-up" fashion – an effective system, under normal conditions, can only be born of a carefully planned, rational design that looks down from the top, the natural vantage point of the managers who will use it."

In most instances the current operational systems have been justified and have been performing effectively for their specific intent, even though their maintenance and interfaces may have become unmanageable. An information system plan should allow a modular approach to implementation, providing confidence that each module will fit and function properly to form an integrated system and will interface properly with the present operational systems. The plan should also allow for better decisions concerning the efficient and effective commitment of information systems development resources. With such a plan, the required information can be more readily obtained.

Business Systems Planning (BSP) is geared to help provide such a plan through:

- A top-down approach to (1) getting people committed and involved (starting with top management and working down through the organization) and (2) studying the business (working from the overall to the detail level)
- A bottom-up approach to implementation
- Use of a structured methodology proven in hundreds of studies

[1] *Harvard Business Review*, November/December, 1970

- The translation of business objectives into information requirements

The effectiveness of the BSP methodology can be attributed to two components:

- The fundamental principles and concepts – the unvarying ideas and logic that form the basis for BSP, including the standards upon which the procedures are based

- The sequenced activities, techniques, disciplines, time, output, planning, team composition, etc., established to fill a particular organization's need and situation (although consistent with BSP's basic principles and concepts, the procedures are flexible and vary with the particular environment)

Origin of IBM's Business Systems Planning Function

Learning from its own mistakes and those of other companies that attempted to implement large information systems in the 1960s, IBM realized that a disciplined approach was required, using proven principles and methodologies. In 1966 a business-wide Information Systems Control and Planning Department was established at IBM's Data Processing Group headquarters. The Data Processing Group was a total business unit comprising the engineering, manufacturing, marketing, and service divisions responsible for all of IBM's domestic data processing business.

Until the control and planning department was established, IBM had little overall direction in the internal use of computers. In fact, little coordination took place between divisions; most data processing activities were confined to locations and units within divisions. Consequently, each manufacturing plant and marketing region developed and operated its own system. Although the individual systems carried out similar functions, they differed in design and performance; they could not be used interchangeably and could not communicate with each other.

The result was a redundancy of data and excessive use of the data processing resources required to develop and maintain such systems. Even with this large expenditure of resources by each division, the systems were mainly satisfying the local department needs of the business, rather than doing an overall data processing job.

When steps were taken to improve the data processing within a division (for example, development of a consolidated order entry system within the marketing division), little serious attention was given to an effective interface of that system with input from engineering and output to manufacturing. The business was not getting the return on investment from data processing that it could have because the information needs of the business, and particularly those of the general manager responsible for the business, were not being accommodated.

The first effort of the control and planning department was to inventory and profile the systems existing within the business and the plans for the future. At the same time, recognizing that the data processing effort must be directed toward satisfying business needs and not solely toward individual functions and departments, the control and planning department established a set of information system strategies covering five major areas:

1. Fixed data responsibility
2. Single source and parallel distribution of data
3. Central control and planning of information systems
4. Organizational independence of data
5. Resource sharing of data, equipment, and communications

With the knowledge of what was being done with data processing, and the direction established through the set of strategies, the department defined an integrated set of information systems and assigned responsibilities for the development of the systems. These systems addressed the operational, functional, and general management needs for information.

As the definition and design efforts for this business-wide set of information systems got under way in the late 1960s many of IBM's customers showed interest in learning how they might better manage their information system resources. In an effort to assist these interested customers, IBM established the Business Systems Planning (BSP) program in 1970.

To conduct the program, the nucleus of the control and planning department that developed the internal information systems plan was transferred to the Data Processing Division headquarters. This group proceeded to document proven methodologies and institute a training program to educate regional and branch office personnel and customers in the approach. Executive briefings and seminars were established to show customer executives the potential benefits of the

approach and the reasons why their involvement was vital to successful implementation of information systems.

The methods used in developing this information systems plan and the lessons learned have since been used by many IBM customers. Application of BSP methodology has helped them to formulate their information system plans and control mechanisms and to improve their use of information and data processing resources Studies using the BSP methodology and guidelines have been conducted successfully by profit and nonprofit organizations, of varying size, in many industries.

Objectives and Potential Benefits of BSP

With a reasonable amount of planning and control, the user of this BSP approach should be able to attain its objectives and realize its potential benefits.

Objectives

The first and most important objective of BSP is to provide an information systems plan that supports the business's short- and long-term information needs and is integral with the business plan. There are other objectives that help to justify and clarify the approach:

1. Provide a formal, objective method for management to establish information systems priorities without regard to provincial interests.

2. Provide for the development of systems that have a long life, protecting the systems investment, because these systems are based upon the business processes that are generally unaffected by organizational changes.

3. Provide that the data processing resources are managed for the most efficient and effective support of the business goals.

4. Increase executive confidence that high-return, major information systems will be produced.

5. Improve relationships between the information systems department and users by providing for systems that are responsive to user requirements and priorities.

6. Identify data as a corporate resource that should be planned, managed, and controlled in order to be used effectively by everyone.

Figure 1 shows the overall flow of BSP and relates the BSP study to the activities that follow. This relationship is expanded upon in Chapter 16, with an explanation of the follow-on activities.

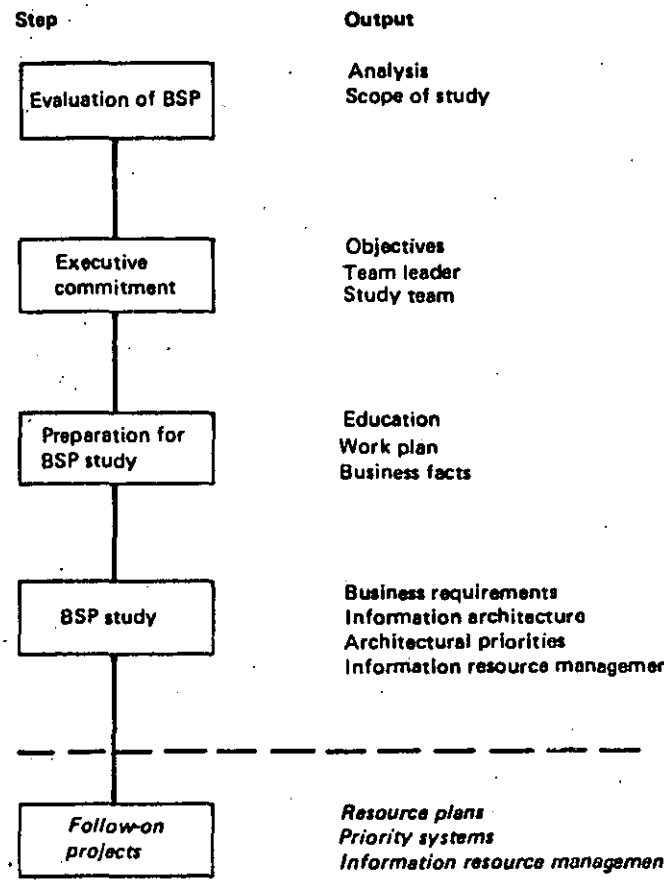


Figure 1. Relationship of BSP study to follow-on activities

Potential Benefits

Application of the approach and methodology contained in this planning guide offers many potential benefits to three management groups:

To executive management:

- An evaluation of the effectiveness of current information systems
- A defined, logical approach to aid in solving management control problems from a business perspective
- An assessment of future information system needs based on business-related impacts and priorities
- A planned approach that will allow an early return on the company's information systems investmer

- Information systems that are relatively independent of organization structure
- Confidence that information system direction and adequate management attention exist to implement the proposed systems

To functional and operational management:

- A defined, logical approach to aid in solving management control and operational control problems
- Consistent data to be used and shared by all users
- Top management involvement to establish organizational objectives and direction, as well as agreed-upon system priorities
- Systems that are management and user oriented rather than data processing oriented

To information systems management:

- Top management communication and awareness
- A better long-range planning base for data processing resources and funding
- Personnel better trained and more experienced in planning data processing to respond to business needs
- User involvement in information systems priority setting

The plan that results from a BSP study should not be considered unchangeable; it simply represents the best thinking at a certain point in time. The real value of the BSP approach is that it offers the opportunity to (1) create an environment and an initial plan of action that can enable a business to react to future changes in priorities and direction without radical disruptions in systems design, and (2) define an information system function to continue the planning process.

Chapter 1. BSP Concepts

Business Systems Planning is most often thought of as a structured approach or methodology. This methodology, however, is based upon some fundamental concepts, a good understanding of which can give the BSP study team members:

1. A better appreciation of the "why's" of the methodology
2. Improved confidence in applying variations to meet specific situations
3. A better background with which to communicate the objectives and eventual recommendations to senior management

The premise for conducting a BSP study is that there exists within the organization a need for significantly improved computer-based information systems (I/S) and a need for an overall strategy to attain them. BSP is concerned with how these information systems should be structured, integrated, and implemented over the long term. The basic concepts of BSP can be related to the long-term objectives for I/S in an organization.

- An I/S must support the goals and objectives of the business.
- An I/S strategy should address the needs of all levels of management within the business.
- An I/S should provide consistency of information throughout the organization.
- An I/S should be able to survive through organizational and management change.
- The I/S strategy should be implemented project-by-project to support the total information architecture.

Each of these items is discussed in turn.

Support the Goals and Objectives of the Business

This most basic concept underlies the "top down" philosophy of the methodology as well as several of the specific steps, such as executive interviews and system priorities.

Since information systems can be an integral part of a business and be critical to its overall effectiveness, and because they will continue to represent major investments of time and money, it is essential that they support the organization's true business needs and directly influence its objectives. BSP, then, can be thought of as a vehicle or process to translate business strategy into I/S strategy (see Figure 2).[2]

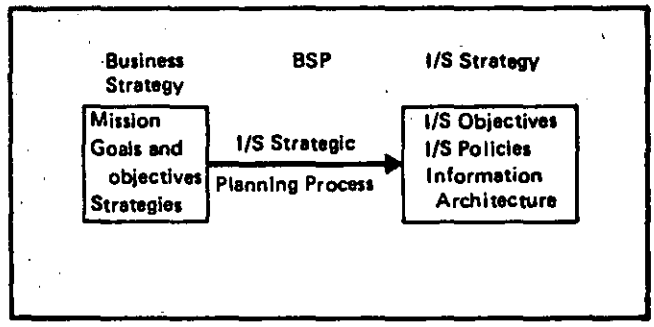


Figure 2. Translation of business strategy to I/S strategy

Obviously, it is important that an organization be willing and able to express its long-term goals and objectives. For some organizations, this can be done principally through the business plan. For others, where a business plan is not available or current, it can be done as a part of the BSP methodology. In either event, a recognition of this basic need by senior management is critical, for only with that recognition will their commitment and involvement be great enough to guarantee a meaningful BSP study.

Address the Needs of All Levels of Management

This requirement has several implications relative to I/S structure. First, it is important to recognize the varying characteristics of information as needed by different activities and management levels. Typically, lower levels need considerable detail, volume, and frequency, higher levels need summaries, "exception" reporting, and inquiries, and still higher levels need cross-functional summaries, special requests, "what if" analyses, and "external" requirements. It would be impractical to construct a single system to accommodate all activities or management levels, and it would erroneous to associate any one type of information requirement solely with one management level. Clearly, there is need to establish some reasonable framework upon which the I/S can be defined.

[2] King, W.R. "Strategic Planning for MIS." *MIS Quarterly*, March 1978.

First, the emphasis in I/S should be in support of management decision making. This is in contrast to more traditional bookkeeping or recordkeeping functions. Business decisions are made for various purposes, but most can be associated with either *planning* or *control*. Planning, of course, is the establishment of various missions, objectives, and policies, and it occurs at all levels; good information is essential to the establishment of good plans. Control decisions, by contrast, are made in order to guide an activity toward some implicit or defined (by the plan) objective. The I/S can provide the measurements of the current or actual condition to the decision maker. We thus complete a planning, measurement, and control cycle with I/S potentially an integral part. Since planning and control are the keys to decision making, a framework for I/S based upon these activities can be used. It has been proposed,[3] and well accepted today, that three distinct but concurrent planning and control levels exist in any organization:

Strategic planning is the process of deciding on objectives of the organization, on the resources used to attain these objectives, and on the policies that are to govern the acquisition, use, and disposition of resources.

Management control is the process by which managers assure that resources are obtained and used efficiently in the accomplishment of the organization's objectives.

Operational control is the process of assuring that specific tasks are carried out effectively and efficiently.

Further characteristics of these areas are outlined in Figure 3. An advantage of this framework is that it does not restrict planning and control activity to any particular industry, function, or management level. A conclusion at this point is that an I/S could conveniently address itself to any one of the above three planning and control levels.

Resource management is also key to this philosophy and represents a major vehicle for I/S definition. The specific resources to be managed vary in nature and relative importance from one organization to the next. Examples of traditional resources to be managed are people, facilities, materials and money. Their requirements are based upon the needs to support the prime mission area of the organization, for example, its products or services.

[3]Anthony, R.N. *Planning and Control Systems: A Framework for Analysis.* Harvard Business School, Division of Research, 1965.

Because an organization's product or service area has all the attributes of a resource, that is, a life cycle of activities and decision points, and yet drives the other resources, it is referred to as the "key resource." Each resource, including the key resource, is managed through planning and control decisions of the three levels previously discussed. Resource management has the desired characteristic of cutting across organizational boundaries – vertically across management levels and horizontally across functional lines. Thus a framework based on resources as well as planning and control levels can be established, and an I/S architecture can be applied within this framework.

Provide Consistency of Information

The keyword in this objective is *consistency*. The implication is that information derived from more traditional data processing applications is not necessarily consistent, particularly when applied to new business problems (decision areas) of broader scope. The problems in data consistency normally arise as a result of a historical evolution of computer usage. Isolated and independent application areas are selected and mechanized, typically to reduce operational costs. The data files are defined as necessary to support the specific needs of each application without regard to one another or to future applications. The data itself is converted from manual files located and maintained by the using organization.

As computer applications are added, new data files are usually required since the data requirements for different applications are rarely the same. These are usually created from spinoffs of existing mechanized files plus any additional data that may be required from the using area. Summary-type reports for higher management levels are the result of sorting and merging various existing data files together to create new ones. Rarely is any existing data file of the form or content required to provide newly requested information of any magnitude. Thus new data files are born. Data redundancies and file update requirements multiply. The continual cry from management that "I know the data that I need is somewhere in data processing, but I can't get at it" may be basically true from their point of view. The data may be there, but not necessarily defined as needed, of adequate summary level or sequence, or of proper time period or currency.

Data, then, exists in most organizations in varying *form, definition,* and *time*. The *form* may be uncaptured raw data, mechanized data files, detailed DP reports, summarized DP reports, business documents, or knowledge in someone's head. The *definition* of any given data can have as many variations, and thus

Decision Characteristic	Planning and Control Level		
	Strategic Planning	Management Control	Operational Control
Management involvement	General management Functional management	General management Functional management Operational management	Functional management Operational management
Time horizon	Long range (1—10+ years)	Year-to-year Monthly	Day-to-day Weekly
Degree of structure	Unstructured and irregular; each problem different	More structured, cyclic, largely repeating	Highly structured, repetitious
Data requirements	Summaries, estimates, difficult to pre-define, much external to business	Summaries, definable, need for unanticipated forms, largely internal	Detail, operational, definable, internally generated
Resource management	Establishment of policies pertaining to the resource	Allocation of the resource	Efficient use of the resource

Figure 3. Characteristics of planning and control levels

inconsistencies, as there are users of that data. For example, "salary" to the payroll department may mean an employee's actual monthly pay, to the project manager an annual figure plus burden to be charged to a customer, to the department manager a budget line item representing total expense for all reporting employees. In addition, a *time* inconsistency is very likely to exist between what may otherwise be comparable data. Data may be captured by such varying methods as the mail, telephone, data terminals, or satellites. It may be "batched" over varying lengths of time by the user or by DP before processing, or it may be entered "online" as each transaction occurs, directly from its source. Data may be processed daily, weekly, or monthly on a predetermined schedule, or it may be incapable of being processed until a series of prior computer runs are completed (for example, the month-end closing). The output itself may be mailed or it could be immediately available as the result of an online inquiry. Finally, the report may be up to the second, as when reading from a terminal, or it may have lain in a desk for three weeks after last month's processing. The combinations of time deviations between data capture, data processing activity, and actual data usage are many.

With all these potential data inconsistencies, it is no wonder that reports frequently don't "match" between using departments and managers. This becomes a problem most often during interdepartmental decision-making or at higher reporting levels where consolidation of multifunction activities is important. Attempts to provide better data consistency usually result in "resystematizing" or "consolidating" existing applications into larger ones with broader problem scope and data definitions. This may yield a satisfactory system within the defined scope, but again, as still broader problems are addressed, there will undoubtedly be data inconsistencies between the larger systems. Resystematizing at this scope may be extremely expensive and difficult to justify, let along accomplish. Comments prevail such as "our systems cannot talk to one another."

What has been described is the classical "bottom up" evolution of data processing systems. In order to begin to address the data consistency problem, a different philosophy must be adopted relative to data management. This is commonly referred to as *managing data as a resource*. This concept suggests that data is of considerable overall value to an organization and should be managed accordingly. It should be potentially available to and shared by the total business unit on a consistent basis. It should not be controlled by a limited organizational segment but by a central coordinator, much like other corporate resources, such as cash and personnel. The management function would include formulating policies and procedures for consistent definition, technical implementation, use, and security of the data.

Survive Organizational and Management Change

Many data processing systems and applications are set up to provide the information needs of a specific department or other organizational entity. Others are

built solely on the specific output report requirements of a particular manager. Both types can become immediately obsolete upon a reorganization or management change. A new manager may have his own ideas as to what information is needed to run the department. Although this kind of change is inevitable, it can be expensive from a data processing standpoint. The data processing system, however, should in no way inhibit management flexibility in a dynamic enterprise. Thus, the I/S must be capable of evolving through the long-term organizational and management changes of a business with minimum impact if the expected return on investments is to be realized.

This objective cannot be realized without the proper support vehicle for I/S, and this vehicle must be independent of the various components of the organizational structure. The BSP vehicle is the *business process*, that is, a basic activity and decision area irrespective of any reporting hierarchy or specific management responsibility. A logical set of these processes can be defined for any type of business and will undergo minimum change as long as the product or service area of the business remains basically the same.

One example of a business process is *purchasing*. A particular business might define this as "the process by which raw materials are acquired from vendors." There may or may not be a separate organizational unit to accomplish this process, or indeed there may be several. Inherent within this process are the various activities and decisions necessary to accomplish the process.

Defining the organization's business processes is one of the most important parts of the BSP methodology, and the method for doing so is tied directly to the previously discussed I/S framework, that is, one based on resources and planning and control levels. With this in mind it is convenient to define an organization's business processes in association with each of its defined resources.

Emphasis in BSP is normally placed upon those processes necessary to manage the *key resource*. Each resource of a business can be thought of as having a "life cycle" made up of several stages. A product life cycle, for example, has four stages: requirements, acquisition, stewardship, and retirement. The time spread of the life cycle can vary greatly with the particular product area but is of no consequence in this approach. Business processes can be identified to describe the major activities performed and decisions made by the business in the course of managing the resource throughout its life cycle. These can normally be organized into a process hierarchy, and this is done without regard to organizational involvement or responsibility.

This approach results in process definitions that encompass the three planning and control levels previously discussed -- namely, strategic planning, management control, and operational control. Using a product resource as the example again, the decision to pursue a particular product area would be "strategic planning," the planning and control decisions relative to product volumes or advertising expenditures would be "management control," and the decisions in areas such as engineering control and manufacturing efficiency would be "operational control." By using this approach for all the resources, it is possible to define all the business processes that take place within any organizational segment. This may be tempered by practicality in the actual BSP as there may be little I/S support interest for some of the resources.

Implement Project-by-Project to Support the Total Information Architecture

There are several implications associated with this concept. The first is that a total I/S to support the entire business unit's needs is too big to build in any single project. However, because of the many problems associated with a "bottom up" evolution of systems (such as data inconsistencies, nonintegrated system designs, expensive resystematizing, priority difficulties), it is very important that long-range goals and objectives for I/S be established. The basic concept, then, is *top-down I/S planning with bottom-up implementation* (Figure 4).

With this implementation strategy (the BSP approach), the information support is implemented in a modular building-block fashion over time, while remaining consistent with the organization's business priorities, available funds, and other shorter-term considerations. This philosophy can be likened to the detail design and construction of a large office building, which would be unthinkable without an architect's approved drawing of the finished product.

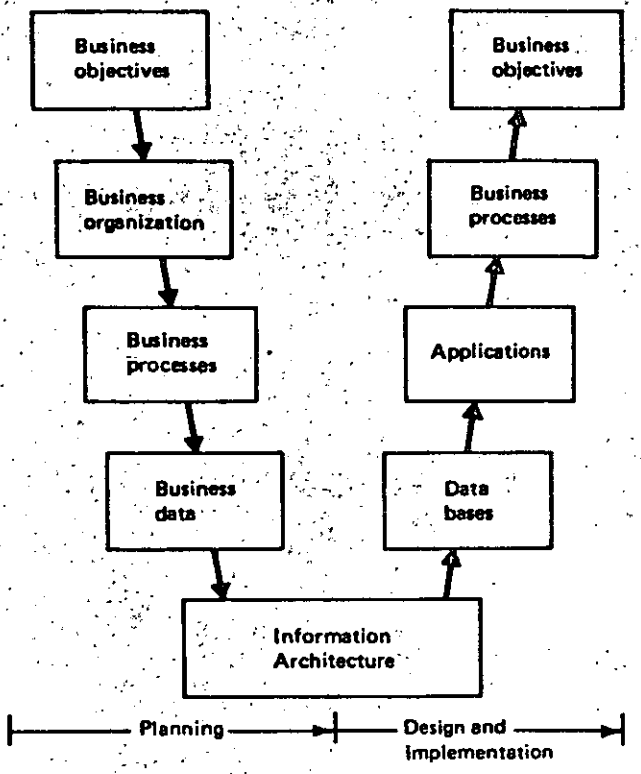


Figure 4. Top-down analysis with bottom-up implementation

The BSP methodology, although consisting of considerably more steps and detail than shown in Figure 5, is consistent with this philosophy. Step 1 of Figure 5, defining the business objectives, is intended to ensure agreement among all executive levels as to where the business is going, so that the I/S strategy can be in direct support. Step 2, defining the business processes, establishes the prime long-term basis for I/S support in the business. Step 3, defining business data, is done by identifying what things (entities) are important to the business and what data is required to manage these entities. Logical groupings of this data about entities are called data classes. Step 4, defining the information architecture, becomes a statement of the long-term I/S objective. From the information architecture the individual modules can be identified, scheduled, and built according to the I/S plan.

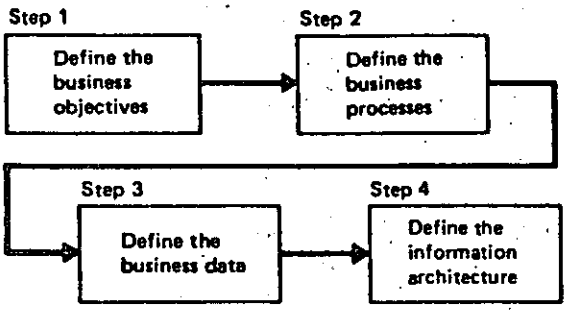


Figure 5. General I/S planning approach

In summary, there are a number of basic concepts and philosophies relative to information systems upon which the BSP methodology is based. Although the methodology itself should be flexible (that is, certain steps and techniques could be altered to adapt to specific situations without detriment to the final outcome) these basic concepts themselves should be inviolate. They, in effect, *are* BSP.

Chapter 2. Overview of the BSP Study Approach

The keys to success in planning, developing, and implementing an information architecture that effectively supports the business goals are:

- Top-down planning with bottom-up implementation
- Managing data as a corporate resource
- Orientation around business processes
- Use of a proven, comprehensive methodology

Chapter 1 examined the first three at some length. This chapter gives an overview of a proven methodology for the BSP study which will be covered in more detail in the following chapters.

Major Activities

As Figure 6 indicates, there are two major activities that precede a BSP study and eleven in the study itself.

Although these activities may be carried out in varying degrees, none can be omitted. Figure 6 shows these activities and their most logical arrangement. When one becomes fairly familiar with the BSP approach, however, he can, as appropriate, do part of a given activity and delay the balance. The remainder of this chapter is a brief commentary on these major activities.

Gaining Commitment

A BSP study should not be started unless a top executive sponsor and some other executives are committed to being involved in it. The study must reflect their view of the business, and the success of the study depends upon their providing an understanding of the business and its information requirements to the team. Most of the input will come directly or indirectly from these executives.

Since approval of study recommendations commits the company for several years to a certain direction in the use of its data processing resources, it is important at the outset to get agreement on the scope and objectives of the study and on its expected deliverables, so as to minimize future misunderstandings.

The most important action following commitment regards selection of the team leader, an executive who will work full time in the study and direct team activities. He sees that contact with other executives is on the proper level and that input from them is interpreted correctly. A letter from the sponsor to all participating executives sets the tone and signifies commitment.

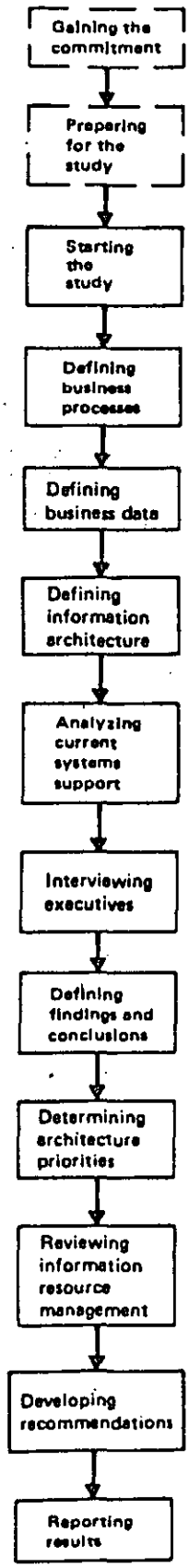


Figure 6. Flow of the BSP study

Preparing for the Study

Considerable saving of time, avoidance of frustration, and higher quality of output can be gained by proper study preparation. All executive participants and the team need to know what will be done, why, and what is expected of them. Proper education and orientation will provide the best input from the executives and the best use of it by the team.

Interviewees are selected as soon as possible so as to allow for their orientation, scheduling of interviews, and the providing of information to the study team. For maximum efficiency on the part of the team in working together full time during the study, information on the company and on data processing support is gathered before starting the study.

A control room is established so that the team may work together, display relevant material on the walls, and conduct interviews.

The major output should be a study control book containing: a study work plan; a schedule of interviews and a schedule of checkpoint reviews with the sponsor; an outline of the final report from the study; BSP and business and information systems data, analyzed and charted, and ready for the study start. This activity should end with a review by the BSP study sponsor.

Starting the Study

The BSP study itself and the full-time participation of the team members starts with a business review meeting, which consists of three presentations. First, the executive sponsor reiterates the objectives, expected output, and perspective of the study with relation to other company activities and objectives.

The next presentation is concerned with the main purpose of the review, which is to provide that each team member is conversant with the information that has been gathered and to discuss those facts that are not part of the information supplied. To accomplish this the team leader "walks through" the business facts that have been gathered and makes subjective comments and additions on facts that cannot be readily documented – politics and sensitive issues, and changes planned and in process. He should also cover the decision process, how the organization functions, key people, major problems, the user's view of DP support, and the image of the DP department.

The third presentation is made by the information systems director or one of his managers, who gives the team a view of data processing analogous to what was presented for the business. He should also cover project status and project control, history of major data processing projects started in the last two years, major current activities, planned changes, and major problems.

These three presentations, added to the facts that have been gathered and made readily available to the team, should give the team an overall understanding of the business and the present and planned data processing support.

Defining Business Processes

No other activity during the study can be quite as overwhelming or as important as identifying of the business processes. Since these processes form the basis for executive interviews, the information architecture, problem analysis, data class identification, and various follow-on activities, everyone on the team must acquire an understanding of all the processes, and they can do so by assisting full time in their identification and in the writing of their descriptions. The major output from this step will be a list of all the processes, a description of each, and the identification of those that are key to the success of the business.

Defining Business Data

The defining of business data involves the identification of entities (things that are significant to the business) and the grouping of data about these entities into logically related categories called data classes. This classification, and its subsequent modification during the follow-on projects, helps the business develop data bases over time with a minimum of redundancy and in a manner that allows systems to be added without a major revision to the data bases. Since data must be recognized as a corporate resource, it deserves the attention recommended here.

Defining Information Architecture

The definition of the information architecture involves the relating of processes to data classes. This enables the evaluation of data sharing within the enterprise. The architecture also provides the foundation for follow-on resource and tactical planning, which enables the orderly implementation of the information architecture.

Analyzing Current Systems Support

The main purpose of this activity is to show how data processing currently supports the business in order to

develop recommendations for future action. The currently existing organizations, business processes, information systems (applications), and data files are analyzed to identify voids and redundancies, help clarify responsibilities, and further the understanding of the business processes.

The main analysis tool is the matrix, and various matrices are developed using combinations of the four elements. The key matrix for the executive interviews is the process/organization matrix, and its intercepts denote the decision makers, major and minor organizational responsibility for a process, and current data processing support.

This activity will not only prepare the team for discussions with executives but will also help them determine requirements for information support.

Interviewing Executives

This activity is an integral part of the top-down approach. Its purpose is to validate the work done by the team, determine the objectives, problems, and information needs and their value, and gain executive rapport and involvement. The executive interviews provide the business understanding necessary for information systems planning.

The major output consists of notes from the interviews, an update of the control room charts, and a new or improved rapport between the executive and the BSP study team.

Defining Findings and Conclusions

Some of the business problems were supplied as input during the fact-gathering step. These were expanded upon and complemented by the knowledge of the team and finally validated, explained, and added to in the executive interview. The problems must now be analyzed and related to the business processes so as to give guidance to the setting of project priorities and to show clearly that better information will help to solve the problems.

The problems must now be divided into several categories, and findings and conclusions must be drawn from them. The results of these tasks provide the base for developing recommendations and for setting priorities among the subsystems of the information architecture.

Among the problems identified by the executives are many that do not pertain to information systems support. These will be delineated and given to the executive sponsor for follow-up while the information systems support problems continue to be addressed in the BSP study and in the follow-on activities.

Determining Architecture Priorities

Since a total information architecture cannot be developed and implemented at one time, the team must establish priorities for the development of applications and data bases. By deciding which of the data bases should be designed and implemented first, the team establishes those applications that will be defined during the follow-on phase. This allows an early implementation for a "pay-as-you-go" foundation and helps establish credibility for the balance of the output from the BSP study.

Priorities are determined by developing a list of projects from the applications that make up the information architecture, then establishing a set of criteria and rating the prospective projects against them. The findings and conclusions are a major contributor to this process.

Reviewing Information Resource Management (IRM)

The purpose of IRM is to eventually establish a controlled environment in which the information architecture can be developed, implemented, and operated efficiently and effectively — and to define the environment wherein data is managed as a corporate resource. The information systems functions are examined during the BSP study to identify (1) any changes that could be made immediately to enhance success in the follow-on projects, (2) changes that are necessary to properly manage and implement the high-priority information architecture projects, and (3) major activities that will become projects in the follow-on to the BSP study. Fulfillment of this step is essential to the successful support of the business by the data processing function.

Developing Recommendations

The purpose of this step is to assist management in its decisions regarding the recommended follow-on activities. The follow-on activities will come from the architectural priorities and from the information resource management recommendations.

Reporting Results

The purpose of the final report and executive presentation is to obtain executive management commitment

and involvement for implementing recommendations from the BSP study. The format of the report was agreed upon before the study was started, but may have to be modified slightly now, on the basis of the study results. Various parts of the report should be written during the BSP study and completed at this time.

The report should be prepared as follows:

1. It should include an executive summary.

2. Supporting detail, such as descriptions of the business processes, should be contained in appendixes.

(13) 3. It should be possible to extract highly confidential or sensitive material easily and still have the balance of the report usable by all people participating in the follow-on activities.

The report provides the basis for the executive presentation and the distribution of the final results to those people designated by the sponsor. After the executive presentation, which is normally given by the team leader, all other relevant material that is not a part of the report should be indexed and filed for ready availability in follow-on projects.

Chapter 3. Gaining the Commitment

(14)

The success of a Business Systems Planning study depends heavily upon the commitment of the participants to complete all the relevant activities and heed the recommendations of the study team. The steps described in this chapter should be completed and an assessment made of the commitment of all participants before the final decision is made to move into the resource dependent activities. Certain situations and environments may be considered unsuitable for a BSP study at this stage. If such is the case, the team should be prepared to postpone BSP until the situation becomes more suitable.

Gaining the commitment starts with a BSP presentation to executive management relating the objectives, advantages, expected output, and resource requirements. One or more of the following activities will provide the executives with a base for understanding the need for a BSP study and assist them in their decision:

- Arrange an executive visit to another business that has successfully completed a BSP study. If so, the following actions should be considered:
- Conduct an executive orientation session specifically for your executives
- Conduct an executive planning session – a service offered by IBM
- Invite the executive to an executive information session conducted at an IBM Customer Executive Education center

The IBM representative can help set up any of these activities. Either the executive planning session or the executive briefing is an excellent preliminary to a BSP.

Following is the list of tasks to be performed in gaining commitment for a BSP study:

- Establish the study scope
- Set the study objectives
- Develop the business reasons for the study
- Select the team leader
- Select the study team
- Brief the study team
- Educate the team

Establish the Study Scope

Normally the executive sponsoring the study (called the *executive sponsor*) will determine the scope of the study for a particular organization. Conversely, it should be recognized that the scope may determine the executive sponsor since he is usually the chief executive officer of the unit being studied. This business unit must be defined so that all participants can recognize the boundaries within which their activities will be concentrated.

Businesses whose activities span multiple functional units tend to gain more from a BSP study than those that are more simply structured since BSP deals well with complexity. It is designed to identify the requirements for data integration across multiple functions.

Most companies performing a BSP study choose a major operating division (which may also be a profit center within a division), a group of divisions, or the entire organization (see Figure 7, options 1 to 3 respectively). In each case it should be possible to define boundaries for the study scope, including the measurements for assessing effectiveness.

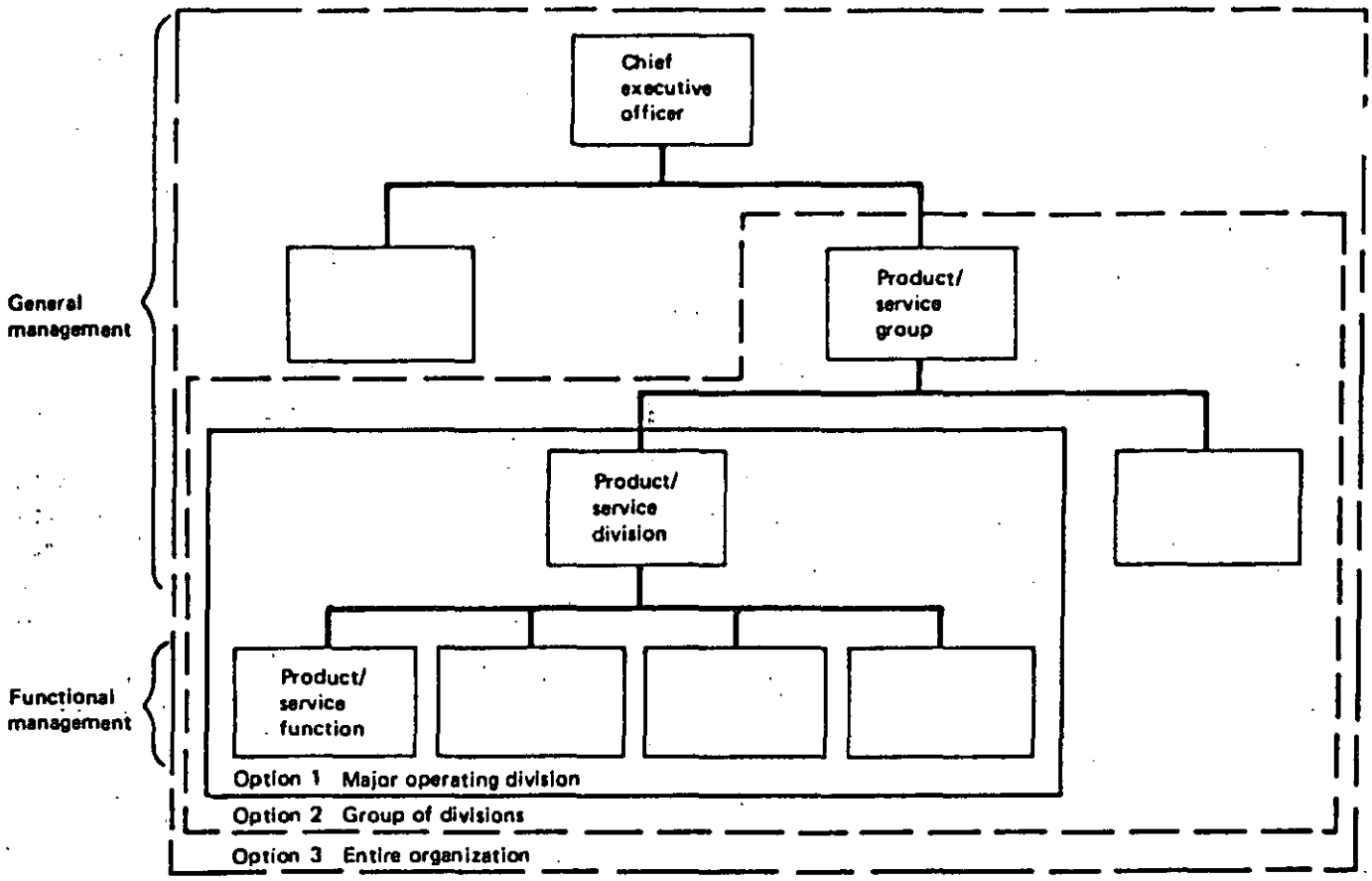


Figure 7. BSP organizational business unit options

Set Study Objectives

Study objectives should now be defined. Good objectives are clear and concise and serve as criteria for measuring the degree to which they have been met. In addition, they are applicable to the specific requirements of the organization to be studied and can be met realistically by the BSP study team working within the defined study scope.

Generally, a study team's objectives may be to:

- Provide a formal, objective method for management to establish information systems priorities without regard for provincial interests.
- Provide for the development of information systems that have a long life, protecting the systems investment
- Provide that the data processing resources are managed for the most efficient and effective support of the business goals
- Increase executive confidence that high-return information systems will be produced
- Improve the relationship between the information systems department and the users by providing systems responsive to the users' requirements and priorities
- Identify data as a corporate resource that should be planned and controlled in order to be shared effectively by everyone
- Identify those data processing functions, such as query, report generation, and personal computing, that the user department personnel may perform for themselves when adequately supported by the data processing department

Develop the Business Reasons for the Study

Those persons promoting or planning for a study should discuss the following factors with an appropriate senior executive of the organization to gain agreement on the following:

- Scope of the of study
- Study objectives

- Positive contributions of a BSP study
- Obstacles to successful completion
- Recommendation either to proceed with BSP or postpone it

The benefits of a BSP are well defined under "Potential Benefits" in the "Introduction" chapter. They may be made more specific and quantified for a given business.

There are several factors that may make one business a more difficult subject for a BSP study than another:

- Major reorganization or transfer of control taking place
- Geographical diversity, necessitating significant travel time
- More than 20 executives who must be interviewed
- Multiple independent suborganizations with autonomous decision-making powers and with unique lines of products/services serving diverse markets

Should the senior executive agree to proceed with the BSP study, the meeting should result in an agreement to:

- Appoint a study team leader
- Provide the study team resources as appropriate
- Establish open communication of business facts and plans between executives and team members
- Write a study announcement letter to executives of the organization in the form shown in Appendix A

Select the Team Leader

The success of the study depends largely on the person selected by the executive sponsor to lead the team. The team leader will be knowledgeable in business matters and have a broad perspective of the business. He can save the team valuable time with his first-hand knowledge of how the various departments of the business interact, and where detailed information about the operation of the business can be obtained. Organizationally he should be a functional vice-president or equivalent.

Activities of the team leader include:

- Conducting the study team orientation session
- Conducting a business review for team members at the start of the study
- Setting and confirming executive session schedules
- Managing the logistics of the study, including day-to-day administration
- Providing guidance and business perspective
- Resolving conflicts and impasses
- Providing resource material
- Maintaining the schedule
- Making the presentation to management at the completion of the study

Select the Study Team

Characteristics of Team

Since the study team is responsible for determining the information needs of the entire business and recommending the nature of its data processing operations for years to come, the selection of team members is very important.

The people chosen should:

- Have several years' experience within the organization, a sound knowledge of their own area, and an appreciation of the rest of the business
- Be able to understand and deal analytically with problems
- Be willing to commit to conclusions and recommendations that will have a far-reaching effect on the organization
- Be perceived by other management within the organization as competent and responsible businessmen whose opinions will carry considerable weight
- Be from upper and middle management

In short, the best people for the study team are those who are already in great demand and least available.

Structure and Responsibilities

The structure of a study team is depicted in Figure 8.

Although minor variations are possible, the following fundamental responsibilities need to be assigned:

- Executive sponsor
  - Sign the study announcement letter
  - Visibly and verbally endorse and support the study effort by making introductory remarks at the executive orientation
  - Review the state of the business with the study team at the start of the study
  - Participate in an interview
  - Review progress and findings during the study
  - Act as the communications contact between the study team and other executives in the organization as appropriate
  - Receive the final report and make necessary decisions
- Team leader
  - See that the study is successfully completed
  - Provide overall direction, make key business judgments, and act as liaison with other executives of the organization
  - Direct the study effort on a day-to-day basis and organize all necessary administration for the team
  - Take a lead role in executive interviewing
- Team members
  - Participate in at least some, if not all, interviews
  - Analyze the data collected and executive perspectives expressed during the study
  - Draw conclusions and perform report writing tasks
- Team secretary
  - Do typing, filing, and general secretarial services
  - Assist in getting graphics, such as matrices and information architecture
  - Confirm and follow up on executive interview schedules.

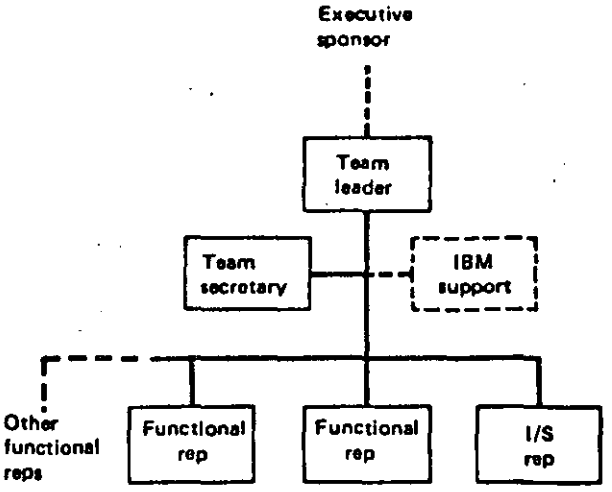


Figure 8. Sample study team structure

Considerations for a Part-Time Study

Upon completion of the BSP study, the business has only just begun the continuous planning process of managing data as a resource. Like any planning activity, it demands the attention of a full-time staff. If a full-time staff is not committed to planning, an environment is created whereby people are split between planning for the future and "putting out fires." People cannot effectively perform in such an environment and, consequently, planning becomes secondary.

The commitment and involvement of the chief executive officer are probably the most significant requirement for a successful BSP study. If he is not willing to make a full-time effort for the 8 to 12 weeks, his commitment is questionable. This lack of commitment can doom the BSP study to failure. In short, BSP should not be attempted on a part-time basis if it can be avoided. Its importance to the business demands a full-time activity.

However, should it become apparent that the study will be made on a part-time basis, then assurance should be gained that:

- The time, effort, and skills required to tightly control and maintain the work schedule will be available and committed.
- Momentum will not be lost as a result of periods of BSP inactivity.

- Effective use will be made of the administrative support available to prepare and type drafts, update charts, and perform related activities when the team is not in session. One full-time member may be helpful to manage this activity and provide continuity between sessions.

- Executive commitment is gained to relieve team members of some of their normal responsibilities.

Manpower Requirements

If the intent and methodology of BSP are followed conscientiously, and the study team is working full time, the resources required should fall within the following limits:

- Number of team members: 4-7
- Number of weeks: 8-12

These differences can be explained by variations in organizational types and sizes and in study objectives.

Communication to Management

The team leader is now in a position to compose a study announcement letter to be signed by the chief executive officer and distributed to the executives who control the major functions within the scope of the study. The letter should cover:

- Objectives and scope of the study
- Team leader and team members
- Potential value to the organization
- Need for functional executives to be involved
- Need for candor and cooperation of executives

See Appendix A for a sample announcement letter.

Brief the Study Team

The team leader, assisted by experienced IBM people, should hold a half- to full-day orientation session to brief the study team and discuss such topics as:

- Overview of BSP. This can include development of the BSP concept, study methods used, output that may be expected, and references to results obtained in other BSP studies.

- Review of activities to date. The team leader presents the scope of the study, states its objectives, and discusses the business reasons for conducting the BSP studies at this time.

- Study preparation plan and schedule. The team leader will have prepared a general schedule. After agreement on the schedule, the team members should start clearing their calendars and *prepare to attend class*.

Educate the Team

To enable a smooth, coordinated entry to the BSP study, the team members should understand the principles of the BSP methodology. Some understanding of the method of implementation, plus a review of the nature of the output that can be expected, will also be of advantage to the team members.

The team leader, working with IBM personnel, should establish an education plan for the team. Formal IBM education is available and the local IBM office can supply details.

It is recommended that all team members attend the same education session. This allows for a daily review to assure understanding and discussion of specifics applicable to their particular study. It also helps to mold the team into a productive, responsive unit.

Chapter 4. Preparing for the Study

Preparation covers that time from the completion of education to the actual start of the study. By now the scope has been established, the team leader designated, the team selected and educated, and the study start and completion dates established.

Preparation is not a full-time team effort; much of it is done by the team leader. It takes a few meetings of the team and may cover a period of two to four weeks. Many of the preparation activities are accomplished in one planning meeting after returning from the Information Systems Planning class. The following list of tasks will give a quick view of what is to be done in this phase of the study. The recommended sequence is on the first task control sheet in Appendix J.

- Obtain and equip the control room
- Review the study objectives
- Outline the final report
- Determine the facts to be gathered
- Select and orient interviewees and develop schedule
- Develop a study work plan
- Complete the task control sheets
- Set up a study control file
- Establish administrative support
- Review the status with the sponsor
- Prepare for starting the study

These tasks, many of which can be performed concurrently, are described in the remainder of this chapter. Task control sheets for these activities can be found in Appendix J. Each of the sheets is perforated for easy removal for use in scheduling.

Obtain and Equip a Control Room

The effectiveness of the BSP study depends on having a work room assigned for its duration. This room is used both as a work location for the team and for executive interviews away from the executive's office. This approach has proven very effective in allowing executives to review the work that the team has done and to interview executives without interruption.

The team will also need adjacent work space during interviewing, since all team members do not participate in each interview. The other members are either summarizing the last interview, preparing for the next interview, or performing some corollary task. This extra space is also an advantage for breakout activities after all the interviews are completed. It can be another smaller room or office, but should be in the immediate vicinity of the control room. It should not be needed before interviewing starts.

The control room should be:

- Located away from traffic, noise, and interruptions
- Large enough to accommodate the study team plus one or two other people
- Have a large amount of wall space for the posting of charts
- Be equipped with a telephone only if it can be disconnected during executive interviews
- Secure to protect confidential or sensitive material

The control room should be equipped with:

- A large table, blackboard, two flipchart stands, and lockable storage or filing cabinet. (Flipcharts are preferred to blackboards since most work is saved.)
- One-inch lined flipchart paper, magic markers, correction tape, scissors, glue, and a long straight edge.
- A supply of matrix forms, IBM form numbers GX20-2350 and GX20-2351.
- Three-ring binders with dividers, scratch pads, plenty of pencils, pencil sharpener, and erasers.
- Sheets of acetate to cover matrices and grease pencils for annotation. (These are not mandatory but do facilitate changes.)
- Coffee maker and paper towels.

Review the Study Objectives

The team should expand the study objectives developed by the team leader. Great care should be taken that every member of the team understands the objectives and agrees that the output will meet these objectives. Although the BSP study objectives in Chapter 3 are comprehensive, they may not be specific enough for every study. As an example, the sponsor should make it clear to the team members whether they are expected to recommend changes to the management systems of the business or to restrict their recommendations solely to information systems changes.

Outline the Final Report

The culmination of a BSP study should be an executive presentation to gain approval of the team's recommendations and to set plans in action. Depending upon the management practice of the business, the team may make an initial presentation to the executive sponsor, supported by a draft of a final report, or they may create a final report and make the presentation to a group of executives. The approach should be agreed upon by the executive sponsor and the outline of the report should be established during the preparation state. The outline delineates the contents and establishes the sequence of the sections. Thus, as material is generated, it can be placed easily in the report.

At this time, team members should be assigned specific responsibilities for sections of the report. It is their responsibility to coordinate and deliver their sections. Appendix I is a recommended outline of the report.

Determine the Facts to Be Gathered

To provide a basis of information, the team should establish a reference library of both business and information systems documentation. With the BSP methodology, all of the information required for the study must be available in the control room, either in the knowledge of the team or in the documents that have been gathered; the exception is the information provided through executive interviewing. Therefore, no time is allotted during the study for going outside the control room to gather facts.

Facts will be of two types—general business and information systems. All the items listed below under "General Business Facts" will be appropriate also for the information systems area, and those unique to information systems will be listed afterward.

The team should consider the use of a data dictionary for recording and manipulating gathered and derived facts during a BSP. The benefits of this are discussed in Appendix L.

General Business Facts

The major business facts to be gathered may be classified into seven categories.

- Environment (defined as those items outside the scope of the study and over which the business has little or no control) – economy, government regulations, labor, consumerism, competition, industry position and industry trends, suppliers, and technology.

- Objectives – statement of company mission, major goals and objectives for the business and objectives for each of the major functions and product lines.

- Organization – organization charts for the total company and for each of the people to be interviewed showing position, names, numbers of people, expense budget, and expected changes.

- Planning – planning process and planning calendar, business plans showing major projects, resources, schedules, and financial. Also major studies, task forces, and committees formed during the previous three years. (Each of these should show a reason for the activity, members, status, and results.)

- Measurement and control – names and general contents of all major business measurements and controls such as reports, project initiation and funding, and type of project control. These should provide measurement of the status of each of the critical success factors.

- Operations – products and markets, geographic distribution of the company showing major physical locations and a general indication of functions performed there, financial statistics, and sizing statistics showing the number of vendors, number of employees, orders, customers, shipments, purchase orders, and related information.

- Information systems support – availability to the user of major applications and data processing facilities, such as terminals, programming languages, text processing, word processing, financial and business planning systems, and query services.

Information Systems Facts

Although most of the information systems facts to be gathered are the same as those for general business, the following is a list of items that are unique to information systems.

- Mode of operation (data base, conversational, distributed, user capabilities, others)
- Systems justification requirements
- History of major projects started in the last two years
- Current systems descriptions
  - Title
  - Major functions performed
  - Major input and output
  - Users supported
  - Technology (batch, online, query, data base, other)
  - Age
  - Summary of known weaknesses
  - Summary of committed enhancements
  - Source (other location, developed on-site, purchased, leased)
  - Critical factors ("Must execute and process all orders by 2:00 p.m. in order for warehouse to pull stock.")
- Planned systems and data bases
- Sizing statistics such as number of programs, jobs per period, users, files, data elements, and turnover

Many of these business and information systems facts will be valuable in assessing the ability of the company to implement the recommendations of the study. The business priorities need to be matched to the information systems abilities.

Responsibilities for gathering data on each of the items on the list should be assigned to the team members, but they should not be expected to do all of the data gathering themselves. In many cases, executives will be asked to prepare and submit facts, particularly if the facts are voluminous and the team needs summaries or extracts. As many of the facts should be charted as practical and should be put on the control room wall for ready reference, particularly those that will be used in interviewing, such as objectives or critical success factors.

After the facts are gathered, code each one of the items, make a list of them, and get them prepared for review or explanation on the study start day. During the presentations by the team leader and the director of information systems at the start of the study, they may wish to refer to each of these items so that all the team members know of the existence of that data, its pertinence, and general content. Since much of it will be of a confidential nature, it should be filed in a lockable cabinet or within a lockable control room.

Select and Orient Interviewees and Develop Schedule

Selecting the Interviewees

A preliminary list of executives to be interviewed should be prepared at this time. The executives selected will normally be within the top two or three levels of the company and be responsible for the major functions included within the scope of the study, or they may occupy major staff positions. The number of interviewees will range from 20 to 30, with an average of 25. Conducting more than 30 interviews should be done with great caution as each additional interview seldom adds new or significantly different information. Also, more and more detail is added that is not needed and may detract from the general objectives of BSP.

Scheduling Executive Interviews

When scheduling the executive interviews, the team should consider all factors that may inhibit the full participation of the executives, such as vacations, annual planning meetings, or public holidays. Although the chief executive is usually interviewed last, there is no set rule for the interview sequence for the other executives. The team should schedule an easy interview with an amenable executive for the first and possibly the second interview.

Schedule the interviews to allow time for a checkpoint review of the results and procedures. A good way to do this is to schedule the first three interviews and then allow a half day for review. While the third interview is going on, all the material from the first two interviews can be completed, typed, and ready for review and discussion. Also, allow a half day midway through the interviews to analyze the problem analysis sheets and summaries of the interviews to date and to determine if the team is getting everything from the interviews that it should and to allow time for making any changes. Lastly, allow a half day for the extra preparation that goes into the chief executive officer's interview. Since he should be interviewed last, this half-day slot will be scheduled near the end of the interview process.

Plan six to eight hours to prepare for, conduct, and summarize each interview. Allowing time for the three half-day reviews makes it possible to conduct 17 interviews in 10 days' time.

Developing the Executive Orientation

All interviewees should be briefed prior to the start of executive interviewing. Executives other than interviewees may also attend the meeting. The executive orientation is prepared, the presenters selected, and exhibits prepared so that the orientation may be reviewed with the executive sponsor.

Major topics that should be included in the orientation are:

- Executive level overview of BSP
- Study scope
- Objectives of the study
- Business reasons for conducting the study
- Review of the schedule and expected output
- Executive involvement – why and what is expected of the executive
- Schedule for the start and completion of the interviews
- Topics to be discussed in the interview

The executive orientation should be scheduled as soon as possible after the sponsor's review so that the executives understand the importance of the involvement of their subordinates in the study, the requests for information that they may receive during the fact-gathering, and to minimize misunderstandings of the BSP study that might otherwise arise.

Conducting the Orientation

The orientation is normally given by the team leader and may involve one other team member. The executive sponsor should open the meeting, introduce the team, emphasize the business reasons for initiating the study, reiterate the objectives, scope, and expected output, and offer his commitment and support to a successful completion of the study. The executive sponsor should make it clear that the study team will not complete its work in isolation in the planning room. The role of the study team is to be a catalyst to provoke and document the best thinking of all the executives in the business. Their final recommendations will be based on the needs of the entire business, independent of parochial interests.

If all of the interviewees cannot get to the one session because of conflicts in schedules or residence at remote locations, then a second briefing should be conducted or a personal call made on the executives by the team leader.

The allotted time should be one to two hours for this session and should concentrate upon the objectives, the output, and the expected executive participation. The team should prepare a handout outlining the major topics to be discussed.

Executives should come out of the briefing with a good knowledge of what is going to happen during the interview, what material they will see, the types of questions that will be asked, and what will be expected of them.

Develop a Study Work Plan

Now that the team knows how many people it will interview (pending approval by the sponsor), what the presentation and final report will look like, and the facts that must be gathered, it can proceed to the development of a work plan. The plan covers all activities and their schedules, from preparation to final reporting. The detailed tasks to be completed in each of the phases are used to further delineate this work plan at the appropriate time. The general flow of the study and the control points are shown in Figure 9.

Control Point	Participants	Activities
①	Study team	Review study to date, documentation standards, and team understanding of results. Confirm resource allocations for the next stage.
②	Executive sponsor	Results to date. Update the study plan. Review the executive interview objectives.
③	Executive sponsor	Report on both qualitative and quantitative results of the executive interviews. Present and validate the assessment of business problems and benefits. Update the study plan.
④	Study team	Team agreement on all major issues. Review all supporting documentation to be completed. Confirm resource allocation. Update the plan.
⑤	Executive sponsor	Review all major findings and recommendations. Demonstrate an understanding of the business and its requirements. Gain executive sponsor's agreement that the team is qualified to present the recommendations.

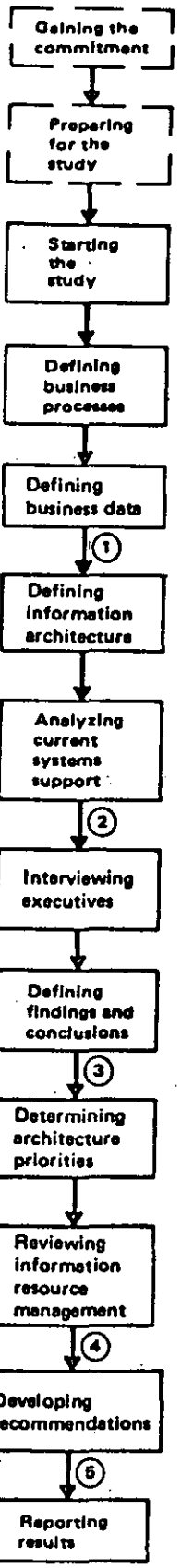


Figure 9. BSP study control points

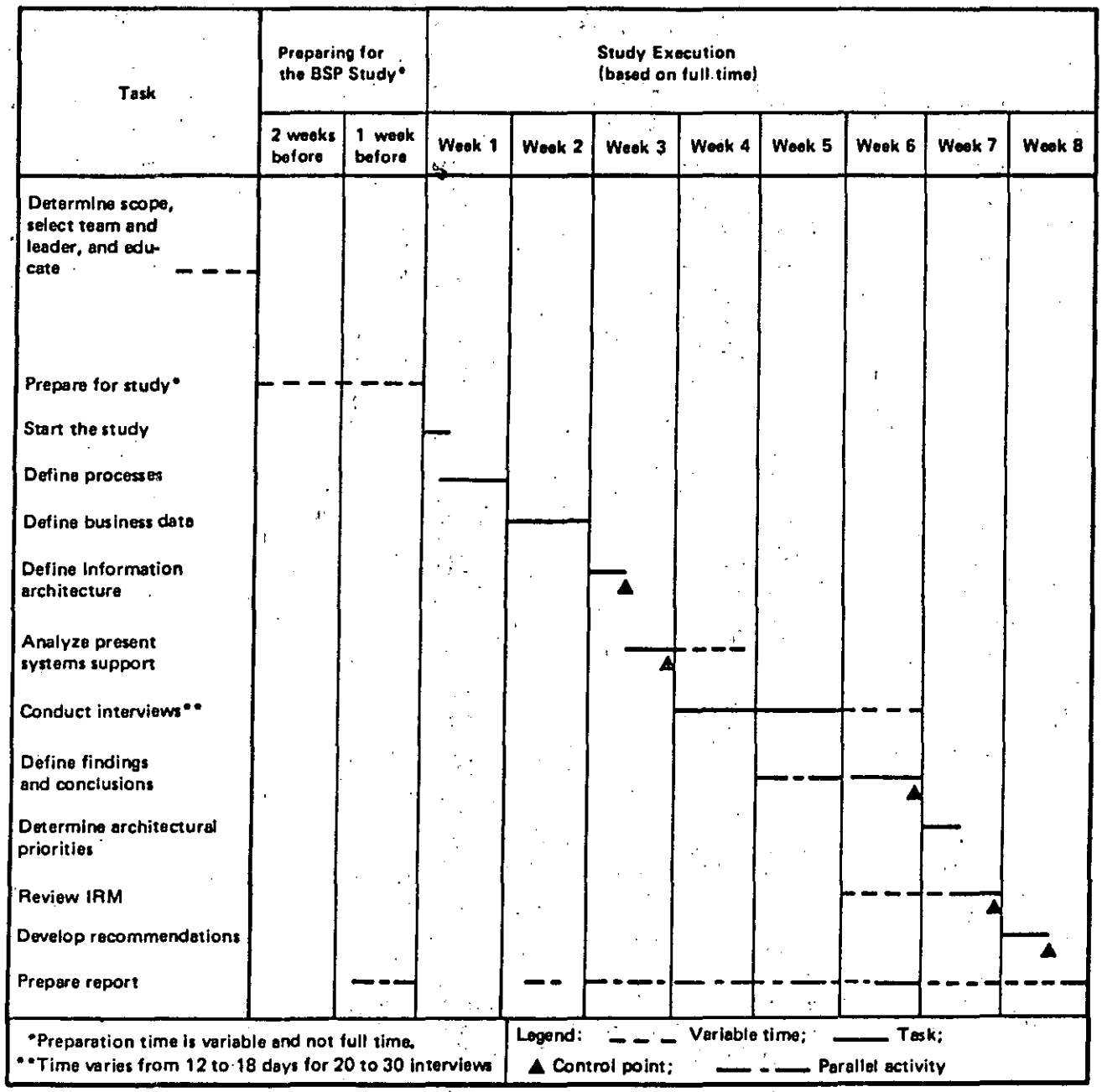


Figure 10. Work plan for the BSP study

When the activities of this study and the project control points have been established, they should be combined into a single action plan and represented on a Gantt chart similar to that shown in Figure 10. The final plan should be put on a chart on the wall so that it can be referenced during the interview. A copy should be made for review with the sponsor.

Complete the Task Control Sheets

If the study is to have an adequate project control system, the BSP study team should develop task sheets showing scheduled completion dates and the team member(s) responsible for each task. The task control sheets in Appendix J list the activities the team normally will perform. The team may wish to add other items or to separate some of these items into subtasks. The first task control sheet in Appendix J is for the preparation activities. The other task control sheets cover activities ranging from starting the study through analyzing the current information systems support, executive interviewing, and the last phase ranging from the development of the recommendations and action plan through reporting results.

One or more members of the team will be responsible for each of the activities on the task control sheets.

This responsibility includes ensuring that the completed task represents the best thinking of the team members after analysis and any required documentation. After completing the task control sheet, check the final date against the study plan to be sure that there is no conflict in the completion dates for that phase.

Set Up a Study Control File

A control file should be established to contain all material pertaining to the study. Having and using this file avoids considerable clutter in the control room and prevents loss of materials. Because of the nature and number of documents involved, they should be stored in a filing cabinet that can be locked. Suggested contents include:

- Administrative
  - Key correspondence
  - Objectives and scope of the study
  - Study work plan
  - Task control sheets
  - Sponsor reviews
- Data gathered during preparation
- Master copies
  - Processes
  - Business entities
  - Data classes
  - Matrices
- Analysis
  - Executive interview notes
  - Executive interview summaries
  - Problem analysis sheets
  - Findings and conclusions
- Report and presentation

In addition, the team may find it beneficial to give each member a three-ring notebook with dividers for such things as study work sheets, task control sheets, processes, entities, data classes, matrices, interview summaries, and problem analysis sheets. These notebooks are particularly handy when the team members are working in different places, as during interviewing.

Establish Administrative Support

Typing and general administrative support (mainly copying) will be required by the study team and should be estimated and organized before the study. The typist should have access to a typewriter with a memory device because of the number of corrections to such items as the descriptions of business processes and of data classes. If the format of the final report is not established early, the memory device will aid the typist in retyping in the format and type style required.

Administrative activities are usually concentrated toward the latter part of the study and will include executive discussion summaries, work plans, status reports, results of analyses, charts and matrices, final report, confirmation of interviews, and the generating of the final presentation materials.

Since the BSP methodology develops information on many subjects that are interrelated (for example, objectives, critical success factors, processes, problems, organization, data classes, subsystems), the team may wish to consider the use of an automation tool such as a data dictionary. The advantages of using the IBM Data Dictionary are covered in Appendix L and in the IBM Los Angeles Scientific Center Report, G320-2705, dated July 1980.

Review Status with the Sponsor

Before conducting the executive briefing, gathering of data, and scheduling of the executive interviews, all plans should be reviewed with the sponsor. Any issues that may have arisen during the preparation should be resolved during this time. Items to be reviewed with the executive sponsor are:

- Study work plan with emphasis on dates to eliminate conflict with other major company activities
- Planned participation by the sponsor in starting the study (a general list of topics for his discussion with the team should have been previously prepared and reviewed with him at this time)
- List of executive interviewees including a tentative interview date for the sponsor
- Study objectives
- Facts to be gathered
- The approval process so that it can be determined what form the presentation will take, who it will have to be presented to, and in what sequence in order to gain approval
- Outline of the final report
- Outline of the executive orientation

Prepare for Starting the Study

Up until this time, many of the activities had been done by the individual team members or had not demanded the presence of everyone. At the start of the study, all

team members must be present whenever the team is meeting. In preparing to start the study, recheck to ensure that the sponsor is ready to present his overview of the business, see that the material is ready for review by the team leader, and ascertain that the information system director is ready to present a profile of present data processing. If all of the activities listed in this and the preceding chapter have been strictly adhered to, then the team should be well prepared for a successful study.

Chapter 5. Review the Business Environment and Objectives

Conducting the start-up meeting is the first major activity in the formal execution of a BSP study. It is the start of full-time activities by the team; until now the team has met a few times to prepare for the study but has probably not worked continually as a team. This, and the next five major activities (see Chapters 6 to 10) are all aimed at understanding the business requirements and data processing support as they exist today as well as the business requirements for the future.

The primary purpose of the start-up meeting is to review the business and the data processing support. This can be accomplished through three presentations. The executive sponsor should present the status of the business and his views of the future of the business unit. The team leader then provides more detail by discussing the material that has been gathered and his views of the business. The director of Information Systems gives a review of the data processing support. These three presentations and the review of the study work plan are covered in the following paragraphs.

Obtain the Sponsor's View

The executive sponsor gives the first overview to the team. His presentation gives the team a base and direction for many of the succeeding activities. He should cover the following topics:

- Industry trends
- Major strategies of the business
- Activities that are critical to the success of the business
- Key measurements
- Direction of the business in the future
- Major problems or inhibitors today and in the foreseeable future
- How better information can help the business succeed
- What output (results) does he expect from the BSP study

Since the executive sponsor is one of the last people interviewed, this presentation allows the team to understand his views as an aid to interviewing the other executives. The team can then check other views of the business with those of the executive sponsor in the formal interview. In essence, his is the first and last of the executive perspectives gained by the team.

Review the Business Facts

The team leader should be prepared to present the business-related facts gathered during the study preparation phase. This information should serve to broaden the business perspective of the team. While time will not permit all the material to be reviewed in detail, each item should be discussed as to content and importance. This will give the team an awareness of what information is available to them. To facilitate recovery from the study control file, each item may be coded and a list may be given to each team member.

Some of the material will be used frequently throughout the study. During preparation for the study, much of this material will have been charted and put on the wall of the control room. The team leader may also elect to distribute selected handouts to each team member for review and/or research. All presentation material, including detailed backup, should remain in the control room for continued reference.

Review the Information Systems Facts

The director of information systems should present a summary of the information systems facts gathered during preparation for the study. It should be directed toward helping the study team understand the support that information system provides the enterprise.

The major projects installed and being developed should be discussed with the study team, as well as the applications that are being developed and used, that is, batch, conversational mode, and data base technology. The discussion should include whether these projects have met or will meet their objectives.

Another area that should be discussed is the information systems interface with its users, the responsibility and involvement the user has in the development and justification of applications, and who has responsibility for data within the company.

A copy of the presentation should be made available to each of the team members as it will help them in interviewing, in developing findings, and in drawing conclusions and recommendations.

The following might be used as an outline for presentation of the information systems function:

- Mission and objectives
- History and background of data processing
- Organization of the information systems function
- Overview of major applications systems installed and planned
- Present distribution of data processing and data ownership
- Functions now available to users (for example, APL, query, text processing, and other applications)
- Relationship with users
- Project approval process
- Key problems and challenges

Review the Study Work Plan

If the work plan was not developed by the whole team, the team leader reviews both the plan and the Gantt chart developed during the study preparation phase. The work plan is used as a point of departure for discussing and completing:

- Major events, major output, and schedules
- Individual assignments
- Project control

This review can also serve as an opportunity to discuss administrative details such as secretarial support, handling of phone messages, working hours, office security, expense charge numbers, and the like.

Chapter 6. Defining Business Processes

Business processes are defined as *groups of logically related decisions and activities required to manage the resources of the business*. They are studied and identified without regard to the organization responsible for them. The reason for defining the processes is that doing so will provide or lead to:

- Information systems that are largely independent of organizational changes
- An understanding of how the business accomplishes its overall missions and objectives
- A basis for defining required information architecture, determining its scope, making it modular, and setting priorities for its development
- A basis for defining *key* data requirements

Lists of processes, along with some sample descriptions from both the public and private sectors, may be found in Appendixes C, D, and E.

Prerequisites to Defining Processes

The following points are particularly relevant to developing good process definitions:

- All members must be present and participate throughout the exercise and there should be general agreement on the expected output before the exercise begins.
- Note-takers should be designated right from the beginning so that decisions and definitions of processes will not be forgotten or misunderstood later.
- The team must understand the concept of resources and their life cycle and must identify the key product(s) and/or service(s).

Product and Resource Life Cycle

A four-stage life cycle of the product/service and of each of the supporting resources is used to logically identify and group the processes. The life cycle normally used is:

- Stage 1 – Requirements, planning, measurement and control
- Stage 2 – Acquisition or implementation
- Stage 3 – Stewardship
- Stage 4 – Retirement or disposition

A description of each of the life cycle stages follows:

1. *Requirements* – activities that determine how much of the product or resource is required, the plan for getting it, and measurement and control against the plan.

2. *Acquisition* – activities performed to develop a product or service or to get the resources that are going to be used in its development. In manufacturing this would include procurement and fabrication; in personnel, the hiring or transfer of people; in education, the development of a curriculum and the enrollment of students.

3. *Stewardship* – activities to form, refine, modify or maintain the supporting resources and to store or track the product/service. In the insurance industry this could be policy maintenance, premium notices and dividend statements; in the distribution industry it could include inventory control and warehousing.

4. *Retirement* – those activities and decisions that terminate the responsibility of an organization for a product or service or signal the end use of a resource. This might include the selling of space on an airline, the discharge or retiring of an employee, the scrapping or selling of a capital asset, or the discontinuance of a service by a government agency.

The life cycle serves as a vehicle for structured, logical, comprehensive identification of the processes by the team.

Basic Steps in Defining Processes

An overview of process identification is provided by Figure 11. Taking the product/service and each of the support resources through their life cycle provides a definition of their respective business processes. Because strategic planning and some management control are not solely product or resource oriented, they must be considered separately to ensure that all processes of the organization are identified. Two examples of processes that involve all enterprise products/services and resources are *determine corporate strategy* and *formulate resource policy*.

After the product and each of the supporting resources have been taken through their respective four-stage life cycles and the processes have been identified, it is no longer necessary to retain the process/stage relationship for process definition, but it will be helpful later in defining the information architecture. Since the life cycle is an artificial means for directing the team's thinking, very little time should be spent deciding in which stage a given process appears. The recording of the process is more important than the stage is appears in.

The measurement and control processes identified under the requirements stage will contain both management control and operational control processes. These requirement processes should be compared to the management control processes identified first and any redundancies should be eliminated.

When identifying the processes, the team should use a verb/object form for the process name. For example, rather than calling a process *engineering*, the process should be called *engineer product*. This verb/object forms helps eliminate confusion between the process and an organizational unit.

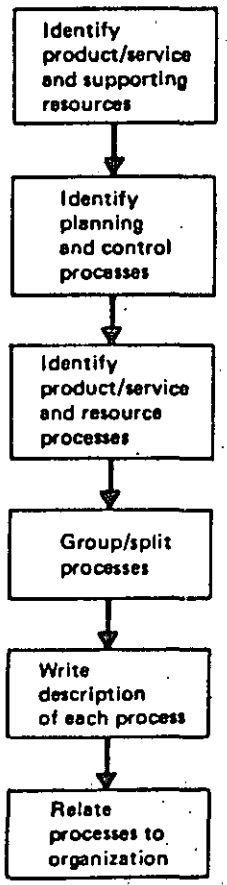


Figure 11. Definition of business processes

Identify Product/Service and Supporting Resources of the Organization

For purposes of this exercise it is assumed that there is one major product group or that the products are managed through similar business processes. In the case of the public sector and some service organizations, it will help to go back to the goals to best specify the product or service. For example, one might think that the product of a health insurance organization would be its policies, but an examination of its goals might show the products to be (1) service to the subscriber and (2) payment to the provider. In like manner, an examination of the goals might keep one from assuming that the curriculum is the major product of a university.

If there are two or more groups or families of products and services, then each group or family should be taken through the identification analysis.

In addition to the product/service, the support resources also need to be identified. A resource is best described as that which a business consumes or uses in meeting its goals. There are four basic resources that we deal with:

- Materials
- Money
- Facilities/equipment
- Personnel

Although the use of these four should be adequate, some organizations may find it appropriate to break them down further. For example, a university may choose to divide personnel into faculty and administrative personnel; a manufacturer may choose to divide facilities/equipment into buildings and production equipment. Such divisions can be useful if the team feels that the management of these resources is handled differently by the organization.

Identify the Processes of Strategic Planning and Management Control

With the preparatory work that was done in collecting the information on planning, critical success factors and their measurement, and samples of the organization's plans, it should not be difficult to identify the processes involved. They will normally be grouped into strategic planning and management control. Strategic planning may be referred to as the long-range plan, the seven-year plan, or the development plan. Management control may be referred to as the operating plan, the management plan, the resource plan, and sometimes the contract plan. In some companies the budget may serve

as a major tool for management planning and control. Examples of planning and control processes are shown in Figure 12.

Strategic Planning	Management Control
Forecast economy	Forecast product
Develop organization plan	Plan working capital
Plan divestures/acquisitions	Plan staff levels
Develop organization goals and objectives	Develop operational plan
Determine product line	Develop budget
	Measure and review performance

Figure 12. Planning and control processes

For a detailed discussion of the theory of strategic planning and management control the reader may wish to refer to Robert N. Anthony's book entitled *Planning and Control Systems: A Framework for Analysis.*

Identify the Product/Service and Resource Processes

The general approach in identifying processes is to take each product/service and support resource through the life cycle, starting with the requirements stage and working through the succeeding stages. Although there is no prescribed number of processes in each of the stages, most studies result in 30-60 processes for the organization being studied. The team should tend to identify more processes than necessary, with the idea of grouping them later as necessary.

With the entire team working together, it is helpful to draw a large matrix with the product/service and support resources listed across the top and the stages of the life cycle down the vertical axis. There is no secret formula for this identifying of business processes, so one should not be surprised if the first attempt results in many more processes than are needed and a great inconsistency in the levels. Do not let this deter you; it will be corrected during the grouping/splitting of processes step.

It is very important that every team member participate at all times, since business processes are the basis for virtually everything that follows in the BSP study. Getting started is far more important than any theoretical discussions on the definition of a process group or a process, logic, theory, or other diversions.

Figure 13 is an example that might result from performing this analysis with a manufacturer.

Group/Split the Processes

The processes have been identified by considering what processes are required for strategic planning and management control, product/service, and the support resources. There may be a need to group or split some of these processes to:

	Product/Service	Supporting Resources			
	Men's Apparel	Raw Materials	Facilities	Cash	Personnel
Requirements	← Establish Business Direction →				
	← Comply with Legal Requirements →				
	Forecast Product Requirements Analyze Marketplace Design Product	← Plan Seasonal Production →	Determine Facilities and Equipment Requirements	Determine and Control Financial Requirements	Determine Personnel Requirements
Acquisition	Schedule and Control Production	Purchase Raw Materials	Acquire Facilities and Equipment	Manage Cash Receipts	Hire Personnel
Stewardship	Control Product Inventory Ship Product	Control Raw Materials Inventory	Maintain Equipment Manage Facilities	Determine Product Profitability Manage Accounts	Manage Personnel
Disposition	Advertise and Promote Product Market Product Enter and Control Customer Order	Schedule and Control Production	Dispose of Facilities and Equipment	Manage Cash Disbursements	Terminate Personnel

Figure 13. Sample process identification for a manufacturer

- Reduce inconsistencies in level. If, for example, a manufacturing firm has identified a process called *produce the product* and also one called *administer health benefits*, this would indicate a difference in level. The produce-the-product process should be broken down into several processes and/or the administer-health-benefits process should be grouped with other personnel processes to bring these processes to the same level.

- Combine the processes where commonalities occur. As an example, purchasing may have been identified as a process in the acquisition stage, not only for the product but also for materials and facilities. Assuming the processes are similar, these should be combined into one process. As the process were grouped during the requirements stage, there may have appeared processes entitled manpower loading, develop workflow plan, and schedule supply materials. These three might be grouped with the management control process of *develop operational plan*.

Write a Description of Each Process

A member of the team should have been making notes on the decisions and activities associated with each of the processes as they were defined by the group. Although this is a laborious task, it is absolutely mandatory if the process descriptions are going to reflect the knowledge of the team and the amount of time that has been spent in defining them. The descriptions may be either in list or narrative form.

The following are examples of process descriptions (further descriptions may be found in Appendix E):

Plan production – The activity of planning for and coordinating materials, personnel, and machines in order to produce the finished products needed to meet forecasted requirements. Included in this process are the activities of:

- *Capacities and capabilities planning* – the balancing of production need with ability

- *Scheduling* – the scheduling of labor and material needs to meet production and shipping requirements; also, the scheduling and balancing of the product mix

- *Material requirements* – the calculation of raw material needs to meet a schedule, recognizing optimum inventory levels, economic order quantities, substitutions, and related information

- *Costing* – establishing standard raw material costs and product costs on the basis of these and other manufacturing and administrative cost factors

Purchase raw materials – The activities of acquiring materials, machinery, and supplies of specified quality on a timely basis at the best price. Included in this process are the activities of:

- *Supplier evaluation and selection* – the searching for, evaluation of, and selection of suppliers who meet requirements for materials, packaging, equipment, and delivery at competitive prices (includes internal search)

- *Order placement and follow-up* – the actual placing of purchase orders with approved suppliers for quantities of raw materials as specified (may include raw material routing) by production planning, or for equipment acquisitions approved by management (includes release of customer-purchased materials)

- *Receipts and inspection* – the receiving or returning of purchased materials, machinery, and supplies, verifying quantity and quality, and documenting the activities

Relate the Business Processes to the Organization

Once the business processes are agreed upon and described, they can be related to the organizational structure of the business to help the study team identify any additional people that should be interviewed and to further clarify their understanding of the business processes. Some teams may wish to complete this matrix before completing the process descriptions. Relating the organization to the processes also helps the team determine the information needed from the interviews, such as verification of executive involvement in the processes.

To relate the business processes to the organizational structure of the business, the team develops a process/organization matrix. Essentially, this is a graphic representation of one aspect of the management system of the organization because it illustrates who makes the decision in each of the processes.

To prepare the process/organization matrix, the study team uses the business processes already identified. Using their knowledge and perspective of the business and the organization charts from the study preparation phase, the team members identify the organizational units involved in the processes. To simplify this task, the following IBM forms are available: BSP Matrix Form, Large Grid (GX20-2350) and BSP Matrix Form, Small Grid (GX20-2351).

Since the BSP study is intended to provide a broad overview of the business, not every organizational unit is identified. Furthermore, common similar organizations can sometimes be represented as one organizational unit. For example, 100 sales offices may be listed as a single unit. Where feasible, plants and laboratories should also be grouped into units. One-unit representation is also generally appropriate when organizations of different scopes are doing the same job. For example, a financial planning organization may have three departments, each focusing on separate divisions. The mission of all three is still financial planning, and they can be represented by one organizational unit The process/organization matrix is illustrated in Figure 14.

Once the processes and organizational units are arranged on the matrix, the study team completes the matrix by indicating the degree to which each organizational unit is involved in the processes. The following symbols are used in Figure 14 to indicate the degree of involvement:

☒ Major responsibility and decision maker
☒ Major involvement in the process
☑ Some involvement in the process

The above definitions define the levels of decision making within the organization. If these definitions are not suitable, the team should customize them.

Such indicators do not describe the actual responsibilities of each of the organization units but serve only as a guide to assigned responsibility for and involvement in a process. Some businesses have found this matrix to be valuable after the study as an index to a management system manual, in which they develop responsibility and activity statements for each of the process/organization intersects.

The process/organization matrix helps the study team validate the list of individuals to be interviewed and determine the questions to be asked of the individuals responsible for the processes. To authenticate the matrix, each person interviewed by the team should be asked to confirm or correct the portion of the matrix showing his responsibility or involvement.

The process/organization matrix sometimes will show overlapping responsibility and decision-making authority for a process or a lack of decision-making responsibility for a process where it normally would be appropriate. Such potential problem areas are clarified later during the executive interviews.

Results (Output) and Their Uses

The output to be expected from defining the business processes consists of:

1. A list of processes
2. A description of each of the processes
3. A process versus organization matrix
4. Team understanding of how the organization under study operates and is managed and controlled

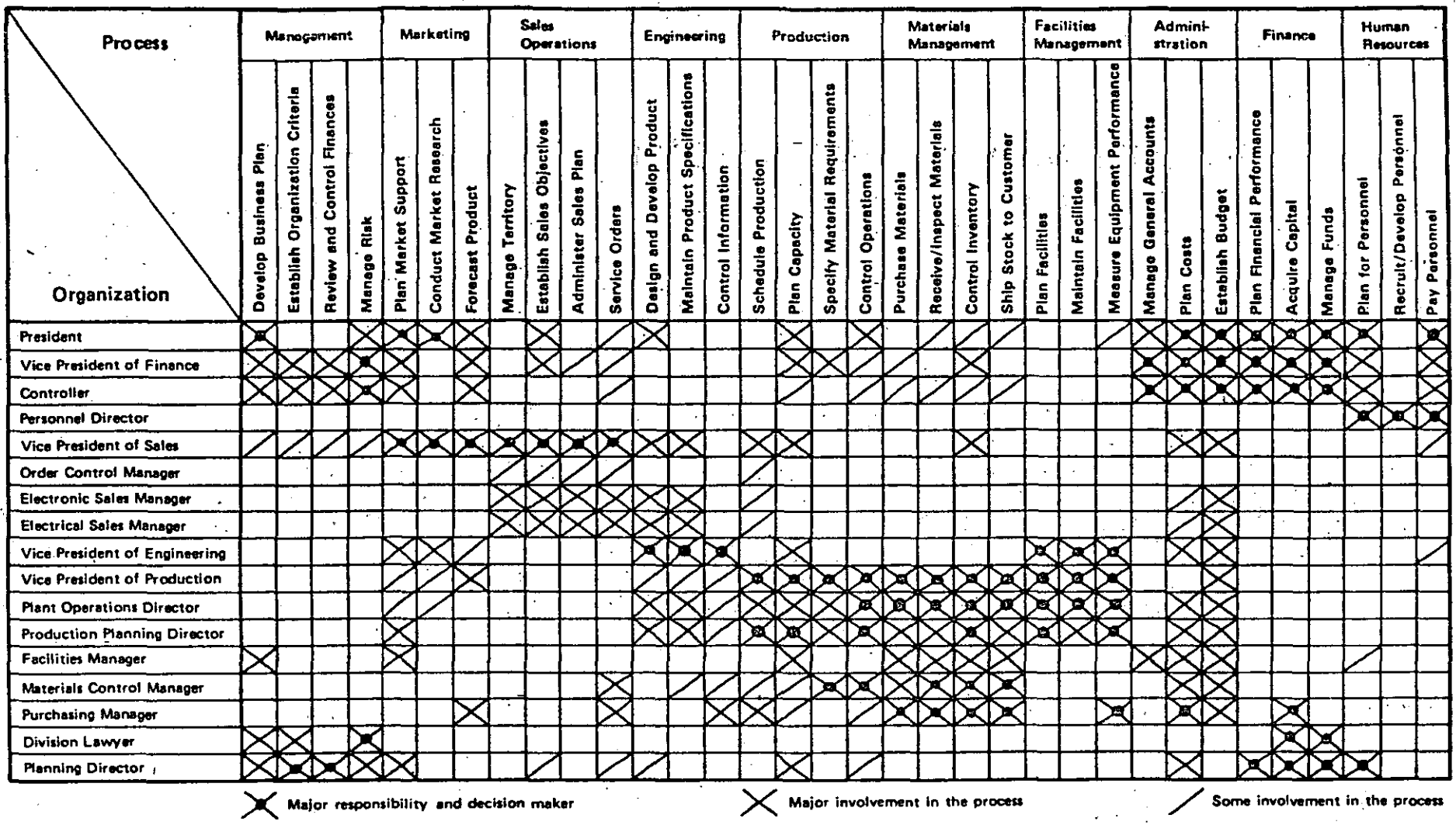


Figure 14. Process/organization matrix

As mentioned earlier, business processes serve as the base for most of the activities that follow. The ultimate use of the business processes is to identify opportunities and requirements for the use of information systems to support the business. That is also one of the basic goals of the BSP study.

Expansions and Variations in Approach

Although following the above procedure should result in well-defined processes and supporting material, other considerations and variations are worth mentioning. The more salient of these are covered in the remainder of this chapter.

Develop/Analyze Product and/or Transaction Flowcharts

A flowchart of the product/service processes (see Figure 15) serves several purposes:

- It helps in verifying that all the processes have been identified.
- It helps in determining if the team really understands the business processes associated with the product/service.
- It serves as a model for the subsequent definition of the information architecture.
- It helps in identifying the processes involved in managing the supporting resources.

Outside Assistance

Even though the team members were selected because of their knowledge of the entire business unit, there may be gaps in that coverage. If so, the team should get outside help where necessary for a thorough understanding of the processes.

Alternate Methods in Identifying Business Processes

Predefined checklists of processes may be used for the business being studied. These may be found in Appendixes C, D, E, other BSP studies, trade publications, or IBM industry publications such as those for the Consumer Based Information System (CBIS) or the Communications Oriented Production Information and Control System (COPICS). If this method is used, care should be exercised so that the processes are well understood by all the team members and represent the activities and decisions of the business being studied.

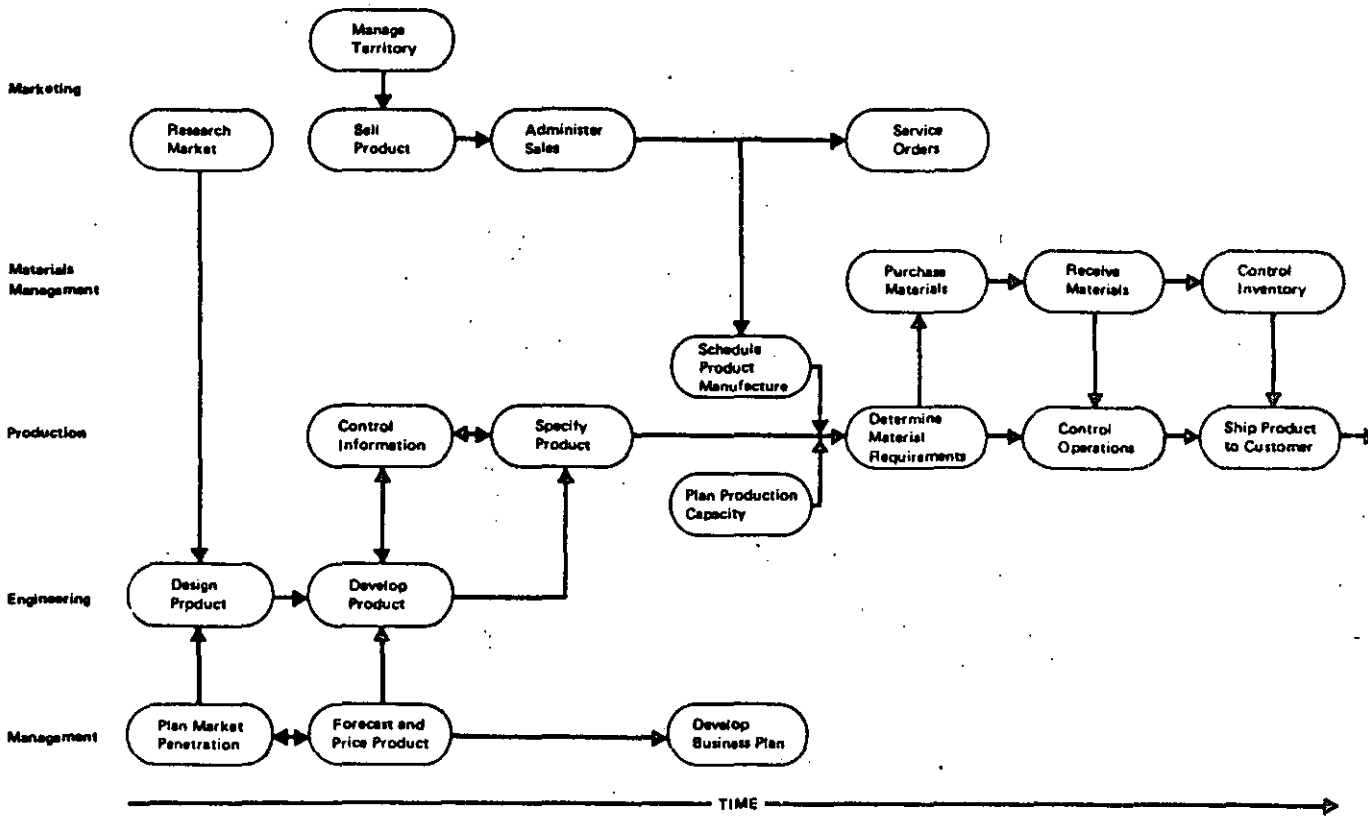


Figure 15. Example of the flow of a product/service through a business

Chapter 7. Defining Business Data (Business Entities and Data Classes)

Once the business processes have been identified, the next step is to identify and define business entities, data classes, and their relationships.

A *business entity* is something of lasting interest to the enterprise — something about which data can be stored and which can be uniquely identified. Business entities are what the enterprise manages and they serve as a basis for identifying the data needs of the enterprise. In addition, business entities provide a foundation for the definition of the enterprise's information architecture. This architecture, which normally is developed following the BSP study, will provide the guidance necessary for ongoing data base design. Therefore, the identification of entities and the definition of those entities are a valuable output of the BSP study.

A *data class* is a logical grouping of data related to things (or entities) that are significant to the organization. Such grouping permits a long-range information architecture to be identified. The data classes should represent data that must be available for business activities and decision making. They should *not* represent a particular format (e.g., report or display) of how the data is currently being used, since this would confuse the existing implementation with the data itself.

Data classes are identified in order to:

- Determine data sharing requirements across processes.
- Determine data that is necessary but either unavailable or insufficient for business use.
- Establish the groundwork for data policy formulation (including data integrity responsibility).

To enable assignment of responsibility for data integrity, data classes must be defined so that there is one and only one process that creates each data class.

The BSP study should include all required data. Currently, some of it may be automated, some may be processed by manual systems, and some may not be recorded at all. Further BSP analysis (particularly executive interviewing) will enable the team to determine which data classes need to be enhanced (or upgraded) to improve business decisions.

The study team should define each data class identified. The definitions should be in complete sentences, with enough detail to differentiate clearly one data class from another. In most cases, the study team will identify 30 to 60 data classes.

Once business entities are identified, each business process is examined for data that the business process uses and creates. This data is then associated with the entity that it describes. Data classes are then identified by grouping data in such a way that one and only one process creates each kind of data. Finally, the definitions for each data class can be written. Data class and business entity examples can be found in Appendix E.

In summary, the definition of data classes takes four steps, as follows:

1. Identify and define business entities
2. Determine data usage and creation for each process
3. Identify data classes
4. Define data classes

The balance of this chapter covers these steps in detail.

Identify and Define Business Entities

A business entity is *something* of lasting interest to an organization — something that an organization wishes to keep data about. Entities may be internal or external to an organization, and can be categorized as one of the following: person, place, thing, concept, or event. An organization must be able to identify each occurrence of an entity uniquely. Therefore, each entity should have a unique identifier associated with it. For example, the unique identifier for the employee entity could be employee number.

The BSP team can use the above categories of entities to help structure their entity discussions. A partial list of entities for a manufacturing company might include:

- *Person*: employee, customer, supplier
- *Place*: retail store, warehouse, plant
- *Thing*: equipment, part, product
- *Concept*: job, legal requirements, organizational unit
- *Event*: purchase order, shipment, customer order

After identifying an initial set of entities, the team should review the list and combine and/or split entities as necessary to arrive at a useful number of entities — comprehensive enough to cover the scope of the study and roughly equivalent in level of importance. Each entity should be carefully defined in detailed and complete sentences.

Determine Data Usage and Creation for Each Process

The second step in data class definition is to identify what data must be available and what data is created by each business process. To do this, a data usage analysis sheet is completed for each process (see Figure 16). The BSP team must identify, for each process, the types of data that are used to perform the process and the types of data that the process generates. Each type of data identified is matched to the entity it describes.

At least two advantages are gained by first identifying what data is needed and then relating that to an entity. First, the exercise forces a clarification of business entities — omissions and inconsistencies becoming readily apparent. Second, when all usage analyses are complete, the relationships between processes and data about entities will have been established.

Identify Data Classes

This knowledge of the relationship of data to processes leads directly to the identification of data classes.

A data class represents a category of information about an entity. Therefore, an obvious choice of data classes would be to assign one category of information to each entity. However, in order to be able to manage the integrity of data, there must be *no more than one* source for its creation — one source that will be held accountable for its accuracy, currency, etc. Thus, data about each entity is broken into multiple data classes whenever multiple processes create different data about the same entity. Typically, a BSP study will define 30-60 data classes.

Looking at the "Data Created" column in data usage analysis, it is possible to identify entities for which there must be more than one data class. For example, since the analyses in Figure 16 show three different processes creating three different kinds of information about

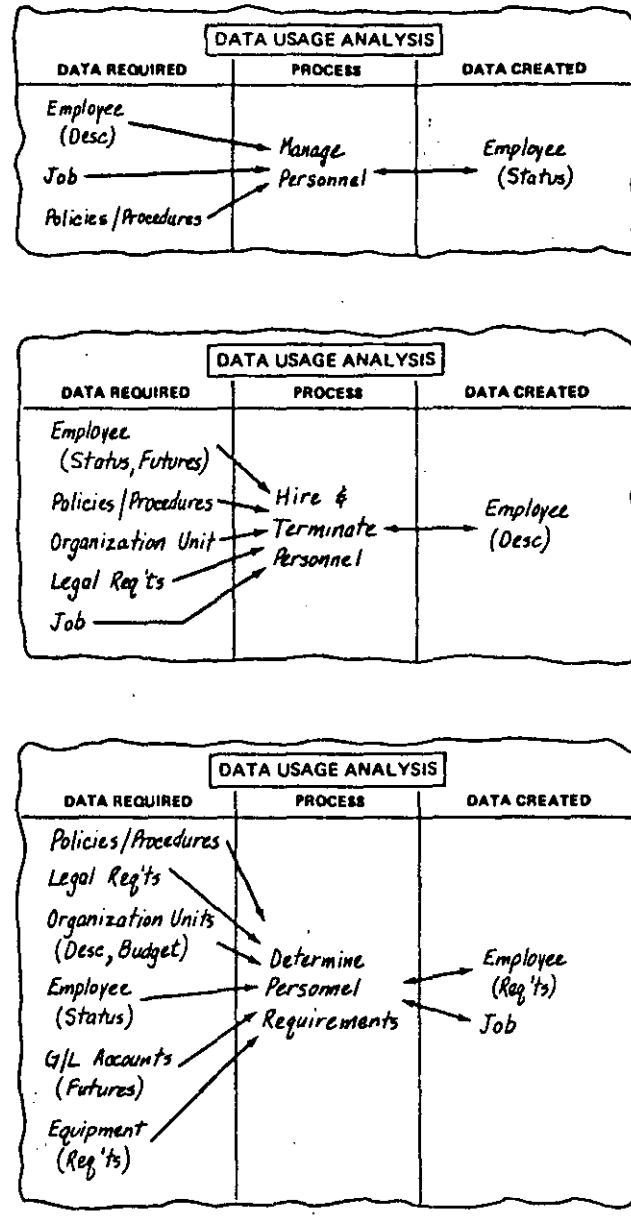


Figure 16. Sample data usage analysis sheets

employees, there should be three data classes associated with the employee entity: Employee Requirements, Employee Description, and Employee Status. In the same example, only one process creates any data about the entity "job;" therefore, only one data class needs to be associated with "job." Each business entity must have at least one data class associated with it. Conversely, each data class can be associated with only one entity.

Sometimes it will appear that two processes are creating the *same* data about an entity, making it impossible to

define two separate, distinct data classes. In such cases, one of the following strategies should be used:

- Recognize that the same data could be describing two different entities. For example, the equipment entity may really be two entities: production equipment and transportation equipment. Status data about each of these entities would most likely be created by different processes. If a new entity is discovered, it should be named, added to the list of entities, and defined. Appropriate data classes should then be created for it.

- Factor the processes. Separate that part of the different processes that creates the same data. In a BSP study for a petroleum company, three processes created purchase orders: engineering, production, and production equipment maintenance. To resolve this problem, the purchase-order creating part of each process was factored to form a new process called purchasing.

- Combine processes into one process. If the processes are performed in succession, it might make sense to combine them into a single process.

Associating responsibility for data integrity with a particular process is really the first step in establishing an organizational data policy. The analyses done at this point also serve to refine the definitions of both processes *and* entities, and will prove helpful later in establishing data dependencies.

Define Data Classes

As a final step, each data class should be defined in complete sentences (see Appendix E). The definition should include a description of the type of data included in the data class and some specific examples thereof.

Chapter 8. Defining Information Architecture

After data classes have been identified, the relationships between data classes and business processes must be established. This is done to ensure that:

- All needed data classes (and processes) have been identified.
- One and only one process creates each data class.

The tool used to establish process/data class relationships is the *information architecture* or process/data class matrix (see Figure 17). The steps for creating this matrix are:

1. List the processes down the vertical axis. Begin with the processes of strategic planning and management control; then list the processes associated with product/service in life-cycle sequence; and, finally, list the processes for managing the supporting resources.

2. List the data classes across the horizontal axis. Begin with the first process and list the data classes "created" by this process (put a "C" at the intersection of the appropriate process row and data class column). Continue until all data classes are listed. Where possible, group data classes by business entity, but ensure that the sequence of data creation prevails (this is important). For example, in Figure 17, notice that employee requirements and other employee data classes are not grouped because of creation sequence. Figure 18 reveals that these data classes are actually part of different data groupings, controlled by different information systems supporting different areas of the business (processes).

   Note: Make sure "C" (create) is defined correctly. It should mean that the identified process is the *best source* for ensuring the integrity of the data class.

3. Across the row for each process, place a "U" in the column for each data class used by that process.

4. Verify that all required data classes are present and that each data class has one and only one creating process.

When completed, the process/data class matrix becomes an important analytical tool for:

- Verifying data class identification
- Communicating data sharing concepts
- Analyzing data problems
- Determining dependencies between applications in the architecture.

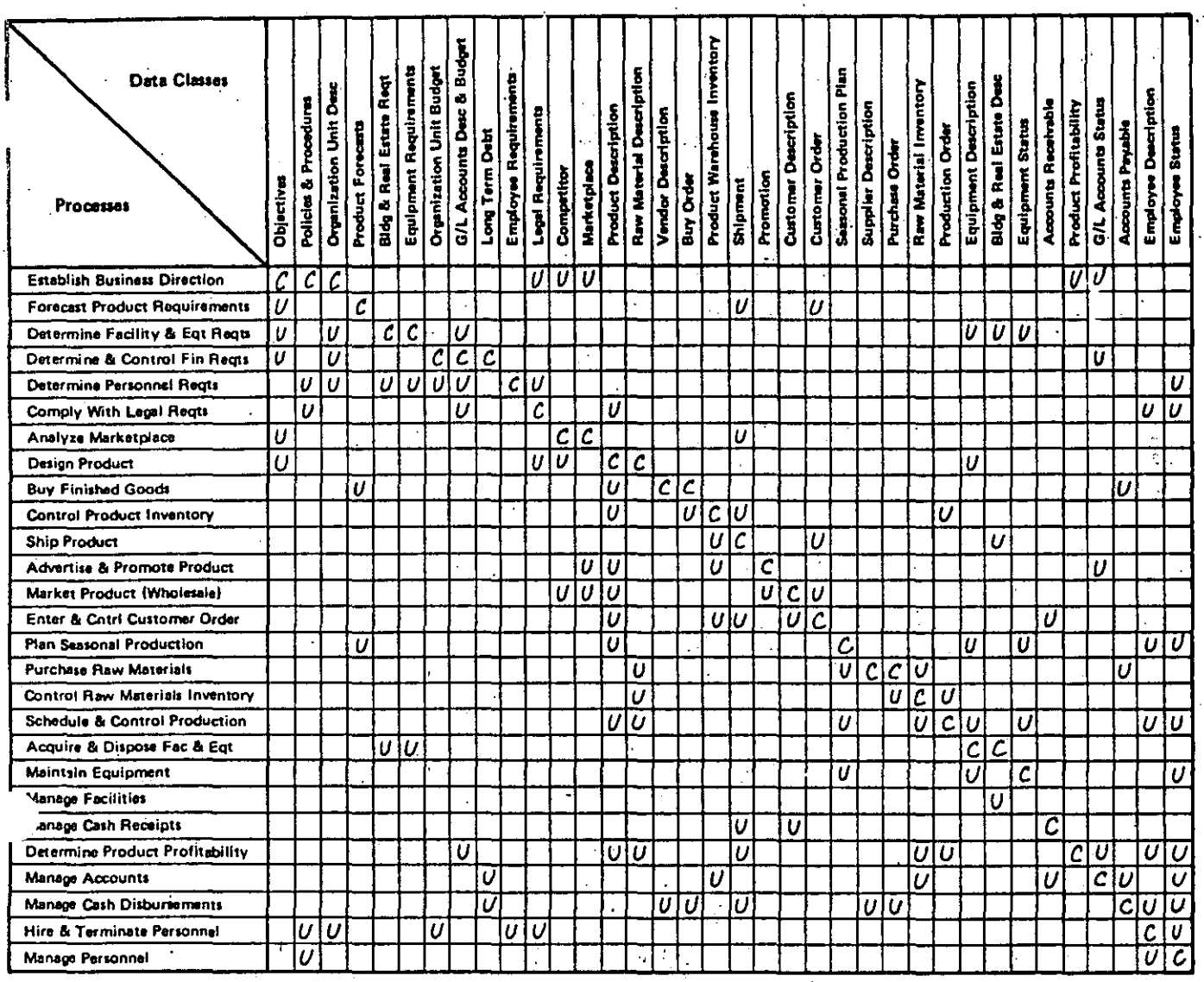
Some organizations find it useful to represent the information architecture as a flow diagram developed from the process/data class matrix. Such a diagram simplifies the overall picture of required information flow in the organization by grouping closely related processes and data classes and indicating how data flows between them. In this way, the diagram provides a vivid illustration of the need for sharing data across the entire organization. Figure 18 shows an information architecture flow diagram developed from the matrix in Figure 17. The rest of this section explains the intervening steps needed to develop this diagram.

Note: The grouping of processes and data classes may be a useful vehicle for subdividing the post-BSP work. The implementation team may choose to develop plans to implement applications and data bases for one group at a time.

Developing the Flow Diagram

An information architecture flow diagram can be developed by following these steps:

1. Rearrange the axes of the process/data class matrix (if necessary).
2. Determine process groups.
3. Determine data flow between process groups.
4. Simplify and complete the graphic.



Processes \ Data Classes	Objectives	Policies & Procedures	Organization Unit Desc	Product Forecasts	Bldg & Real Estate Reqt	Equipment Requirements	Organization Unit Budget	G/L Accounts Desc & Budget	Long Term Debt	Employee Requirements	Legal Requirements	Competitor	Marketplace	Product Description	Raw Material Description	Vendor Description	Buy Order	Product Warehouse Inventory	Shipment	Promotion	Customer Description	Customer Order	Seasonal Production Plan	Supplier Description	Purchase Order	Raw Material Inventory	Production Order	Equipment Description	Bldg & Real Estate Desc	Equipment Status	Accounts Receivable	Product Profitability	G/L Accounts Status	Accounts Payable	Employee Description	Employee Status
Establish Business Direction	C	C	C								U	U	U																		U	U			
Forecast Product Requirements	U			C															U			U													
Determine Facility & Eqt Reqts	U		U		C	C		U																			U	U	U						
Determine & Control Fin Reqts	U		U				C	C	C																							U			
Determine Personnel Reqts		U	U		U	U	U	U		C	U																								U
Comply With Legal Reqts		U						U			C			U																				U	U
Analyze Marketplace	U											C	C						U																
Design Product	U										U	U		C	C												U								
Buy Finished Goods				U										U		C	C																U		
Control Product Inventory														U			U	C	U							U									
Ship Product																		U	C		U							U							
Advertise & Promote Product													U	U				U		C												U			
Market Product (Wholesale)												U	U	U						U	C	U													
Enter & Cntrl Customer Order														U				U	U		U	C								U					
Plan Seasonal Production				U										U								C					U		U				U	U
Purchase Raw Materials															U							U	C	C	U							U			
Control Raw Materials Inventory															U									U	C	U									
Schedule & Control Production														U	U							U			U	C	U		U				U	U
Acquire & Dispose Fac & Eqt					U	U																					C	C						
Maintain Equipment																						U				U	U		C					U
Manage Facilities																												U						
Manage Cash Receipts																			U		U									C				
Determine Product Profitability								U						U	U				U							U	U				C	U		U	U
Manage Accounts									U									U								U				U	C	U			U
Manage Cash Disbursements									U							U	U		U				U	U								C	U	U
Hire & Terminate Personnel		U	U				U			U	U																							C	U
Manage Personnel		U																																U	C

Figure 17. Process/data class matrix

Rearrange the Axes of the Process/Data Class Matrix

If necessary, arrange the processes on the process axis such that those sharing a lot of data are next to each other. This is facilitated by listing the management control processes first, followed by the remaining processes, in life-cycle sequence for each product, service, and resource. In Figure 17 the first six processes listed are management control processes. The next eight relate to the product and are listed in life-cycle sequence: the requirement processes of "analyze marketplace" and "design product" are followed by the acquisition process of "buy finished goods," the stewardship process of "control product inventory," and the disposition processes of "ship product," "advertise and promote product," "market product," and "enter and control customer order." Following the product-related processes are those for raw materials, facilities, cash, and personnel, listed in life-cycle sequence for each resource.

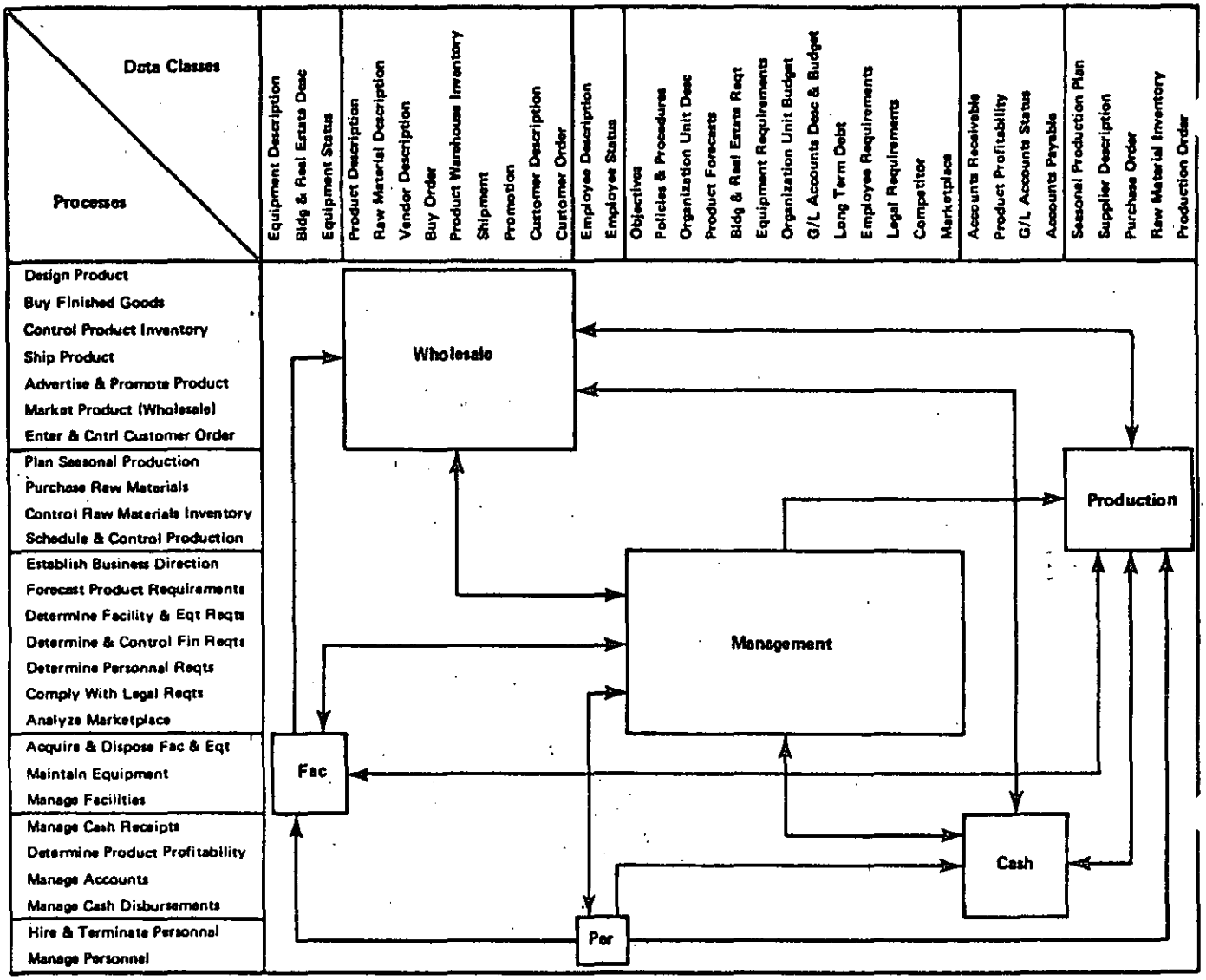


Figure 18. Information architecture flow diagram

Rearrange the data class axis so that the data classes closest to the process axis are those created by the first process listed, the next closest are created by the second process, and so forth. In Figure 17, the first data classes listed — "objectives," "policies and procedures," and "organization unit description" — are all created by the first process listed, i.e., "establish business direction." The next data class listed, "product forecasts," is created by the second process listed, "forecast product requirements." This arrangement of data classes will cause the C's in the matrix to appear along the diagonal.

Identify Process Groups

Identify groups of processes that have similar patterns of data usage. For each group, identify all the data classes created by the processes in that group. In Figure 19, the management control processes were grouped together because they were seen to have similar patterns of data usage. The "analyze marketplace" process was added to this group because it uses some of the same data as the management control processes and it creates two of the data classes used by these processes. The data classes created by this first group are easily identified because of the sequence in which the data classes are listed. The first 13 data classes are created by the processes in the first group and, as such, are associated with this group.

Determine Data Flow Between Process Groups

Identify the flow of data between process groups. Whenever there is data used by a process and that data

is created by a process in some other group, draw an arrow from the creating group to the using group. In Figure 20, the "shipment" data class is used by the "forecast product requirements" process, but it is created by the "ship product" process. This relationship requires that an arrow be drawn showing flow from the group to which "ship product" belongs to the group to which "forecast product requirements" belongs. All U's outside group boundaries represent the need for data flow. When all U's are examined and the necessary data flows represented, the result will be a flow diagram (Figure 21).

Processes \ Data Classes	Objectives	Policies & Procedures	Organization Unit Desc	Product Forecasts	Bldg & Real Estate Reqt	Equipment Requirements	Organization Unit Budget	G/L Accounts Desc & Budget	Long Term Debt	Employee Requirements	Legal Requirements	Competitor	Marketplace	Product Description	Raw Material Description	Vendor Description	Buy Order	Product Warehouse Inventory	Shipment	Promotion	Customer Description	Customer Order	Seasonal Production Plan	Supplier Description	Purchase Order	Raw Material Inventory	Production Order	Equipment Description	Bldg & Real Estate Desc	Equipment Status	Accounts Receivable	Product Profitability	G/L Accounts Status	Accounts Payable	Employee Description	Employee Status
Establish Business Direction	C	C	C								U	U	U																		U	U			
Forecast Product Requirements	U			C															U			U													
Determine Facility & Eqt Reqts	U		U		C	C		U																			U	U	U						
Determine & Control Fin Reqts	U		U				C	C	C																							U			
Determine Personnel Reqts		U	U		U	U	U	U		C	U																								U
Comply With Legal Reqts		U						U			C			U																				U	U
Analyze Marketplace	U											C	C						U																
Design Product	U										U	U		C	C												U								
Buy Finished Goods				U										U		C	C																U		
Control Product Inventory														U			U	C	U							U									
Ship Product																		U	C			U							U						
Advertise & Promote Product													U	U				U		C												U			
Market Product (Wholesale)												U	U	U						U	C	U													
Enter & Cntrl Customer Order														U				U	U		U	C								U					
Plan Seasonal Production				U										U									C				U		U				U	U
Purchase Raw Materials															U								U	C	C	U						U			
Control Raw Materials Inventory															U										U	C	U								
Schedule & Control Production														U	U								U			U	C	U		U				U	U
Acquire & Dispose Fac & Eqt					U	U																						C	C						
Maintain Equipment																							U					U		C					U
Manage Facilities																													U						
Manage Cash Receipts																			U		U										C				
Deter Product Profitability								U						U	U				U							U	U						C	U		U	U
Manage Accounts									U									U								U					U		C	U		U
Manage Cash Disbursements									U							U	U		U					U	U									C	U	U
Hire & Terminate Personnel		U	U					U			U	U																						C	U
Manage Personnel		U																																U	C

Figure 19. Process groupings

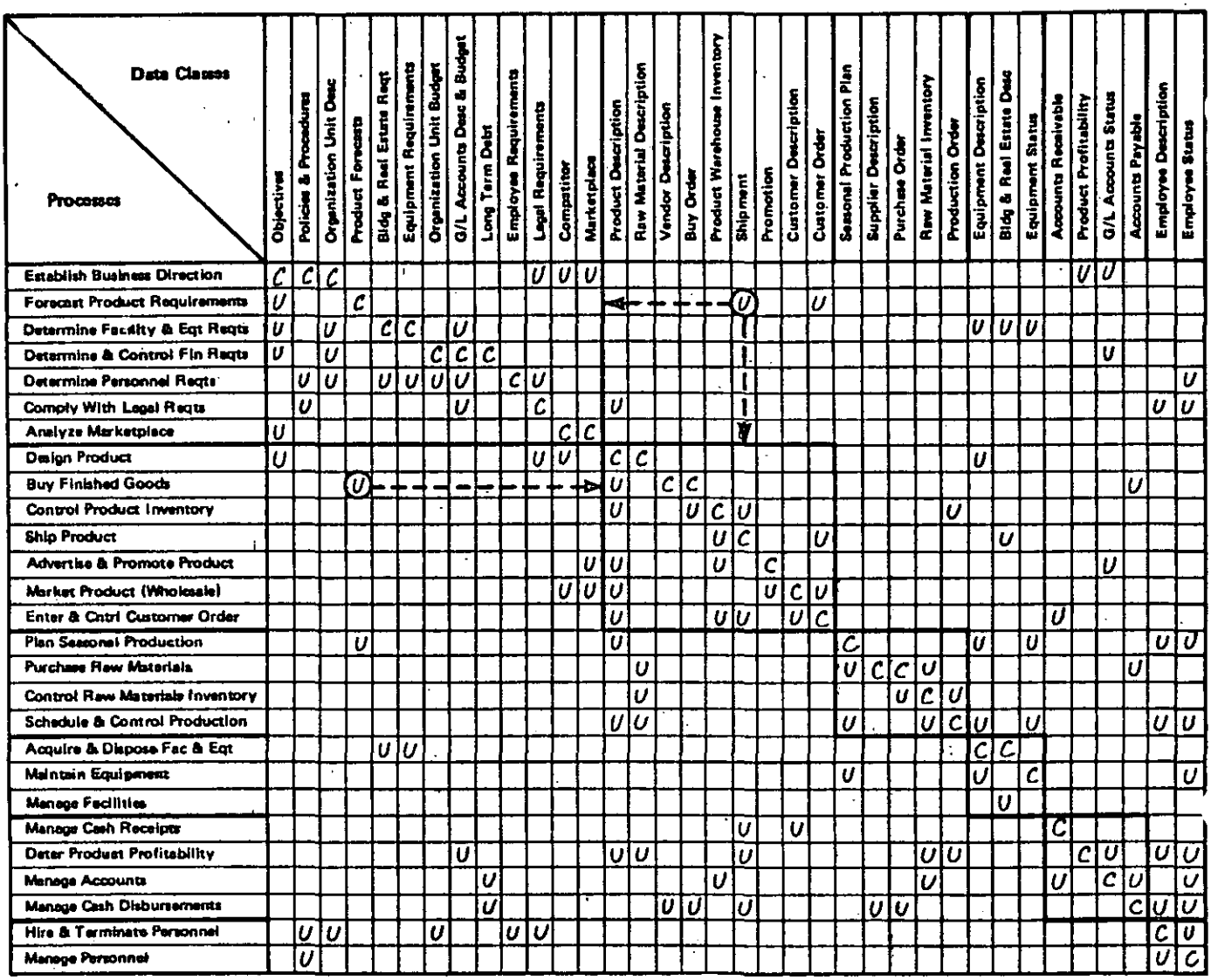


Figure 20. Data flow determination

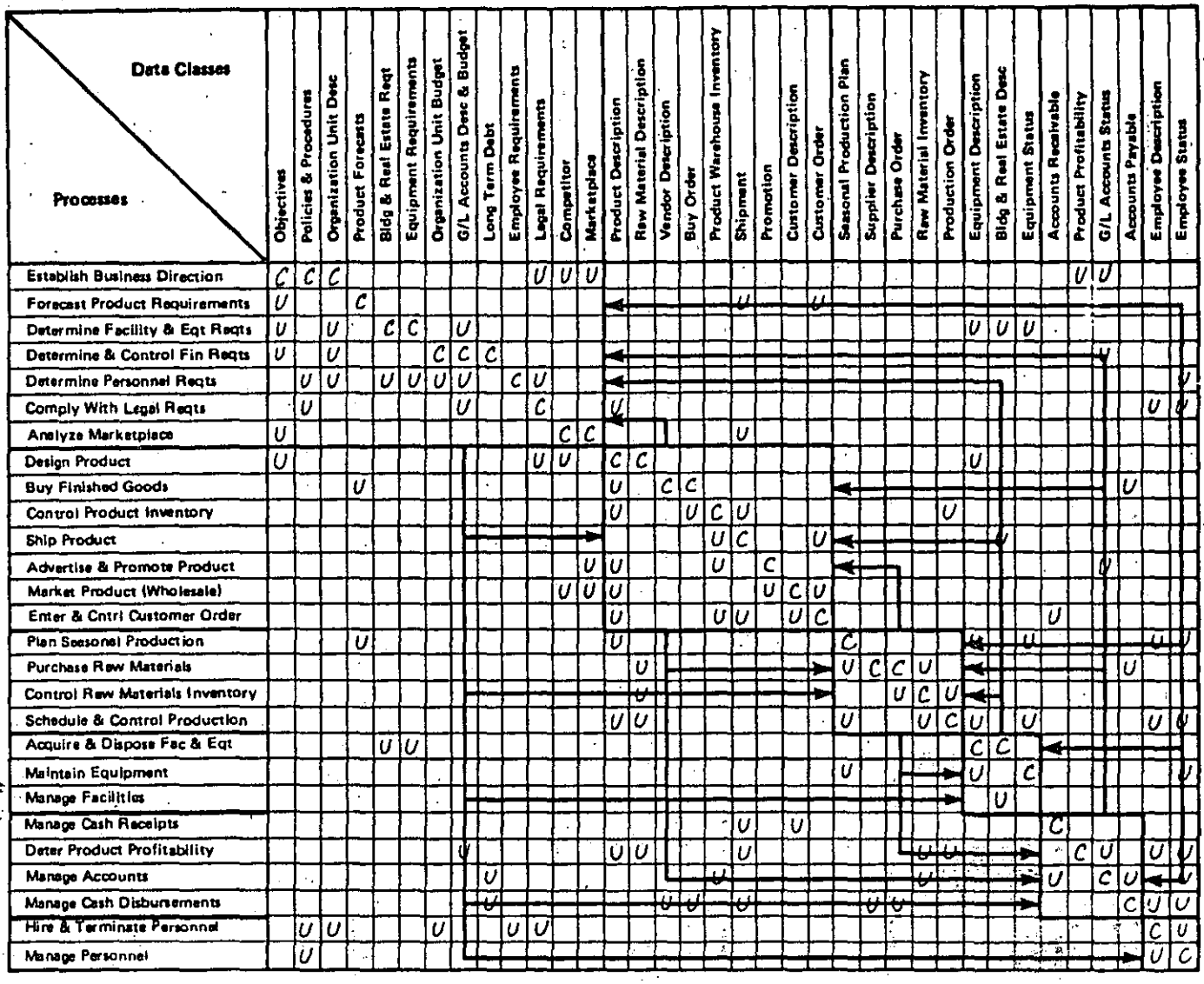


Figure 21. Completed data flow

Simplify the Graphic

(45)

For presentation purposes, it is useful to simplify the flow diagram. The most common ways to do this simplification are to:

- Remove the C's and U's.
- Use two-way arrows.
- Move the groups of processes and data classes to conform to the stylized information architecture shown in Figure 18.

Figure 18, which appears near the beginning of this chapter, is the result of applying these techniques to Figure 21.

The Completed Graphic

The completed architecture drawing is a very useful management communication tool because:

- It is the team's recommendation for long-range information systems implementations.
- It identifies the information systems (the blocks or boxes) that form the long-range plan.
- It shows the data *controlled* by each information system (reading vertically).
- It shows the business processes *supported* by each information system (reading horizontally).
- It shows the flow of information between the various information systems (the lines and arrows) and thus shows the flow of information through the business itself.

Chapter 9. Analyzing Current Systems Support

Up to now, the team has been developing a new perspective of the business – learning to look at the business in terms of business processes and the data classes necessary to perform them. Now the team must develop a firm understanding of how data processing currently supports the business, in order to develop recommendations for future action. This section describes the use of data gathered and presented as part of the I/S review as well as the organization, process, and data class information to develop perspectives on:

- I/S support of processes. The organization/process matrix developed in Chapter 6 is annotated to indicate which current systems support the business processes.

- Usage of current data. A data file versus data class matrix should be developed to identify the data available in existing applications. These data classes should then be related to the entities about which the enterprise needs to store data in order to manage effectively.

Although the matrices provide an overview of the current and planned data processing support of the business, they cannot indicate present problems, the extent of support needed, or the value of this support to each of the processes. Such information is obtained later during the executive interviews. These matrices, together with the output from the executive interviews, help the study team determine the architecture priorities.

Appendix F contains examples of matrices from other industries.

Review I/S Support of Processes

To obtain a business-wide picture of existing and planned data processing support, the study team should create a system/organization matrix (Figure 22) or a system/process matrix (Figure 23). These matrices identify which organizations involved in the processes are receiving application support. This enables the study team to identify:

- Processes receiving no current systems support
- Processes receiving systems support in some organizational units, but not all
- Potentially redundant systems

Identify the Use of Current Data

The team needs to understand what portion of the data classes is currently automated and which systems utilize which data. The next matrix developed by the study team, the present system/data class matrix (Figure 24), provides this understanding. The systems shown in Figure 22 form the vertical axis of this matrix, while the data classes, grouped by similarity, form the horizontal. An X is placed in each appropriate box to show which data classes support which systems.

This matrix sheds more light on how much data is shared by various systems. This, in turn, helps point out the need for a data base approach to provide consistency of data. The information gathered here will also be useful later in developing implementation priorities.

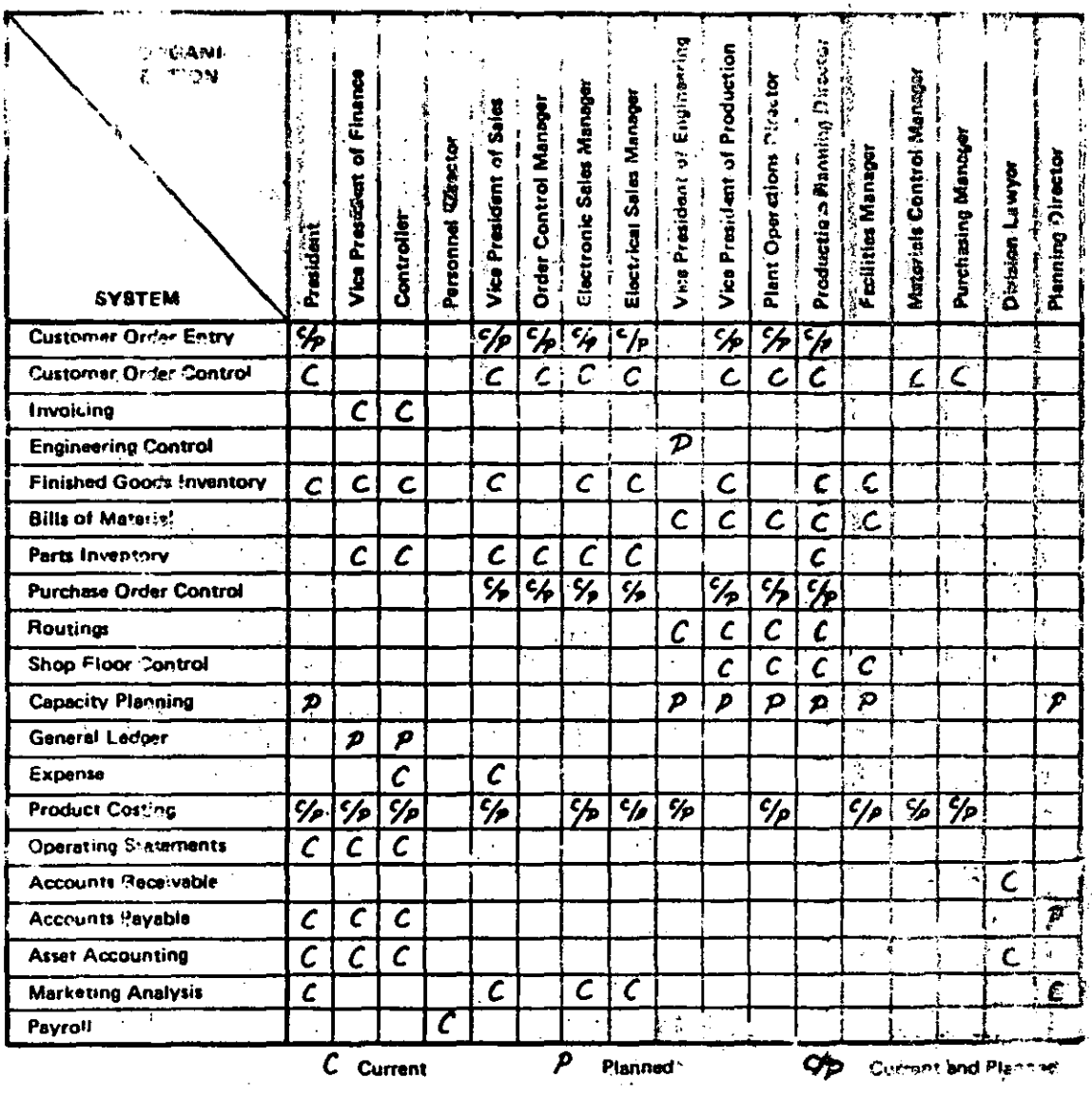


Figure 22. System/organization matrix

Process / System	Management				Marketing			Sales Operations				Engineering			Production				Materials Management				Facilities Management			Administration			Finance			Human Resources		
	Develop Business Plan	Establish Organization Criteria	Review and Control Finances	Manage Risk	Plan Market Support	Conduct Market Research	Forecast Product	Manage Territory	Establish Sales Objectives	Administer Sales Plan	Service Orders	Design and Develop Product	Maintain Product Specifications	Control Information	Schedule Production	Plan Capacity	Specify Material Requirements	Control Operations	Purchase Materials	Receive/Inspect Materials	Control Inventory	Ship Stock to Customer	Plan Facilities	Maintain Facilities	Measure Equipment Performance	Manage General Accounts	Plan Costs	Establish Budget	Plan Financial Performance	Acquire Capital	Manage Funds	Plan for Personnel	Recruit/Develop Personnel	Pay Personnel
Customer Order Entry	C/P							C/P			C/P				C/P	C/P	C/P																	
Customer Order Control											C			C	C		C		C															
Invoicing																										C			C					
Engineering Control														P																				
Finished Goods Inventory																			C	C	C					C	C							
Bills of Material													C	C	C				C								C							
Parts Inventory											C								C		C													
Purchase Order Control															C/P	C/P	C/P		C/P															
Routings														C		C		C																	
Shop Floor Control														C		C		C			C													
Capacity Planning																P		P																
General Ledger																										P					P			
Expense								C																		C								
Product Costing														C/P				C/P			C/P						C/P							
Operating Statements	C	C		C																						C								
Accounts Receivable				C																						C								
Accounts Payable	C			C																						C								
Asset Accounting				C																								C		C				
Marketing Analysis			C					C	C		C																							
Payroll																										C		C						C

C Current P Planned C/P Current and Planned

Figure 23. System/process matrix

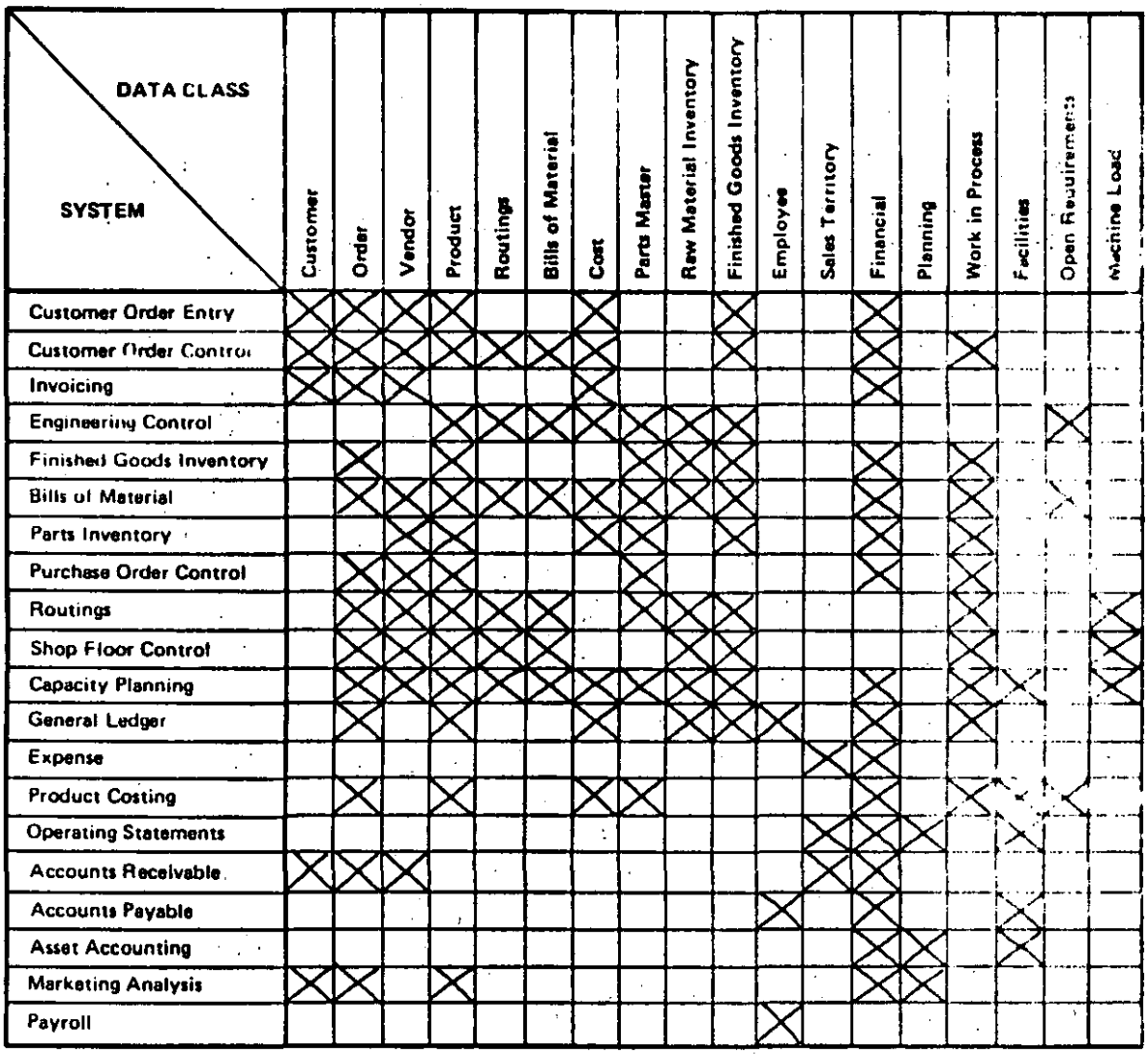


Figure 24. Present system/data class matrix

Chapter 10. Interviewing Executives

The top-down approach of BSP dictates that there be executive input. Executive perspective is gained by conducting two- to four-hour interviews with 20 to 30 executives from the top levels of management. The purpose of the interview is to:

- Validate the processes, data classes, organization, and their interrelationships.
- Clarify the future direction of the business and its impact on the information requirements.
- Identify and document the business problems so that they may be related to business processes and data classes.
- Quantify, where possible, the value of solving the business problems.

Each interview is conducted in the BSP control room where the charts and matrices are displayed for validation and discussion, and where interruptions are least likely to occur.

There are four major phases to the BSP interview process:

1. Make general preparations for interviewing.
2. Make specific preparations for each individual interview just prior to conducting it.
3. Conduct the interview.
4. Summarize each interview and analyze the results.

Each of these phases is discussed in the following sections. The tasks to be performed are listed at the start of each section and the task control sheets for all phases of interviewing are found in Appendix J.

For purposes of discussion, it is assumed the team is large enough (at least six people) to have two groups of interviewers. If the team is smaller, it may not be able to conduct two interviews per day and it will have less time that can be allotted to corollary activities.

With two interview groups, each group should have time to prepare, conduct, and summarize each interview. Additionally, the executive is more likely to be completely frank and at ease with only three or four interviewers. Because a split team introduces a communication problem, it is important that the entire team review the interview immediately upon its completion.

Make General Preparations

The executive interviews are the primary source of information for determining the business problems and management's need for I/S support in overcoming those problems and supporting new opportunities. Interviewing and interview analysis consume more time than any other activity in the study. Therefore, considerable preparation is warranted before starting the individual interviews. This section deals with that preparation.

The team should assume that the tasks outlined in Chapter 4 were performed during the study preparation; specifically, that the executives has been briefed, a list of topics to be discussed has been given to them, and executive interviews have been scheduled.

The importance of the information derived from interviewing and its relationship to the other categories of information that are developed during the study are represented in Figure 25. There are three distinct categories of information — that which is:

1. Gathered before the study
2. Developed during the pre-interview activities
3. Developed/validated/clarified during the interview

As can be seen in Figure 25, before the beginning of the interview, facts are gathered on the business operations, environmental impacts, objectives, critical success factors, planning methods, control methods, and business resources. Prior to the interviewee's entrance into the control room, the team has derived the major corporate-wide problems and opportunities mainly through an examination of the environmental impacts, objectives, and critical success factors; they have identified and described the business processes and data classes by an examination of the planning methods, control methods, and the business resources.

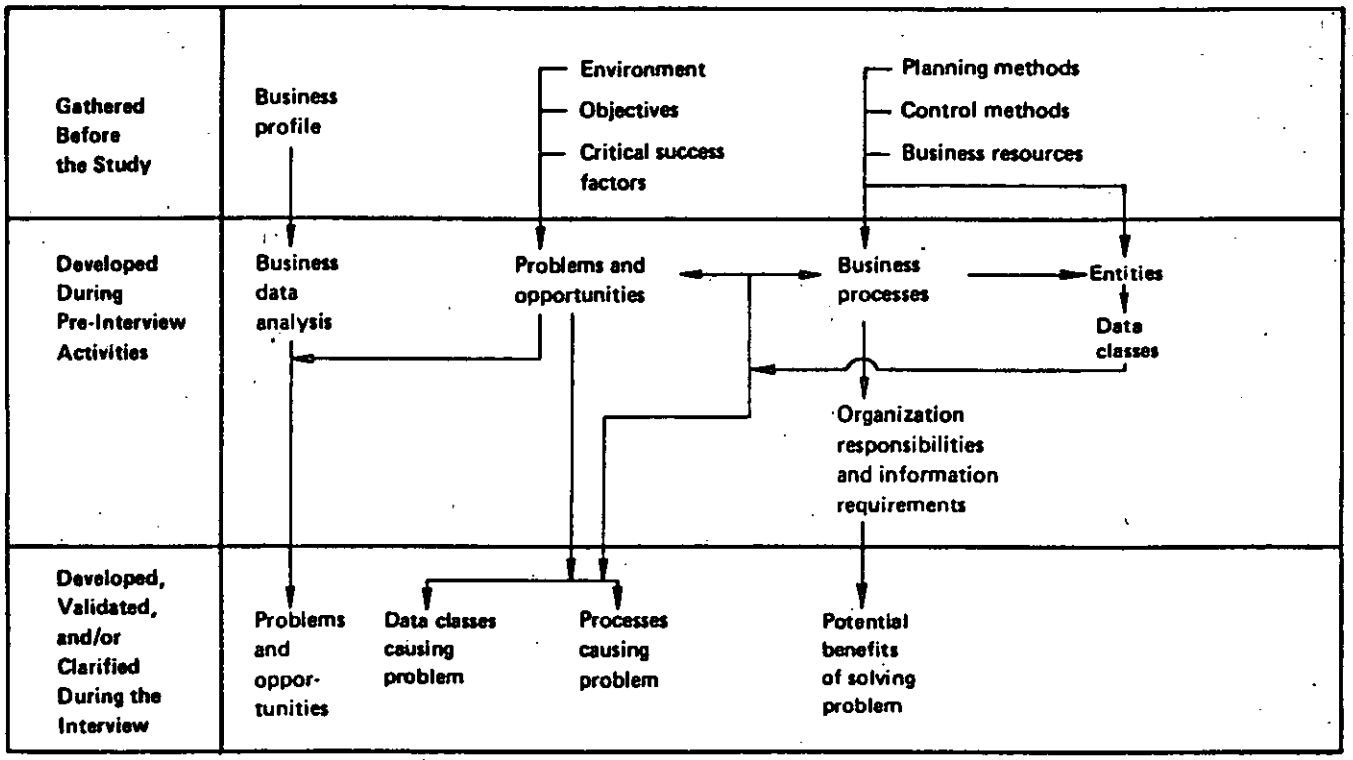


Figure 25. Flow of information in problem and opportunity analysis

During the interview, and through an analysis of the organizational responsibilities, the team is able to delineate the problems and opportunities and establish the problem and process/data class relationships. They then extend these to the output shown in Figure 25 as problems, data classes causing, processes causing, and potential benefits of solving.

The tasks to be performed in getting ready to interview are:

- Define expected output from interviewing.
- Confirm the interview schedule.
- Prepare the team leader's letter to interviewees.
- Establish note-taking procedures.
- Make charts for interviews.
- Set up the control room for interviewing.
- Prepare a set of questions.
- Establish administrative procedures.
- Conduct a practice interview.
- Plan corollary activities.

Each of these tasks is discussed in the following sections.

Establish Expected Output from Interviews

Using the previously defined objectives of the BSP study and the outline of the final report, the team members should list the output they expect to get from the interviewing of executives. For example, they might list:

- Each executive's critical success factors, objectives, and his environmental impacts.
- Problems and opportunities identified along with potential solutions and values.
- An understanding of the impacts of planned changes.
- More thoroughly defined business processes and data classes.
- Better understanding of each executive's information needs related to data classes.
- Identification of apparent inadequacies in present information resource management.

The team should make a list of these as a point of reference for future checkpoints during interviewing to determine if the required results are being obtained from the interviews.

It is important to note that later in the BSP process, the team will reduce the results of the interviews to a set unique information problems, relate those problems to

areas of the information architecture, and then sequence these problem areas via a prioritization process. The prioritization criteria should be well understood so that during the interview the team will pay attention to gathering data that will prove helpful during prioritization. See the discussion under "Problems" later in this chapter for some ideas regarding executive value judgment as a source of data for prioritization.

Confirm the Interview Schedule

During the BSP study preparation, the team identified the executives who should be interviewed. Now, aided by the organization/process matrix, the team can confirm its earlier list of interviewees and identify any other executives that should be included. Two questions will aid in this confirmation:

- What specific parts of the organization are involved in each of the processes?
- Within the organization, must other people be interviewed to adequately determine the problems, objectives, and information requirements?

Any additional interviewees should be selected judiciously to keep the number at a minimum. A decision for additional interviewees should be made by the executive sponsor upon the advice of the study team.

After the interviewee list has been confirmed and new names added, the interview schedule should be confirmed with each of the executives. The new candidates should be briefed prior to the interview.

Prepare the Team Leader's Letter to the Interviewees

The purpose of the team leader's letter is to confirm the time and place of the interview and to let the interviewee know what to expect. The letter should cover:

- Time, place, and expected length of the interview
- Objectives of the interview
- Subjects to be discussed

The sample letter in Appendix B assumes that the interviewee was at the executive orientation, that the interview was previously scheduled and confirmed, and that the subjects for discussion were covered in the executive orientation. The letter should be sent approximately five days before the interview to give the interviewee time for preparation and yet not so far ahead for it to lose its effectiveness.

Establish Note-Taking Procedures

Good note-taking is as important as good discussion-leading during an interview. After two or more interviews, it is difficult to remember all of the major points and the context in which they were made. In a team of three or four people conducting an interview, at least two of them should be responsible for taking notes to ensure adequate reconstruction of the issues during the analysis and documentation of the interview.

Make Charts for the Interviews

Charts in the interview room serve many purposes:

- Reinforce the purpose and approach of the study in order to prepare the interviewee to accept and support the final study recommendations.
- Act as a reference for either the interviewee or the team to use during discussion.
- Expose the interviewee to the findings and conclusions made to date.
- Act as an outline to lead the interviewee through all of the subjects necessary to get the information required.

Most of the charts have been discussed in previous sections of this manual. Many of them were developed for the start of the study, while others, such as the business processes, the business process/organization/systems matrix, and the process/data class matrix resulted from the pre-interview tasks. The following paragraphs discuss points that may not be obvious.

The business process charts should show the process groups and the processes under them so as to allow discussion and understanding of the term "business process." It should further give the interviewee a broad overview of all business processes before any detailed discussion of each business process is done using the process/organization matrix.

The process/organization matrix should be large, clearly printed, and have heavy (or colored) lines dividing process groups and organization segments so as to make it easy to follow. In addition to the lines, a piece of thin light-colored acetate to cover the line or lines being reviewed by the executive will reduce the confusion in trying to focus on the correct intersection on the matrix.

Set Up Room for Interviewing

After the charts are made and put in place, prepare the rest of the room. Make the necessary arrangements to ensure that there will be no interruptions during the interview. If the telephone cannot be disconnected or switched to prevent ringing, it should be removed. A sign should be placed on the door to indicate that an interview is in progress. In those cases where the interviewee may not know the first names of each of the interviewers, the team may wish to use name cards.

Prepare a General Set of Questions

Before the interviews are conducted, the study team prepares a list of questions that serve as an outline for the interview and help to ensure that all the necessary points are covered.

The following is a sample list of questions, with the reasons for asking them. If an interviewee's response does not provide the information desired, it may mean that the question should be worded differently.

Objectives

- What are your objectives?

This question is useful, not only for acquiring information about objectives, but because it sets the stage for follow-up questions that will link directly to the process/data class matrix. If the interviewee is prepared for this question and knows his or her objectives well, the team will derive a clear perspective of what is significant to him/her.

- What are your responsibilities?

This is a good question for interviewees who may not be properly prepared for the objectives question above. The answer here will help validate the process/organization matrix.

Either question should make the interviewee feel comfortable and thereby promote an openness of communication.

Because of the limited time available during the study, it is not likely that the study team will be able to do much analysis on the answers to this question. Also, it is difficult to get the interviewees to structure their answers to this question in a consistent fashion that lends itself to analysis. Therefore, relatively little time should be spent on this part of the interview — just enough time to "break the ice" and orient the interviewee for ensuing questions.

Measurements

- What measurements do you apply?

This question should uncover which data classes or combinations thereof are of significance to the interviewee. The measurement question usually derives from the objectives question. If you know the objectives, you can derive measurements. It doesn't matter whether the question is interpreted as "How do I measure?" or "How am I measured?". Both interpretations will focus on data that is significant to the interviewee.

Unfortunately, the measurement question can be a difficult one to answer, especially if the interviewee has not defined his or her objectives. And, even if the interviewee has a clearly defined set of objectives, he or she might not have a well-defined set of measurement criteria.

This is another area where it is difficult to analyze answers because of the inconsistent quality of those answers. Again, relatively little time should be spent here, although the question itself is a thought-provoking one that may yield some value at a later time.

Information Requirements

- What information do you need?

This is a direct, "point-blank" approach to focusing the interview on data classes. Although it may yield substantive responses in some cases, more than likely it will prove unfruitful because the information systems community has been asking management this same question for years with few satisfactory results. Management usually responds, "That's what I thought you (the study team) were going to tell me as a result of the study."

Once again, limited time should be devoted to this question.

Note: Although the three questions we've seen so far appear to have little real value for the study, they are, in fact, *very useful* for preparing the interviewee for the key questions that follow.

Problems

- What are your (business) problems?

This is the key question to ask and should dominate the interview insofar as the time allocated to it.

The problem question lends itself to further analysis because it can be easily structured and, therefore, documented consistently for each interviewee. Each interviewee, without exception, will have problems that he or she is willing to discuss. Hence, everyone will find the question easy to answer provided they do not feel threatened by the interview.

Furthermore, with some careful and diligent probing, the interviewer can determine whether the problem can be tied to the process/data class matrix by specifically identifying the processes and/or data classes that are causing or aggravating the problem.

A simple format that is useful for structuring the problem statement is:

Because: (*the cause*), the result is: (*the effect*)

If interviewers cannot spare the time needed to "fit" problems into this format during the interview, they should construct these problem statements later during interview documentation/analysis. The important thing at this point is to probe the problems and gain the understanding needed for later analysis.

The interviewer should determine if the root of the problem is some process ("Because we can't *do* something, the result is...") or some data class "Because we don't *know* something, the result is..."). If the team already knows the processes and data classes, they can insert actual process and/or data class names in the problem documentation, which makes it easy to link the problem directly to the process/data class matrix.

When problems are documented in this fashion for every interviewee, analysis of the total problem set gathered from all interviews is facilitated. One can group all problems having a common "cause" and see the overall, organization-wide impact of the causing process and/or data class. This is helpful in prioritizing the process/data class matrix — that is, in identifying the "zones" on the matrix that are significant to the business.

The "result" clauses provide value-judgment information for the analysis. If these value judgments could be quantified, they would provide an indication of the potential benefits to be realized by solving the problems. In a grouping of problems with common "causes," the organization-wide effect (or value) is the aggregate of all the associated "result" clauses.

High-level management usually has difficulty quantifying the results/value of the problems. In general, the higher the level of the issue, the more difficult it is to quantify, but the more significant the issue is. In contrast, the lower the level of the issue, the easier it is to quantify, but the less significant it is.

The interviewer should try to obtain some judgment of value associated with each problem for prioritization purposes. These values could be expressed in terms of:

- An absolute (e.g., $250,000)
- A range (e.g., between $200,000 and $400,000)
- A percentage (e.g., 30% of current costs)
- A benefit (e.g., industry position, share of market, etc.)
- A relation (e.g., small, medium, large, etc.)

In summary, the bulk of the interview should be devoted to problems because this area, if diligently pursued and methodically documented, will yield substantial benefit to the study team in relating the interviews to the structured description of the business (in terms of processes and data classes).

Changes

- What changes do you see in the future (that would impact the infrastructures of the business)?

Any change in the infrastructure (e.g., organization structure, product structure, distribution structure, geographic structure, control structure, etc.) has a dramatic impact on the information systems of the business. Insight into these kinds of change might reveal the value of expending resources to design and manage certain data classes, regardless of function. Or, they might reveal that systems could be designed for certain processes, regardless of organization.

The question on change, therefore, provides a useful supplement to the question on problems. The information obtained can be used during analysis to determine whether there is reason to expend the resource required to "design for change."

Critical Success Factors (CSF) — Optional

- What are the limited number of areas in which satisfactory results will ensure successful, competitive performance for the organization — the key areas where things must go right in order to successfully achieve objectives and goals?[1]

The key to this approach is to find the things that are truly "critical." As potential candidates for a Critical Success Factor are identified, it may be useful to apply a test to their criticality. A useful test may be the converse, "If this factor does not go right, will the organization fail?"

After the Critical Success Factors are identified, each one is examined creatively to develop measurements that describe successful (or unsuccessful) performance of the organization or its environment with regard to the CSFs.

A framework has been developed to help locate the organization's CSFs and give a reasonable degree of assurance that all of them have been identified. It specifies that there are some CSFs inherent in the industry in which the organizations under analysis participate. Some CSFs are peculiar to the organization itself as a result of the strategies it has selected. Some are temporal. There are some internal and some external. Some tend to be specific to the interviewee's responsibility, etc.

A great deal of literature is available on CSFs as an independent subject. It is not the intent of this publication to duplicate it, but merely to reference it as an alternative to some of the interview questions.

The utility of CSFs from a BSP standpoint is that they, like the problem analysis, focus the executive's perspective on the structural descriptions of the organization — the process/data class matrix. The CSFs themselves are likely to point to the key processes; the CSF measurements point to the data classes.

To be successful, the CSF approach must permit:

1. Easy recognition as a true CSF.
2. Reconcilability among different CSFs identified by different interviewees.

[1] *A Primer on Critical Success Factors.* Christine F. Bullen, John F. Rockart, June 1981, CISR No. 69, Center for Information Systems Research, Sloan School of Management, Massachusetts Institute of Technology.

It takes considerable interview time to properly identify CSFs and develop measurements for them (i.e., 4 to 8 hours). But, identifying and analyzing CSFs can be a fruitful experience for the interviewee, as well as the organization.

A useful interview strategy may be to employ the C. approach in the CEO interview and the problem approach in other interviews. (If this strategy is used, it would be advisable to "practice" on some other executive to gain experience with CSFs before interviewing the CEO.)

Additional Questions

If appropriate for a given study, additional questions may be posed by the interview team, including:

- What is the most useful information you receive?
- How would you rate your information support with respect to adequacy, validity, timeliness, consistency, cost, volume, etc.?
- What is your assessment of Data Processing? This study?

Caution: These additional questions must not detract from the interview team's efforts (or use up their time) to obtain the executive's perspectives on business problems and to focus on the process/data class m description or the organization.

Establish Administrative Procedures

Some attention to administrative procedures at this time can save considerable confusion during the study. Without proper administrative organization, copies of interviews may get mislayed and thereby breach security rules, extra copies of materials pile up, and materials are not available when they are needed. Among the administrative items that need attention are:

- Set up a notebook with dividers for each of the team members so that they can retain copies of material that they reference continually, such as copies of the charts that are in the control room, overall schedule and study work plan, task control sheets, interview schedules, business processes, data classes, summaries of interviews, and problem analysis sheets.
- Provide for immediate typing of the interview summaries and the problem analysis sheets.
- Maintain a file of all materials.

Conduct the Interview

All interviews should be conducted in the study control room; here, interruptions are least likely and the matrices and charts can be displayed on the walls for reference. If the team members have properly prepared for the interview, they should be able to use the charts sequentially and cover all of the necessary material. All charts should have been updated with the input from the previous interviews. The rooms should be cleared of miscellaneous material and name cards should be in place if they are being used.

After the interviewee is introduced, he should be seated so that he can see the charts that will be used during the interview. This should make the interviewee most comfortable and enhance rapport. Be sure that the phone is disconnected and that preparations have been made to prevent interruptions. The interview is one of the major sources of input to the BSP study; therefore, do whatever is necessary to put the executive at ease and to extract the maximum value from the interview.

The following tasks are performed in conducting an interview:

- Give the background and work done to date.
- Explain the purpose of the interview and how it is conducted.
- Validate the process/organization/system matrix.
- Cover the key questions and check the discussion points.
- Conclude the interview.

Give Background and Work Done to Date

Assume that the interviewee does not recall all of the points made at the BSP orientation. Review the BSP objectives and schedules, review the interview schedule and note the interviews completed to date, and quickly review the work done to date. The purpose of this is to bridge the gap between this and the last contact with this executive and to let him know who has been interviewed before him. An easy way to do this is to give him a quick review of all the charts.

Explain Purpose of Interview and How It Is Conducted

Review with the executive the purpose of the interview. The following will serve as a guide:

- Validate material developed to date.
- Understand his responsibilities/objectives, measurements, and information requirements.
- Identify his business problems.
- Gain an idea of company direction and the effect it will have on information requirements.

The interviewee should understand enough about how the interview is to be conducted so that he is at ease. He should realize that the charts form the structure for the interview and that a large amount of the time will be spent on understanding his problems and information needs. Explain the role of each of the individuals in the interview, the confidentiality of what he says, and whether or not the notes will be sent back for his review.

From this time until the end of the interview, the interviewee should be doing more than 80% of the talking!

Validate Process/Organization/System Matrix

Considerable use of this matrix will enhance the study by providing the opportunity to discuss responsibilities, information requirements, present data processing support, and the problems in relation to each of the processes that the interviewee is involved in.

Care must be taken to avoid using too much time in validating the matrix. A significant portion of interview time must be reserved for identifying and analyzing the interviewee's problems.

Cover Key Questions and Check Discussion Points

Ask the questions as suggested earlier in this chapter and ensure that all issues identified during interview preparation are discussed. This is the primary focus of the interview. The question on problems, in particular, is designed to provide all the data needed to draw conclusions and make recommendations regarding post-study activities.

Conclude the Interview

Be sure that the interviewee has had plenty of time to cover all points he wished to discuss. The interviewer should ask each team member for his last question and then conclude the interview by thanking the executive and explaining to him what will follow.

Document the Interview

After concluding the interview, it is important that the interview team document it as soon as possible, while it is still fresh in their minds. The documentation of the interview is as important as the interview itself. Without good, well-structured documentation, it will be impossible to draw supportable conclusions and make recommendations.

The documentation must be well-structured in order to facilitate analysis and integration of all data collected during all the interviews. Unstructured, narrative documentation is difficult to use in further analysis.

The set of questions suggested earlier provides an outline for the documentation, as follows:

- Objectives
- Measurements
- Information Requirements
- Problems
- Changes

Remember that the primary focus should be on the problems that are likely to yield the most substantial data for further analysis within the study timeframe.

The problem statements should be structured in their cause/effect formats, with care being taken to identify the "root" of the problem in terms of the data class or process that causes it. It is likely that some discussion will be required by the interview team to arrive at a common perception of the cause and effect. It may require almost as much time to document the problems meticulously as it took for the interview itself. However, the quality of this problem documentation will determine how easily and how well the problems can be analyzed and recommendations suggested to affect a solution.

The cause/effect format is also very useful for obtaining validation from the interviewee and for communicating the "net" results of the interview to study team members who were not present during the interview.

A useful technique for summarizing problems is to use the format suggested by the Problem Analysis Sheet shown in Figure 26.

Update Control Room Charts

Be sure that all the charts in the control room are updated as required before the next interview. One person should be responsible for updating both the charts and the related material, such as process descriptions, data class descriptions, and the process/data class matrix.

Int. No.	Problem Cause	Problem Result	Value	Causing Process	Causing Data Class	Suggested Solution
12	Because we can't analyze adequate alternatives in business planning . . .	. . .we can't identify optimum capital investment alternatives.	"Medium"; over $1 million	Plan capital investments	–	Financial modeling
12	Because we don't know the cost of our products . . .	. . .we can't measure profit contribution by product and eliminate low-profit producers.	"Huge"; 10% profit increase; over $10 million	–	Product data (cost)	Cost accounting system. Acquire/manage labor/material cost data related to product.
12	Because we can't effectively process the high volume of orders we receive . . .	. . .we are not meeting our delivery commitments, losing customers, and paying high clerical labor costs in marketing.	"Medium"; 50% of clerical effort in marketing	Process orders	–	Automat- the order processing/tracking system.
12	Because we don't know the skills/experience of existing employees/ . .	. . .we have to hire from the outside and don't always get the best-qualified candidate.	"Small"; clerical cost of hiring under $100,000	–	Personnel (skills and experience)	Build skills and experience inventory for each employee and provide access to all responsible managers.
12	Because we don't maintain consistent parts inventory across all warehouses (or within any one warehouse, for that matter). . .	. . .we are experiencing production line stoppages almost weekly.	"Huge"; over $10 million	Control warehouse inventory	Parts (inventory)	Redesign the inventory control system. Establish data administration to control inventory data Design parts data base to support inventory control as well as purchasing.

Figure 26. Problem analysis sheet sample

Chapter 11. Defining Findings and Conclusions

The fact gathering is complete — from the early data gathering before the start of the study through the last interview. Now it is time to arrange the facts, analyze them, and draw conclusions. The purpose of the findings and conclusions in the report is to:

- Confirm to management that the points they made in the interview were understood, accepted, and are a part of the total business analysis.
- Provide the basis for recommendations and an action plan.
- Help set architectural priorities.
- Provide input for the description of the subsystems of the information architecture.

Methodology outlined in this chapter is based upon the following assumptions:

- Each interview was summarized to a predefined structure and all summaries are available for analysis.
- Problem analysis sheets were completed after each interview.
- Business process and data class descriptions are complete.

Each interview provided two results: (1) the interview was summarized and approved by the interviewee, and (2) the major problems were extracted and posted to the problem analysis sheets. Each of these items is used in defining the findings and conclusions. The problem analysis sheets form the main structure, with support coming from the interview summaries. Additional input might come from the material gathered before the study, particularly the wall charts used in interviewing. Defining the findings and conclusions involves the following steps:

- Review assumptions for completeness.
- Determine findings and conclusions categories.
- Sort problems by category.
- Write findings and conclusions statements.
- Sort problems for architecture priorities.

Each of these items is discussed in the following paragraphs.

Review Assumptions for Completeness

Ensure that all interview summaries are accounted for. Although work can be started on the findings and conclusions before all of the interview summaries are returned from the interviewees, the last ones should be expedited so as to not slow up this process. Check the problem analysis sheets for each interview to be sure that they are accounted for and complete.

By this time in the study the business processes and data classes should have received enough attention so that no further changes need be made. Therefore, update all descriptions and have them typed and ready for reference in the development of findings and conclusions.

Determine Findings and Conclusions Categories

Productivity of the team is increased and confusion and frustration are reduced if there is consistency in the various tasks of the BSP study where categorizing is required. Hence, findings and conclusions should follow the pattern set in fact gathering, interviewing, and interview summarization. The following are additional questions the study team might use to develop conclusions.

Objectives

- Do well-defined objectives exist for the total business and for each major function?
- Do corporate and divisional objectives provide direction for information systems planning?
- What correlation exists between each of the objectives and the current information systems?
- Is information available to measure attainment of objectives?

Organization

- Are management philosophies well understood?
- Are responsibilities and accountabilities well defined?

- Are planned and evolving changes in organization well defined and their impact on information systems understood?
- Will functional area responsibilities for new information systems be easy to define?
- Are organizational guidelines for information resource management consistent with those for the business?

Planning

- What is the degree of formalized planning?
- What is the relationship of long-range, short-range, and operational planning?
- Are the business plans adequate as a base for information systems planning.
- Are dependencies upon information systems explained in the functional plans?
- Is the computer used in planning?
- Will the planning process lend itself to automation?
- What type and amount of "what if" planning is done? Are computer models used?

Measurement and Control

- How effective are present measurements for controlling the business?
- What other measurements would be made if the data was available?
- What measurements support the control of the critical success factors?
- What additional data is needed for adequate control of the critical business areas?
- What measurements are made to determine the fulfillment of objectives?
- How are budgets used for measurement and control?

Operations

- What major difficulties are encountered in performing the operational processes (sales, production, distribution, branch office operations)?
- What problems were found relative to such items as low productivity, loss of revenue, excessive time and cost, and schedules not met.

Current Information Systems Support

- What are the major information requirements not currently being met (and not covered in the other categories)?
- What is the general state of the data that exists and the information received by the users (that is, accuracy, timeliness, format, accessibility, redundancy, excessive manual processing)?
- What is the condition of current applications (statements not covered in the other findings and categories)?
- What are the environmental impacts (management systems impact, restrictions and limitations, image of DP among users, impact on the business direction)?
- What is the present architecture (shared data, common systems, centralized systems, distributed systems, hardware/software, organizational independence of systems, vertical and horizontal integration of systems, and support of all levels of management)?
- What is the current systems design (data base, flexible and expandable, generalized systems for data retrieval and analysis)?
- How effective are operations (interface with users, support of company goals and objectives, turnaround, response, and currency)?
- What information systems planning now exists (degree of formalized planning, integration with the business plan)?

Sort Problems by Category

Identify each problem by its category and post the category to each problem on the analysis sheet if this has not already been done during the interview summary process. The team will find some problems that tend to fall into more than one category. If so, assign the most likely category and note the other category next to it. The problem might then be used to support more than one finding and conclusion.

The sorting of problems can be most easily done by cutting the problem analysis sheets into "problem strips." Annotate each strip with the interview number to provide an audit trail, then sort the strips into problem categories.

Write Findings and Conclusions Statements

Organize the problems into logical groups under each of the categories. Divide the interview summaries in a like manner. If the interview was summarized by the categories of findings and conclusions, this is an easy task. If not, then separate each summary as well as possible and use it for reference when writing the detailed discussion of each conclusion.

Write a general conclusion for each logical problem group and use the specific problems within that group, along with the interview summaries, to form a detailed discussion of the conclusion. Conclusions should be written so that they lead easily to recommendations. Be careful that the recommendation does not become a part of the conclusion.

Not every problem will lead to a logical conclusion; some problems may have been mentioned in only one interview and could not be substantiated in subsequent interviews. These may be discarded or grouped under a general cover statement at the end of the findings and conclusions.

The problems dealing with information systems support may be further subdivided into four groupings:

- Specific information requirements
- Present program inadequacies
- Data processing service
- Information resource management

This logical grouping will feed directly into the recommendations. Information requirements can be covered by a general findings and conclusion statement, but it is more important to ensure that these information requirements will be fulfilled by the new information architecture.

The present program inadequacies should be covered by conclusions that will set the stage for recommendations for program changes that should be effected before the implementation of the corresponding portion of the information architecture.

Service covers the set of problems resulting from the operating interface between the user and the data processing department because of such difficulties as late reports, inability to get a special job run, and inaccessibility of the system. The whole area of invalid data will be covered in this area.

The last category, information resource management, covers items such as organization, training, management systems, and administration that would be apparent to the team and the executives interviewed. The conclusions covering information resource management will set the stage for recommendations on steering committees, user interface groups, data administration, education, and project management. These are discussed in Chapter 13, "Information Resource Management."

These findings and conclusions should provide a framework that will ensure executive management involvement in the planning and measurement of information processing.

Sort Problems for Architecture Priorities

The same set of problems that formed the basis for findings and conclusions can be used now to determine architectural priorities. Sort the problems by the "process causing." All problems can be rectified by addressing the processes that are causing them. Moreover, sorting by "process causing" provides a direct relationship between the problems and the subsystems of the information architecture since the subsystem was established to support given processes. Hence, if we know the value of solving problems as designated on the problem analysis sheets, and we know what problem solutions are assisted by what subsystems, we can do a better job of setting architectural priorities.

Chapter 12. Determining Architecture Priorities

To begin implementation as early as practicable, the team should select and recommend to management the portion of the information architecture to be implemented first. This will establish a "pay-as-you-go" foundation for the information system.

Selection of the first application(s) should be based primarily on a value analysis following the interviews. To facilitate this analysis, use the findings and conclusions and the problem analysis sheets. Sorting the problem analysis sheets by "Problem Causing" (from Chapter 11) indicates the process that must be supported by the application to help solve the problem. If the problems are sorted by "Process Affected", the application can be related to all the problems they help solve and hence to the benefits derived. Each application is then evaluated against a set of criteria to determine its worth to the business.

The tasks to be performed in setting architectural priorities are:

- Determine selection criteria
- Apply criteria and list applications
- Document or describe the recommended applications
- Consider implementation options

Determine Selection Criteria

Since data is now being treated as a business resource, management should be able to evaluate information systems projects in the same way that other business resource projects are evaluated. Therefore, it is recommended that the team use whatever justification technique already exists within the organization for evaluating new projects. This will ease decision making for executive management as it considers the necessary trade-offs between, say, developing a new product, expanding a facility, making an acquisition, or implementing a major information system.

Some of the questions to be answered in determining application priorities are:

- Will the application provide a significant near-term saving and a substantial long-term return on investment?
- Whom will it impact, and how many people will be involved?
- Will it lay the groundwork for an initial data base architecture?

A method of determining logical priorities is to group the major criteria into four categories: potential benefits, impact upon the business, probability of success, and demand.

Potential Benefits

The team must determine the relative value of each of the potential applications in order to establish a set of priorities. Doing so entails making judgments in the areas of:

- Near-term
- Long-term
- Competitive advantage

Generally, the level of detail in a BSP study is not sufficient to do precise ROI (return on investment) calculations for each application. Therefore, the team must rely upon the major benefits cited in the interviews. For example, the sales executive might indicate that sales could be increased by 2% if certain information were more readily available. The team can translate this 2% into a precise dollar benefit figure for use in setting priorities.

In some cases, specific, tangible, benefit-to-cost ratios can be quantified and applied to an ROI calculation for each application. This adds significantly to the selection process. The team should also consider intangible benefits which, of course, must be estimated. These estimates should be based upon the interviews that have been conducted with the executives.

Impact upon the Business

Several items should be considered here including:

- Business economic trends
- Critical success factors
- Number of problems addressed
- Major problems solved

The team needs to describe and quantify how each proposed application will change the existing situation or problem within the business for the better. The team should determine whether the application will have an

affect on the quality of goods or services. Other considerations are whether there will be visible positive results, how many departments and how many employees will be affected by the systems changes, and in what ways they will be affected.

Probability of Success

The team should consider:

- Political climate
- Technical and organizational complexity
- Prerequisites
- Length of implementation
- Risks
- Resources available

This wide range of considerations is necessary because the chance of successfully implementing a particular application is important in establishing priorities. The team must consider how many and what type of resources would be necessary to implement a particular system.

- Are there hidden factors that need to be considered?
- How long would it take to design and implement the applications
- Is there a dependence upon unavailable or costly technology?
- How receptive to changes will the individual users be?

Some estimate of the degree of risk should be determined. The greater the risk appears to be, the greater the potential return that must be shown for the individual applications being considered. These factors must be considered not only to establish the priorities, but also to ensure that there will be a complete understanding of what the applications will and will not provide.

Note: An analysis of prerequisite applications should be performed after a list of high-priority applications has been prepared. Determination must be made as to which applications/systems must be in place before others can be created. By using the information architecture previously developed, and the team's understanding of the business, these interdependencies can be analyzed. This may mean, for example, that an application that provides data must be built prior to a high-priority application.

Demand

This area includes consideration of:

- Value of existing applications
- Relationship with other applications
- Political overtones
- Number of users
- Regulatory requirements

The decision makers must recognize a demand from within the organization before they can consider a particular data processing investment. They should consider the extent to which a proposed application specifically supports corporate goals and objectives.

There should be a correlation between the application being considered and the high priority needs identified in the executive interviews.

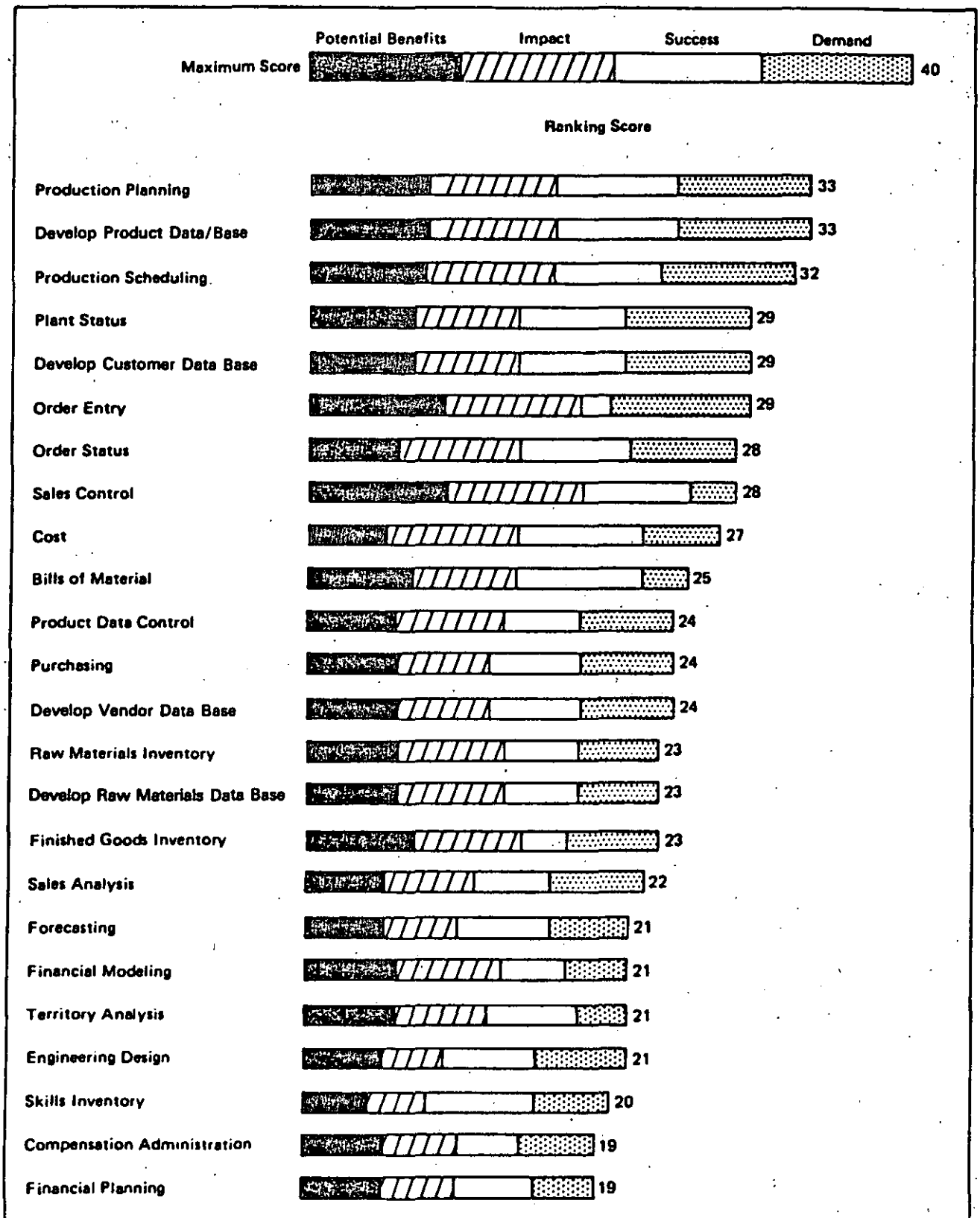
Apply Criteria and List Applications

The applications constituting the architecture can then be analyzed and ranked on a scale of 1 to 10 for each of the four categories above to help determine the implementation sequence. A pictorial representation can then be drawn to emphasize the most needed applications. Figure 27 is an example of this pictorial representation.

After the implementation of the first application has been completed, the priorities for the remaining applications should be reassessed. For example, after the first four have been implemented, the fifth may no longer be at the top of the list because the requirements and problems of the business have changed.

Document Recommended Application

Finally, the recommended first application should be documented in sufficient detail that the executives can properly evaluate it. This documentation should be based on the business process and data class descriptions, the problem analysis sheets, the process/organization/system matrix, and the findings and conclusions.



Note: Data base development is carried out in parallel with application development. Data bases scheduled in conjunction with an application are also designed to be part of the corporate data base.

Figure 27. Sample application ranking

The description should include an overview of functions, major objectives and processes supported, problems solved, and anticipated benefits. It should identify any new technologies and/or special skills that need to be acquired or developed to implement or operate the new applications. A general description of perceived benefits anticipated should also be included. The recommendation, documentation, and presentation should be in a format consistent with standard company practice.

Generally, the recommended first application(s) should be described broadly enough so that a business case evaluation may be made. For each application provide:

- Description
- Objectives
- Major problems
- Potential benefits
- Business processes affected
- Input
- Output
- Organizational levels affected
- Prerequisites

Description and Objectives

The description and objectives should provide a general framework of the application. For example, in the description it would be appropriate to include a list of the individual programs that this application would contain. Objectives might include the level of service expected or the parameters within which the application must operate. This section should also contain a general description of the basic activities or functions, such as what records are maintained, specific calculations, and what controls are necessary.

Major Problems

The documentation of major problems would draw heavily on the problem analysis sheets developed from the executive interviews. Here, there would be specific reference to problems cited by the executives which this particular application would be expected to address and solve. However, solutions should not be limited to the problems cited in the executive interviews. For instance, a recommended application may solve problems that were identified by the team while documenting the business environment. Also, the findings and conclusions will serve as an excellent recap of both the interview and the knowledge of the team.

Potential Benefits

This area also relies heavily on the executive interviews. Here the team should document as specifically as possible the benefits cited by the executives. Benefits should be documented in as much detail as possible. Improvements to revenue or specific cost reductions should be cited. Both near- and long-term benefits should be listed, such as expectations for reduced production down-time, or better supply costs, or increased management reporting, or improved quality.

Business Processes Affected

This may be identified from the information architecture. The team should describe the specific processes being supported by this application and the relationships to other business processes and other applications.

Input

A description or list of the data classes used by this application should be included as part of the application description. This is obtained from the data class input to the processes supported by this application.

Output

The team should document specifically what is expected as output of this application. This should include data or information created, as well as report output. Refer to the processes supported by this application and the data created.

Organizational Levels Affected

Using the process/organization matrix, the team should document the organizational units that are specifically affected or supported by this application. The responsible executive of the supported organization should sponsor the implementation project for this application.

Prerequisites

Once a first application has been identified, any steps or projects necessary for its successful implementation should be sequenced and listed. This is necessary to develop a meaningful action plan. Prerequisite projects should be clearly described and their relationship to the selected first application defined.

Implementation Options

Chapter 13, "Information Resource Management," covers the aspect of the BSP study that relates to analyzing current I/S management activities, including

implementation considerations. The team should also consider the develop-or-buy question and, if possible, make specific recommendations.

Decision to Buy

Obviously, the quickest way to implement a given application would be to purchase existing code that might require little or no modification for this particular application. If code does exist, the team should make a preliminary review and determine if this is a viable option.

Decision to Develop

If the application is to be developed from scratch, there are several considerations that might reduce the resources required for implementation. The assistance available from an information center or from Application Transfer Teams (ATTs) should be considered.

Information Center

An information center provides a great deal of application development support while keeping all of the development effort in-house. Under the information center concept, the data processing department makes available a group of individuals who provide packages and services that allow user departments direct access to the computer. In this fashion, features such as query, report generation, and personal computing can be provided directly to the user department without tying up large amounts of data processing application development resources. It would also reduce the lead times to provide certain specialized data processing support.

Application Transfer Teams (ATTs)

An ATT study is performed jointly by IBM and the customer to determine whether an existing application can be adapted economically for use by that customer. The result of an ATT study is a well-documented, cost-justified recommendation for implementing a specific application solution which is:

- Designed primarily by end users with the support of the data processing staff
- Cost-justified and supported by financial management
- Presented to user executives by their own personnel

Expansions and Variations in Approach

Another method that may be used for priority selection is to focus primarily on financial justification. Using this technique, the team focuses on tangible benefits as provided in the value statements during the executive interviews. This technique, called potential-benefit analysis, is described in Appendix G.

Chapter 13. Information Resource Management

The growth and profitability of an enterprise has become increasingly dependent on effective data processing. This is so because information is a *basic* resource of the business. Data, the raw material of this resource, can and should be controlled as an asset. But problems in data processing are also increasing, both in number and in severity. In many instances, money is being spent on applications that prove to be either unprofitable or of minimal real value. This could happen with the BSP-developed plan if we do not apply the same general management principles that have been successfully applied to the other basic resources of the business.

Therefore, in addition to determining an information architecture and setting application priorities, the BSP study team must ensure that the information resource is managed properly to support the functional needs of the business — i.e., that:

- The information architecture is implemented in an orderly fashion.
- I/S is consistently effective is servicing the information and data needs of the enterprise.
- Provision exists for an overseeing function to assure the responsiveness of information processing to the enterprise.
- A viable information resource plan exists — one that integrates business needs, personnel, hardware, software, communications, and office automation within the scope and financial resources of the enterprise. A basic premise of information resource management is the ability to make information available to whomever needs it when and where it is needed. Therefore, the information resource management environment must include a structure with the function of managing data/information.

In effect, the BSP study team must evaluate the information resource environment and recommend changes deemed necessary for the effective management of that resource. In particular, the team should address the following subject:

- Information resource management mission
- Steering committee
- Information resource organization

Information Resource Management Mission

This subject covers the many facets of the information resource and the concept that a single function must be responsible for office automation, communications, and data processing. Since these "technologies" are interrelated, the concept of a single integrated plan and implementation schedule is viable and necessary for their maximum effectiveness. Although the integration of these technologies is a long-range plan, the BSP study can point the way and define the mission.

The team must also consider the visibility of the information management function within the organization. It is possible that this function has been "buried" in the financial or administrative service area and that it more appropriately deserves its own area, consistent with its resource management charter. In addition, consideration should be given to the level at which responsibilities are focused so that comprehensive systems plans closely tied to both corporate and unit business plans will be continuously generated. In a company with multiple business units or divisions, this is particularly critical. Corporate influence on division information systems, business unit autonomy, the centralization/decentralization question, and similar issues should be discussed by the team to developed recommendations appropriate to existing organizational levels. Additional information about these principles can be found in the IBM publication titled *A Management System for the Information Business; Volume I, Management Overview,* GE20-0662.

Need for a Steering Committee

Figure 28 represents a general approach for providing the enterprise with adequate information resource support. The steering committee acts as the overseer of the information resource organization and represents the functional groups of the enterprise. Its primary concerns are policy setting (that is, establishing the direction of information use in the enterprise), exercising control mechanisms to ensure that the desired results are achieved, and monitoring to measure the effectiveness the information system (see Figure 29).

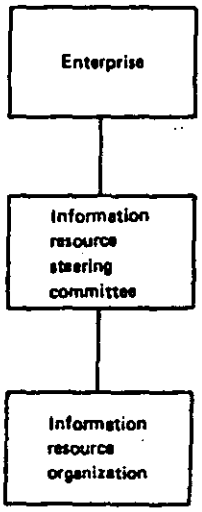


Figure 28. Information resource support organization

Policy Setting	• Data responsibility/accountability
	• Long-range planning
	• Funding strategy
	• Application development strategy
	• Performance objectives
Control	• Budget approval
	• Project authorization
	• Performance appraisal
Performance Monitoring of IR Function	• Budget monitoring
	• Project monitoring
	• Operations monitoring

Figure 29. Functions of information resource steering committee

The need for a steering committee can be determined by examining information needs and usage in the enterprise. Is there a standard way of doing business with data processing? Are applications developed in a priority sequence in today's environment, and what are the criteria for those priorities? Do long-range DP plans support the business requirements?

The policy-setting function of the steering committee defines a course of action necessary to guide the information resource group in providing the enterprise with DP support. Policy setting includes establishing responsibility and accountability for the collection, retention, accuracy, and availability of data. It defines the security measures that are necessary to protect the enterprise.

The steering committee also delineates the data responsibilities of the functional areas and the data processing groups. Long-range planning concerning hardware, optimum use of software (that is, distributed data processing versus centralized data base), new software applications, and data availability is also part of this committee's responsibilities. Policy-setting functions also include funding strategies, such as establishing a charge-for-services policy, determining the value of data to the enterprise, deciding to buy or lease equipment or to make or buy applications, and setting operational objectives.

The control function of the steering committee ensures that the information resource meets the business needs on an ongoing basis. Thus it needs to authorize projects and set priorities for them, approve budgets, and appraise system performance.

The monitoring function supports the control function. Monitored are the funds being expended on development and operations, the projects under development, and the operation of production systems to ensure continuing cost-effectiveness.

Figure 30 suggests the composition of a steering committee.

User Community	• Finance
	• Engineering
	• Marketing
	• Personnel
	• Manufacturing
	• Departments that use data processing resource
Information Resource Organization	• VP responsible for IR (chairman of committee)
	• Data processing manager, secretary of committee
	• Data administrator

Figure 30. Composition of the information resource steering committee

Review of the Information Resource Organization

Prior to this review of the information resource organization, the team has identified problems related to the business processes and separated those problems dealing with current information systems support. It has determined that the proper implementation of the information systems architecture would provide solu-

tions to those problems. (See "Write Findings and Conclusions Statements" in Chapter 11.) However, unless an organization is in place that is responsive to business needs and can react to the priorities of the functional areas of the business, the implementation of the information architecture and the specification of follow-on systems may not be carried out expeditiously.

This phase of the BSP study is designed to review the information resource organization in place to ensure that adequate resources have been deployed to implement the information systems architecture, to better exploit the data resource, and to overcome other inadequacies. The BSP team should analyze the strengths and weaknesses of the current information systems organization. It should consider the requirements of the information architecture (in a prioritized list of applications) and the problems voiced by the interviewees regarding present service inadequacies, both in data processing and in information resource management. This review can also be recommended as a follow-on activity for the steering committee. Additional organizational considerations can be found in the IBM publication titled *A Management System for the Information Business; Volume IV, Managing Information System Resources,* GE20-0751.

formation Resource Organization Responsibilities

The information resource organization should be structured to carry out the policies of the steering committee, to promulgate the precepts of "data is a resource," and to ensure that data is managed throughout its life cycle. Therefore, as its major functions, it should have data planning and control, data acquisition, and data stewardship. Data retirement activities can be divided between data acquisition and stewardship. Figure 31 depicts such an organization. The following paragraphs cover some of the tasks for which each organizational function should be responsible.

Data and Application Planning and Control

This function is responsible for the continuing planning necessary in a dynamic organization. It maintains the I/S architecture, sets priorities for applications, plans for improvements to systems software and hardware, and develops the uses of data. In addition to planning, this area is also concerned with measuring and auditing the performance of the development and production groups. It needs to know how much of the machine's capacity is being used and is available for additional use, and it needs to know whether the service levels are being achieved since it also is the user interface. This group also sets standards for both performance and documentation.

Data Administration and Application Development

Both data administration and application development are concerned with the acquisition and use of data; therefore, these two functions are grouped together. Data administration is defined as the function responsible for the planning and control of the enterprise's data, whether computerized or not. Its objectives are to maximize the availability of data and to control the use of data. The control of data needs to be exercised in the areas of acquisition, storage, access and disposition. Further discussion of data administration can be found in Appendix K.

Application development uses data to create information for a specific purpose. It, therefore, requires close

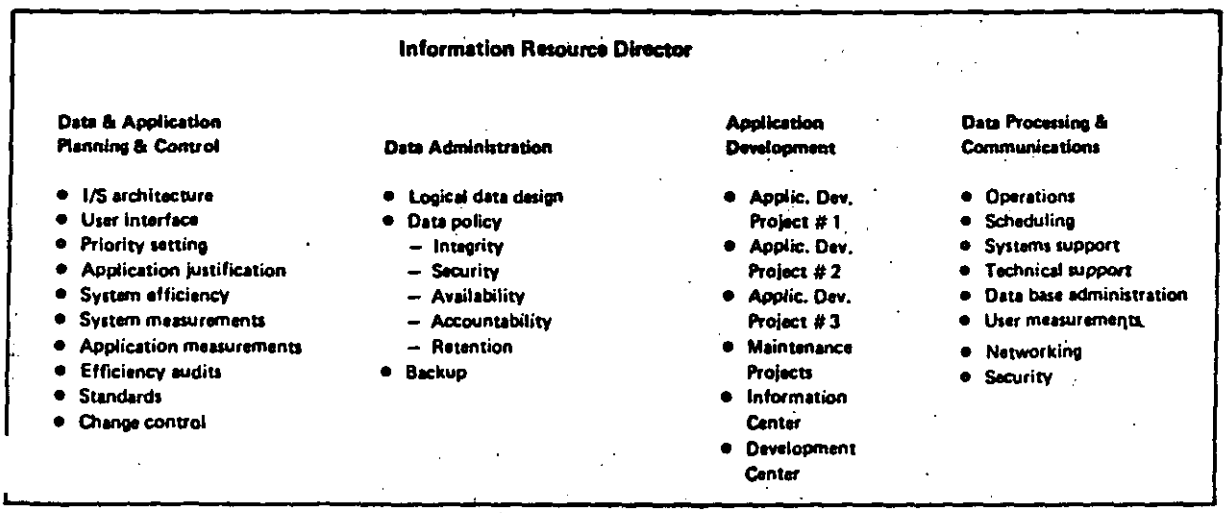


Figure 31. Suggested information resource organization

interaction with the data administration function in the logical data base design. Traditionally, development projects have defined their own data base requirements, but with the establishment of shared data bases, this requirement becomes one of ensuring that the data required has been properly identified and is available for access and use in the application.

The application development group is responsible for the project throughout the development cycle. The group is also responsible for the maintenance of existing applications. In a DB/DC environment, maintenance is the addition, modification, or rearrangement of transactions. Modification of the data base is controlled by the data base administrator.

IBM has developed an independent study program, *Managing the Application Development Process*, SR20-7245, which describes the application development process that has been successfully used by IBM's Data Processing Services organization. This independent study program is in two parts. Part I has three modules. Module 1 covers the need for a structured application development process, explains the complexity of development, and offers insight into the application development methodology. Module 2 presents the concepts that are key to establishing a structured process, one that promotes successful application development. Module 3 provides some techniques for evaluating progress and for implementing changes that may benefit your organization.

Part II of the study program applies the project management techniques to a specific case, the development of an online order entry system at a publishing company. This part spells out the roles, responsibilities, and interaction among the people involved in an application development project.

Data Processing and Communication

The data processing function is the production arm of the information resource organization. It is concerned with providing an uninterrupted flow of service and information to the users. Some of the functions necessary to support this function are operations, technical support (systems), data base administration, measurements (systems utilization, available capacity and service level), and scheduling.

For a more complete discussion of data processing's role and responsibilities refer to *A Management System for the Information Business*, Volumes I through IV (GE20-0662, -0749, -0750, and -0751, respectively). These volumes apply fundamental management principles to the data processing organization. The insight thus provided is useful in analyzing the organizational and personnel skills necessary to properly manage, that is, plan and control, the information resource and data processing function. The information systems management processes of these volumes are listed in Appendix H.

Chapter 14. Developing Recommendations

Specific BSP study recommendations center on:

- Information architecture
  - Accept the information architecture as the base for current direction and future I/S planning.
  - Make required changes to applications currently under development.
  - Make interim improvements to current applications.
  - Prioritize areas of the information architecture to which I/S resources should be applied.
- Information resource management
  - Implement and enhance data administration to control the data resource of the organization.
  - Improve the information planning process to support the business more effectively and utilize resources.
  - Provide a control system to measure the success of implementations.
- End-user computing — Enhance the decision-making capability of the functional staff through the use of available business data.

These recommendations form the Strategic Information Plan. I/S now has a master plan for the data resource — a plan driven by the information architecture within which all new development and modifications to current applications must fit. The plan presents recommendations concerning the planning and management of the information resource and presents architectural priorities sequenced from a perspective of benefits to the enterprise.

The strategic plan should be presented to management for validation and approval (see Chapter 15, "Reporting Results"). For each recommendation, there may be an associated project, such as the implementation of an information center. Depending on the team's perspective, other short-term recommendations may flow directly into implementation. For each such project, an action plan should be developed identifying the key decisions and activities required to help management provide proper direction. The action plan should include the following for each project:

- *Project Scope:* Describe the subject of the project, including the size, impact, and purpose.
- *Potential Benefits:* Describe anticipated benefits or reasons for doing this project.
- *Deliverables:* List and define each output or result expected of the project.
- *Schedule:* Identify project duration to the extent possible at this stage.
- *Business Tools and Techniques:* Describe any required practices or methods as specifically as possible.
- *Training:* Describe orientation, education, and study necessary for successful execution of the project.
- *Communications:* Identify coordination, liaison, interface points, or functions, as well as documentation requirements.
- *Control:* Define project responsibilities as well as review/approval requirements.

Once management approves the BSP recommendations (the strategic I/S plan), a tactical plan should be developed to facilitate the implementation of those recommendations. The tactical plan will contain the following resource architectures (which represent a detailed subdivision of the information architecture):

- *Data Architecture:* A data development plan that identifies the required data by defining the important business entities and the activities that link them.
- *Application Architecture:* Illustrates how applications and sources of data relate.
- *Geographic Architecture:* Shows where information originates and where it is used geographically.

The development and subsequent analysis of these architectures may cause a resequencing of application implementations. For example, the prioritization may have been sequenced initially from a benefits view if little information about implementation expense or technical prerequisites were available. In this and other areas, the resource architectures will help generate additional information, which, when combined with the

BSP prioritization information, give a more complete basis upon which to sequence implementation.

The tactical plan can be developed by using IBM's Business Systems Planning Implementation (BSPI) technique, which is presented in the BSPI class of the IBM Information Systems Management Institute (class number W9930).

The completion of these activities enables the BSP study team to proceed to its final activities: preparation of the study report and presentation of the study results to executive management. Thorough preparation of the action plans, potential benefits, and costs enables the executive sponsor to evaluate the recommendations. The study team should ask for and expect a prompt approval of recommendations and a prompt commitment by management to implement them.

Chapter 15. Reporting Results

Having defined the information architecture, identified the architecture priorities, reviewed and assessed I/S management, and developed recommendations and the action plan, the BSP study team is ready to complete its mission by preparing the study report and preparing and delivering an executive presentation. The purpose of the report and presentation is to obtain further executive management commitment and involvement for implementing recommendations from the BSP study.

The following activities are performed in preparing the study report and executive presentation:

- Review the report outline
- Prepare the report
- Select the presentation medium
- Present to executives

Review the Report Outline

During the initial steps of preparing for the study and developing a work plan for conducting the study, the team prepared a preliminary report outline. Since the report now materializing from that outline is to be a consensus of the entire team, each of its sections should be reviewed by each of the members so that the final document reflects all their comments.

The report may be structured in many ways, depending upon precedent and methods of presentation within the business conducting the study. However, to assist the team in its consideration of pertinent areas, an extensive list of topics is presented in Appendix I. For a sample report see *Mecca International Suit Corporation Business Systems Planning Report (Sample)*, IBM document G320-6503.

The most significant findings, conclusions, and recommendations should be summarized in the first few pages of the report for the use of top management. Supporting details should be included later and in the appendixes for other members of the organization and for team members who will participate in follow-up activities.

Prepare the Report

The primary writing responsibility for each of the sections of the report was assigned during study preparation. As the study progresses, changes, additions, and deletions can be made to the preliminary outline. By the time the executive interviews have been completed, agreement should have been reached on the final table of contents, and the individual study team members should have available to them most of the information needed to complete their assigned sections. The study team leader usually assumes the responsibility for writing the background and overview because much of this material comes directly from the orientation information presented to the team at the kickoff meeting.

The conclusions, recommendations, and an action plan should be reviewed with the executive sponsor before the team drafts the final report. Controversial areas or areas affecting the business most may be reviewed with the executive involved, to determine how best to present the recommendations.

Select the Presentation Medium

The principal factors to consider in determining how the report should be presented are the type and size of the audience and the accepted ways of making such a presentation in the particular business environment. Consultation with the executive sponsor early in the study to obtain his advice on this subject can be most helpful in establishing the proper direction.

If the presentation is made to a small group, flipcharts are adequate and popular. If the audience numbers more than a dozen, viewgraph foils or slides should be considered. If slides are to be used, adequate time should be allotted to have them prepared, whether in-house or by an outside vendor.

Present to Executives

The executive presentation can be developed completely from material contained in the final draft of the report. The principal aims of this presentation are to inform management of the study's findings, make recommendations, and secure approval of the action plan.

The presentation should be concise, preferably no longer than two hours. It should be logical and factual and should end with recommendations for the follow-on activities. For example, it may take the following form:

- Introduction
  - Background and overview
  - Objectives
  - Scope
  - Study team

- Study approach
  - Business and I/S review
  - Business processes and data classes
  - Matrices
  - Executive interviews
- Major problems identified
- Conclusions and recommendations
  - Information architecture and priorities
  - IRM requirements
- Action plan for follow-on project activities
  - Description(s)
  - Deliverables
  - Resource requirements
  - Benefits
  - Schedules

Chapter 16. Overview of Follow-On Activities

The BSP study ended with the development of an action plan for the follow-on activities and a presentation for management approval to proceed in the execution of that plan. This final chapter of the BSP planning guide gives an overview of the follow-on activities and serves four purposes:

1. To relate the follow-on activities to the BSP study.
2. To show how to capitalize on the results of the BSP study.
3. To further explain the major follow-on projects.
4. To identify several IBM courses that may be particularly helpful to those responsible for implementing the BSP recommendations.

Perspective on Follow-on Activities

Relation of BSP Study to Follow-on Activities

The follow-on activities are a continuation and expansion of the major activities in the BSP study. The major thrust of the study was one of understanding, developing findings and conclusions, and making recommendations. Although the thrust in follow-on projects is still to understand, greater emphasis is put on detailed definitions and planning for implementation of the project.

There is a great need for communication among the project teams if project implementation is to be successful. Therefore, the information resource director should take overall responsibility for the projects, since the functions performed during the follow-on activities are a part of the IRM responsibilities. Because the follow-on activities build upon the results of, and the information gathered in, the BSP study, there is a need for continuity of team members. At least one person on the BSP study team should have been chosen from the data processing function, with the idea that he would remain with the follow-on activities.

The functions performed by the executive sponsor of the BSP study should be performed by a steering committee in the follow-on projects, unless there is a management committee already established to undertake those tasks.

Preparation for Follow-on Activities

Because of the variations in the findings and conclusions that may result from BSP studies, the follow-on activities and the emphasis placed on each of them will vary from one BSP study to another. For purposes of this discussion, the following assumptions are made:

1. IRM functions must be changed or added to provide a controlled environment for the development and implementation of the information architecture identified in the BSP study. A detailed information systems plan will be developed that will reflect these planned changes.
2. The information architecture must be further defined and data bases identified.
3. The distribution of information systems may be a consideration.
4. One or more major applications have been chosen for development and implementation.

To aid in accomplishing these objectives, IBM's Information Systems Management Institute offers a course called Business Systems Planning Implementation (BSPI) — course code W9930. This course teaches methodologies, and discusses issues and policies necessary to convert BSP recommendations into functioning applications and data bases. Among the methodologies taught are those for creating plans for application and data base development as well as the geographic distribution of these applications and data bases. Relevant issues regarding policies on I/S planning, data administration, development, and organization are explored.

Information Resource Management

IRM is the keystone of effective support of the business by information systems. The importance of continued emphasis on IRM is summarized below:

- Development of the complete information architecture will take place over a period of years.
- Changes to business strategies and plans and to information technology will continue throughout the development of the information architecture.
- The processes of managing information will be refined and changed continually to properly plan,

measure, and control required information resources.

- The BSP study is a one-time effort that should be followed by continual information planning to fully capitalize on the results.

IRM in Perspective

The IRM follow-on projects will emanate from (1) requirements to support IRM objectives, (2) changes resulting from the development and implementation of the first application, and (3) recommended changes to solve I/S support problems.

From this, one can assume that the recommendations included:

1. Emphasis on information systems planning and control.

2. The establishment of a data administration function.

3. Some near-term projects, such as the establishment of a steering committee and refinements to project control. Capacity planning is also a natural follow-on to a BSP study since it examines the data processing capacity and determines whether changes are required to accommodate the architectural priorities or to solve some of the I/S support problems.

4. A major project(s) in IRM — for example, establishing an action plan to move to distributed data processing by defining the geographic architecture.

An important action after the BSP study is the establishment of a continuing information planning function using the BSP study as a base. Although the BSP study culminated in an approved action plan, the overall objective of an IRM project is to develop a long-range information plan that will direct the design, development, and implementation of the information architecture. It should include sufficient detail on projects, resources, and schedules to guide all levels of management on what is to be done, when, and by whom in the organization.

Since data administration is a complex area requiring long lead times for implementation, the data administrator should be appointed as early as possible so that this function can be given proper attention. Two major activities should be addressed immediately:

1. Establishment of a data policy that fixes responsibility and accountability for data accuracy, consistency, and timeliness to a specific organization.

2. Establishment of a data dictionary by the data administration function to catalog the meaning and use of data.

The importance of information planning and data administration cannot be overemphasized, and they will nearly always be a part of the follow-on activity. The data administration activities are covered thoroughly in IBM, GUIDE, SHARE, and trade publications, and are discussed in Chapter 13 and Appendix K.

Some of the near-term projects should be done immediately after the BSP study recommendations are approved. The establishment of a steering committee and a project control system are excellent examples of near-term projects that are required. A steering committee will direct the follow-on activities, while a project control system will help the information resource director adequately control and coordinate activities.

IBM's Information Systems Management Institute offers a 3½-day course that deals specifically with these issues and is intended for managers and executives responsible for the I/S function. The course is called Organizing the Information Systems Business — course code W9935.

Information Architecture

The information architecture is one of the key results of the BSP study. Its primary purpose is to provide direction to the information resource organization in information systems planning, design, and implementation and in managing and controlling the data resource. Therefore, it should be implemented as described in Chapter 8 to ensure maximum utility.

The refinements to be accomplished in a follow-on project should include a confirmation of the information systems architecture, a decomposition of each application, application interrelationships, and information flow among applications.

Other activities that should be included in the information architecture project are:

- Examination of current data processing systems to determine how they can evolve into the new architecture.

- Documentation of alternative architectures that were tried or investigated and rejected, with the reasons for rejection.

- Evaluation of the technical implications of the information architecture, including control systems, data base/data communications requirements, and potential distribution of both information and systems.

Architecture Refinement

Using the information architecture defined in the BSP study, each application description should be expanded. Each description should include the purpose of the application, business problems addressed, data created and used, dependencies on other applications, general requirements for implementation, and priority of the application.

Current Systems Examination

Data processing applications currently in use or under development should be carefully examined to determine how each one relates to the applications described in the information architecture. Possible modifications to current applications should be documented to show how they can be used in the future architecture. Where modified applications could be used in lieu of new ones, a description should be included in the architecture documentation. For applications already under development, an evaluation should be made of necessary modifications to make the application compatible with the information architecture. The development schedule should be reviewed to determine the effect of the changes in the applications under development when compared with changes to be made after application implementation. Both present commitments and future priorities should be considered in this evaluation.

Data Base

Integral to the information architecture definition in the BSP study were the data classes supporting the business processes. A logical grouping of related data classes will yield the set of data bases that will support the architecture implementation.

A data base is defined as a nonredundant collection of interrelated data items processable by one or more applications. In this definition:

- *Nonredundant* means that individual data items appear only once (or at least less frequently than in normal file organizations) in the data base.

- *Interrelated* means that the files are constructed with an ordered relationship that allows data elements to be tied together, even though they may not necessarily be in the same physical record.

- *Processable by one or more applications* means simply that data is shared and used by several different subsystems.

Development of a data base has some obvious benefits. By consolidating files, the user can obtain better control of data and reduce storage space. Equally important are the resultant data synchronization and timeliness. Use of a single information source makes processing more accurate because all applications refer to the same data.

A data base system can help overcome some of the complexities of data management. It can provide additional data relationships while minimizing storage redundancy.

A comparison of the data base environment with the traditional approach to systems development and maintenance reveals the advantages of the data base concept. In the traditional approach, a system is usually designed, programmed, tested, and then implemented as a total entity. Its advantages cannot be realized by the end user until the entire system is completed. The amount of time involved can cause frustration, since business requirements cannot be kept frozen long enough to avoid changes and delays. Also, when data or logic changes are required, considerable testing may be necessary to determine how the changes affect other programs or systems functions.

By contrast, the data base approach allows a gradual transition from existing applications to online, transaction-driven applications. With gradual implementation of transactions, user department signoff can be obtained more easily and the user can enjoy the benefits earlier. Changing data needs can be accommodated without affecting programs that do not use that specific area or segment of the data base. Thus, the data base environment can be a more effective way to accommodate change, deliver benefits to the user, and control development costs.

Data Dictionary

The data dictionary is the primary tool of data administration to manage the data inventory. Data administration begins to implement the data dictionary by discussing with the creator of data, as defined by data policy, how the data should be described in terms of data elements, and then developing the standards and

procedures to be followed by the other users. These are then documented into a data dictionary which provides the guidance on what data is available, how it should be used, and who is allowed to use it. Further information on the use of a data dictionary is contained in Appendix L.

End-User Computing

End-user computing is the direct use of data, tools, and services by business people to meet business needs observed as extensions of corporate information and data requirements. With the advent of personal computers and interactive computing via terminals and workstations linked to a central computer, end-user computing has become more and more prevalent in the business professional community.

Since end-user computing is becoming a significant activity in many corporations — and promises to become even more significant over time — organizations are finding that to effectively support the objectives of the enterprise end-user computing must be managed as a business activity as much as a technical activity. Thus, end-user computing requires both a support structure and a management system. This involves:

- Enterprise-level management direction for end-user computing.
- A way of identifying and prioritizing user needs and ensuring the selection of the appropriate tools and a delivery system for them.
- A means for providing each user the appropriate workstation.
- A process for giving users access to corporate data and other types of needed information.
- A support organization to provide effective guidance, training, and on-going assistance for users.
- The inclusion of an information center as part of the end-user strategy.

Along with the information center, the development center is another technique for providing end-user computing capability. As described below, the information center is directed toward the business professional and the development center is a tool that will enhance the productivity of the data processing professional.

Information Center

The information center is a department within the DP organization which provides end users with packages and services that enable them to access the computer directly to improve their productivity and/or their decision-making capability. End-user products include report writing, query processing, financial planning, and packages for training and document preparation. The key to a successful information center is the quality of support provided to the users.

If an information center is being recommended, IBM's Information Systems Management Institute offers a 2½-day course for prospective or newly appointed information center managers. The course is called Information Center Implementation — course code W9934.

Development Center

One form of assistance the application development group will want to consider is the development center, which offers dedicated resources (both hardware and software) and special tools that accelerate application development and increase programmer productivity. Effective development centers provide unconstrained hardware resources to allow interactive development with time-sharing systems. Application generators are used extensively. In larger groups, there is increasing use of dedicated host system resources for application development.

Using the Information Architecture Chart

The information architecture chart (Figure 18) serves as a management communication tool by providing an overview of the major elements of the architecture and their relationships. The chart also provides a firm foundation for later development of individual portions of the architecture.

The architecture chart provides a wealth of information and is interpreted in the following fashion: The major blocks represent individual management information systems that may be implemented in logical building block sequence. Reading across from an individual system identifies those portions of the business (business processes) which are specifically supported by the information systems. Reading upward to the data axis identifies those data bases which may be controlled by the specific information systems. Finally, the lines and arrows connecting the information systems represent the flow of data among the systems.

Thus, data is managed as a corporate resource and is provided to the appropriate portions of the business as needed by means of systems that may be implemented piece by piece as business justification warrants.

Distributed Information Systems

The primary purpose of distributed information systems is to provide the user with a level of computing resource best suited to his operational requirements. The elements that can be considered for distribution include hardware, program execution, program development, and data. The degree of distribution can vary from totally centralized to totally decentralized. The information architecture defined in the BSP study provides a good foundation for the planning necessary to address I/S distribution requirements.

To add distributed I/S parameters to the architecture, the project team must:

- Thoroughly understand the degree of distribution of each element
- Understand the reasons for and against distribution of the various elements
- Be able to put into perspective all the factors that affect distribution and sharing of data processing resources
- Appreciate the need for a long-range I/S plan and for development of a facilities plan to support the I/S plan

Several of the matrices developed in the BSP study can be very useful in developing the architecture extensions required to define distributed systems. The process and data flow analysis should be extended to include physical locations for systems and data requirements. Location versus organization, location versus process, and location versus data matrices should be prepared. A process versus data matrix for each location (or group of similar locations) should be prepared with additional parameters on the axes to include data use, security, auditability, data occurrence (all or partial), activity volumes, response required, and criticality. Figure 32 shows this type of matrix for a manufacturing plant site.

These additions to the architectural description of the information systems can then be input to the individual application requirement studies to determine detailed distributed systems requirements. They will also give an overall feel for the applicability of distributed systems to the information architecture.

The reader may wish to refer to the IBM document entitled *BSP – Planning for Distributed Information Systems*, GE20-0655. This document outlines extensions to the basic BSP methodology and shows the development and use of the matrices cited. It then examines the possibilities for distribution of information systems resources and for choosing the most effective plan for implementation.

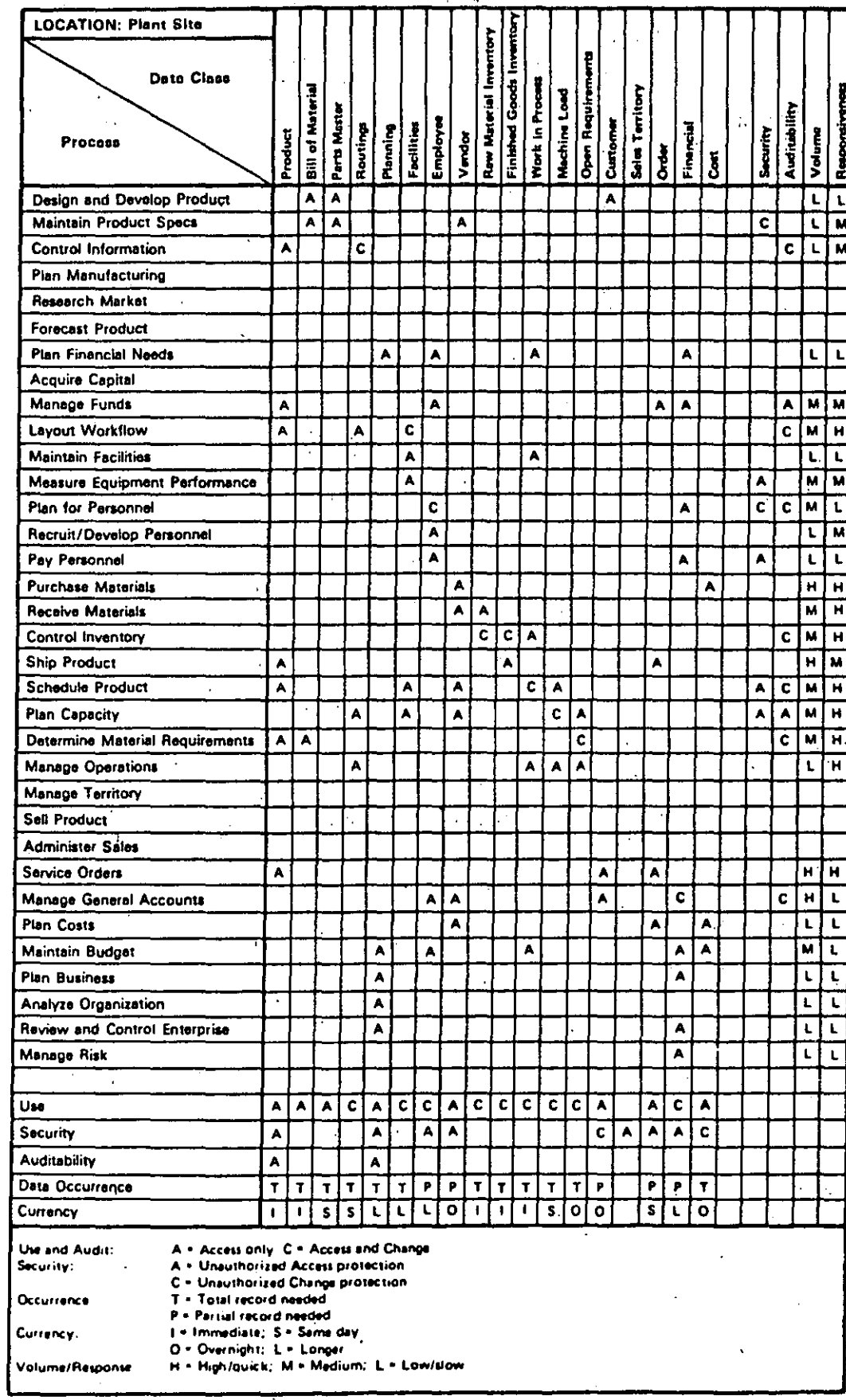
Developing the First System

As discussed under "Implementation Options" in Chapter 12, information requirements may be satisfied in several ways. The quickest, least expensive way (that is, using the fewest people) would be to buy or otherwise procure existing code, as in an application package. Second, some user requirements might best be satisfied by giving the user direct access to the computer through the establishment of an information center. Third, if an application must be developed, then consider the IBM Application Transfer Team approach to reduce the resource requirement and accelerate the application design and justification.

If none of the above are applicable, then an application will have to be developed. As development begins, care should be taken that the application is not developed "by itself." The BSP study has set a certain direction for implementation of applications. This first development and implementation must support the business processes and must:

1. Fit properly into and interface with the other (future) applications of the overall architecture described in the BSP study
2. Provide for proper implementation of the appropriate data base(s)

These concurrent projects must be designed and managed to ensure that they lead toward an integrated whole (the information architecture).



LOCATION: Plant Site																								
Process \ Data Class	Product	Bill of Material	Parts Master	Routings	Planning	Facilities	Employee	Vendor	Raw Material Inventory	Finished Goods Inventory	Work in Process	Machine Load	Open Requirements	Customer	Sales Territory	Order	Financial	Cost		Security	Auditability	Volume	Responsiveness
Design and Develop Product		A	A											A								L	L
Maintain Product Specs		A	A					A												C		L	M
Control Information	A			C																	C	L	M
Plan Manufacturing																							
Research Market																							
Forecast Product																							
Plan Financial Needs					A		A				A						A					L	L
Acquire Capital																							
Manage Funds	A						A									A	A				A	M	M
Layout Workflow	A			A		C															C	M	H
Maintain Facilities						A					A											L	L
Measure Equipment Performance						A														A		M	M
Plan for Personnel							C										A			C	C	M	L
Recruit/Develop Personnel							A															L	M
Pay Personnel							A										A			A		L	L
Purchase Materials								A										A				H	H
Receive Materials								A	A													M	H
Control Inventory									C	C	A										C	M	H
Ship Product	A									A						A						H	M
Schedule Product	A					A		A			C	A								A	C	M	H
Plan Capacity				A		A		A				C	A							A	A	M	H
Determine Material Requirements	A	A											C								C	M	H
Manage Operations				A							A	A	A									L	H
Manage Territory																							
Sell Product																							
Administer Sales																							
Service Orders	A													A		A						H	H
Manage General Accounts							A	A						A			C				C	H	L
Plan Costs								A								A		A				L	L
Maintain Budget					A		A				A						A	A				M	L
Plan Business					A												A					L	L
Analyze Organization					A																	L	L
Review and Control Enterprise					A												A					L	L
Manage Risk																	A					L	L
Use	A	A	A	C	A	C	C	A	C	C	C	C	C	A		A	C	A					
Security	A				A		A	A						C	A	A	A	C					
Auditability	A				A																		
Data Occurrence	T	T	T	T	T	T	P	P	T	T	T	T	T	P		P	P	T					
Currency	I	I	S	S	L	L	L	O	I	I	I	S	O	O		S	L	O					

Use and Audit:	A = Access only C = Access and Change
Security:	A = Unauthorized Access protection
	C = Unauthorized Change protection
Occurrence	T = Total record needed
	P = Partial record needed
Currency	I = Immediate; S = Same day
	O = Overnight; L = Longer
Volume/Response	H = High/quick; M = Medium; L = Low/slow

Figure 32. Distributed information requirements analysis

Managing the Application Development Process

Application development is a complex process. It requires a phased approach and a well-defined project management process. The following examples are from the IBM independent study program *Managing the Application Development Process.*

A phased development approach must be used so that functional executives and the information resource director can review and approve the products resulting from each phase before the project proceeds. Figure 33 is an example of such a phased approach and reflects the people resource requirement by stage.

The project management process specifies the planning and control tasks necessary to manage the phase, or group of phases, which make up a project, as shown in Figure 34. All products and tasks must be clearly defined so that they can be performed, measured, and managed. From a user standpoint, it means a process that will deliver the application on time and within budget.

Three IBM courses are available to support application development efforts:

1. Project Management (3½ days, course code W9924) is directed toward DP management.
2. Project Implementation (4½ days, course code W9885) is intended for the project implementation team.
3. Project Planning and Control for Users (3 days, course code W9896) is intended for non-DP personnel.

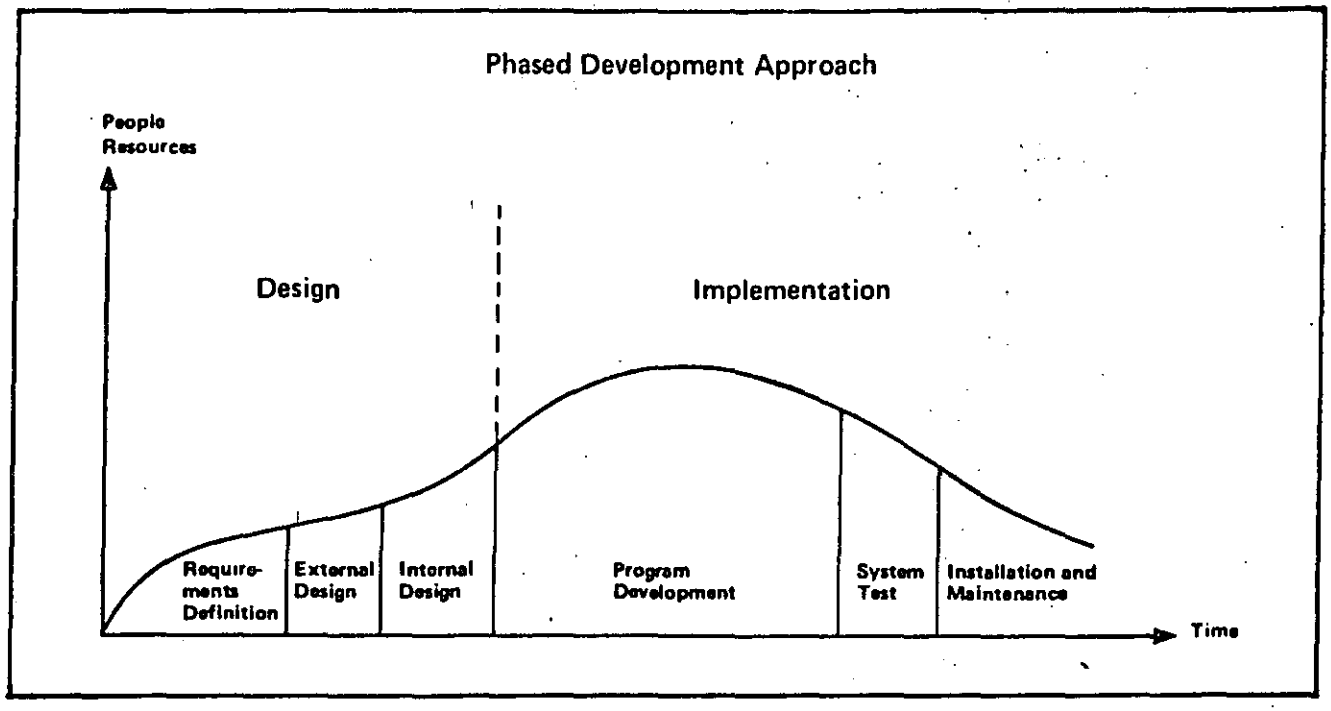


Figure 33. Application development process

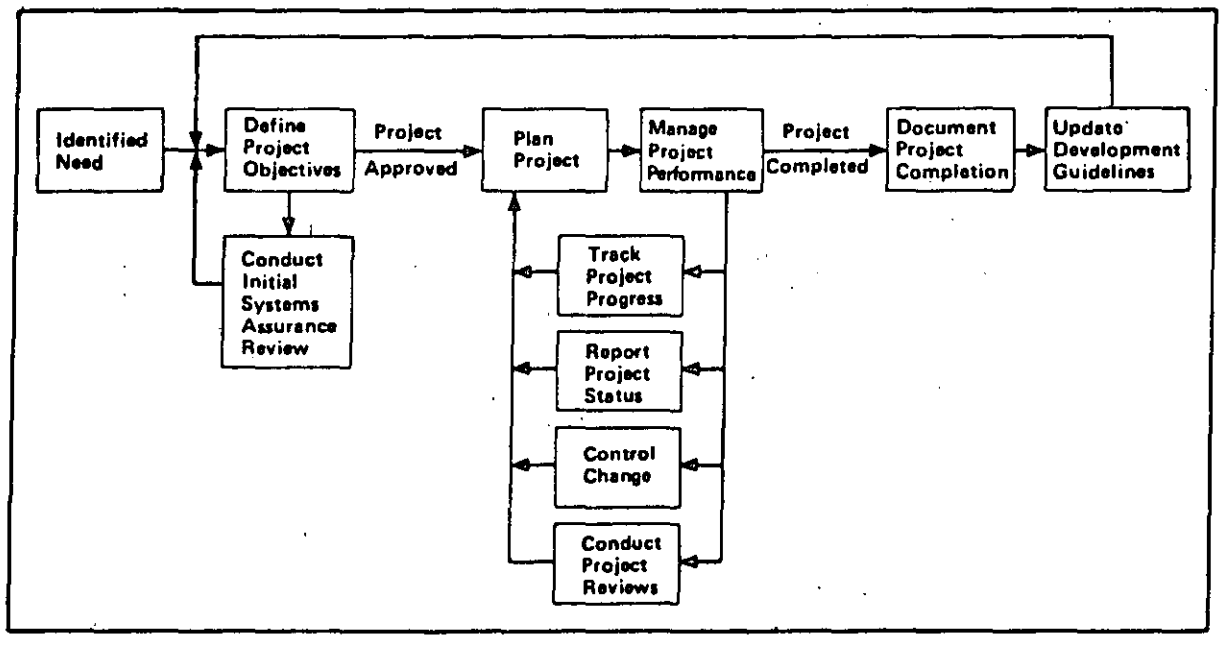


Figure 34. Managing the application development process

DISEÑO DE BASES DE DATOS

INTERACTIVE SPECIFICATION AND FORMAL VERIFACTION OF USER'S VIEWS IN DATA BASE DESIGN"

MAYO, 1985.

INTERACTIVE SPECIFICATION AND FORMAL VERIFICATION OF USER'S VIEWS IN DATA BASE DESIGN

C. Baldissera, S. Ceri, G. Pelagatti, and G. Bracchi

Politecnico di Milano
Istituto di Elettrotecnica ed Elettronica
I-20133 Milano
Italy

Abstract

Among the different phases of the data base design process, the phase of modelling user's views has a particular relevance.
This paper describes an interactive methodology for designing the views starting from the elementary sentences that specify the requirements of the application.
The methodology generates a canonical representation, and provides verification algorithms for detecting inconsistencies, redundancies and ambiguities, and for restructuring and optimizing the model.

1. Introduction

The process of logical data base design consists of many different phases: requirement collection, view modelling, view integration, schema optimization, and schema mapping /1-2/. Particular emphasis has been devoted in the literature to the view modelling phase /2-8/.

The goal of this phase is to obtain a formal representation of each user's view starting from semantic information about the application, and using a common modelling technique; once the user's views are represented, they can be compared and integrated by the data base designer (view integration).

In order to obtain the formal representation of the user's view, a modelling procedure and a verification procedure have to be performed /1/.

In the modelling procedure, using the primitives of the target data model, a representation is developed which embodies the user's view; in the verification procedure this representation has to be checked, evaluated, restructured, and optimized.

At the current state of the art, the problem of view modelling is being tackled by the data base designers in a manual manner with little discipline. However, real world experience shows that computer support is needed for the design of today's large data bases /9/. Fully automatic design techniques are not viable, since they may produce results that do not express the actual user's perspective, if a human is not actively involved /2,10/.

The basic assumption of this paper is that it is possible to help the user in performing the view modelling process through an interactive system which:

a) takes the responsibility of linking together the different facts (requirements) that the user is expressing, thus allowing the user to collect them in any order and to concentrate on the completeness of the collected requirements and not on their organization;

b) checks the formal correctness of the representation which is being built and detects inconsistencies and redundancies;

c) shows the user those requirements that have already been collected and that can be interpreted in different ways, and asks the user to solve the ambiguities;

d) produces, at the end of the whole process, a schema which is best suited to the subsequent design phases (the design process implies a choice among different possible representations of the user's requirements).

The view modelling process starts, in the interactive design methodology that is proposed here, with the collection of the elementary sentences that express the user's requirements. The sentences can be collected and introduced by the user in any order. The system reorganizes the elementary sentences to finally obtain a canonical formal representation of the user's view. A goal of a canonical representation is to eliminate the arbitrariness of the perspective of different designers in modelling the same view, at least as far as syntactic aspects are concerned. (Note that if two designers see the problem in two "semantically" different ways, two different canonical representations will result. However, who can tell in this case that they are looking at the same problem?). The parti-

This work was partially supported by Consiglio Nazionale delle Ricerche, Italy.

CH1405-8/79/0000-0262$00.75 © 1979 IEEE

cular canonical representation that has been chosen will be defined in Section 2.

It is recognized that requirement collection and view modelling are operations which deal with the "meaning" of data. Unfortunately, there is a lack of precise definitions in this field. One of the aims of the interactive algorithms that will be presented in Section 3 is to force the user to express the intended meaning of data in a precise and unambiguous way, thus clarifying also to himself what he really means. In fact, while the syntactic portion of the verification procedure is entirely automatic, the semantic portion generates assertions about the view representation that must be affirmed by the user.

The main emphasis of this paper is on the design method and on the algorithms that can be used for view modelling, and not on the particular binary data model which has been adopted to represent the view. The main features of the data model will be briefly recalled in Section 2. This specific binary model has been selected since it is founded on a sound theoretical basis, it permits incremental specifications of basic facts, it is easy to formulate, to manipulate, to restructure, and to understand, it allows flexible matching and merging of different user's views and efficient mapping to other data models, and it permits the definition of user's intended semantic /8, 11-13/.

In describing the view design methodology, we will mainly deal with the problem of generating the abstract model of the user's information structure, while we will not consider the problem of specifying and representing the various static and dynamic constraints; some of the constraint modelling problems are treated in /14/.

2. The Binary Data Model

In this section the main properties of the data model which will be used in the view design methodology will be defined in a formal way. The definitions will be given synthetically, since many of the modelling concepts can be found spread in other papers in the literature /2-8, 15/, and particularly in /13/.

The present paper deals with design of logical schemas. A logical schema is a set of sentences in some language which define the semantic properties of data as intended by the user. Unfortunately, such sentences are usually ambiguous. Therefore, in section 2.1 a basic formal model is defined; this model will be used in section 2.2 to represent the possible meanings of user's sentences. The algorithms which will be presented in Section 3 aim at forcing the user to give additional specifications, so that only one interpretation becomes possible.

Section 2.3 extends the basic model in order to deal with various semantic properties. Of course, other features could be added, but here only those features which will be used in the modelling process of Section 3 are considered. Finally, section 2.4 defines a canonical form of a schema which allows elimination of further ambiguities which may be present in the user's semantics.

2.1 The basic model

Any object or fact of the real world to be represented in the data base constitutes an entity. The set of all entities that may exist in any possible data base state is $E = \{e \mid e \text{ is an entity}\}$. The set of entities which exist in a given state is $[E]_s$.

The basic unit of information is represented by the fact that two entities are grouped in a pair having some semantic meaning. For instance, 'owns $\langle e_1, e_2 \rangle$' and 'lives-in $\langle e_1, e_2 \rangle$' are two pairs on the same two entities, having different semantic meaning. A relation $R_i \subset E \times E$ is the set of all pairs having the same semantic meaning. Usually, the relation name is the element through which we express this meaning. A relation instance $[R_i]_s \subset R_i$ is the set of pairs of R_i which exist in a given state. The above concepts are considered to be primitive in our model; the following definitions specify the derived concepts of the model.

Given a relation R_i, the set of all entities which constitute the first (second) entity of its pairs is called its first (second) projection R_i^1 (R_i^2).

Two projections R_i^e, R_j^m are equal if they contain the same elements, i.e. $R_i^e = R_j^m$. Given a set of relations, their first and second projections can be collected into groups of equal projections. A type T_r is then associated with each such group, called the derivation group of T_r, in such a way that $T_r = R_i^j$, for all R_i^j of the derivation group of T_r.

The usual set operators can be applied to types; therefore

- Type T_1 is a subtype of type T_2 iff $T_1 \subset T_2$
- Type T_1 is a supertype of type T_2 iff $T_1 \supset T_2$
- Type T is the union type of types $T_1 \ldots T_i \ldots T_N$ iff $T = \bigcup_{i=1}^{N} T_i$.

Complex types hierarchies may consequently be derived.

Given a relation instance $[R_i]_s$ of a relation R_i, we call its first (second) domain $[R_i^1]_s$ ($[R_i^2]_s$) the set of all entities which are the first (second) entity of its pairs.

Of course, for each possible state: $[R_i^1]_s \subseteq R_i^1$ and $[R_i^2]_s \subseteq R_i^2$.

The global domain or population of a type T_r in a possible data base state is

$$GD_s(T_r) = \bigcup [R_i^j]_s, \text{ for all } R_i^j \text{ which belong to the derivation group of } T_r.$$

2.2 The model as an interpretation of users' Semantics

A Schema is a definition of all possible data base states. Through the definition of a Schema a user defines which data base states are allowed; the purpose of the algorithms which are presented in the next section is to guide the user in performing a schema definition which reflects as precisely as possible his intended semantics.

The properties which are defined in the basic sentences of a Schema (without considering additional constraints) are essentially those which express the possibility of a same entity to participate in different relations.

The following two cases can be defined:

a) Any entity which can participate in a projection of a relation R_1 can also participate in a projection of a relation R_2. This is expressed as a condition on types. For example, if the two projections are the first projection, then $R_1^1 \subseteq R_2^1$.

b) Any entity which exists in the domain of a relation instance of relation R_1 in a data base state must also exist in the domain of a relation R_2 in the same state. This is a condition on two domains in any possible state. For example, if the first domain is considered, then $[R_1^1]_s \subseteq [R_2^1]_s$ in any possible state.

The above two types of semantic properties are expressed in the Schema in the following way.

Each relation has a Relation Name and each type has a Type Name. For each relation the type names of its first and second projection are declared. The subset relationship between two types is declared. Then in case a) the condition that $R_1^1 \subseteq R_2^1$ is expressed by relations R_1 and R_2 being connected with the same type; the condition $R_1^1 \subset R_2^1$ is expressed by explicitly declaring that the type associated with R_1^1 is a subset of the type associated with R_2^1. Case b) is expressed by means of Domain Names. A domain name is given to the domain of a relation when needed, say to $[R_2^1]_s$. Then the relations, for which the property of case b) applies are referred to this domain.

When a user expresses the properties of data, he commonly uses the Relation, Type and Domain names. Of course, he is not aware of this fact. The model is a precise definition of his intended semantics. The algorithms that will be presented in Section 3 are a guide to define data semantics correctly. For instance, if a user says:

Sentence 1 : "Persons have Names"

Sentence 2 : "Persons are Managers"

Sentence 3 : "Managers manage Departments"

three interpretation are possible:

1) Persons, Names, Managers and Departments are types.
   Sentences 1, 2 and 3 express then three relations.

2) Persons, Names, Departments are types, Managers is a subtype of Persons. Sentence 2 expresses then a subtype relationship.
   This implies that there are Persons that cannot be Managers.

3) Persons, Names, Departments are types, Managers is a domain.
   In this case, a Manager is a Person, any Person can be a Manager; but only a Person which is a Manager can manage a Department. Sentence 2 expresses a relation producing a loop on the type Person.

2.3 Some extensions to the model

The following extensions to the basic model are useful:

- the concepts of path and of semantically equivalent paths
- the concept of external and internal types
- the concept of identifying set
- the concept of external invariant type.

Given a set of N relations $R_1, \ldots R_i \ldots R_N$, $N > 1$, such that $R_{i-1}^2 \equiv R_i^1$ and $R_i^2 \equiv R_{i+1}^1$ for all $1 < i < N$, a Path $(R_1 \ldots + R_i + \ldots R_N)$ is the derived relation which is built through the ordered composition of relation R_i with relation R_{i+1}.

An instance of a path p_i in a data base state, $[p_i]_s$, is the derived relation instance obtained through the composition of the instances of the relations contained in the path.

Two paths p_i and p_j are semantically equivalent if, in any possible correct state of the data base, their instances are equal, i.e. they contain the same pairs of entities.

Types may be classified as external or internal types: A type is external if it is possible to synthesize the information possessed by the entities belonging to the type through alphanumerical strings; on the opposite all types constituted by entities related to abstract concepts which cannot be expressed by an alphanumerical strings are internal types. This classification was introduced in /6,11, 12/, where additional features of internal types were detailed; observe that each domain of a type takes the same properties of the correspondent type. Internal types are meaningful only if they are related to at least one external type through a path; besides, internal types must be provided of an identifying set, while external types are self-identified by their mapping to alpnanumerical strings.

According to the definition given in /5/, the identifying set (IS) of a type or domain consists of all the relations which can be used to uniquely describe the entities of a type or domain in opposition to all other entities of the same type or domain.

We only recall that the identification of an entity is obtained through relations, and it may need

the and combination of two or more relations. The relations which constitute an IS have to satisfy a set of formal properties /see 13/.

Some external types have an additional property: assume that the relation R_i is given, where R_i^1 is an internal type and R_i^2 is an external type. Assume also that R_i is bijective and each entity of R_i^1, as long as it exists in the data base, is related to only one entity of R_i^2 (consequently, a pair of $[R_i]_s$ can only be inserted or deleted). In this case we say that R_i^2 is an external invariant type through R_i.

2.4 The Canonical Form

The goal of the view design methodology, which will be presented in Section 3, is to obtain a canonical form, starting from a description of the real world given through a set of elementary sentences.

Let us first recall the basic properties of the reference model:

a) Each relation appears only once in the data base; it can connect both types or domains. This condition implies that all the equivalent relations must be grouped during the design process.

b) Each type is defined as internal or external; each internal type is provided of an identifying set and each external invariant type is detected.

Two additional properties are required in order to obtain a canonical form:

c) each external type is connected only to one internal type. This implies that two external types cannot be connected via a relation.

d) Relations are switched to the proper type or domain, using the "minimal update propagation" criterium.

The "minimal update propagation" criterium must be explained.

Given two relations R_a and R_b, where $R_a^1 = R_b^1$, this criterium is used in order to decide the substitution of R_b by R_c, where $R_c^1 = R_a^2$, $R_c^2 = R_b^2$. Provided that the relation R_b is equivalent to the path $(R_a \rightarrow R_c)$ and that the relation R_c is equivalent to the path $(R_a^{-1} \rightarrow R_b)$, R_b will be substituted by R_c if the stability of R_c is greater than that of R_b. By stability of $R_b(R_c)$ we mean that an update of R_a does not imply an update of $R_b(R_c)$.

Observe also that this criterium is equivalent to the intuitive concept of attribution of a relation to the "best" type. The advantages of the above properties with respect to the design of a database will not be discussed here /see 13/; however, the view specification methodology which is described in the next section shows that the canonical form represents a rationalization of the initial user's view, since most of the hidden implications of the elementary input sentences are made evident, and semantic properties of data are formally expressed.

3. The Design Methodology

The design methodology will be described in the present section from an algorithmical view point, by illustrating the sequence of operations which manipulate the input elementary sentences given by the user.

Although this manipulation reflects both automatic and interactive processes, the formal notation adopted here will not stress the interaction aspects.

Four main stages in the proposed view design, each described in a subsection, may be distinguished:

stage 1) gathering and preliminary manipulation of elementary sentences

stage 2) identification of hierarchies among concepts

stage 3) definition of internal and external types and of domains in hierarchies; obtaining the canonical representation of the data base

stage 4) formal simplification of the canonical representation.

The human mind has the capability of aggregating entities within an abstract concept, and is accustomed to think in this way. It is therefore possible for the user to list relations between abstract concepts; this list of relations between concepts is the input to our algorithm, and it is obtained as a simple enumeration of relations and concepts, which can be performed in an arbitrary sequence by the user following his intuition. This list is not in direct correspondence to a correct model, because:

- many relations with the same semantic meaning may be present
- consequently, it is not clear whether the concepts correspond to types, subtypes or domains
- many semantic aspects may be ambiguous, incorrect or unexpressed.

In order to obtain a correct logical schema in canonical form, the modelling process transforms the input list of relations, and classifies concepts into types, subtypes, and domains. However this classification can be done only in stage 3, using the results of the analysis of the hierarchy of concepts which is done in stage 2. The canonical form, obtained as a result of stage 3, is formally simplified in stage 4.

3.1 Gathering and preliminary manipulation of elementary sentences

In order to illustrate the design methodology, we must introduce the main features of an elementary set of language sentences by means of which the user can describe the real world; we recall that in this paper we will deal with the collection of information on the structure of the intended data base, disregarding the specification of constraints; the problem of specifying constraints is treated in /14/.

The common structure of the considered sentences is in strict correspondence with the binary model, since in the sentences two concepts or sets of concepts are linked through a verbal construct. We can distinguish three main classes of elementary sentences:

(1) A "verbal construct" b

(2) A is b

(3) A has B

where lower-case letters are used for simple concepts, and upper case letters are used for sets of concepts.

The use of sets of concepts in the sentences allows the user's expression of partial aggregations; for instance, a user can express the sentence:

manager, employee has name, address

Construct (1) relates any concept a ∈ A to the concept b through a generic verbal construct, for instance:

manager, employee works in department

Construct (2), based on the fundamental verb "is", forces the designer to explicitly express hierarchies between concepts; as a consequence of the sentence "A is b", each concept a ∈ A is a subconcept of b; for instance:

manager, employee is dependent

In fact, hierarchies of concepts constitute a basic information to the design method.

Construct (3) specifies the property that every concept a ∈ A possesses a concept b ∈ B; for instance:

manager, employee has name, address

The two relations linking managers and employees to name and to address respectively have in effects a different semantic meaning; therefore the first (automatic) step of the design process decomposes the construct (3) into

(3') ∀ b ∈ B : A has b b

The previous sample sentence is then transformed into:

manager, employee has name name

manager, employee has address address

The construct (3) is introduced in order to avoid undesired homonyms and synonymies in relation names; for instance:

manager has address

employee lives at address

If both relations were written using the same verbal construct, further analysis would have been avoided in order to recognize their equivalence.

As a conclusion, the verbs "is" and "has" should be used whenever possible, as the first introduces a basic information, the second avoids multiple names for the same relation.

The set of concepts A for each of the three constructs is a candidate to constitute a supertype.

Let us investigate now in more depth the nature of each concept.

A concept a can be elementary, as "student" and "course", or it can be complex, as "student in a course".

As complex concepts correspond to the cartesian product of elementary concepts, the sign '*' will be utilized for their expression ("student * course").

The existence of complex concepts corresponds to the existence of composite primary keys /16/. This situation may be handled in a binary model introducing an internal type corresponding to the complex concept and generating new relations between it and the component concepts /6/. These relations will constitute the identifying set of the complex concept.

The set of elementary sentences between concepts may be represented through a graph, where each node corresponds to a concept and each arc corresponds to a verbal construct. Verbal construct of type (2) will be represented in the graph by special arcs (⇒). In fig. 1 a set of concepts and of sentences is represented; decomposition (3') has been performed.

ELEMENTARY SENTENCES:

employee has address,name,employee-n,manager
manager has name
person has name,address
employee-n works in department
manager is person
manager lives at address

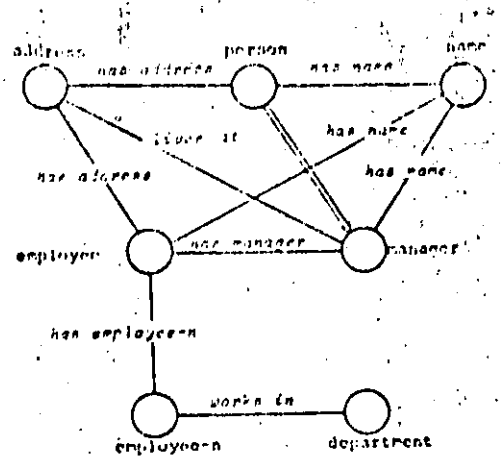


Fig. 1 - Set of concepts and sentences

All sentences that represent the data base in the user view constitute a set $R\langle R_i\ (A_i, b_i)\rangle$; they must now be analyzed in order to eliminate homonymies in relation names.

During this process, the semantic equivalence of all the relations must be tested; the relations having the same name are subdivided into groups, in each of which relations have the same semantic meaning, and each group is then forced to have a different name. All sentences with the same name

linking the sets A_i of concepts to the same concept b must be transformed into one sentence linking the set union of the sets A_i to b. In a formal way:

$$\forall R_i(A_i,b_i), R_j(A_j,b_j) : R_i \in R \text{ and } R_j \in R$$

if name (R_i) = name (R_j), then

(if semantic meaning (R_i) ≠ semantic meaning (R_j)

then change name (R_j) else

if $b_i = b_j$ then $(A_i = A_i \cup A_j$; delete $R_j)$)

EXAMPLE Manager manages activity
Employee manages activity
Servant manages hotel-room
↓
Manager, Employee manages activity
Servant keeps in order hotel-room

3.2 Identification of hierarchies among concepts

This second stage of the view design methodology operates upon the elementary sentences that have been obtained after executing the first stage, described in the previous subsection. It may be divided into two steps:

(a) testing the hierarchies of concepts that are explicitly defined and eliminating involved redundancies

(b) detecting hidden hierarchies and eliminating consequent redundancies.

Step (a). The test of the semantic correctness of a relation "$\bar{A}$ is b" consists of checking that all the relations applied to the superconcept $\bar{b}$ may be applied also to the concepts $\bar{a} \in \bar{A}$. In a formal way:

if $\exists(R_i(\bar{a},b_i)$ is meaningful, $\forall R_i(A_i,b_i): R_i \in R$

and $\bar{b} \in A_i)$ and $(R_i(A_i,\bar{a})$ is meaningful,

$\forall R_i(A_i,\bar{b}): R_i \in R) \forall \bar{a} \in \bar{A}$,

then $\bar{A}$ is $\bar{b}$ is correct

The procedure eliminates now redundant relations, i.e. relations linking a concept c to the concept $\bar{a}$ where c is also linked to the concept b with an equivalent relation and $\bar{a}$ is a subconcept of b.. For instance:

EXAMPLE 1 Employee is person
employee, person has name name
↓
employee is person
person has name name

EXAMPLE 2 Person has department department
employee is person
employee works in department
↓
person has department department
employee is person

Examples 1 and 2 correspond to different operations which may occur in the modelling process.

In the first situation, where the superconcept $\bar{b}$ and at least one of his subconcepts $\bar{a}$ are present in a same relation, no interaction is necessary; the operation to be performed may be expressed as:

$$\forall R_i(A_i,b_i) : R_i \in R \text{ and } \bar{b} \in A_i \text{ then } A_i = A_i - \bar{A}$$

In the second situation synonymies within the relations $R_i \in R$ must be detected; this implies the detection of the semantic equivalence between relations, that can be obtained only through interactive requests to a human operator. It is clear that the synonymies analysis is unpracticable over the whole set R of relations in large data bases, as the complexity of the analysis is quadratic with the number of relations. The algorithm therefore limits this analysis to the set of relations linking a concept and the corresponding superconcept to a generic common concept. Most synonymies are in this way detected with limited efforts. When a synonymy is discovered, we assign a common name to both the involved relations, and redundancies are furtherly eliminated. In a formal way:

$$\forall R_i(A_i,b_i), R_j(A_j,b_j) :$$

$(R_i \in R$ and $R_j \in R)$ and $(\bar{a} \in A_j$ and $\bar{b} \in A_i)$

and $(b_i \neq b_j)$ and (semantic meaning (R_i) =

semantic meaning (R_j))

then name (R_j) = name (R_i); $A_j = A_j - \bar{A}$;

if $A_j = \emptyset$ then delete (R_j)

The final operation performed in step a) for each relation "$\bar{A}$ is $\bar{b}$" consists of transferring to a new superconcept $\bar{b}'$, where $\bar{b}' \subseteq \bar{b}$, equivalent relations linking two or more concepts $\bar{a} \in \bar{A}$ to a common concept.

This operation leads to the creation of hierarchies of concepts; it may be formalized as follows:

$$\forall \bar{A} \text{ is } \bar{b}, \forall R_i(A_i,b_i), R_j(A_j,b_j) :$$

$(b_i = b_j)$ and $(\bar{a}_k \in A_i$ and $\bar{a}_\ell \in A_j)$ and

(semantic meaning (R_i) = semantic meaning (R_j))

then delete R_i, R_j;

create $R_i(\bar{b}', b_i)$;

if $(\bar{b}' \neq \bar{b})$ then create $\bar{b}'$ is $\bar{b}$;

create $\bar{a}_k, \bar{a}_\ell$ is $\bar{b}'$;

$\bar{A} = \bar{A} - (\bar{a}_k, \bar{a}_\ell)$

We will explain the essence of this operation referring to the example shown in fig. 2.

Given the situation depicted in fig. 2(a), which corresponds to the set of sentences R, we can see that relation R2 is extended to all the subconcepts

of $\bar{b}$; since $\bar{b}$ is the union concept of a1, a2, a3, then relation R2 can be extended to $\bar{b}$ and removed from a1, a2, a3.

Relation R3 constitutes a quite different case, as it does not insist upon all entities of the superconcept $\bar{b}$, but it affects only concepts a1 and a2. It is then necessary to join these concepts into one concept $\bar{b}'$, to which relation R3 must be applied, removing it from a1 and a2. In fig. 2(b) the results of this analysis are shown, together with the set R' of sentences so generated.

Step (b). The search of hidden hierarchies must now be performed; it seems reasonable to limit the analysis to the sets of concepts of each relation, as the process of grouping relations with the same meaning has already been performed in the first stage of the methodology.

For all the relations whose set of concepts A is constituted by more than one element, we must define a superconcept $\bar{b}'$ equivalent to A. If $\bar{b}'$ represents the same concept of an $a_i \in A$, i.e. $a_i \equiv \bar{b}'$, then a hierarchy is discovered, and without introducing any new concept, we can define the new relation:

$$A - (a_i) \text{ is } a_i$$

Otherwise we will have:

$$A \text{ is } \bar{b}'$$

In a formal way:

$$\forall R_i (A_i, b_i) : R_i \in R$$

then if ($A_i = a_i$ and $a_i \in A_i$)

then create "$A_i - (a_i)$ is a_i"

else ($b_i = A_i$;

create "A_i is b_i")

The possibility that $\bar{b}'$ is a subconcept of $\bar{b}$, where $\bar{b}$ is a superconcept of a generic $a \in A$, must be tested. If possible, the relation "$\bar{b}'$ is $\bar{b}$" must be inserted, while the relation "a is $\bar{b}$" must be deleted.

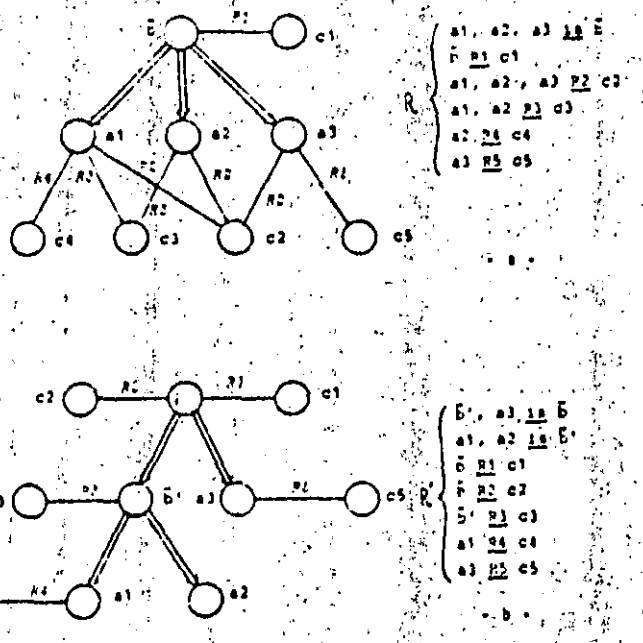
The elimination of redundancies involved in the detected hierarchies can be obtained through the operations already described for Step (a).

This process is shown in fig. 3. Starting from Fig. 3a, we analyze relation R2, which involves concepts a2 and a3; the concept $\bar{b}'$ (see fig. 3b) represents the union of concepts a2 and a3 and it is equivalent neither to a2 nor to a3.

Fig. 3c shows how the hierarchy is transformed if the information that "$\bar{b}'$ is $\bar{b}$" is interactively obtained.

3.3 Deriving the canonical representation of the data base

A preliminary test must be performed upon the hierarchies before starting the process of deriving the



where for instance:

$\bar{b}$: person	c1 : name	R1 : has name
$\bar{b}'$: manager	c2 : address	R2 : has address
a1: sr.mgr.	c3 : project	R3 : is chief of
a2: jr.mgr.	c4 : department	R4 : manages
a3: employee	c5 : employee-n	R5 : has employee-n

Fig. 2 - Creation of hierarchies of concepts

canonical form.

Homogeneity between concepts belonging to the same hierarchy of concepts must be checked. Suppose that the following set of sentences is given :

employee number, manager is person

In this case, while employee number may be synthesized through a number, manager cannot be mapped to an alphanumeric string. As they are not homogeneous, we must divide the concept employee number into two concepts, the first corresponding to the abstract concept employee, and the second corresponding to the number of the employee, and relate then these concepts through a relation. The result of this operation is:

employee, manager is person
employee has number employee-n

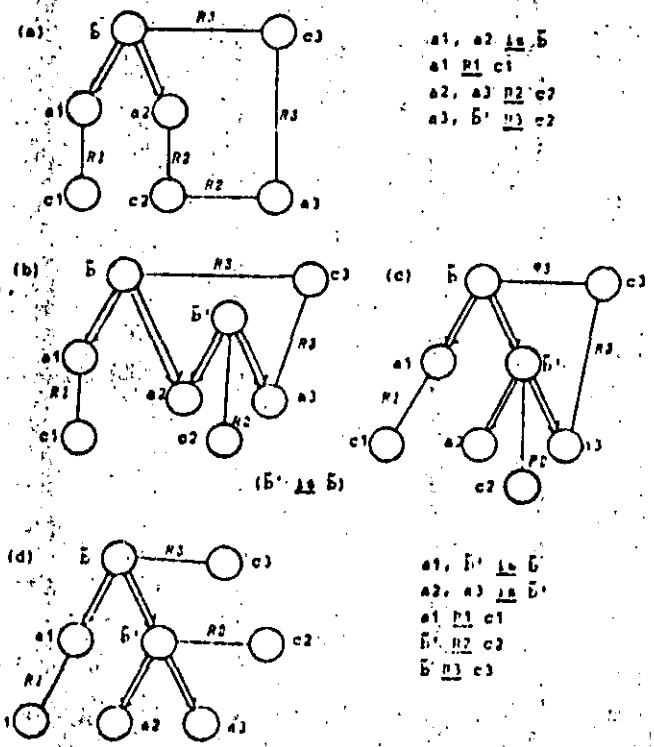
Referring to the properties of domains and types discussed in section 2.2, the hierarchies of concepts must also be analyzed in order to detect type-domain or domain-domain dependencies.

Consider the following examples:

(1) male is person

(2) manager is person

(3) senior manager is manager



where, for instance:

Б : person	c1 : employee-n	R1 : has employee-n
Б': manager	c2 : project	R2 : is chief of
a1: employee	c3 : name	R3 : has name
a2: sr. mgr.		
a3: jr. mgr.		

Fig. 3 - Research of hidden hierarchies and further elimination of redundancies

and assume that

- not all persons are males
- each person may become manager
- each manager may become senior manager

According to the previous definitions we deduce that person and male are types, (where type male $\subset$ type person), while the concepts manager and senior manager must be considered domains of person, because they may include any entity of person.

Since in the previous examples the verbal construct is represents three different facts, it is correct to distinguish them as follows:

(1) male is subtype of person

(2) manager is domain of person

(3) senior manager is subdomain of manager

The first example represents a type-type (T-T) dependency, the second a type-domain (T-D) dependency, and the third a domain-domain (D-D) dependency.

We can give a formal rule for the classification of dependencies :

Given the sentence

$$\alpha : C_a \text{ is } C_b$$

let $S_a(S_b)$ be the set of all sentencies of the concept C_a (C_b):

if $S_a \subset S_b$

then α is a T-T dependency

else (comment : $S_a \equiv S_b$;

if (C_b has not superconcepts or is related to them via T-T dependency)

then α is a T-D dependency

else(α is a D-D dependency;

comment: C_b has a superconcept and is related to it via a T-D or D-D dependency))

All concepts that are present in hierachies can therefore be defined as types or domains through the previous analysis; the remaining concepts may now be considered as types, because they now possess the required properties.

All sentences, that are obtained through formanipulation and interactive testing from the original set R of user's sentences, may now be regarded as binary relations compatible with the basic model; sentences linking a type T_1 to a domain D_2 represent only a linguistic construct expressing both the relation between T_1 and the type corresponding to D_2 and the constraints on the projection of any instance of the relation.

Thus a logical schema is obtained which on one hand is correct with respect to the basic model, and on the other hand is clearly derived from the original sentences of the user.

The canonical form is obtained in this third stage of the methodology. Three steps are considered:

(a) definition of internal types

(b) minimal update propagation analysis

(c) identifying set definition

The first two steps are repeated for each type that is linked to at least two other types. A type T is intended to be linked to a type T_1 also when a supertype of T is linked to the type T_1. The third step is repeated for each internal type. The data base of Fig. 4 illustrates the process.

Step (a). Given a generic type T, linked at least to two other types, the designer evaluates if it may be an internal type, i.e., if it is not possible a mapping of T to alphanumerical strings. If T cannot be an internal type, it must be splitted into two types T' and T", where T' is internal and T" is external; a relation R (T', T") must be introduced, and the invariance property is given to T" with respect to T'; finally all

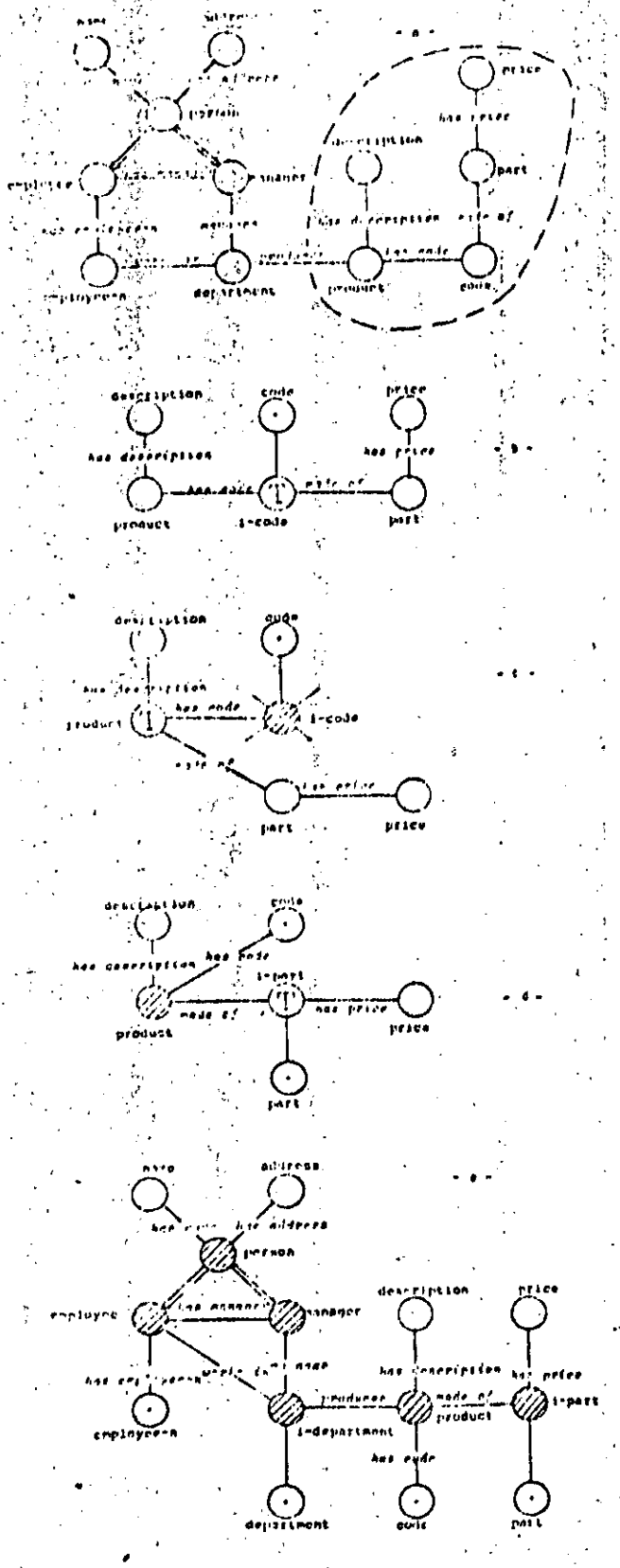


Fig. 4 - Derivation of the canonical representation

relations R_i (T, T_i) are transformed in $R_i(T', T_i)$.

These operations have been performed for instance on the type "part" of fig. 4a,e; note that in the figure invariant external types are denoted through a dot in the middle of the correspondent node of the graph, while internal types are dashed.

Step (b). The analysis of the minimal update propagation of relations is now performed; all the relations R_k departing from all types T_i linked to the type T, are considered.

This analysis is avoided if T_i and T belong to the same hierarchy, as the optimal allocation of relations in this situation has already been obtained in a previous stage of the design methodology. The designer must interactively evaluate, for each relation R_k, if R_k (T, T_k) is 'better', according to the minimal update propagation criterium, than the existing relation R_k (T_i, T_k). If this condition is verified, then R_k (T_i, T_k) is substituted by $R_k(T, T_k)$; all relations starting from T_k must be further-ly analyzed.

The possibility of suppressing T_i, if it shows to be now unnecessary, must also be analyzed: this may occur only if two relations depart from T_i linking it to T and to a generic T'. An interaction with the designer may lead to the creation of a direct link between T and T', suppressing T_i. Figure 4c shows such a situation; in fact the type 'i-code' has been suppressed.

The above two steps are described in figures 4b, 4c and 4d for the subset of the data base that is enclosed in a dotted circle in Fig. 4a. A non optimal sequence of types T has been performed in order to show how the final result is independent of the order of execution of the analysis.

When these steps have been accomplished for all the possible types, the remaining undefined types must be analyzed: They are candidate to be external types, but this hypothesis must be interactively confirmed. The designer must also declare the property of invariance for some of the external types. In Fig. 4e the type "employee-n" has such property with respect to the type "employee".

Step (c). The definition of the identifying sets must be finally performed. We remember that external types are self-identifying, while at least one identifying set must be provided for each internal type. The identification process must avoid loops, i.e. chains of internal types where no member of the chain is at least indirectly identified by external types.

This result may be obtained interactively looking for an identifying set of an internal type only among relations linking it to either external or already identified internal types. The process is repeated until no new identifying sets can be defined; if this procedure terminates and some internal types have not been identified, a lack of information is revealed and the designer is required to define new external types and/or new relations.

In Fig. 4e the following identifying sets can be defined for person, manager, and employee:

IS (person) = (has name A has address)
IS (manager) = (has name A has address)
IS (employee) = (has name A has address),(has number)

The final result of this analysis is the canonical form derived from the input sentences, which represent the data base in the user's view.

3.4. Formal simplification of the canonical form

The canonical form can be simplified, eliminating internal types which are related to external invariant types. This is obtained by substituting the internal types with the corresponding invariant external types.
The feature of minimal update propagation, which is a requirement for the canonical model, is saved in the new derived structure, due to the properties of external invariant types. This operation must be suspended if the internal type belongs to a hierarchical structure, in order to respect the homogeneity of the hierarchy.

The fourth stage of the design methodology is shown in Fig. 5, where the rationalization of the model has been extended to the maximum degree. All external invariant types, save "employee-n", have been substituted by the corresponding internal types.

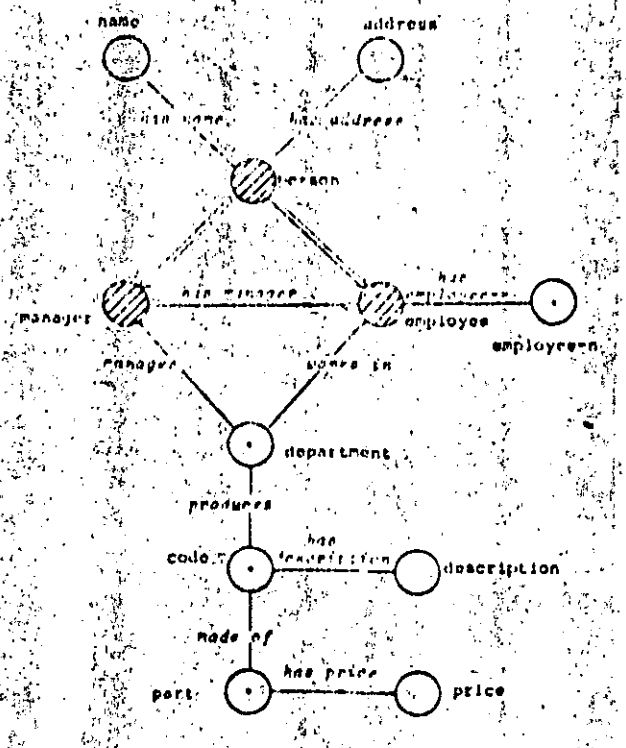


Fig. 5 - Simplified final canonical form

Conclusions

This paper has described an interactive methodology for designing user's views, starting from the elementary sentences that specify the requirements of the application to be modelled.

The methodology includes algorithms for checking the formal correctness of the model that is built, and for detecting inconsistencies, ambiguities and redundancies.

An important feature of the methodology is the generation of a canonical formal representation of the user's view, that eliminates the arbitrariness of the perspective of different designers in modelling the same universe of discourse, at least as far as syntactic aspects are concerned.

A particular binary data model has been adopted to represent the views; however, the concepts and the algorithms of the methodology could be applied to other different data models.

Future developments of this research will include the extension of the interactive design methodology to the subsequent processes of integrating the several and possibly conflicting user's views into one data model that represents a community view of the real world, and of mapping the canonical model to the logical structures available in existing data base system environments.

References

[1] Yao, S.B., Navathe, S.B. and Weldon, J.L.: "An Integrated Approach to Logical Data Base Design", Proceedings of NYU Data Base Symposium, New York, 1978.

[2] Navathe, S.B. and Schkolnick: "View Representation in Logical Data Base Design", Proceedings ACM-SIGMOD Conference on Management of Data, 1978.

[3] Chen, P.P.: "The Entity-Relationship Model: toward a Unified View of Data", ACM Transactions on Data Base Systems, vol.1, n.1, 1976.

[4] Roussopoulos, N. and Mylopoulos, J.: "Using Semantic Networks for Data Base Management", Proc. of the International Conference on Very Large Data Bases, ACM, (Ed. Berr, D.S.), 1975.

[5] Biller, H. and Neuhold, E.G.: "Semantics of Data Bases: the Semantics of Data Models", Information Systems, vol. 3, n. 1, 1978.

[6] Bracchi, G., Paolini, P. and Pelagatti, G.: "Binary Logical Associations in Data Modelling", in Modelling in Data Base Management Systems, (Ed. Nijssen, G.M.), North-Holland, 1976.

[7] Smith, J.M. and Smith, D.C.P.: "Data Base Abstractions: Aggregation", Communications of the ACM, Vol. 20, n. 6, 1977.

Nijssen, G.M. : "Current Issues in Conceptual Schema Concepts", in Architecture and Modelling in Data Base Management Systems, Proc. IFIP TC-2 Working Conference, North-Holland, 1977.

Kaver, N. and Hubbard, G.U. : "Automated Logical Data Base Design: Concepts and Applications", IBM Systems J., vol. 16, n. 3, 1977.

Mylopoulous, J., Bernstein, P.A. and Wong, H.K.T. : "A Language Facility for Designing Interactive Data Bases", Harvard University, Cambridge, MA 02138.

Pelagatti, G., Paolini, P. and Bracchi, G. : "Mapping External Views to a Common Data Model", Information Systems, vol. 3, n. 2, 1978.

Pelagatti, G., Paolini, P. and Bracchi, G. : "Mappings in Data Base Systems", Proc. 1977 IFIP Congress, (Ed. Gilchrist, B.), North-Holland, 1977.

Baldissera, C., Ceri, S., Pelagatti, G. and Bracchi, G. : "A structured methodology for designing static and dynamic aspects of data base applications", Istituto di Elettrotecnica ed Elettronica, Politecnico di Milano, Int. Rep. N. 10, 1979.

Bracchi, G., Furtado, A. and Pelagatti, G. : "Constraint Specification in Evolutionary Data Base Design", Proc. IFIP Working Conference on Formal Methods and Practical Tools for Information Systems Design, (Ed. Schneider, H.J.) North-Holland, 1979.

Abrial, J.R. : "Data Semantics", in Data Base Management, (Ed. Klimbie, J.W. and Koffeman, K.L.), North-Holland, 1974.

Date, C.J. : "An Introduction to Data Base Systems", Addison Wesley, 1977.

DATABASE CONCURRENT PROCESSOR

Emilio LUQUE*, José J. RUZ**, Ana RIPOLL* and Alfredo BAUTISTA**

* Dept. Electricidad y Electrónica, Fac. Ciencias, Univ. Autónoma Barcelona, Spain. ** Dept. Informática y Automática, Fac. Físicas, Univ. Complutense, Madrid, Spain.

The organization of an autonomous processor supporting database management functions is presented. The DataBase Concurrent Processor (DBCP) can be thought as a back-end data management machine of a general purpose host computer: it supports relational data model directly in hardware, and is able to run concurrently a number of programs written with a relationally complete instruction set. DBCP is composed of a parallel organization of cells and a Coordination Unit (CU). Each cell supports and processes tuples of an unique relation and consists of a special purpose microprocessor and a circulating serial memory. Along a full memory revolution each microprocessor accomplishes an access on its own memory block while, at the same time, the CU transmits all necessary information to the access to be made in the next revolution. CU is allowed by microprocessor functional independence to organize different concurrency control strategies, trying to make maximum use of cellular organization processing capability. This concurrent processing capability at backend level fits better to the database system multiuser nature (shared resources) and consequently higher capabilities and performance rates are hoped to be achieved.

Introduction

In the latest years, there has been a trend to build processor architectures problem-oriented, decreasing as far as possible intermediate software. Recent appearance of so-called "data-base machines" is a clear proof of the above statement. The objectives are to implement many of the traditional software database management functions in hardware in order to increase systems performance and capability. The main reason of this approach is found in existing mismatch between conventional computer architecture conceived for numeric applications and non-numeric nature of problems raised in present database systems. This mismatch coupled with the fact that software costs continue to rise, while hardware costs continue to decrease makes it imperative to take maximum advantage of recent developments achieved in hardware architecture and technology to find new solutions to the database management problem.

Within this research line, a DataBase Concurrent Processor (DBCP) is being developed at the Department of Computer Science and Automatic Control of the Complutense University in Madrid. The project, presently in phase of simulation, gathers the foremost contributions appeared in previous processors embodying in addition architectural features enabling inter and intra-query concurrent processing. This concurrency capability makes DBCP specially useful to be incorporated to on-line and multi-user database systems.

The approach of decentralizing database management functions of main computer was initiated by Canaday with XDMS system[1]. On this approach DBCP is concerned only with DBM functions. Therefore, DBCP organization is pointed to optimize such functions.

As many others proposed database back-end processor architectures -RAP[2], RARES[3], CASSM[4], RAPID[5]- we have used for DBCP the distributed logic principle as alternative to high costs of otherwise desirable associative memories. This principle was applied firstly by Slotnic[6] who coupled a logic element to each track of a conventional mass memory (logic-per-track). Bearing in mind present and middle-dated forseenable mass memory architecture, this alternative allows us to achieve a pseudo-associative memory device with capacity enough as required in DBM systems.

The logic element in DBCP will be a non-numeric microprocessor purposely designed to perform database elemental functions; basically, context searching on its associated memory block.

Conventional systems suffer overhead caused by processing and maintenance of auxiliary data structures which enable mapping data logic model upon support physical structure. This overhead to be avoided, we have implemented the data logic directly on memory in DBCP. RARES[3], RAP[2] and DIRECT[7] systems have employed this approach, implementing data relational model directly on memory. CASSM system[4] has employed an intermediate model among those currently used: relational, network and hierarchical.

Processor cellular organization avoids passing large amounts of data between mass memory and central processor. Processing capability is distributed around the whole memory in such a way that data are processed there where they lie. However, current systems using this principle do not take advantage of all processing capability allowed by high parallelism of cellular organization. For instance, all cells in RAP system[2] perform the same function on their associated memory blocks along a revolution. Since every cell holds tuples of a unique relation and RAP instructions refer to one or two relations, only the cell set supporting the relation referenced by the instruction will be processing data, the remainder cells will be idle.

Trying to make maximum utilization of the processing potential inherent in the parallel arrangement of cells, we have designed DBCP pretending to keep cell microprocessor idle the least time possi-

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

DISEÑO DE BASES DE DATOS

LOGOS AND THE SOFTWARE ENGINEER

MAYO, 1985

Palacio de Minería Calle de Tacuba 5 primer piso Deleg. Cuauhtémoc 06000 México, D.F. Tel.: 521-40-20 Apdo. Postal M-2285

LOGOS and the software engineer*

by C. W. ROSE

Case Western Reserve University
Cleveland, Ohio

INTRODUCTION

Most of us consider a well-engineered product to be one which is structurally sound; which communicates with its environment in a predictable, well-disciplined manner; which has been thoroughly tested; and which is reliable and easily maintained. In any engineering field, the structural philosophy, design disciplines, and checkout methods which yield such a product are called "good engineering practices." Software engineering is the application of good engineering practice to the design, implementation and final checkout of large programs. The result of effective software engineering should be:

(1) The production of a correct program (certifiable)
(2) The availability of means of efficiently determining the correctness of a program (certification)
(3) The ability to modify a program so that recertification is possible.[1]

The goal is to organize complexities, master multitude, and avoid its bastard chaos as effectively as possible.[2] However, unlike many types of engineers, the software engineer has had few tools, either for implementation or analysis, with which to accomplish his task.

Many of the problems in operating systems which occurred during the mid-sixties can be traced to an inability to enforce the design disciplines indicated by good engineering practices, or to determine after the fact that they had been applied. In some cases the problems appeared years after the system was in the field. Higher level implementation languages for software relieve many trivial coding errors and deal effectively with the problem of storage allocation; however, they do little in reducing the major problem of complexity—inter-module communication, software/hardware interface conflicts, and mishandling of real and apparent concurrency within the hardware/software system. This is more obvious when one remembers that most large design efforts are multiperson, and that software modules and hardware designed by many people must communicate properly at the many interfaces.

The hardware designer is somewhat better off since he can call upon switching algebras, flow table analysis[3] and register transfer languages[4,5] to aid him in the design. Unfortunately, these tools are not amenable to the design and analysis of very large systems, and the designer soon learns to modularize his system and to apply his techniques to several modules of manageable size. It is at the interfaces of these modules though, that problems equivalent to those in software arise.

The net result of this inability to systematically deal large scale complexities has been the late delivery of expensive and buggy computer systems. This is not to suggest that the several successful structural approaches to systematic operating system design[6,7] are insignificant, but rather that the difficulty of enforcing their requisite structural and communication disciplines becomes very great as the size of the target system increases.

Hardware engineers encountered this problem of complexity very early in terms of implementation, and responded by developing computer-aided design (CAD) systems for logic diagram production, package placement, wire routing and mask generation, and simulation and test generation.[8] Many other engineering disciplines have also turned to the computer to help deal with the complexities of large systems.[9] It is ironic, however, that the computer, which has great analytic capability, doesn't often forget details, and can enforce structural and communications disciplines by syntactical analysis, has not, to date, been applied to the conceptual and detailed design of computer systems.

*This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Department of the Army under Contract No. AMSEL-[illegible]-2(CAE).

Project LOGOS was begun in 1968 at Case Western Reserve University to exploit these capabilities by creating a computer-aided design environment for both the hardware and software of large-scale computer systems. An integrated hardware/software design system was chosen because mismatches at that particular interface in a computer system are the most costly and time-consuming to correct. The goals of LOGOS can be simply stated: the creation of a multi-designer environment in which computer system designers can define a system in which a high degree of parallelism or concurrency exists, verify its logical and functional consistency, evaluate its expected performance before implementation, and finally implement the hardware and generate the code for the software. Inherent in any CAD system is a philosophy of target systems structure and an associated representation system which both embodies that philosophy and has a well-defined syntax and semantics. It would be helpful here to briefly describe both to set the stage for a discussion of LOGOS' contributions to the software engineer.

A LAYERED VIEW OF SYSTEM STRUCTURE

From a user's viewpoint, a computer system presents an environment to each user which is characterized by a collection of service facilities.[10] Each facility may be activated and directed according to a well-defined communications discipline. Since users do not, in general, act in coordination, the system facilities must cope with multiple and asynchronous requests for services.

Response to a request activates the facility, an instance of which we shall call a task, and the method of handling multiple requests depends upon the nature of the facility. A single-user facility would queue all requests in excess of one, while a limited resource facility such as a magnetic tape controller with six tape drives would allow six concurrent activations before queueing requests. On the other hand, a fully reentrant software procedure would have no limit to its activations although exhaustion of some other resource such as core memory would impose a limit externally.

Users of facilities very often do not care about how a facility is implemented internally, but rather how it interacts with its environment. This concern with the input/output function of a facility is the "external" or "primitive" view. Conversely, a user and, in particular, a designer may need to know the details of implementation as well as the I/O function of a facility. This is the "internal" view.

A facilities approach to viewing systems immediately gives rise to a hierarchical structure. Many facilities in a system provide essentially identical services, or equivalently, have identical subtasks. These subtasks could be viewed as instances of activation of separate facilities shared among those requiring the particular services. The most primitive shared subtasks in a software system are the machine instructions. By the same token, however, the reading of a text file appears primitive to a compiler using the file system facility, although an internal view of the file system shows that the read file operation is quite complex and uses other shared facilities such as the disk channel.

It is natural, therefore, to structure a system as a partially ordered hierarchy of layers, the highest layer being the interface with the system users, and the lowest, the system primitives. A system primitive for a software system might be a machine language instruction or a library subroutine, while a hardware primitive might be a NAND gate or a four-bit MSI adder. A facility on a lower layer may be activated by a task of a facility existing on a higher layer. Its tasks may, in turn, activate facilities on still lower layers.

Between any two layers, there is an interface partitioning the system into facilities below the interface and users of those facilities above it. Users above present an environment of service requests and arrival rates, while facilities below present an environment of service available and service times.

The ability to "collapse" or look at a facility as a primitive suggests that consistency analysis of a facility could be done by exposing the internal structures, analyzing it, collapsing it, and then analyzing its interactions with its environment as a primitive. This is the only practical way of analyzing large systems, and the representation which accompanies this philosophy allows just that.

Implied in all of this is the existence of an interfacility communications discipline for both data and control. Several might be defined such as Dijkstra's P-V discipline[11] or Multics' mailbox scheme.[12] What is important is that whichever one is chosen, it must be enforced, or the layered model will break down, and the analytic capability afforded by this scheme will be lost.

A facility in general consists of four elements:

(i) Resources of one or more types which may be required by the facility subtasks.

(ii) An enclosing control which determines, based upon resource availability, if a subtask should be activated or if the request should be queued or dismissed.

(iii) A set of algorithms defining the subtasks. Algorithms are called activities; instances of their activation are called processes.

(iv) An interpreter which accepts user directives and determines appropriate action.

A facility need not have all of these elements. A wholly software facility would not have local resources, while a storage allocator would control a resource but have no set of algorithms to be selected by a user.

This philosophy of system structure can be applied to both hardware and software. It is consistent with the structural approach to proving program correctness[13,2,14,15] which is to force the structure of the program text (or representation) to correlate strongly with the structure of the actual computation, thus allowing analysis of the computation by analyzing the structure of the representation by stepwise decomposition.

THE LOGOS REPRESENTATION SYSTEM

The central part of any CAD system is its *representation system* which consists of the design data base in which the description of the target system is accrued, the external representation of this design information, and the translators between the external and internal form. The representation system must satisfy several global constraints.

First, the representation must be sufficiently general to describe all interesting and desirable objects in the set of design objects, while at the same time, it must be sufficiently specific to allow algorithmic consistency and performance analysis. Second, the internal representation must be decomposable into elements which may be implemented directly. Finally the designer must be comfortable with the external representation and the constraints it places upon his freedom of expression.

In the case of LOGOS, the target objects are facilities, which can be described by a number of algorithms implemented in either hardware, software, or a combination of both. Therefore, the representation must be suitable for describing both and must yield the target system implementation directly.

The representation must be consistent with the hierarchical, layered view of system structure. It must, therefore, be declarative in that it must express both the structure and function of the target facility to allow algorithmic consistency and performance analysis. It must allow the design to be described in multiple levels of abstraction to accommodate the primitive and internal views of facilities required for stepwise analysis. This feature is especially important to designers since they tend to work "around" in a design rather than in a strictly top-down or bottom-up manner.

Finally, since many of today's computer systems and those proposed for the next generation contain parallel processing capabilities, the representation must allow the specification of parallelism or concurrency in a natural way and be capable of analyzing its effect on consistency and performance.

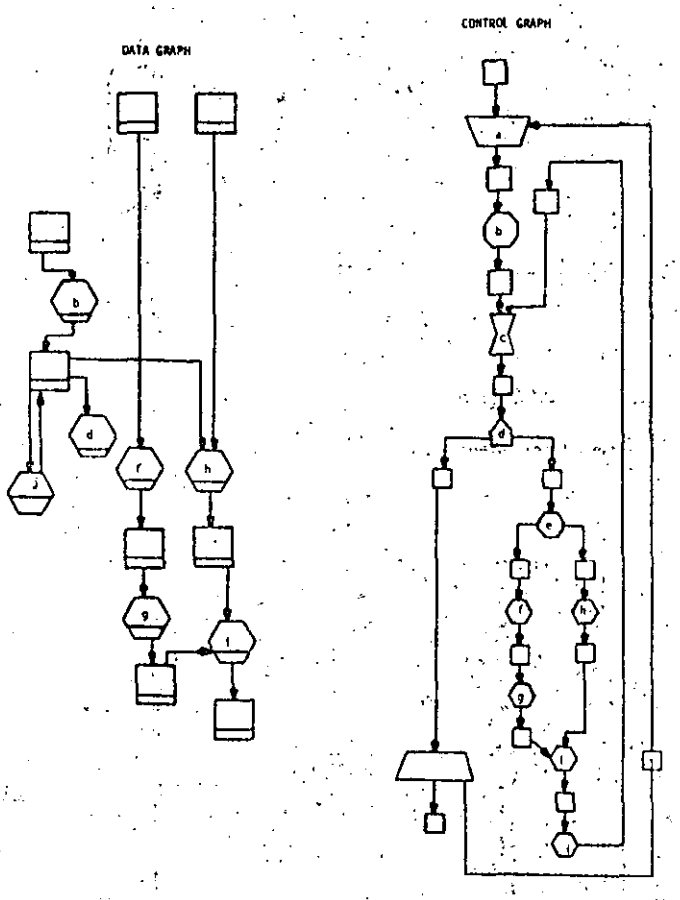


Figure 1—Example of an activity.

LOGOS chose a graph-theoretic system of representation which satisfies the above constraints. The system is a synthesis and extension of valuable work done by Petri,[16] Karp and Miller,[17] Holt,[18] Luconi,[19] and others. The extensions were required because (1) LOGOS deals with very large systems and must localize analyses, (2) little work had been done in the representation and analysis of data structures in graph models, and (3) because LOGOS must actually implement rather than merely analyze the target algorithms or systems.

A complete treatment of the representation may be found in References 20 and 21. Briefly, the representation of an algorithm consists of a pair of directed graphs. The data graph (DG) defines the algorithm data structures and the transformations upon them, while the associated control graph (CG) sequences the transformations and defines the control flow. The schema formed by a CG-DG pair is called an activity and will be seen

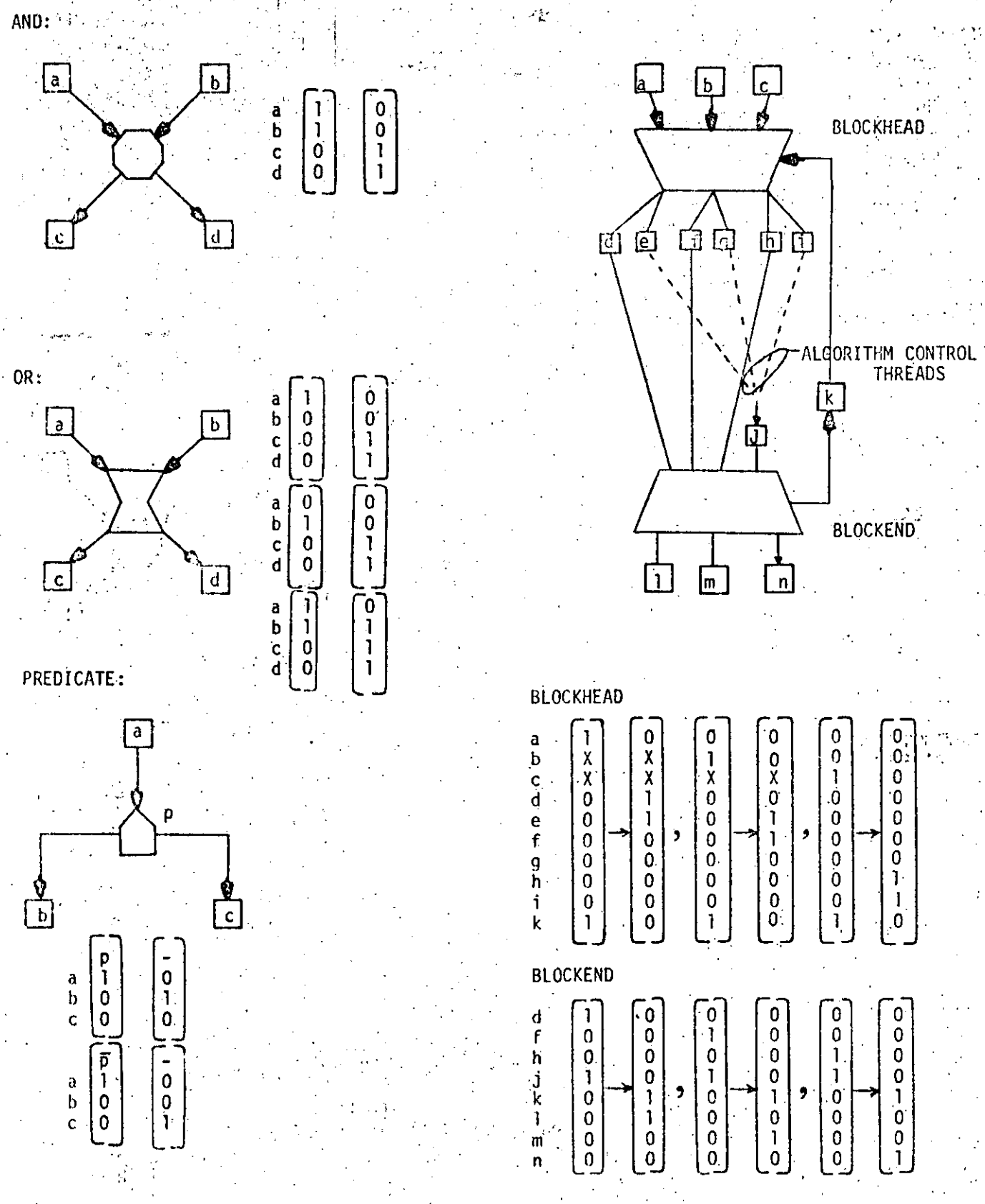


Figure 2—LOGOS atomic control operators

to be the static template of a task. Figure 1 is an example of an activity.

The CG consists of two node types: the squares are control variables or c-cells, and the remaining nodes are control operators. Cells must be connected only to operators by directed arcs and vice versa. There are several types of control operators as denoted by the different shapes in Figure 1. Each type of control operator has an associated enabling or transfer function defined on its input and output c-cells.

The DG consists of cells (squares) which represent the information structures of the activity, and data operators which perform the transformation upon them (e.g., Add, Move, Integrate, etc.). Each data operator is associated with a unique control operator which determines when the data transformation may take place. The initiation of a control operator initiates the associated data operator which then reads its input cells (data structures), performs the data function, and writes the results into its output cells. Upon writing, the data operator communicates to the control operator that it has terminated, and the control operator terminates by alterings its c-cells appropriately.

The flow of control in the CG is determined by the values in the c-cells and the nature of the c-operators to which they are connected. The c-operators are defined so that asynchronous or synchronous control and data flow can be represented. The atomic or first level control operators are shown in Figure 2 together with their transfer functions in vector form.

The AND operator of Figure 2 is used to resynchronize parallel control-paths and functions analogously to an AND gate in hardware. The OR operator is asymmetric in that if both of its input c-cells contain 1's, the initiation of the operator preserves the 1 in the second c-cell. It will then reinitiate as soon as possible. This "conservation" of 1's is required to insure determinacy, a property of consistent systems with concurrency which will be discussed later. The OR operator is analogous to an OR gate in all other ways.

The PRED operator is the interface between data values and the control flow in the CG. It is a data dependent control branch whose associated data operator performs a test on its input d-cells. The result of this test conditions the branch in the control.

The blockhead (BH) and Blockend (BE) operators are paired to delineate an activity and form the enclosing control for the facility task being represented. The control algorithms must perform the following functions:

(i) arbitrate access to the facility
(ii) provide a communication discipline between the facility and its users
(iii) define the number of concurrent users which may be served by the facility.

The BH/BE pair described in Figure 2 act as a P-V pair. The arbitration algorithm shown is a fixed left-to-right priority, but round-robin and other disciplines have been implemented also. The BH and BE communicate primarily via the feedback c-cell, which initially contains, if it is present, the number of concurrent activations possible. All control flow is restricted to enter and leave the activity via the BH/BE pair with the exception of nested subroutines or procedure calls (i.e., calls upon activities on the same layer of the system) which are controlled by Call/Return operator pairs constructed from a common control primitive.

These control operators may all be constructed from a common primitive control operator whose definition is logically complete. This primitive operator may be realized directly in hardware, but for the purposes of the software engineer, it is sufficient to state that other, higher-level control operators may be constructed from the primitives and placed in a macrolibrary.

The activity of Figure 1 is shown in Figure 3 with interpretations placed upon the data structures and data operators of the DG (these are informal interpretations; a formal syntax will be introduced later). The activity is the representation of an ALGOL 60 FOR statement with a parallel DO ⟨statement⟩ part. When the task is activated, the stepping variable is initialized to 5, and the loop head is passed. Note that there is no data operator associated with the loophead OR. If the predicate is false, the parallel DO ⟨statement⟩ is executed which allows the sequence of data functions ϵf to be time independent of h. The threads are resynchronized at i whose data operator uses common results. n is decremented and the loop is re-entered. Thus we have represented:

FOR N=5 Step−1 until Ø DO ⟨statement⟩

The 1 in the feedback c-cell indicates that the activity may be initiated only once before terminating.

The nesting of activities on a layer allows the imposition of an ALGOL-like block structure upon the representation. If the activity in Figure 3 were in a block structured environment, data cells A, B, and C might be global to the block while n, m, p, 5, and r are local.

Thus, a collapsed or primitive view of the activity is that of a single control-data operator pair as shown in Figure 4. For most types of analysis performed by LOGOS, the local structure of an activity is analyzed in its internal or expanded form, and the activity is then collapsed. All further interactions with its global environment are analyzed in the collapsed form. In this

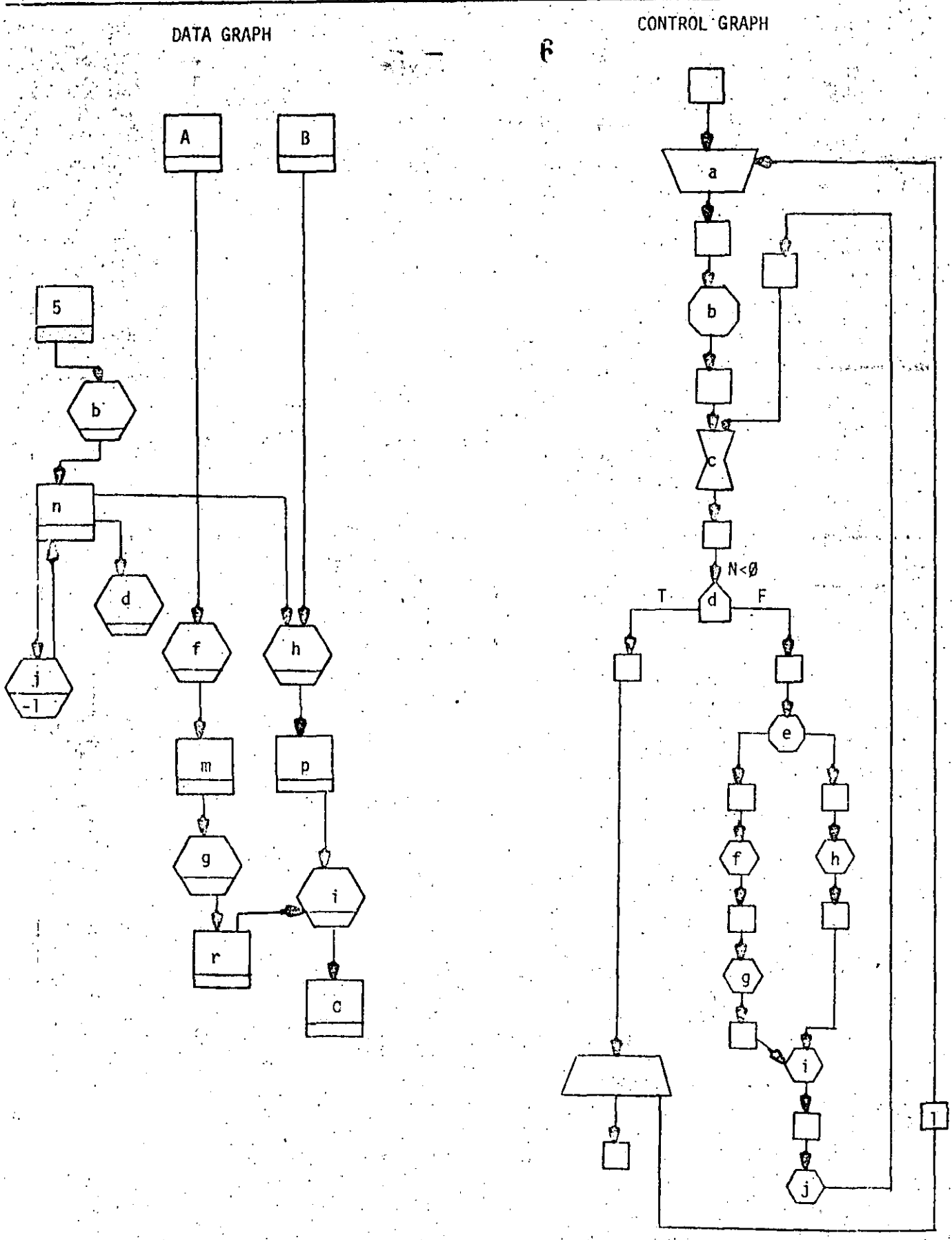


Figure 3—Example of Algol 60 for statement representation

way the analysis of a software (or hardware) system can be carried out in a stepwise, computationally efficient manner.

The imposition of an ALGOL environment is optional, of course, and does not affect the representation itself. To do so does imply the existence of an ALGOL-like run time environment layer which implements the necessary storage allocation and other semantics. A cactus stack is required to keep track of the concurrently active and executing tasks.

The representation may be generalized to allow control variable contents to be non-negative integers with the control operator definitions changed to allow decrementing of input c-cells and incrementing of output c-cells by greater than one. The constraint that output cells be Ø before initiation of the control operator is removed, and the initiation and termination of control and data operators are made distinct to allow multiple initiations of data operators before termination of preceding activations as resources allow. This generalization is useful in describing higher level software processes and hardware such as pipeline systems.

Thus far, only an ALGOL-type level structure has been suggested. Where does layering enter the picture? The concept of layers enters the representation at the data operator. The function performed by a data operator may be truly a system primitive or it may be a "call" on a lower layer facility. That is, its data cells may be parameters to a task existing on a lower layer which is activated by the initiation of the data operator. This may in turn activate other tasks on still lower layers, but the entire data function appears primitive at the layer on which it is initiated. This is an explicit call upon a lower layer. An implicit call would be the activation of the storage allocator upon activation of an activity in a block structured environment.

A formal syntax and semantics for data structures

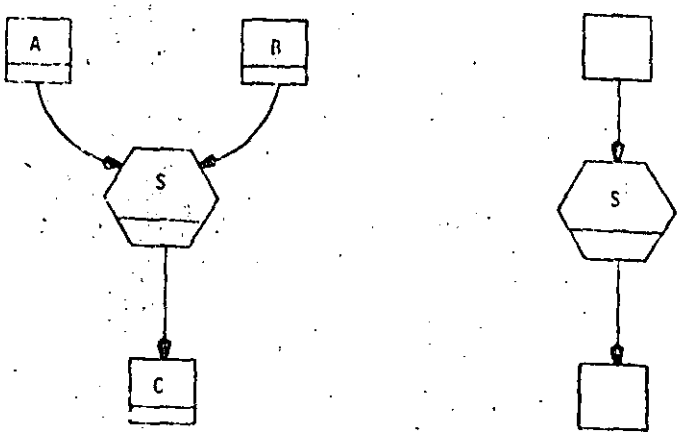


Figure 4—Collapsed activity

```
CDF WORD = (36);
(a)

CDF DOUBL_WD = <CONTIGUOUS> WORD (2);
(b)

STRUCT INTEGER = <WORD>;
(c)

STRUCT LISTEL = <DOUBL_WD> (INTEGER:
  DATA, <WORD> REF LISTEL: PTR);
(d)

WORD CONSTRAINT. INTEGER SIMPLE
(e)
```

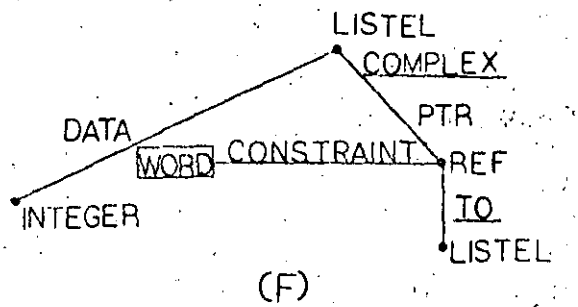


Figure 5—Example of data structure declaration

and a syntax for data operator declarations is being developed. The declarative language is similar to ALGOL 68 and the resulting graphic representation of the data structure descriptors resembles those of Early's VERS.[22] The language consists of six basic building block structures—SIMPLE, MULTIPLE BIT STRUCTURE (MBS), ARRAY, REFERENCE, UNION, and COMPLEX. Examples of SIMPLE structures are integers, reals, etc. MBS's are used to define fields in words or tables. REFERENCE structures denote address variables. UNION is meant in the Set sense, and thus UNION is a place holder for one of a set of structure types. A COMPLEX structure is heterogeneous, consisting of more than one type of basic building block.

Another fundamental concept is that of a constraint. The data structure declarations define logical relationships only. Constraints are used to relate these to physical realities such as words, right half words, etc. These two primitive constraints are: *WORD* and *CONTIGUOUS*.

As an example, consider Figure 5. The length of a WORD is defined in Figure 5a. The constraint

DOUBL_WD is defined in Figure 5b, and an integer is Figure 5c. A complex structure LISTEL (list element) is defined in Figure 5d. It is constrained to occupy a double word, one being an integer, and the other a reference to a LISTEL. The terms DATA and PTR are accessing function names. Figures 5e, f, and g show the graphic representation of the resulting templates. Instances of these data structures may be declared which create descriptors based upon the template but with nodes for allocation information added.

Data operators are defined in terms of the types of their input and output data structures. LOGOS has no formal semantics for data operators, so functional definition is not possible at present. However, an informal semantics is being developed to enhance inter-designer communication and to allow simulation of activities if desired.

The intent of this brief and incomplete description of the structural philosophy and representation system of LOGOS has been to set the stage for a discussion of the use of LOGOS and the analysis tools it provides the software engineer and systems designer.

THE DESIGNER'S ENVIRONMENT

Before discussing the types of analysis tools LOGOS provides the software engineer and system designer it would be helpful to examine the LOGOS environment by describing a typical design scenario.

The systems architects, two or three highly skilled analysts, will either be given or will create a specification for the target system in terms of capabilities, number of users, service times, arrival rates, etc. They will pick the design parameters, block the system into facilities, and identify any obviously common facilities such as memory. In the case of a software system built on an existing computer, the system primitives—the machine instructions—will be specified in advance. The facilities will probably be specified in terms of their external characteristics and will have required performance parameters associated with them.

The information will be given to a group of designers (perhaps the architects themselves) who will define these facilities in the LOGOS design data base from their graphics consoles. The individual tasks performed by facilities will be roughed in, and performance parameters defined for them from those on the facility itself.

The designers may define canonical schemata and store them in a macrolibrary to be inserted and expanded during the design process. These may include structures such as IFTHENELSE and DOWHILE, the primary control elements of structured programming.[23] In terms of hardware, these macros will include the set of MSI functions available to the designers.

As the description of a task becomes complete on a layer, the resulting activities can be analyzed, and the activity collapsed. Common tasks may be grouped into facilities on lower layers and defined accordingly, each having a performance specification derived from above. The designers will make their work available to each other by placing it in a common global data base. Here, lower layer facilities common to several designers may be identified. Duplicate and similar facilities and tasks will be replaced by commonly shared facilities.

Modifications may be evaluated along the way by substituting modified tasks into the data base and re-analyzing the affected portion of the design. This process is continued across descending layers until the data operator functions are in terms of the software primitives of the target system, i.e., machine instructions, implementation language statements, or a trial set of instructions if the entire hardware/software system is being created. Code optimization can then be attempted using one of the newer graph-oriented optimization techniques. Code generation will be discussed briefly in the next section.

If the hardware and/or implementation language exists, actual times will be available for the software primitive operations. These can be reflected up and across layers to determine if the performance requirements were met. If not, redefinition of tasks and/or layer boundaries will be required to attempt to meet the specifications.

If the hardware has not yet been designed, it can be begun at this point with the trial instructions and their required times as the target. The process is identical to the one above, but the lowest layer hardware primitives will be hardware elements such as NAND gates, MSI chips, etc. Once again, actual performance information becomes available and is backed out to the software layers.

If the resulting performance is inadequate, the interpreter (hardware) can be speeded up by increasing the degree of parallelism or upgrading the technology. On the other hand, the hardware/software interface can be adjusted by redefining as primitives certain key data operators which were originally implemented as calls upon lower layer facilities. These procedures may be applied interactively in combination to reach the desired performance, if indeed, it is attainable at all. Once the instruction set is frozen, code may be generated, and the necessary steps taken for implementation of the hardware.

Note that this series of events is a departure from the usual practice of defining the target instruction set as almost the first step in system design and then sending the hardware designers away to build a computer and the programmers to build an operating system. The

integrated design approach advocated here should (1) reduce the hardware/software interface mismatches, and (2) allow cost/performance tradeoffs to be intelligently evaluated at the proper time—before commitment to hardware and code.

The data structures, data operators and resulting code of the operating system are simply data in one of the data structures—memory—of the interpreter (hardware processor). This is true of all program/interpreter systems. If the interpreter were not to be implemented in hardware but on, say, an 1108, then the data operator primitives would be 1108 machine instructions, and code rather than hardware would be generated.

In addition to a framework for representing layered systems, LOGOS will provide the designer with several types of consistency and performance analyses. Further, code generation of target system software, and ultimate implementation of target system hardware are goals which appear attainable.

The analyses can be separated into two classes—uninterpreted and interpreted. Uninterpreted analysis implies that no interpretation is placed upon the function performed by the data operators for purposes of the analysis. Thus, uninterpreted analyses deal primarily with the control graphs and are topological in nature.

The addition of parallelism or concurrency to an activity raises several analysis questions. Of primary interest is whether multiple activations of a parallel activity (schema) with a given initial control state (contents of its c-cells) and data values will result in the same final values in a set of "result" locations. This condition is called determinacy and was formulated originally by Karp and Miller.[17] This condition, even after formalization, is mathematically difficult to prove. Another condition, more stringent but easier to verify, has been formulated by Karp and Miller.

1. A schema is determinate if, given an initial state, q_0 and an initial set of values, each data location has a fixed sequence of values.

With this condition satisfied, then a schema will surely produce consistent values in the "result" locations provided that the algorithm terminates. Karp and Miller further showed that the above condition is equivalent to the following two conditions.

2. (i) No two data operations can be concurrently enabled to "write" into a common data location. (ii) No data operation can be enabled to "write" into a data location while another data operation is simultaneously enabled to "read" from the same location.

From conditions (i) and (ii), a schema is determinate provided that it is free of "race" conditions of two types. This situation should not startle hardware designers who have always faced this problem.

Karp and Miller gave conditions on a parallel schema which allow determinacy analysis to be conducted on the control graph. The analysis tool is a mathematical construct called a "vector addition system" (VAS); for a given schema, the vectors used have one component corresponding to each c-cell in the control graph. A vector q_0 gives the initial control state, and, for each control operator, e, a vector δ_e gives control state changes when control operator e occurs. These change vectors may be derived from those shown in Figure 2, but may be generalized to integers greater than ± 1 for higher level representation. A "tree of nodes" is generated from the root node q_0 which corresponds to the tree of attainable control states of the schema. The algorithm identifies loops in the control and may be used for finite and infinite attainable state schemata. A complete treatment may be found in Reference 20.

The resulting tree can be used to determine those control operators which can be simultaneously enabled, and, hence, those data operators which are concurrent.

By examining the input and output data cells of those data operators, conditions (i) and (ii) above can be verified. The blockhead/blockend of the activity in LOGOS limit the scope of the analysis, and thus can limit the size of the tree to manageable size. The activity can be analyzed for determinacy and collapsed. It will then appear as a single operator pair in more global analyses.

The vector addition system can be used to check for proper termination of an activity, i.e., can a control/data operator pair remain enabled after the blockend of an activity is enabled? Further, is the topology of the control graph such that the activity will not terminate? Remember that this is uninterpreted analysis, and, consequently, the results of predicate operations are not known. Therefore, in some cases, all that can be said is that there exists a path which if taken, will result in no termination.

Similarly, by viewing all activities as primitives, a potential recursion analysis can be carried out using the vector addition system. These types of analyses fall into the category of general control path analysis, and additional algorithms in this family can be identified and easily implemented using the VAS.

A major weakness in the integrity of computer systems has been the management of system resources and the prevention of system deadlocks. This is particularly true in systems with a high degree of real or apparent concurrency. This problem has been extensively studied, and much insight has been gained.[6,11,24,25] Holt[25] has

developed graph models for deadlock and resource allocation which are directly applicable to the LOGOS environment. Resources are represented as control cells, and a topological analysis using adaptations of Holt's results can be performed. Once again, a layered structure tends to limit the scope of analysis.

System performance analysis depends upon knowledge of arrival rate and service request distributions, and, thus is only as good as the model load. However, actual path transit times can be computed in the LOGOS environment, and if model service request distributions and arrival rates are available, performance statistics can be gathered before implementation using a combination of path analysis and simulation, if necessary.

Interpreted analysis deals with the correctness of the algorithms used in implementing the activities. At present, LOGOS has no automated solution to the program correctness problem. The layered structure of target systems, together with the communications disciplines enforced by the syntax of the representation and the various other analysis algorithms tend to assure logical and structural consistency. However, a logically consistent, *but* incorrect algorithm is undetectable. Current work by Scott and Strachey,[26] leading toward a formal mathematical theory of hierarchical systems and semantics, may well be the answer. Results of this work could be adapted to replace LOGOS current data graph syntax and semantics and provide a certifiable representation. In the interim, interpreted data analysis algorithms based upon the functional attributes of the data operators are being considered. For example, a data operator must access data structures of the appropriate type and compute results which correspond to the types of output data structures to which it is connected. This is useful in analyzing data functions which are implemented by interlayer facility activations. In critical areas, actual simulation of the algorithms in question may be performed directly.

Finally, if a global semantic such as ALGOL 60 or FORTRAN is imposed, environmental consistency algorithms such as scope of reference can be included modularly.

CURRENT STATE OF LOGOS

The LOGOS system is being implemented on a Digital Equipment PDP-10 with a Bolt, Beranek and Newman paging box and TENEX executive system. The primary graphics terminals are two IMLAC PDS-1 display systems which communicate with the PDP-10 at 9600 baud. The implementation language is SAIL (Stanford Artificial Intelligence Language). A multi-designer data base management system is being implemented using the LEAP associative data structures of SAIL and the TENEX virtual memory facilities. The system provides for local (single user) and global (shared) data bases with linking between local and global information in a controlled manner. The data base management system is based upon earlier work by M. Pliner.[27]

The graphical representation system is implemented together with the following analysis algorithms: graphical syntax checking, determinacy, halting and termination, and repetition freeness. Implementation of generalized control path analysis is also under way.

The remainder of the control analyses, deadlock and resource allocation, are scheduled to be implemented and integrated by September 1973. It should be noted here that all of the analysis packages are modular and act upon the standard internal representation, thus allowing new packages to be added when necessary.

The implementation specifications for the data structures and data operators are scheduled for completion in December 1972, and implementations should be complete by September 1973 along with the associated analysis routines. These analysis routines assume a FORTRAN environment with a static block structure but may be replaced if another semantic is chosen.

Performance analysis algorithms should be implemented and integrated by September 1973.

Thus, with the very major exception of a formal semantics and corresponding attack on program correctness, LOGOS is scheduled to have a running representation and analysis system by September 1973.

The implementation of target systems requires the production of a code generator for the software and a "hardware compiler" for the hardware portions of the representation. Here again, Scott's work may provide a general solution to the semantics problem for the code generators, but even without such results, if the software primitives in the data graphs are machine language instructions of the target machine, code generation becomes rather straightforward. In addition, the graphic form of the program tasks will allow application of the newer optimization techniques to the target software. A first cut code generation scheme for sequential (rather than parallel) systems should be implemented in early 1974.

Rather than re-create a "hardware compiler" which would require 30-50 man years, LOGOS has chosen to interface with existing hardware CAD systems at the logic equation/logic diagram level. Although much of the information which could help in optimization of the hardware will be lost in going to the equations, the time scale and scope of the project preclude attacking the hardware CAD problem directly. It is felt, however,

that the graphic representation may provide helpful insight in the partitioning and placement operations of hardware CAD, and those problems will continue to be studied. The hardware equation/diagram outputs are scheduled for September 1974.

In parallel with these efforts, an attempt is being made to define one or more programming languages to serve as alternate external representations of the target system rather than the current graphical representation. This is being done because some programmers may feel uncomfortable with the graph form, and because the human engineering and scope management problems become significant as the complexity of the target graphs increases.

The LOGOS representation has been used off-line to describe various types of small systems and subsystems including a PDP-8. The resulting descriptions are concise, and being able to see both the structure and function of the systems in one "picture" aids in understanding the target system.

With regard to implementation, the resident executive in the IMLAC display processors was designed according to the LOGOS structural philosophy.

The IMLAC system provides the designer with facilities of (1) creating a picture and designating it a subroutine for transmission to the PDP-10, (2) editing a subroutine, and (3) deleting a subroutine. The system exists on six layers as shown in Figure 6. The lowest is the PDS-1 hardware used by all higher layers. The next layer facility is the character transmitter (all messages, text and graphics are sent to the PDP-10 as multiple character strings). Layer 4 contains the keyboard character handler and the character receiver both of which are users of the character transmitting facility. The character receiver uses the character transmitter facility to control the transmission of characters from the PDP-10 to the IMLAC. The next layer has three independent facilities—the light pen tracking facilities, the graphics message handler, and the core management facility. All of these facilities are used by the facilities on layers 1 and 2, the subroutine edit and subroutine create and delete facilities. The communications discipline between the facilities are well-disciplined according to LOGOS design principles.

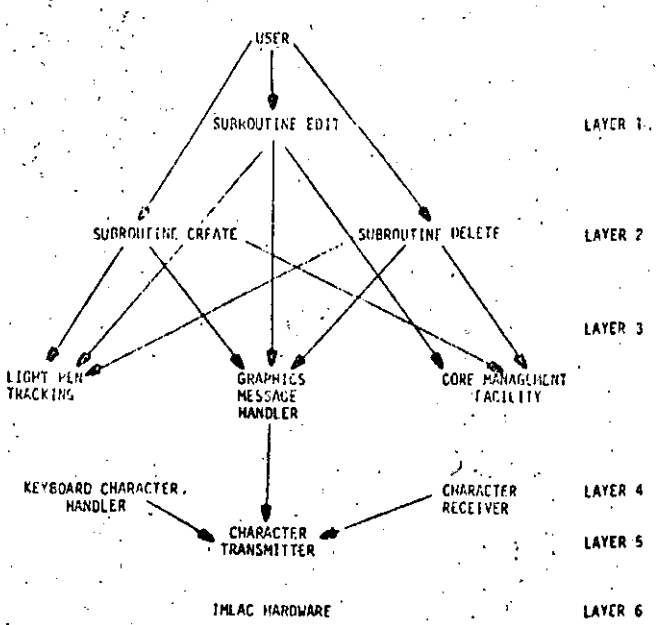


Figure 6—Layer structure of Imlac executive

The design and implementation of the executive required about six man months of effort. It occupies approximately 3000 words in IMLAC core and was coded in assembly language. As with the 'THE'[6] and 'VENUS'[7] systems, coding errors were discovered, but few logical errors were committed in the design. These proved easy to identify and correct.

CONCLUSION

The aim of Project LOGOS is to provide the computer system designer with a computer-assisted design environment in which good engineering practice can be applied to large-scale target systems and verified after the fact. The basis of this good engineering practice is a structural view of computer systems which is a generalization of Dijkstra's[2] and Mills'[23] structured programming for sequential software. Dijkstra's 'THE' system[6] is a result of this philosophy as is the VENUS system.[7] Both these and the IMLAC executive have demonstrated the payoffs of a well-disciplined approach to structure. They were implemented in a fairly short time by small design groups (VENUS required about 6 man years for the design and implementation of the operating system and the support software). They were easily checked out and modified, and have proven to be stable, reliable systems. The primary contribution of LOGOS in this area is that it provides a uniform, analyzable representation in which to express these otherwise abstract notions of system structure, one which leads directly to the implementation of the target software or hardware. It also allows the designer to express the maximum degree of real and apparent concurrency in his target system and provides the analyses required to evaluate its effect.

Both 'THE' and VENUS are small operating systems implemented on small to medium scale machines, yet even they were found to contain a few errors resulting

from breaches of discipline. True, these errors were easily corrected, but as the size and complexity of the operating system and hardware increases, the difficulties of enforcing the disciplines, detecting errors, and correcting them without introducing more will increase nonlinearly. It is because of this complexity explosion that a CAD environment such as LOGOS is required for large scale systems.

A LOGOS-type system can provide several other advantages to the software engineer and system designer. First, because performance measurements can be made before rather than after implementation, modifications to the system can be proposed and their effects evaluated economically. In particular, the final positioning of the hardware software interface can be postponed until quite late in the design cycle and can be made a true function of performance vs. cost.

Second, the design team will tend to be smaller. The computer will act as the "bookkeeper" and will perform many of the analyses which have traditionally been attempted manually or not at all.

Third, the increased degree to which a target system can be certified before implementation (even without formal semantics) should reduce the integration and checkout cycle significantly. It may also be possible to produce more complete diagnostics in a LOGOS environment since the entire system description as well as its implementation is stored within the design data base. This is an area for continued research.

Finally, although this hints of "big brother," valuable management and scheduling information can be extracted from such a system. The effectiveness of designers, the times required to complete various portions of the system, etc., could be used in estimating, staffing, and scheduling future systems.

LOGOS is an open-ended system. Although a first producing system will be complete in 1974, it is expected that the users themselves will enhance, modify and tailor the design environment to their needs as new technology becomes available.

ACKNOWLEDGMENTS

The LOGOS design environment described in this paper is the result of work done over the past three years by Professor E. L. Glaser, principal investigator; Dr. F. T. Bradshaw; S. Katzke; the author and several others. In particular, much of the philosophy of system structure which underlies the LOGOS system was articulated by Dr. Bradshaw, and the syntax and semantics of target system data structures and data operators were developed by S. Katzke during his doctoral research.

REFERENCES

1 F G HEATH C W ROSE
The case for integrated hardware/software design with CAD implications
IEEE Computer Conference Digest September 1972

2 E W DIJKSTRA
EWD249—notes on structured programming
T. H. Report 70—Wsk—03
Technological University Eindhoven Netherlands April 1970

3 T BREDT
A model for parallel computer systems
Technical Report No 5 STAN-CS-70-160 Stanford University April 1970

4 C G BELL A NEWELL
Computer-structures: reading and examples
McGraw-Hill Book Company New York New York 1971

5 M BARAY Y H SU
A digital system modelling philosophy and design language
Proceedings Eighth Annual Design Automation Workshop 1971

6 E W DIJKSTRA
The structure of the T.H.E.—multiprogramming system
Comm ACM Vol 11 No 5 May 1968 pp 341-346

7 B LISKOV
The design of the VENUS operating system
Comm ACM Vol 15 No 3 March 1972 pp 144-149

8 C D MARSH
Automation of the design and manufacturing of a large digital computer
IEE Electronics & Power October 1970 pp 375-379

9 M R CORLEY
The graphically accessed interactive design of thermally stressed pipe systems
Proceedings Ninth Annual Design Automation Workshop 1972

10 F T BRADSHAW
Some structural ideas for computer systems
IEEE Computer Conference Digest September 1972

11 E W DIJKSTRA
Co-operating sequential processes
Programming Languages ed F Genuys Academic Press 1968

12 M J SPIER E I ORGANICK
The MULTICS interprocess communication facility
Second ACM Symposium on Operating Systems Principles Princeton University October 1969

13 E W DIJKSTRA
A constructive approach to the problem of program correctness
BIT Vol 8 1968 pp 174-186

14 C A R HOARE
Proof of a program FIND
Comm ACM Vol 14 No 1 January 1971 pp 39-45

15 N WIRTH
Program development by stepwise refinement
Comm ACM Vol 14 No 4 April 1971 pp 221-227

16 C A PETRI
Kommunikation mit automaten
Schriften des Reinsch-West Falischen Inst Instrumentelle Math und der Universitat Bonn Nr 2 Bonn 1962

17 R M KARP R E MILLER
Parallel program schemata
Journal of Computer and System Sci 3 1969 pp 147-195

18 A W HOLT F COMMONER
Events and conditions an approach to the description and analysis of dynamic systems
Third Semi-annual Technical Report Part II For the Project Research in Machine-Independent Software Programming Applied Data Research Inc April 1970

19 F L LUCONI
Asynchronous computational structures
Doctoral Thesis MIT Cambridge Mass January 1968

20 F T BRADSHAW
Structure and representation of digital computer systems
Jenning Computing Center Report No 1114 Case Western Reserve University Cleveland Ohio January 1971

21 C W ROSE
A system of representation for general purpose digital computer systems
Jennings Computing Center Report No 1113 Case Western Reserve University Cleveland Ohio August 1970

22 J EARLY
Toward an understanding of data structures
Comm ACM Vol 14 No 10 pp 617-627

23 H D MILLS
Mathematical foundations for structured programming
FSC72-6012 Federal Systems Division International Business Machines Corporation Gaithersburg Maryland February 1972

24 A N HABERMANN
Prevention of system deadlocks
Comm ACM Vol 12 No 7 July 1969 pp 373-385

25 R C HOLT
On deadlock in computer systems
Doctoral Dissertation Cornell University Ithaca New York January 1971

26 D SCOTT C STRACHEY
Toward a mathematical semantics for computer languages
Tech Monograph PRG-6 Oxford University Computing Laboratory August 1971

27 M S PLINER
PDMS—a primitive data base management system for representing structured data in an information sharing environment
Doctoral Dissertation Case Western Reserve University Cleveland Ohio September 1971

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

DISEÑO DE BASES DE DATOS

MULTIPLE VIEWS AND ABSTRACTIONS WITH AN EXTENDED-ENTITY-RELATIONSHIP MODEL

MAYO 1985

Computer Languages, Vol. 4, pp. 139 to 154
Pergamon Press Ltd. 1979. Printed in Great Britain.

MULTIPLE VIEWS AND ABSTRACTIONS WITH AN EXTENDED-ENTITY-RELATIONSHIP MODEL*

GERD SCHIFFNER and PETER SCHEUERMANN

Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201, U.S.A.

(*Received for publication* 10 *May* 1979)

Abstract—The features of the Entity-Relationship model are examined, as a means for representing both the conceptual schema and the external schema in a database system. In order to serve in this dual capacity, its modelling features are extended to support multiple user views. We show how to represent a model and present a suitable diagrammatic technique. To represent these concepts at the user level we propose a data definition language with a concise and simple structure. Furthermore, guidelines are given for interfacing the E-E-R schema with existing DBTG-like systems.

Conceptual schema External schema Database systems Data Definition language Data abstraction

1. INTRODUCTION

THE PAST few years have seen a growing awareness of the fact that a conceptual framework, involving a three schema technology, is essential for the development of data base systems. The novel element in this framework, compared to previous database technology, is the emergence of the *conceptual schema*: a set of rules describing the information relevant to a given enterprise. Such a description should encompass object types and roles/relations, as well as consistency and dependency constraints [1, 4, 12, 13]. How this information is presented to a program by various users is described in the *external schema*, while the *internal schema* describes how to represent it in storage (still one step removed from actual physical records, but including physical sequencing, selection of indices, etc.). There is also the need for two mappings, one between the conceptual schema and the internal schema, the other between the conceptual schema and the external schema. Since the process of defining the relevant information to an enterprise is closely related to the system analysis and design activity, the conceptual schema is an important tool in the communication between end-users and managers on one hand and database designers and implementers on the other hand. Additional advantages of the conceptual schema concept are improved semantic control over the data and a higher degree of data-independence, due to the relative stability of the conceptual schema [2, 7, 19].

Although there is not yet a consensus about what constitutes the best set of concepts to be incorporated in a conceptual schema, a number of models have been identified as possible candidates [3]. Among these, the Entity-Relationship (E-R) model proposed by Chen [6] has many of these desired features; it is easy to formulate, it is easy to understand, and in addition it includes a concise diagrammatic technique which can be used in the process of system analysis [6, 7, 10]. In addition, the simplicity of the E-R model and its semantic structure, closely resembling that of sentences in natural languages, also make it a suitable choice for the external schema. However, to use the E-R model in this dual role, it is necessary to expand its capability to support multiple views and abstractions along the lines suggested by Smith and Smith [15, 16].

In this paper we first describe some general properties of the E-R model. Next, we discuss how to extend the model in order to support more abstractions (views) and correspondingly augment the diagrammatic technique. To realize these concepts at the

* This work is sponsored in part by NSF Grant MCS77-03904.

external (user) level we present a suitable data definition language. Finally, in the last section we give some guidelines for translating the Extended-Entity-Relationship (E-E-R) diagrams at the internal level into an executable DBTG network schema.

2. SEMANTIC INFORMATION IN THE ENTITY-RELATIONSHIP MODEL

Basically, the Entity-Relationship model [6, 7] consists of sets of *entities* or *entity sets* which denote real-world entities or "things" which can be distinctly identified. A *relationship* is an association among entities, and a relationship set is a set of relationships of the same type. The role of an entity in a relationship is the function it performs in that relationship. In the E-R diagram an entity set is represented by a rectangular-shaped box, while a relationship set is represented by a diamond-shaped box with lines connected to the related entity sets. In addition, the lines are labelled so as to indicate the type of mapping (*1*:*1*, *1*:*m*, *m*:*1*, or *m*:*n*) for the particular relationship (see Fig. 1). Entities and relationships constitute only the "upper conceptual domain" in Chen's model [6]. Entities and relationships have properties which can be expressed in terms of *attribute-value* pairs. This information resides at the "lower conceptual domain" and to draw it in a diagram, value sets are designated by circles, and an attribute is represented by an arrow directed from the entity (or relationship) set to the corresponding value set(s). Figure 1 also includes partial attribute-value information.

It is worthwhile to remark here that the relational model of Codd [8] deals only with values and relationships. Relations represent the intension (meaning) of the data as well as the extension (representation) of the data [11]. As such the model is sensitive to changes in the nature of the associations involved. For example, if an association type changes from *1*:*m* to *m*:*n* and the relations are maintained in 3NF, new relations will have to be introduced. On the other hand, the relational model does not have any particular naming requirements for relations, and does not provide the control of modeling only intellectually manageable objects. Consider the example given by Smith and Smith [15] that "a student is enrolled in a class in a given room and climbs a certain mountain", which can be modeled by the Codd relation:

R(STUDENT, CLASSROOM, MOUNTAIN)

This relation is in 3NF, but is not "well-defined" in the sense of [15], since it does not relate to a real-world entity or relationship.

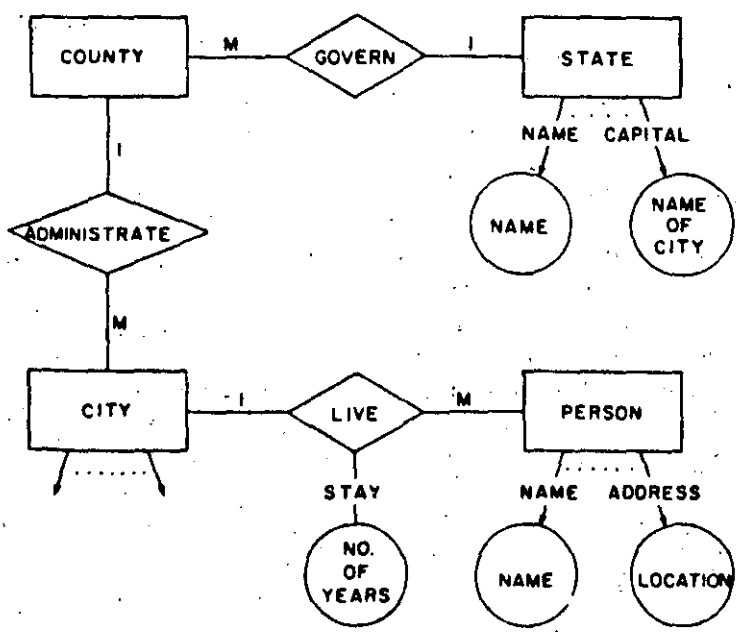


Fig. 1.

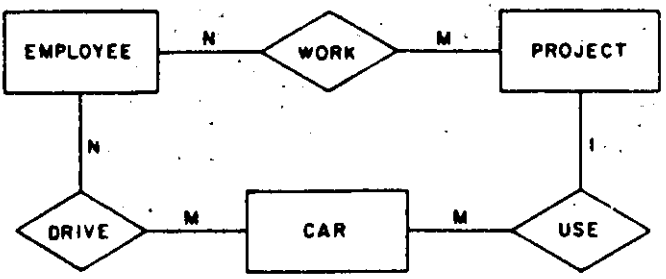


Fig. 2.

The E–R model provides more semantic control over the data and is capable of expressing a more stable view of the information of interest to a given enterprise. In addition, as discussed in [11], the model closely resembles the surface semantics of a sentence, i.e. a complete discourse, which is an essential property for a conceptual schema model. The requirement of Smith and Smith [15] that a naming discipline with single words of the natural language to be adopted can be enforced quite easily. We go one step further and recommend the use of nouns for entity names and verbs for relationship names, as commonly done in the structure of phrases in a natural language, for example, "an employee belongs to a department" and "parts are ordered for projects". However, sometimes it might be difficult to find suitable names, particularly for similar objects. For example, consider the case when one wants to model the attendance at schools of parents and children by two relationships between the entities school, parents and children. Furthermore, it can be recognized that natural names of relationships often favor one direction of discourse, for example, "an employee belongs to a department". With respect to the formulation of queries such an asymmetry might not be desirable. The problem may be resolved along the following guidelines. The use of transitive verbs can support both directions by employing the active and passive voice. According to the application, the names should be selected to match the most frequent queries. Furthermore, synonyms can be used to resolve the asymmetry. More detail regarding this naming discipline can be found in [14].

The diagrammatic representation of the E–R model enables us to consider additional relevant semantic aspects. We define a *logical path* as a sequence of connections (undirected edges) between two entities. Since these connections correspond to existing relationships, a traversal of such lines refers to a sequence of corresponding relationships. Thus a logical path gives us a simple semantic measure of the "relatedness" of entities, which are not adjacent. Since individual connections are explicitly specified in the model, in some cases they may be used to derive extended relationships. Consider again the hierarchical structure depicted in Fig. 1. The relationship "persons inhabiting a state" expressed here in a natural language phrase, has the underlying meaning of relating "state" over the given logical path to "person", an operation which the system can perform unambiguously.

On the other hand, it is rather clear that from a semantic point of view, a shorter path usually carries more exact information than a longer one. For example, the path from "car" to project" via "use" in Fig. 2 reflects the fact that each car is used only for one particular project. The other path over "employee" yields a $m:n$ association between cars and projects. The concepts of logical path and length of a logical path become of major importance when we consider the design of a query language close to natural language formulation for the E–R model [14].

3. MODELING DIFFERENT LEVELS OF ABSTRACTIONS

Although the E–R model provides a quite stable picture of the enterprise view of the data, it is still possible that this view will change in the course of time. In addition to permanent changes, such as those described in [2, 7] we also want to provide the facility for different users to see only part of the data or see it in a different form. Traditionally,

multiple views have been supported in database systems by the DBTG-like (external) schema/subschema facility [9], or by allowing the forming of views via query evaluation [5].

While the information available to a given user (view) could be derived (with greater or lesser difficulty) at evaluation time, it is important to allow various views to coexist and to be able to model this explicitly. Some benefits of this approach are (1) easier reinforcement of consistency and integrity constraints, (2) a more systematic data-base design process is achieved and (3) more efficient internal structures are possible [16].

The current proposed approach to deal with multiple views is to treat them as *abstractions*, i.e. models of the data-base in which certain details are deliberately suppressed [15, 16, 19]. It is also recognized that the notion of abstraction is powerful enough only if we allow abstract objects to be composed (decomposed) in a hierarchic fashion. Two kinds of abstractions which are particularly relevant to data base design have been identified by Smith and Smith: *aggregation* and *generalization* [15, 16]. *Aggregation* refers to an abstraction in which a relationship between objects is regarded as a higher level object. *Generalization* refers to an abstraction in which a set of similar objects is regarded as a single generic object. For example, the employee classes "secretary", "engineer" and "trucker" with presumably different attributes can be generalized to the single object class "employee". Under this structuring discipline the generic object "employee" contains only the attributes which are common to all employee classes, while the attributes distinguishing a particular employee class are visible only at the lower level of the hierarchy.

In this section we shall consider how to extend the E–R model to include variations of the above mentioned kinds of abstraction. It is important to mention here that Chen's [7] "split" operation in which new objects are created to express a greater amount of detail, does not satisfy our notion of abstraction. The new objects are modeled as separate entity sets; therefore, the information about their similarity is lost.

In designing the E–E–R diagram a decision has to be made regarding the tradeoff between the amount of information visible and the manageability of the graphical representation. We opted for the benefit of a two-dimensional picture, rather than a three-dimensional one as employed in [16], and to avoid the representation of cutting lines, as can happen in the conceptual diagrams of Nijssen [12]. These features are particularly important if we regard the E–E–R diagram not only as a tool for the conceptual level, but also for the user (external) level by facilitating the capability of graphical support during interactive query processing.

In the context of the E–R model we will distinguish between abstractions based on entities, and those based on relationships. For the sake of discussion, we shall speak here in terms of transformations applied to the diagram, but the reader should keep in mind that we are referring to different views (subviews). In the next section we shall show how to actually represent these views in a data definition language.

3.1 *Abstractions of entities*

In this group we include abstractions similar to the ones obtained by generalization in [16]. We make a distinction, however, between subsets of entity sets formed by applying a predicate to a value set and subsets with partially different attributes. An example of the first kind of abstraction is given by the entity subsets "old" and "young" employee formed from the entity set "employee". A second kind of entity abstraction is obtained by decomposing "employee" to the distinct status classes "secretary", "trucker" and "engineer" which may exhibit different attributes in addition to the common ones, say "empl-number" and "name". For example, for secretaries we may also be interested in "typing skill", for engineers we may want to know the "type of degree" obtained, while for truckers we are interested in "type of license".

Objects of the first kind are called *selectors*, because such a substructure can be easily described by a selection clause which qualifies attributes of an entity, like "age" in the above example. For simplicity, we assume that only one attribute can be qualified by a

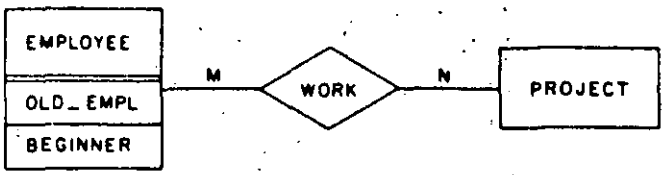


Fig. 3.

selector. Combinations may be obtained by the formulation of a query. Since the subsets are of the same type as the original entity set, it is consistent to represent them in the diagram within the same box. For example, Fig. 3 shows that besides the "employee" set we want to maintain the subsets "old-emp" and "beginner". Thus, selectors represent logical subsets with the same attributes as the general entity, with their members being selected according to the value of a discriminating attribute.

The second kind of entity abstraction correspond to the generalization defined in [16]. We choose, however, a different graphic representation, which is simpler and integrates the ideas of the E–R model. We discuss first how to represent an entity set which is *detailed* into subsets with different attributes and therefore different names (this is conceptually the inverse of the generalization transformation). Figure 4 refers to our previous example and represents employees detailed by different job classes according to the attribute "status". We note that the objects which are derived on the basis of one discriminating attribute ("status" in our example) constitute mutually exclusive entity types ("secretary", "engineer", "trucker"). Such a group of generic objects sharing a common parent was called a *cluster* in [16]. In general, a generic hierarchy such as the one shown in Fig. 4 can have more than one cluster, each containing entity types which are mutually exclusive.

We observe that the associations between a general entity and the detailed entities in a cluster have different meaning from that expressed by a normal relationship. Each instance of the general entity "employee" is related to only one instance of one of the entities in the cluster "status". Furthermore, the association type between a general entity and a whole cluster is always fixed to $1:m$ (one-to-many). Note also that the relationship in this case does not bear any additional information (i.e. cannot have its own attributes); it merely expresses the structure of the generic hierarchy.

As a consequence, we use a modified diamond box to relate a general entity to its detailed entities in a cluster. The diamond box bears the name of the cluster; no association types are specified, and all entities of the cluster sharing a common parent are connected to the same point of the diamond box (see Fig. 4).

The inverse transformations can also be performed if fewer details are to be available. If the detailed entities in a cluster are *generalized* to their parent entity, the cluster name

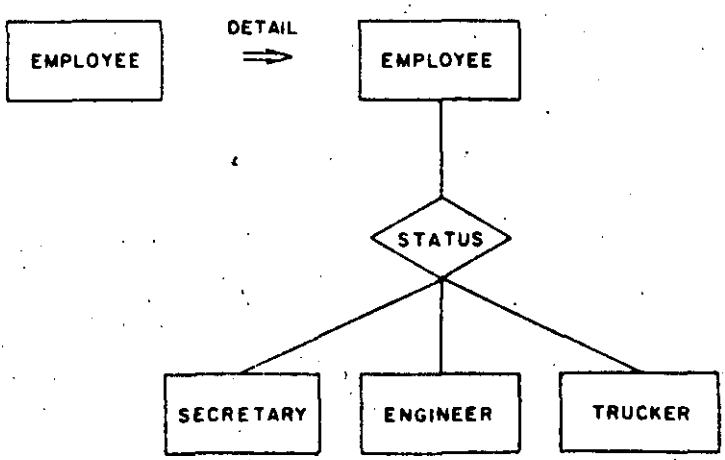


Fig. 4.

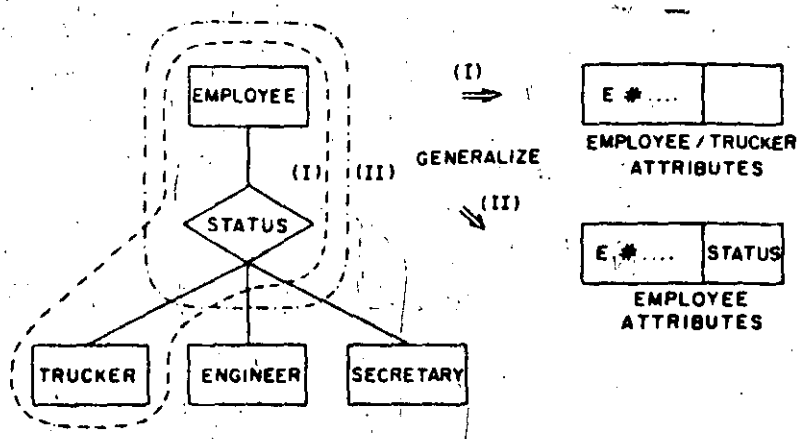


Fig. 5.

becomes a regular attribute again and the substructure is not visible to the user (Fig. 5—transformation II). If, however, only one detailed entity in the cluster is to be generalized to its parent, its attributes are concatenated to those of the parent and the cluster attribute is left out (Fig. 5—transformation I).

3.2 *Abstractions of relationships*

In this group we first include the aggregation as defined earlier. Figure 6 shows the aggregation of the relationship set "work" between "employee" and "project" (represented graphically by the rectangle enclosing the desired relationship set and its associated entity sets). The new object "employee" may reflect the fact that user is interested in knowing only the employee-number, project number, and the attribute of the relationship, say time involved. Aggregating the relationship results in an object with fewer details and it is up to the user to specify which details he wants omitted.

We observe here that the inverse of the *aggregate* transformation, is not included in our abstraction primitives.

An "employee" entity set may have to be decomposed due to a change in functional dependencies, but this should be reflected at the conceptual schema level. From the standpoint of ensuing multiple views, we want to guarantee that we deal only with "well-defined" objects [15], and therefore an entity cannot have multiple values for an attribute (which could justify a decomposition).

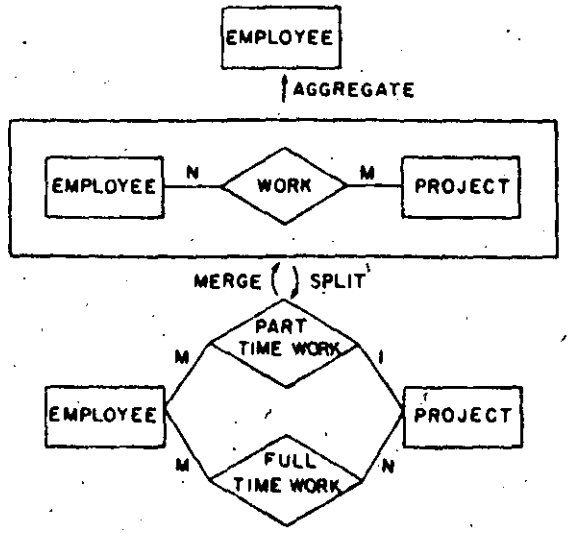


Fig. 6.

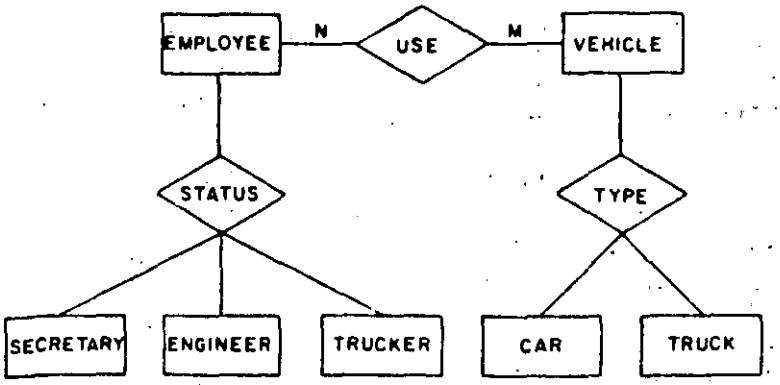


Fig. 7.

Figure 6 also includes the transformations *split* and *merge*, which are applicable only to relationships. Splitting a relationship on the basis of one of its attributes, allows one to view the relationships between subsets of the original entity sets.

The more complex case of a relationship split is the one which involves the detailed entities of a generic hierarchy. For example, consider the case of Fig. 7, where it becomes desirable to model that truckers are assigned to particular tracks as the only employees who are allowed to drive trucks. Since a detailed entity is a subclass of the general one, its relationships to other entities must be subclasses of the relationships in which the general entity is involved. In terms of the diagrammatic representation, it is therefore consistent if relationships among detailed entities are not explicitly expressed. They can be visualized by following the unique path between the involved entities, as shown in Fig. 7. This decision has the disadvantage that a user cannot readily perceive which association type is declared on such a subdivided relationship. This corresponding association type, however can be easily looked up in the full declaration of the scheme, as we shall see. On the other hand, we achieve the advantage of a picture of manageable complexity, which can be represented quite easily on a graphical system, supporting users in a more natural query formulation.

Detailed entities within one cluster can only become related if a relationship is declared recursively on the general entity. Consider, for example, the recursive relationship "cooperate" on "employee" depicted in Fig. 8, which expresses term connections among employees. This relationship could be split up to reflect a particular connection, say "supervise" between the different job types "engineer" and "secretary".

As we mentioned earlier, several clusters can be derived from one entity type, each one containing only mutually exclusive entity types. Notice, however, that the instances to which these types are referring are not necessarily disjoint. For example, if another

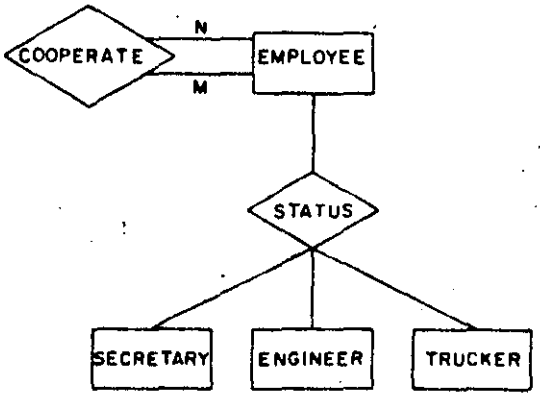


Fig. 8.

cluster is derived from "employee" on the basis of "income" classes, a particular employee instance occurs in both clusters. Therefore, relationships among detailed entities of different clusters within the same generic hierarchy are not allowed since their meaning cannot be defined consistently.

4. A DATA DEFINITION LANGUAGE FOR THE EXTENDED-ENTITY-RELATIONSHIP MODEL

After the information-structure of a given system is designed as an E-E-R diagram and properties are identified for the various objects, the model can be described at the external (user) level in terms of Data Definition Language (DDL) statements. While the schema DDL defines the entire data base that is available, the subschema DDL describes the partial information which is visible to certain users or application programs.

The transformations discussed earlier allow for more or fewer details of information to be modeled. Accordingly, they can be associated with either the schema DDL or the subschema DDL at the user level. Decomposing an object type while also maintaining the global object allows for modeling of multiple views. Thus the representations of selectors, detailed entities or split relationships are eligible for the schema DDL. Associated with the subschema DDL are the operations which suppress some details such as generalization of entities, aggregation, or merging of relationships. This is consistent with the modeling of different views in other models [18, 19].

4.1 *Description with schema DDL*

The DDL proposed in this paper extends the ideas presented in [6]. To illustrate the relevant features of the language we use the sample database depicted in Fig. 9, which describes the information of interest to a given company. The employees are associated with a particular department and may participate in several projects. There are three

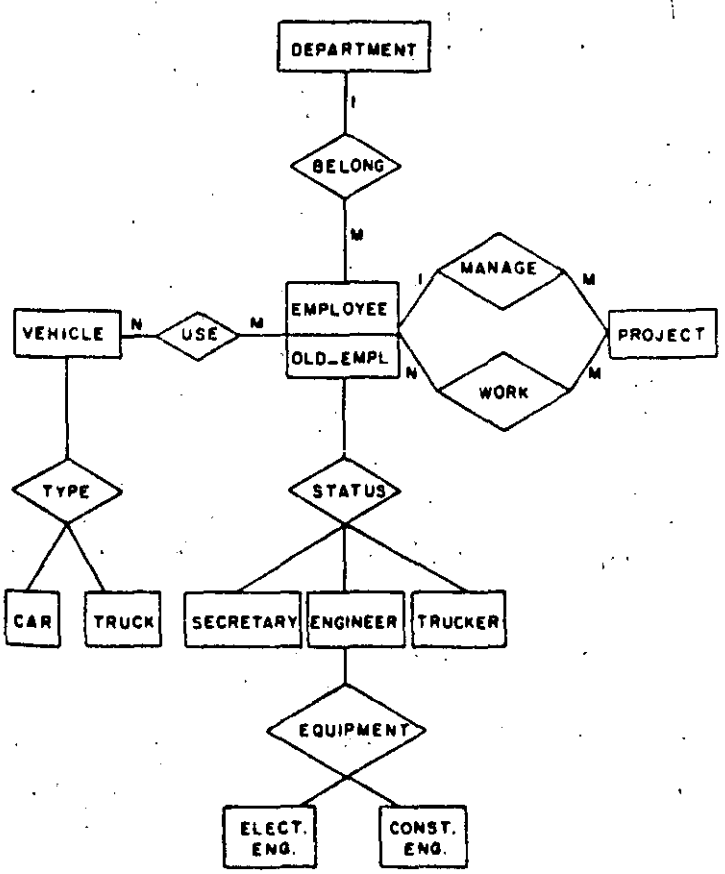


Fig. 9.

distinct employee classes with different properties. Within the class of engineers, two groups are to be modelled separately, according to the technical equipment used. Also, the group of senior employees is of interest and is explicitly reflected. Related to their jobs, employees may use company-owned vehicles of different kinds. This usage is restricted according to the job classes.

The following declarations describe selected parts of this sample database. The DDL uses upper case letters only. Comments which are not part of the schema appear in lower-case letters. The metasymbols "{ }" and "[ ]" are used to denote default and optional elements, respectively.

```
SCHEMA COMPANY                                                    (I)

ENTITY EMPLOYEE ALIAS WORKER                                      (II)
attr. name    specif.    value set        representation  allowed values
ID            KEY        EMPL#            INT(50)
NAME                     NAME             CHAR(20)
AGE                      NO_OF_YEARS      INT(2)          (17;65)
SALARY                   INCOME_P_YEAR    INT(6)
SPOUSE        OPTIONAL   NAME             CHAR(20)
STATUS        GENERIC    JOB_CLASS        CHAR(8)         (SECRETARY,
                                                          ENGINEER,
                                                          TRUCKER)

ENTITY ENGINEER [WITHIN STATUS]                                   (III)
{KEY IS FROM EMPLOYEE}
EDUCATION                DEGREE           CHAR(5)         (PHD, MS, BS)
EXPERIENCE               NO_OF_YEARS      INT(2)          (0;10)

SELECTOR OLD EMPLOYEE OF EMPLOYEE                                 (IV)
WHERE  AGE > 55

RELATIONSHIP BELONG                                               (V)
specif.   entity       role                          type  assoc. key
BETWEEN   DEPARTMENT   ADMINISTRATIVE UNIT           /1    {ON DEPT#}
AND       EMPLOYEE     {EMPLOYEE}                    /100  {ON EMPL#}

Attr. name                value set          representation  allowed values
AFFILIATION               STARTING DATE      INT(6)          ≤ 78 09 01

RELATIONSHIP USE                                                  (VI)
BETWEEN   EMPLOYEE    DRIVER                  /N
AND       VEHICLE                             /M
USAGE                 HOURS_P_MONTH INT(3)    ≤ 200

RELATIONSHIP HAULAGE WITHIN USE                                   (VII)
BETWEEN   TRUCKER                             /1 {ON EMPL#}
AND       TRUCK                               /1 {ON VEH#}
LOSS HOURS            NO_OF_HOURS   INT(3)    ≤ 200

RELATIONSHIP VISIT WITHIN USE                                     (VIII)
BETWEEN   ENGINEER                            /N {ON EMPL#}
AND       CAR                                 /M {ON VEH#}

RELATIONSHIP SERVICE_TOUR WITHIN USE                              (IX)
BETWEEN   ELECTRICAL ENGINEER                 /M
AND       CAR                                 /1
```

A schema is identified by its name (block I). To allow convenient referencing for different users, a number of synonyms can be declared for the same object name (block II). The specification column in the entity declaration of "employee" expresses the following information:

ID is the primary key of the entity.
SPOUSE may have an empty value, since not every employee is necessarily married. All other attributes must have non-empty values.
STATUS is used to specify a generic substructure on the entity "employee" which will consist of a cluster with the name "STATUS" and three detailed entities according to the specified values of the value set.

As defined in the E-R model entities and relationships have unique names throughout the schema, while attributes names are unique at least within their defining object.

The entity "engineer" (III) is declared as a detailed entity in the cluster "status". Although the system can automatically identify "engineer" as a detailed entity from the information given in block I, the optional specification may support a better understanding of the substructure. The next line of block III is default because by the definition of generic substructure, a detailed and a general entity always have a one-to-one association type. Thus, the primary key does not have to be repeated in a detailed entity.

The declaration of a selector in the schema (block IV) extends the view of the "employee" entity to include the subset of "old employee". The case where a subset is to be derived as a reduced view of an entity is shown later within the declaration of a subschema.

With regard to the declaration of relationships (blocks V–VIII) the following explanatory comments can be made. One important purpose of the role name is to qualify those entities in a query which can not be uniquely identified by the values of their attributes. A role has the name of the entity as default value. Only in a recursive relationship, i.e. with both involved entities of the same type, it is imperative to specify distinct role names in order to distinguish between instances of the involved entity type. The association type can be expressed either in the general fashion "$M:N$", or by including explicit numbers to show the maximum number of entities that can be related. Since the association type is normally based on the primary keys of the involved entities, this information can be omitted in the declarations.

Declarations VII and VIII describe relationships which are substructures of another relationship. "Haul" and "visit" involve detailed entities and are therefore subsets of the general relationship "use". It can be noticed that the split relationships may have a more restricted association type than the original one. For example, "haul" expresses that each trucker drives his "own" truck.

All declarations of split relationships use only one level of nesting, independent of the level of the involved detailed entities. Block IX shows as an example the relationship between "electrical engineer" and "car". It is reasonable to refer only to the general relationship "use" because with this information, the path between the involved entities can be uniquely identified by the system.

The benefits of the defined DDL are that the extended modeling features are all consistently expressed by entities and relationships. The complexity of the declarations is controlled by using a hierarchical structure for entities and only one nested level for relationship description.

4.2 *Description with subschema DDL*

The need to provide the capability of declaring subschemas of the user schema was discussed earlier. In addition to allowing various users a view of reduced complexity and detail for their applications; the subschema concept also allows for the incorporation of security and integrity constraints for different classes of authorized users [5]. Furthermore, at the internal level, a different representation may be specified for a submodel to improve the performance of a particular application.

As suggested earlier, some of the conceptual transformations presented can be associated with the derivation of submodels. We shall now illustrate how objects can be derived in a subschema, representing higher-levels of abstraction than in the schema. The declarations in the subschema have basically the same format as in the schema. Note that the new derived objects are visible only to the concerned users.

The following declarations refer to the sample database illustrated in Fig. 9 and to the corresponding "company" schema. The statements do not relate to one single subschema, but rather illustrate how various subviews can be derived.

```
SUBSCHEMA VEHICLE_USAGE FROM COMPANY                    (I)
        :       :

ENTITY EMPLOYEE
FROM STATUS CLUSTER GENERALIZED TO EMPLOYEE             (II)
        :       :

ENTITY DRIVER                                           (III)
FROM TRUCKER GENERALIZED TO EMPLOYEE
        :       :

ENTITY EXPERIENCED                                      (IV)
FROM OLD EMPLOYEE
        :       :

RELATIONSHIP USE                                        (V)
FROM HAUL WITHIN USE
        :       :

ENTITY YOUNGSTER                                        (VI)
FROM EMPLOYEE
SELECT ID, NAME, AGE
WHERE  AGE  < 22
        :       :

ENTITY MANAGER_FUNCTION                                 (VII)
AGGREGATED FROM BELONG RELATIONSHIP
attribute declaration
```

Each subschema is identified by a name and a reference to its schema (block I). The declarations II and III describe generalized entities according to the two transformations presented in Fig. 5. Notice that "EMPLOYEE" in the first line of block II refers to a subschema object, whereas in the second line it refers to the object declared in the schema. Declaration IV declares an entity which is derived from the selector substructure of the schema.

Further, by declaration V only the view of the split relationship "haul" obtained from the general relationship "use" is maintained in the subschema, thus in fact omitting the association between the other employee types and vehicle types.

The selection phrase, following in the sample description (block VI), illustrates how a subview can be created for which there is no equivalent counterpart in the schema.

An object derived as an aggregate of a relationship available in the schema is given in declaration VII, corresponding to the transformation shown in Fig. 6. In this last case, the declaration also has to include the attributes which the user is interested in knowing.

5. REPRESENTATION AT THE INTERNAL LEVEL

The user schema declared for a given database must be translated into an internal schema, in which actual second types, access paths, placing of records, etc. are specified. We have opted to use the DBTG model at the internal level because it includes all the necessary features to accomplish this task, and it also provides a well-defined interface with the E-E-R model. Thus, if we restrict ourselves to a subset of the DBTG modeling features, the guidelines presented here can be used either by a Data Base Administrator (DBA) to perform a manual translation from an E-E-R schema to a DBTG executable schema, or with slight extensions can be incorporated in an automatic translator.

Chen [6] has given some basic translation rules for converting an E-R diagram into a Data-Structure diagram which represents the corresponding DBTG sets. Figure 10 summarizes these rules in a self-explanatory manner. It should be added that attributes of a relationship are translated into data fields of the member record type in the case of a *1:m* relationship and into data fields of the intermediate record type in the case of an *m:n* relationship.

We now shall consider how to extend these rules to cover the structures which are available in the E-E-R diagram presented earlier. The one consideration we are governed by in choosing the corresponding internal structure is to save storage space by eliminating the redundant information which is inherent in a model representing different levels of abstraction. Accordingly, no information is duplicated, which is also consistent with the property of the E-E-R schema.

The proposed translation of a generic hierarchy is shown in Fig. 11. Each cluster becomes a set type with the general entity as owner and the detailed entities as members.

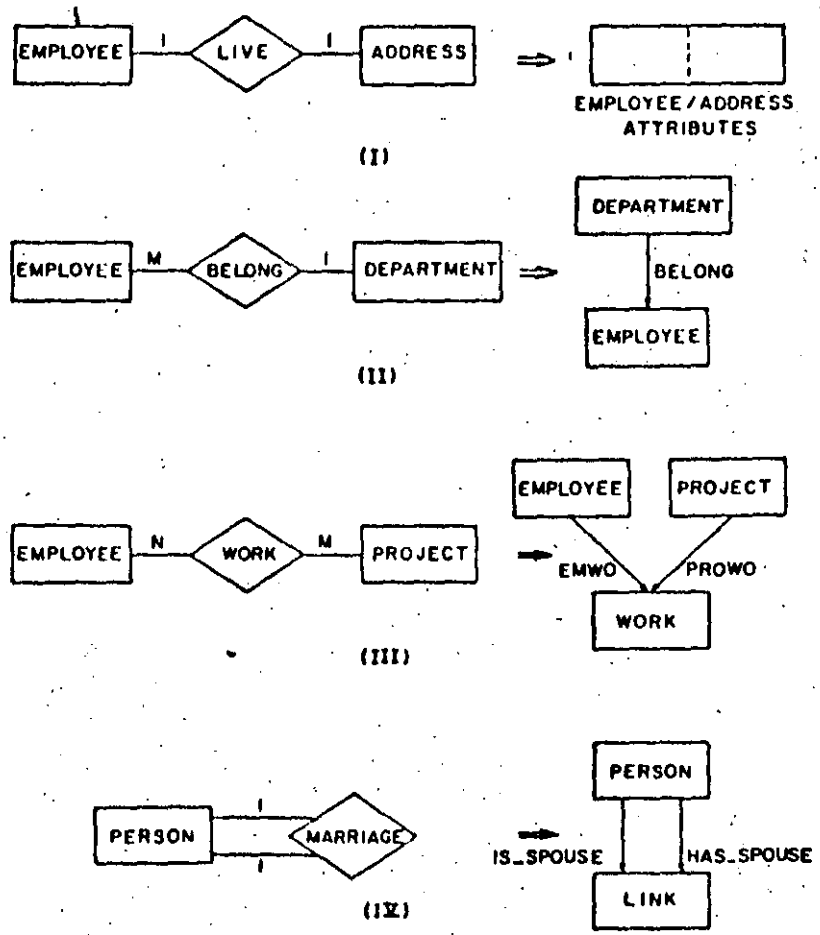


Fig. 10.

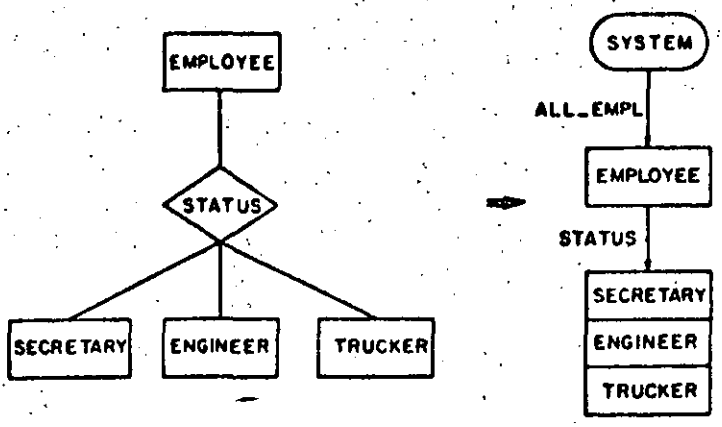


Fig. 11.

According to the DBTG specifications [17], multiple member-record types are allowable in a single set type. Note, however, that each set instance has only one member instance. In addition, to enhance the support of multiple views, i.e. to provide for faster access to the substructure, a singular set is created on the record type corresponding to the general entity. Note that this access path is also available for the other possible clusters declared on this entity. On the other hand, a substructure created by a selector does not introduce any new record types, as shown in Fig. 12, but a singular set is created for this record type.

Let us examine in more detail why Fig. 11 is an appropriate DBTG set structure for the generic hierarchy. First we note that allowing only a single member type in a set would not bring us any advantage since each set has in any case only one member instance. Another possible representation would be to merge the detailed record types with the general one, thus creating multiple record types on a higher level. This, however, is not appropriate since it would necessitate the creation of a number of new set types, instead of the one we have eliminated. Further, we observe that in order to retrieve an instance of a detailed entity in a generic hierarchy, its identifier (primary key) has to be "passed down" from the general entity. Therefore, the creation of a singular set on the detailed record type could mean fewer set accesses only if the majority of queries are of a statistical nature (e.g. find average typing skill of secretaries).

Similarly, relationships involving detailed entities do not cause the creation of additional DBTG sets by default. Thus Fig. 13 shows how the relationship "VISIT" declared in the schema "COMPANY" is to be realized within the existing access paths. Because of the hierarchical access to detailed record types the same number of set accesses would be required for a retrieval if an additional connection was created, i.e. another intermediate record type and two set types.

In order to obtain an executable network schema, a DBA would have to specify a number of other implementation-oriented parameters in addition to record and set types. The declaration of record types has to be augmented with a LOCATION MODE and AREA clause to specify some information about placing and retrieving of records.

In the declaration of set-types, we also have to include an ORDER clause to specify the sequencing of member-record occurrences, a MEMBERship class regarding the rules

Fig. 12.

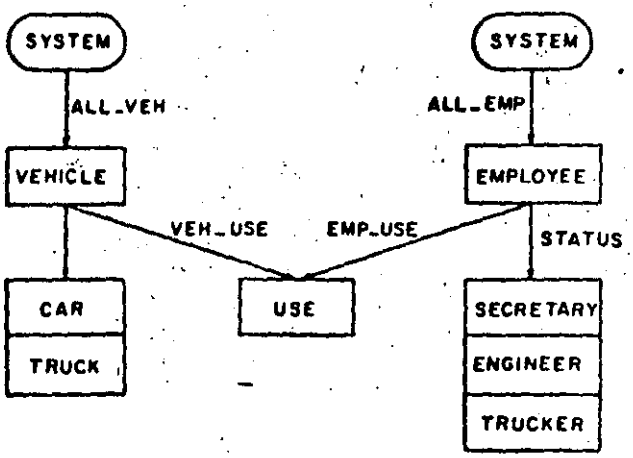


Fig. 13.

to be followed for insertion/deletion, and finally, a SET SELECTION clause to permit the automatic selection of set-occurrences when necessary [17]. Following is a partial description of the DBTG schema corresponding to our sample data-base:

```
RECORD NAME IS PROJECT
  LOCATION MODE IS VIA MANAGE SET
  WITHIN COMPANY AREA
  02 PROJ#  PIC '9999'
  02 TITLE PIC "X(9)"
  02 FIELD PIC "X(6)"
  02 TERM_DATE PIC "99999"
          .
          .
          .
SET NAME IS MANAGE
  OWNER IS EMPLOYEE
  ORDER IS PERMANENT SORTED BY DEFINED KEYS
    DUPLICATES ARE NOT ALLOWED
MEMBER IS PROJECT MANDATORY MANUAL
  KEY IS ASCENDING PROJ#
  SET SELECTION IS THRU MANAGE
    OWNER IDENTIFIED BY CALC_KEY.
```

To facilitate an automatic translation the following default steps can be followed, using only a subset of the above mentioned DBTG features:

(i) use only one AREA.
(ii) for record types which are only owners, specify "LOCATION MODE IS CALC USING primary key", otherwise specify "LOCATION MODE IS VIA set-name".
(iii) use MANDATORY MANUAL membership.
(iv) for member-record types which have been derived from corresponding entities at the E-E-R level, specify an ORDER clause with member occurrences sorted by the corresponding primary keys; for the other member-record types use an IMMATERIAL ordering clause.
(v) use a SET SELECTION clause through the identifier of the owner or through the SYSTEM depending upon the LOCATION MODE chosen in step (ii).

This straightforward algorithm will work correctly in almost all cases, but will have to be modified to take case of situations such as that in [17], where a more complex set-selection clause is to be generated.

In addition to the automatic translation process explained above, it should obviously be possible for the DBA to perform some optimization of the generated network schema in order to improve its performance. Based upon knowledge of the use of the data base, the DBA might want to include additional singular sets, create indices or allow multiple areas, etc. To eliminate the need for an additional translation, one possibility is for the DBA to enter these changes before the actual translation, with a special preprocessor providing for this high-level interface.

6. SUMMARY

The simplicity of the E-R model and its semantic structure, resembling that of sentences in natural language, make it a suitable choice for the conceptual schema, providing a stable picture of the information in a given enterprise. In order also to use the model for the external (user) schema we have shown how its modeling features can be extended to support multiple user views, i.e. different levels of abstraction. The notion of abstraction, based on the decomposition of objects in a hierarchic fashion, can be represented in an Extended-Entity Relationship diagram, while maintaining a graphical representation of manageable complexity. The conceptual transformations, which allow for more or fewer details of information to be available, can be associated with either external schema or external subschema declarations. The data definition language proposed makes use of the implicit hierarchical structure of entities to achieve concise declarations, with only one level of nesting. Finally, we have shown how the E-E-R schema can be translated into an internal schema, by restricting ourselves to a subset of the modeling features in a DBTG-like system. This presents the advantage that casual users can express queries in a language close to English based on the E-E-R model, and these are then translated into a sequence of calls to navigational procedures available in an existing DBTG system.

REFERENCES

1. ANSI/X3/SPARC, Study Group on Data Base Management Systems: Interim Report, ANSI (February 1975).
2. C. W. Bachman, Trends in data base management-75, *Proc. AFIPS 1975 NCC*, Vol. 44, AFIPS Press, Montvale, N.J., pp. 569-576.
3. H. Biller and E. Neuhold, Concepts for the conceptual schema, in *Architecture and Models in Data Base Management Systems*, G. M. Nijssen (ed.), pp. 1-30, North Holland (1977).
4. J. Bubenko, The temporal dimension in information modelling, in *Architecture and Models in Data Base Management Systems*, G. M. Nijssen (ed.), pp. 93-118, North Holland (1977).
5. D. D. Chamberlin, J. N. Gray and I. L. Traiger, Views, authorization, and locking in a relational data base system, *Proc. AFIPS 1975 NCC*, Vol. 44, pp. 425-430, AFIPS Press, Montvale, N.J.
6. P. P. Chen, The entity-relationship model: toward a unified view of data, *ACM Trans. Database Systems*, 1, (1) 9-36 (March 1976).
7. P. P. Chen, The Entity-Relationship model—A basis for the enterprise view of data, *Proc. AFIPS 1977 NCC*, Vol. 46, pp. 77-84, AFIPS Press, Montvale, N.J.
8. E. F. Codd, A relational model of data for large shared data banks, *Comm. ACM*, 13, 377-387 (1970).
9. C. J. Date, *An Introduction to Data base Systems* (second edition), Addison-Wesley (1977).
10. T. I. M. Ho, New perspectives for information systems education, *Proc. AFIPS 1977 NCC*, Vol. 46, pp. 569-573, AFIPS Press, Montvale, N.J.
11. L. Kerschberg, A. Klug and Tsichritzis, A Taxonomy of Data Models, in *Systems for Large Data Bases*, Lockemann, P. C. and Neuhold, E. J. (eds.), pp. 43-64, North Holland (1976).
12. G. M. Nijssen, Current issues in conceptual schema concepts, in *Architecture and models in Data Base Management Systems*, G. M. Nijssen (ed.), pp. 31-65, North Holland (1977).
13. G. M. Nijssen, On the gross architecture for the next generation database management systems, *Proc. IFIP Congress 1977*, pp. 327-335, North Holland.
14. G. Schiffner, Design of a query language based on an extended entity-relationship model, technical report, Northwestern University, Dept. of Electrical Engineering and Computer Science (1978).
15. J. M. Smith and D. C. P. Smith, Database abstractions: aggregation, *Comm. ACM* 20, 405-413 (1977).
16. J. M. Smith and D. C. P. Smith, Database abstractions: aggregation and generalization, *ACM Trans. Database Systems*, 2, 105-133 (1977).
17. R. W. Taylor and R. L. Frank, CODASYL database management systems, *ACM Comput. Surveys*, 8, 67-103 (1976).
18. K. A. Robinson, An analysis of the uses of the CODASYL set concept, in *Data Base description*, Douque, B. C. M. and Nijssen, G. M. (eds.), North Holland (1975).
19. H. Weber, The D-graph model of large shared data bases: a representation of integrity constraints and views as abstract data types, IBM Research Report RJ 1875 (Nov. 1976).

About the Author—Gerd Schiffner was born in Germany in 1951. He received his Dipl. Inform. degree from the Technische Universität Berlin in 1977 and his M.S. in Computer Science from Northwestern University in 1979.

From 1973 to 1976 he worked at Census related institutions on the development of an information system for regional planning purposes, and is currently continuing his work in this area.

About the Author—Peter Scheuermann received the B.S. degree in Applied Mathematics from Tel-Aviv University in 1969 and the M.S. and Ph.D. degrees in Computer Science from S.U.N.Y. at Stony Brook in 1972 and 1976 respectively. From 1975 to 1976 he taught at the College of William and Mary in Williamsburg, and since then he has been on the faculty of Northwestern University where he is an Assistant Professor in the Department of Electrical Engineering and Computer Science. He has authored over a dozen articles in the areas of data structures and programming languages, database systems and performance evaluation and is Associate Editor for the journal of Simulation.

Computer Languages, Vol. 4, pp. 155 to 169

0096-0551/79/1201-0155$02.00/0

17

TESTING AND VERIFICATION ASPECTS OF PASCAL-LIKE LANGUAGES

ANTHONY I. WASSERMAN

Medical Information Science, University of California, San Francisco,
San Francisco, CA 94143 U.S.A.

(*Received* 29 *November* 1978)

Abstract—This paper addresses aspects of programming language design that affect the ease with which programs written in a language can be subjected to systematic testing and/or program verification. The discussion focuses of Pascal and on several languages that have been derived primarily from Pascal, particularly Euclid and PLAIN. Specific language issues addressed include translation-time checking, program readability, flow of control, support for program modularity, data flow, and program immutability. The relative ease of validating such programs is then determined by the style in which the programs are written. The paper presents some guidelines for writing programs in Pascal-like languages for testability and verifiability.

Programming languages Testing Verification Pascal Type checking Programming style Aliasing

1. INTRODUCTION

A MAJOR theme in current software engineering research and development is software reliability. One key aspect of this area focuses upon techniques for determining the correctness of programs, i.e. the extent to which they satisfy their specifications, either through testing or verification. Testing is a collection of activities that provides a practical demonstration of conformity between the program and the specification, based upon systematic selection of test cases, execution of program paths and segments, and inference based upon test results. Verification is a formal mathematical proof that the program conforms to its specification.

Both of these approaches have met with some success. Testing techniques successfully uncovered large numbers of program errors that could be fixed on a one-by-one basis; the vast majority of useful programs in existence today were developed in this manner [1]. More recently, automated tools for testing have been developed, providing environments for testing individual modules and for systematically executing the various paths of a program symbolically [2, 3]. Work has progressed toward developing a theoretical basis for testing, to complement the more pragmatic techniques of the past [4, 5]. Program verification techniques have come into practice more slowly. It is fair to say that the biggest impact of verification has been the *recognition* that one might want to verify a program, rather than actual program proofs.

There are indications that work in verification and testing are increasingly overlapping. Some of the symbolic execution systems, for example, were developed by groups with a professed interest in formal proofs of programs. Indeed, it has been suggested that future demonstrations of program correctness will draw on both techniques of testing and verification [6]. Accordingly, the term "validation" will be used in this paper to refer to both the testing and the verification approaches to determination of program correctness.

Program validation is extremely dependent upon other stages of the software development life cycle. For example, it is extremely difficult to prove anything about a program in the absence of a precise, unambiguous statement of what it is to do, an observation that has led to extensive work in the area of formal specifications [7–9]. Similarly, considerable attention has been given to programming language design and to programming methodology, since the structure and use of programming languages determines the ease with which validation can be carried out.

DISEÑO DE BASES DE DATOS

A METHOD FOR DEFINING INFORMATION STRUCTURES AND TRANSACTIONS IN CONCEPTUAL-SCHEMA DESIGN

MAYO, 1985

A METHOD FOR DEFINING INFORMATION STRUCTURES AND TRANSACTIONS IN CONCEPTUAL SCHEMA DESIGN

Hirotaka Sakai

Hitachi Institute of Technology , Hitachi, Ltd.
5-5-12,Minami-Aoyama, Minato-Ku, Tokyo 107, Japan

ABSTRACT

A method for defining the conceptual schema of databases is discussed within the framework of the ER(Entity-Relationship) model. Analyzing the semantic aspects of the ER schema based on the notions such as the normalization, the generalization, the instantiation, and the typification, the initial schema can be refined to have more natural and clearer meanings. We further analyze the processing requirements for the database and formalize them into the transaction descriptions which reflect the behavioral properties on the database. From the set of the transaction descriptions, the distribution of transaction workloads on the database is easily predicted.

1. Introduction

The objective of the logical design of databases is to construct a conceptual database that is an abstraction of a physical database. The formal description of a conceptual database is a conceptual schema. It is represented here within the framework of Chen's ER(Entity-Relationship) model[2].

First of all the conceptual schema should correctly reflect information structural requirements for the database. The entities and the relationships among the entities should be represented in it so that anyone can understand the meanings of the information structure in a clear and natural way.

Furthermore, in order to maintain the integrity in various applications environments, it is highly desirable to describe the behavioral properties under the processing of transactions which request query and/or update operations on the database. Santos-Neuhold-Furtado[7] and Scheuermann-Schiffner-Weber[8] investigated the characteristics of existence and operational constraints in the ER model. From the practical point of view, however, it is more useful that these constraints are directly reflected in the description of query and update requests to the database.

In this article, we propose a more systematic method for designing the conceptual schema outlined below.

(1) Refine the initial schema according to those notions such as the normalization, the generalization, the instantiation, and the typification.

(2) Formalize the query and update requests in the form of the transaction descriptions, and evaluate quantitatively the workload distribution which the transactions impose on the conceptual database.

2. The ER Model and the ER Diagram

In the ER model, information is represented using three conceptual elements : entities, relationships among entities, and values. The set of similar entities, similar relationships, and similar values in certain contexts are called an E set, an R set, and a V set respectively[2,3,4].

An E set is described in the form E(A1/V1, A2/V2, ..., An/Vn), where E is a name of the E set, Ai/Vi (i = 1,2,...,n) is an attribute/V set pair. The attribute Ai is a property of an E set, and is defined as a function from E into the V set Vi. When X = { A1, A2, ..., Ak} is a minimal set of attributes which gives a one-to-one mapping from E into the Cartesian product V1 x V2 x ... x Vk, we call X an identifier of E. The description of an E set is simply denoted E(A1, A2,...,Ak, Ak+1, ..., An) in which the identifier is underscored. If there exist more than one identifier, we arbitrarily select one and underscore it.

An R set is a set to relate several (not necessarily distinct) E sets E1, E2, ..., Em. It is a set of tuples (e1, e2, ..., em) of mutually related entities ei of Ei (i = 1,2,...,m). An R set is described in the form R(E1/L1, E2/L2, ..., Em/Lm : A1/V1, A2/V2, ..., An/Vn). R is a name of the R set. Ei/Li (i = 1,2,...,m) is an E set/role pair where Li is a name of the role that Ei plays in R. An R set may have attribute/V set pairs Aj/Vj (j = 1,2,...,n) too. In this case, the

CH1701-2/81/0000/0225$00.75 © 1981 IEEE

attribute Aj is a function from R into Vj. The simple version of the R set description is denoted R(E1, E2, ..., Em : A1, A2, ..., An).

The ER model is constructed as a set of E sets and R sets, and is illustrated in the ER diagram, in which the E set and the R set are represented by rectangular- and diamond-shaped boxes respectively as will appear in the examples.

Example 1 : the AIRLINE database (1)

Consider the conceptual schema shown in Figure 1, which is constructed of E sets FLIGHT, PASSENGER, and FDATE, and R sets RESERVE and AVAILABLE. The R set RESERVE represents the fact that a FLIGHT on a certain date (FDATE) has been reserved by a PASSENGER. The R set AVAILABLE has an attribute to indicate the number of available seats of a FLIGHT on a specific date. The formal description of E sets and R sets are given below.

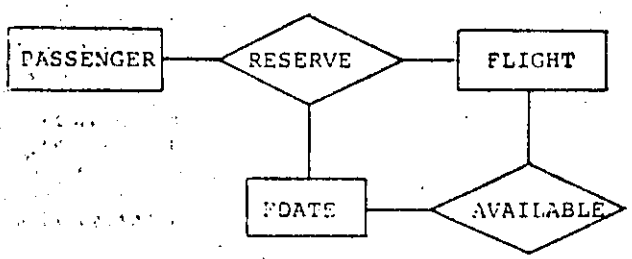


Figure 1 : ER Diagram of the AIRLINE Database (1).

[E sets]

PASSENGER(NAME, ADDRESS, PHONE)

FLIGHT(FLT#, SOURCE, DEST, DEP-TIME, ARR-TIME, PLANE-TYPE, SEAT-CAPACITY)

FDATE(DATE)

[R sets]

RESERVE(PASSENGER, FLIGHT, FDATE : NO-OF-SEATS)

AVAILABLE(FLIGHT, FDATE : NO-OF-SEATS-AVAILABLE)

3. Refinement Procedures of Schemas

The initial conceptual schema of the ER model is usually designed through the steps : (1) Recognize E sets and R sets of interest ; (2) Determine V sets and attributes for the E sets and R sets ; (3) Draw the ER diagram.

In the initial design, the recognition of E sets and R sets highly depends on the designer's intuition. However, by analyzing the initial schema from viewpoints of several notions such as the normalization, the generalization, the instantiation, and the typification, it is possible to refine the schema so that it carries the richer and the clearer meanings.

(1) The Normalization

This notion is similar to the normalization in the relational model, and is defined by the following sentence : "The conceptual schema in the ER model is said normalized if, for any E set E or R set R, (a) the attributes not included in the identifier of E or R are not functionally dependent on each other, and (b) they are fully functionally dependent on the identifier of E or R." The concept of the functional dependency and the full functional dependency between the sets of attributes in E or R are defined as those between the V sets or the Cartesian products of V sets associated with the attributes. In the normalization of an R set R, we take for the identifier of R the set of identifiers of the E sets which are related to each other by R.

The normalization process of the ER model is discussed in [4] and [5].

Example 2 : the AIRLINE database (2)

In the E set FLIGHT in the AIRLINE database (1), the attribute SEAT-CAPACITY is functionally dependent not only on the attribute FLT#, but also on the attribute PLANE-TYPE. We can normalize the conceptual schema by replacing FLIGHT with the new E sets FLIGHT and PLANE, and the R set ASSIGN as follows. The normalized schema is illustrated in Figure 2.

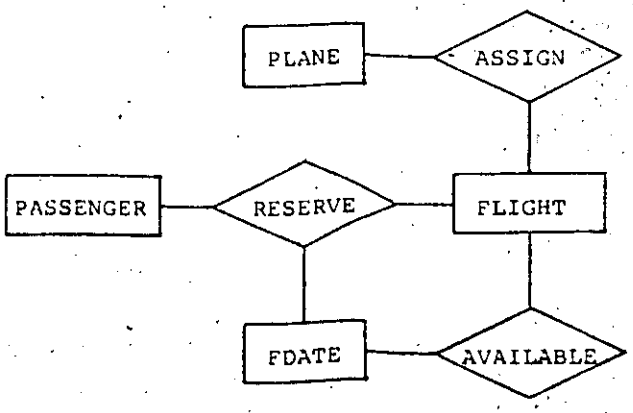


Figure 2 : ER Diagram of the AIRLINE Database (2).

[E sets]

FLIGHT(FLT#, SOURCE, DEST, DEP-TIME, ARR-TIME)

PLANE(PLANE-TYPE, SEAT-CAPACITY)

[R sets]

ASSIGN(FLIGHT, PLANE)

(2) The Generalization and the Specialization

The notion of the generalization introduced by Smith-Smith[10] in the relational model can also be applied to the ER model[9].

The generalization is an abstraction which enables the class of individual E sets under a certain category to be thought of generally as a single E set. Conversely the specialization is a partitioning of a single E set into several disjoint E sets under a certain category. In the ER diagram, as illustrated in Figure 3, categories are represented by ovals, and the generalization is indicated by an arrow from an oval to the E set that is generalized.

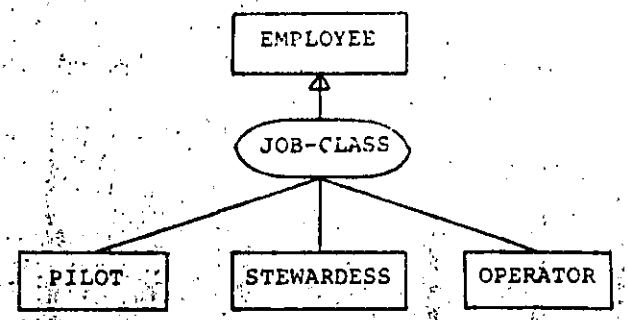


Figure 3 : ER Diagram Representing the Generalization.

Example 3

An E set EMPLOYEE is defined as a generalization of E sets PILOT, STEWARDESS, and OPERATOR in the airline company under the category JOB-CLASS. This fact is illustrated in Figure 3. Each E set that is a specialization of the generic one has the attributes inherited from the E set EMPLOYEE, coexistently with its proper attributes.

(3) The Instantiation

Suppose that two E sets E and F have many-to-many relationships in an R set R. For an entity e0 of E, the set $R/e0 = \{(e0,f) \mid f \in F, (e0,f) \in R\}$ is said the instance set of e0 by F with respect to R. The instantiation of the E set E is a procedure which treats R/e0 as the instance of e0, by taking R for an E set. The notion of the instantiation is particularly useful, when F is an E set representing time factors and it is convenient to treat entities (e0,f1), (e0,f2), ..., (e0,fk) ($fi \in F$) as the time series instances of an entity e0 of E.

The following is the formal procedure for the instantiation.

(i) When we take an element r of R for an entity, we denote it r^*. As a set of these forms, we can define an E set $R^* = \{r^* \mid r \in R\}$. Let X and Y be the identifiers of E sets E and F respectively, and Z be the set of attributes of an R set R. Then R^* has attributes X^*, Y^*, and Z^*, where, for $r = (e,f)$ of R, $X^*(r^*)$, $Y^*(r^*)$, and $Z^*(r^*)$ take the values X(e), Y(f), and Z(r) respectively.

(ii) The E sets E and R^* are related to each other by an R set INSTANCE-OF-E. This R set relates an entity e0 of E to its instance set by F of R^*, namely to the set $\{r^* \mid r \in R/e0\}$.

We call the E set R^* the instance set of E by F with respect to R.

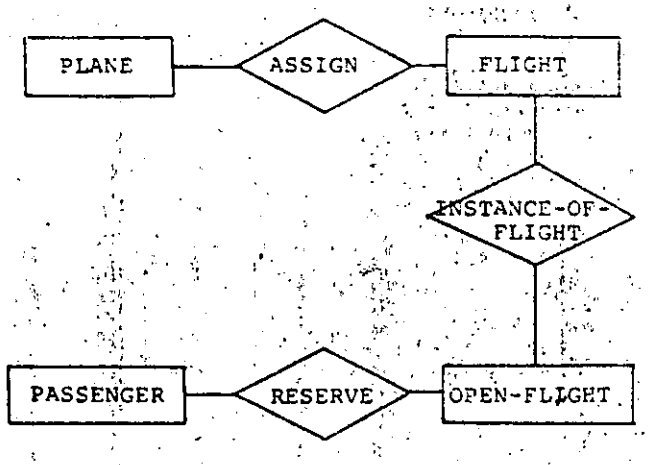


Figure 4 : ER Diagram of the AIRLINE Database (3).

Example 4 : the AIRLINE database (3)

We apply the instantiation to the R set AVAILABLE(FLIGHT, FDATE : NO-OF-SEATS-AVAILABLE) in the AIRLINE database(2). Namely, we define the instance set of FLIGHT by FDATE with respect to AVAILABLE. The R set AVAILABLE is replaced with the new E set OPEN-FLIGHT(FLT#, DATE, NO-OF-SEATS-AVAILABLE) which is related to FLIGHT by the R set INSTANCE-OF-FLIGHT. As the E set FDATE is included as an attribute in OPEN-FLIGHT.

we can delete FDATE from the schema. Next we redefine the R set RESERVE(PASSENGER, OPEN-FLIGHT : NO-OF-SEATS) relating PASSENGER to OPEN-FLIGHT. As Figure 4 shows, the conceptual schema is refined to have more natural meanings.

Example 5

The notion of the instantiation is applicable to the case where an E set representing time factors does not exist. In the schema of Figure 5, the R set MARKET stands for the "sell/sold" relationship between E sets SUPPLIER and PART, while the R set BUY represents that a PROJECT buys a PART from a SUPPLIER. In order that the element of BUY exists, there must exist an element of MARKET including relevant elements of SUPPLIER and PART.

From the fact that the R set MARKET is a many-to-many relationship between SUPPLIER and PART, we define the instance set of SUPPLER by PART and the instance set of PART by SUPPLIER with respect to MARKET respectively.

We thus obtain the schema shown in Figure 6, where PROJECT is related to the redefined E set MARKET by the redefined R set BUY. Through this refinement, the existence dependency of BUY on MARKET is illustrated in the ER diagram in a self-explanatory way.

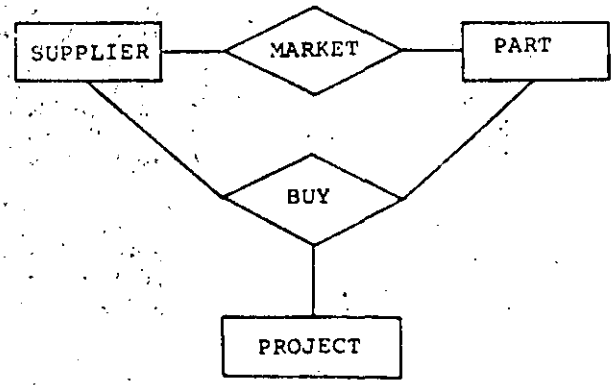


Figure 5 : ER Diagram of SUPPLIER-PROJECT-PART (1).

(4) The Typification

Suppose an E set is classified into several groups under a certain category. The typification is a procedure to define an E set the element of which holds information about the types of each group of E. The formal procedure is expressed as follows.

(i) Let an E set E be classified into disjoint subsets E1, E2, ..., Ek. This classification could obey the specialization, the instantiation, or any other notions of categories.

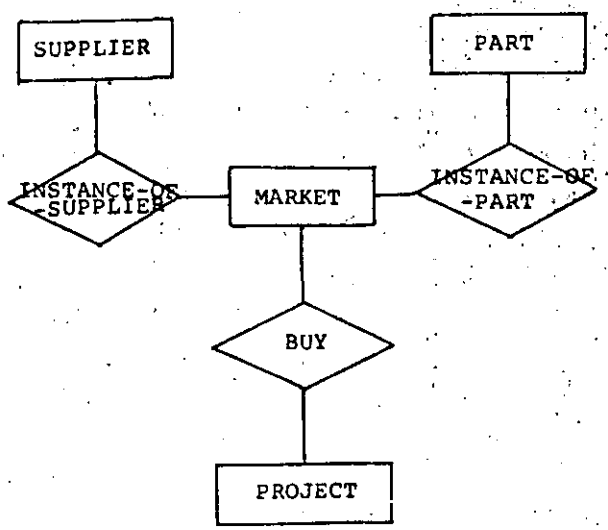


Figure 6 : ER Diagram of SUPPLIER-PROJECT-PART (2).

(ii) Define an E set $\bar{E} = \{\bar{e}i \mid i = 1,2,\ldots,k\}$ where the entity $\bar{e}i$ represents the type of each E set Ei. $\bar{e}i$ is an entity with properties which are common to or derivable from all the elements of Ei.

(iii) Link $\bar{E}$ and E by an R set TYPE-OF-E. This R set relates each $\bar{e}i$ to all the elements of Ei.

Example 6

In Example 3, the E set EMPLOYEE was classified, by the specialization, into the E sets PILOT, STEWARDESS, and OPERATOR. As a typification of the three E sets, we define the following E set EMP-TYPE.

EMP-TYPE(TYPE-ID, JOB-DESCRIPTION, NO-OF-EMP, MAX-SALARY, MIN-SALARY, AVERAGE-AGE).

The E set EMP-TYPE consists of three entities representing the types of E sets PILOT, STEWARDESS, and OPERATOR respectively.

Example 7

In the AIRLINE database (3), the E set OPEN-FLIGHT is classified into groups such that elements with the same value of FLT# belong to the same group. We can define the following E set FLT-RECORD as a typification of those groups.

FLT-RECORD(FLT#, AVERAGE-NO-OF-PASSENGERS, TOTAL-SALES, AVERAGE-COST-OF-FUEL).

On the other hand, the E set FLIGHT could also be seen as a typification of the same groups as above.

We can look upon FLIGHT-RECORD and FLIGHT as two different typifications of the same groups of OPEN-FLIGHT, one having dynamic properties while the other having static properties. In some situations, it would be convenient to combine them into a single E set. However, the meanings of types become clearer by treating them as separate E sets.

4. Transaction Analysis

A database varies with time. The semantic rules of changes are generally described in terms of integrity constraints. Among other things, the existence and operational constraints are essential factors to determine the behavioral properties on a database [1,7]. From practical viewpoints, it is desirable that these constraints are reflected directly in the descriptions of query and update requests to a database.

For this purpose, we formalize query and update requests in the form of the transaction descriptions. By this means, the characteristics of transactions such as types, precedences, and propagations of operations on the database are made clearer and more understandable. Furthermore, the transaction descriptions provide effective rules to measure the workloads on E sets and R sets imposed by the set of transactions.

The AIRLINE database (3) in Example 4 will often be referred to in this section.

4.1 Transaction Descriptions

Query and update requests to a database are called transactions. A transaction which refers to at most one R set on its access path is said a simple transaction. Any transaction T is decomposed into a sequence of simple transactions S1, S2, ..., and Sn. We write this fact T = S1 S2 ... Sn.

A simple transaction is described in either the primary or the secondary form.

(1) the primary form : S = op(U).

In this form, op and U are the type and the object of operations respectively. The form of op is any of the codes get, cr(create), del(delete), and mod(modify). The primary form has the following functional meanings depending on the op types.

(a) S = get(E).

E is a name of an E set . This form represents an operation to retrieve the subset of E which satisfies a specific condition on attributes of E.

(b) S = cr(E)

E is a name of an E set. This form represents an operation to add an entity or a set of entities to E.

In the primary form of (a) or (b), the subset of entities which are retrieved from or added to E are called the object space of S, and denoted S.

(c) S = del(S'), or S = mod(S').

This form stands for an operation to delete or modify the object space S' of a simple transaction S'. S' could be an object space of the secondary form to be defined later. In this case, S' is a subset of an E set or an R set.

Example 8

Let Ta be the update transaction stating, "Reduce the NO-OF-SEATS-AVAILABLE of 'JL001' on 'October 10' to zero."

Then Ta is described in the form :

Ta = Sa1 Sa2

where Sa1 = get(OPEN-FLIGHT), and Sa2 = mod(Sa1).

(2) the secondary form :

S = S1 S2 ..... Sk(R) : op(U).

In this form, op and U represent the type and the object of operations. The code for op is either get or cr. Si (i = 1,2,...,k) is the simple transaction which should have been executed before S, and Si is its object space. R is a name of the R set to be referred to. Si itself could be a secondary form. In such a case, the object space Si also represents the set of elements which have been retrieved or created by the simple transaction Si.

The meaning of the secondary form is as follows.

(a) S = S1 S2 ..... Sk(R) : get(U).

U is a name of an E set or an R set. The object of this operation is the E set U which is related to S1, S2, ..., and Sk by R, or the R set U = R which includes S1, S2, ..., and Sk as participants in R. Making reference to R, the subset of U which satisfies a specific condition on attributes of U is retrieved by this operation.

(b) S = S1 S2 ..... Sk(R) : cr(R).

The effect of this form is to generate a relationship or a set of relationships which relate S1, S2, ..., and Sk in terms of R, and add it to R.

Example 9

Consider the reconfirmation transaction Tb stating, "Reconfirm the reservation status of 'JL001' on 'December 10' for PASSENGER 'A'." This transaction requires to retrieve the relevant entities from PASSENGER and OPEN-FLIGHT respectively, and

retrieve the relationship of RESERVE which relates them. Thus we have the following description:

Tb = Sb1 Sb2 Sb3.

The three simple transactions are given by :

Sb1 = get(PASSENGER)
Sb2 = get(OPEN-FLIGHT)
Sb3 = Sb1 Sb2(RESERVE) : get(RESERVE).

Example 10

Let Tc be the reservation request transaction stating, "Reserve 'AF003' on 'November 3' for PASSENGER 'B'."

The transaction Tc should be processed according to the procedure :

(i) Search for NO-OF-SEATS-AVAILABLE in OPEN-FLIGHT.

(ii) If the seats are available, add 'B' to PASSENGER.

(iii) Add a relationship relating the entity 'B' of PASSENGER with the entity of OPEN-FLIGHT to RESERVE.

(iv) Modify (i.e. reduce by one in general) NO-OF-SEATS-AVAILABLE of OPEN-FLIGHT.

We have the following descriptions :

Tc = Sc1 Sc2 Sc3 Sc4

where the simple transactions are given by :

Sc1 = get(OPEN-FLIGHT)
Sc2 = cr(PASSENGER)
Sc3 = Sc1 Sc2(RESERVE) : cr(RESERVE)
Sc4 = mod(Sc1).

Example 11

Let Td be the cancellation transaction stating, "Cancel all the reservations made by PASSENGER 'C'."

This transaction should be processed according to the following procedure.

(i) Retrieve PASSENGER 'C'.

(ii) Retrieve all the entities of OPEN-FLIGHT that were reserved for PASSENGER 'C'.

(iii) Retrieve the relationships of RESERVE relating 'C' and the entities of OPEN-FLIGHT retrieved in (ii), and,

(iv) Delete these relationships.

(v) Modify (i.e. raise) NO-OF-SEATS-AVAILABLE of OPEN-FLIGHT.

(vi) Delete 'C' from PASSENGER.

The transaction description of Td is :

Td = Sd1 Sd2 Sd3 Sd4 Sd5 Sd6

where the simple transactions are given by :

Sd1 = get(PASSENGER)
Sd2 = Sd1(RESERVE) : get(OPEN-FLIGHT)
Sd3 = Sd1 Sd2(RESERVE) : get(RESERVE)
Sd4 = del(Sd3)
Sd5 = mod(Sd2)
Sd6 = del(Sd1).

Suppose the transaction T is decomposed into a sequence of simple transactions. If each of the simple transactions in this sequence is of type get, then we say T is a query transaction. If a simple transaction of operation type other than get is included in the sequence, we say T is an update transaction. In the conceptual schema design of databases, it is neither necessary nor possible to prepare detailed descriptions of transactions. The transaction description mentioned above provides sufficient information on the behavioral properties of the conceptual schema.

4.2 Transaction Workloads

Transactions impose workloads on the database. We can consider that the workload of a transaction is distributed among E sets and R sets. From the transaction descriptions, the workload distribution can be determined quantitatively.

Suppose the transaction T is described in the form:

T = S1 S2 ..... Sn

The workload of T depends on the following three factors.

(i) The complexity of T, representing the complexity of the process to be executed. For this measure, we use the length of the transaction, i.e. the number n of the simple transactions in the transaction description.

(ii) The volume of T, the measure of which is given by the frequency f of T that occurs per unit of time.

(iii) The responsiveness of T, or the response time required for T. For this measure, we use the quantity r which takes high values as the response time requirement gets severe.

Now we define the workload of T, denoted L(T), as

L(T) = n x r x f.

We assume L(T) is uniformly divided among the simple transactions S1, S2, ..., and Sn. As the

object of each Si operation is an E set, an R set, or a subset of them, we can define the transaction workload imposed on E sets and R sets as follows.

(a) For a primary form Si = op(U), where U is an E set or an R set, the workload of Si on U, denoted w(U,Si), is defined as w(U,Si) = r x f.

(b) For a secondary form Si = Si1 Si2 .... Sik(R) : op(U), where U stands for an E set or an R set, the workload of Si on U under the reference to R, denoted w(U/R, Si), is defined as w(U/R, Si) = r x f.

(c) The total workload on U, denoted W(U), is defined as the sum of quantities w(U, Si) for all primary forms Si's which operate on U. In the same way, we define the total wokload on U under the reference to R, denoted W(U/R), as the sum of quantities w(U/R, Si) for all the secondary forms Si's which operate on U under the reference to R.

The set of W(U) and W(U/R) for all U's serves for clearly understanding and evaluating the distribution of the transaction workloads on the conceptual database.

Example 12

Let us consider the distribution of transaction workloads on the AIRLINE database (3) imposed by transactions Tb, Tc, and Td in the previous examples. We assume the responsiveness of the three types of transactions is uniform (for example, all of them are on-line transactions), and put r = 1 for each transaction. As for the volume of the transactions, we assume the frequencies of Tb, Tc, and Td are 30, 50, and 5(thousand transactions per day) respectively. As Tb, Tc, and Td have been decomposed into 3, 4, and 6 simple transactions respectively, we have by definition L(Tb) = 90, L(Tc) = 200, and L(Td) = 30.

The distribution of the transaction workloads on the AIRLINE database (3) is shown in Figure 7. The total workloads on each E set and R set are, as listed in the bottom line of Figure 7, W(PASSENGER) = 90, W(OPEN-FLIGHT) = 130, W(OPEN-FLIGHT/RESERVE) = 10, and W(RESERVE/RESERVE) = 90 respectively.

	PASSENGER	OPEN-FLIGHT	OPEN-FLIGHT/ RESERVE	RESERVE/ RESERVE
Tb[Reconfirm]	Sb1 : 30	Sb2 : 30		Sb3 : 30
Tc[Reserve]	Sc2 : 50	Sc1&Sc4 :100		Sc3 : 50
Td[Cancel]	Sd1&Sd6 : 10		Sd2&Sd5 : 10	Sd3&Sd4 : 10
W[Total Workloads]	90	130	10	90

Figure 7 : Distribution of the Transaction Workload on the AIRLINE Database (3).

We can evaluate the effectiveness of the schema refinements from the viewpoint of the transaction workload. The following is an example to show a comparison of two schemas, before and after the instantiation procedure, in terms of the transaction workload.

Example 13

Let us compare the total workloads of two schemas, the AIRLINE database (2) in Example 2 and the AIRLINE database (3) in Example 4. The latter was obtained from the former through the instantiation procedure. We shall consider the same transactions Tb, Tc, and Td as appeared in Example 12. Namely Tb, Tc, and Td have the responsiveness parameter of

the same value r = 1, and volume parameters f = 30, f = 50, and f = 5 respectively.

The total workloads of these transactions on the AIRLINE database (3) have been evaluated in Example 12.

In order to calculate the workloads on the AIRLINE database (2),we first decompose the transactions into sequence of simple transactions. This could easily be done as follows.

	PASSENGER	FDATE	FLIGHT	FDATE/ RESERVE	FLIGHT/ RESERVE	AVAILABLE/ AVAILABLE	RESERVE/ RESERVE
Tb[Reconfirm]	Sb1 : 30	Sb2: 30	Sb3: 30				Sb4 : 30
Tc[Reserve]	Sc4 : 50	Sc1: 50	Sc2: 50			Sc3&Sc6:100	Sc5 : 50
Td[Cancel]	Sd1&Sd8 : 10			Sd3 : 5	Sd2 : 5	Sd4&Sd7: 10	Sd5&Sd6 : 10
W[Total Workload]	90	80	80	5	5	110	90

Figure 8 : Distribution of the Transaction Workloads on the AIRLINE Database (2).

the AIRLINE Database (2) (Before the Instantiation)		the AIRLINE Database (3) (After the Instantiation)	
W(PASSENGER) =	90	W(PASSENGER) =	90
W(FDATE) =	80		
W(FLIGHT) =	80		
W(FDATE/RESERVE) =	5		
W(FLIGHT/RESERVE) =	5	W(OPEN-FLIGHT/RESERVE) =	10
W(AVAILABLE/AVAILABLE) =	110	W(OPEN-FLIGHT) =	130
W(RESERVE/RESERVE) =	90	W(RESERVE/RESERVE) =	90
Sum of Total Workloads =	460	Sum of Total Workloads =	320

Figure 9 : Comparison of the Total Workloads between the Two Databases.

(i) the reconfirmation transaction : Tb

Tb = Sb1 Sb2 Sb3 Sb4

where we have :

Sb1 = get(PASSENGER)
Sb2 = get(FDATE)
Sb3 = get(FLIFHT)
Sb4 = Sb1 Sb2 Sb3(RESERVE) : get(RESERVE).

(ii) the reservation request transaction : Tc

Tc = Sc1 Sc2 Sc3 Sc4 Sc5 Sc6

where we have :

Sc1 = get(FDATE)
Sc2 = get(FLIGHT)
Sc3 = Sc1 Sc2(AVAILABLE) : get(AVAILABLE)
Sc4 = cr(PASSENGER)
Sc5 = Sc1 Sc2 Sc4(RESERVE) : cr(RESERVE)
Sc6 = mod(Sc3).

(iii) the cancellation transaction : Td

Td = Sd1 Sd2 Sd3 Sd4 Sd5 Sd6 Sd7 Sd8

where we have :
Sd1 = get(PASSENGER)
Sd2 = Sd1(RESERVE) : get(FLIGHT)
Sd3 = Sd1 Sd2(RESERVE) : get(FDATE)
Sd4 = Sd2Sd3(AVAILABLE) : get(AVAILABLE)
Sd5 = Sd1 Sd2 Sd3(RESERVE) : get(RESERVE)
Sd6 = del(Sd5)
Sd7 = mod(Sd4)
Sd8 = del(Sd1).

From these sequences, we have the distribution of transaction workloads on the AIRLINE database (2) as shown in Figure 8.

Figure 8 (for the AIRLINE database (2)) and Figure 7 (for the AIRLINE database (3)) distinctly illustrate how the transaction workloads are distributed over each of the two databases. Figure 9 shows the comparison of the total workloads between the two, suggesting the effectiveness of the instantiation quantitatively.

5. Conclusion

A method for the conceptual schema design within the framework of the ER model is proposed. The following procedure make the basis of the method.

(1) The refinement of schemas based on the notions such as the normalization, the generalization, the instantiation, and the typification.

(2) The formalization of the transaction description which not only reflects the existence and operational constraints, but also provides means for the measurement of transactions workload.

These procedures are practically applicable to the logical design of databases as a meaningful method. Furthermore, the transactions workload which represents the processing requirements for databases quantitatively, can be used as a useful measurement in the detailed logical design such as the transformation from the ER model to other data models underlying target DBMS's[5,11].

We consider the conceptual schema should describe not only characteristics of the information structure, but also characteristics of transactions. In this sense, two types of descriptions should be consolidated into a single schema. We intend to evolve the procedures to a design tool with a comprehensive schema definition language to express the semantic aspects of databases.

REFERENCES

[1] G.Bracchi, A.Furtado, G.Pelagatti : Constraint Specification in Evolutionary Database Design, in : Formal Models and Practical Tools for Information Systems Design, pp.149-165(North-Holland, Amsterdam, 1979).

[2] P.P.Chen : The Entity-Relationship Model : Towards a Unified View of Data, Transactions on Database Systems Vol.1, No.1 pp.9-36(1976).

[3] P.P.Chen : The Entity-Relationship Model : A Basis for the Enterprise View of Data, AFIPS Conf. Proc. Vol.46, AFIPS Press, pp.77-84 (1977).

[4] H.Sakai : On the Optimization of the Entity-Relationship Model, 3rd USA-JAPAN Computer Conf. Proc. pp.145-149 (1978).

[5] H.Sakai : A Unified Approach to the Logical Design of a Hierarchical Data Model, in : Entity-Relationship Approach to Systems Analysis and Design, pp.61-74 (North-Holland, Amsterdam, 1980).

[6] H.Sakai : Entity-Relationship Approach to the Conceptual Schema Design, Proc. ACM-SIGMOD 1980, Santa Monica, California, pp.1-8 (1980).

[7] C.S.dos Santos, E.J.Neuhold, A.L.Furtado : A Data Type Approach to the Entity-Relationship Model, in : Entity-Relationship Approach to Systems Analysis and Design, pp.103-119 (North-Holland, Amsterdam, 1980).

[8] P.Scheuermann, G.Schiffner, H.Weber : Abstraction Capabilities and Invariant Properties Modelling within the Entity-Relationship Approach, in : Entity-Relationship Approach to Systems

Analysis and Design, pp.121-140(North-Holland, Amsterdam, 1980).

[9] G.Schiffner, P.Scheuermann : Multiple Views and Abstractions with an Extended-Entity-Relationship Model, Journal of Computer Language Vol.4, pp.139-154 (1979).

[10] J.M.Smith, D.C.Smith : Database Abstractions : Aggregation and Generalization, Transactions on Database Systems Vol.2, No.2 pp.105-133 (1977).

[11] J.T.Teorey, J.P.Fry : The Logical Record Access Approach to Database Design, ACM Computing Surveys Vol.12, No.2 pp.179-211 (1980).

INFORMATION SYSTEMS DESIGN APPROACH INTEGRATING DATA AND TRANSACTIONS

M. Léonard and B.T. Luong

Centre universitaire d'informatique
Département d'économie commerciale et industrielle
University, Geneva, Switzerland

Abstract

Our objective is to show clearly the interdependance between the study of data and the study of transactions in information systems design. We have used Codd's model to describe the data and obtain their possible partitionings, and we have adapted Petri's model to model the transactions.

We have shown how the parallelism of the transactions depends on the partitioning of the data and how one partitioning allows more parallelism than another.

Introduction

Our approach is part of the general framework of information systems design.

Previous studies [4], [6] have already demonstrated the importance of the semantic properties of the information during a first stage; indeed, efficacious data shemata can be deduced from them, as current research work has proven. These data schemata can then be described either in terms of a data description language for database software [2], [19], or terms of files.

These studies do not take into account the transactions. We are interested in the way the study of the transactions could contribute to an improved information system design and more particularly concentrated on the transactions which can be executed in parallel; from this, we have deduced a data structure which, on the one hand, is compatible with the preceding results and, on the other hand, satisfies the requirements of parallelism as much as possible.

Some research work has already been done on the relationships between the data and the transactions of an information system. Part of it was concerned with the specifications of the transactions through the semantic relations existing between the data [6], [1]; others proposed a general description of the information system [12] by taking into consideration the aspects of parallelism and the synchronization of the transactions.

[25] is more specifically interested in the events concept and the triggering of transactions by events. By introducing the chronology concept, [14], [13] isolate an information typology and a transaction typology, and describe an algorithmic method for the design of a data structure verifying certain conditions. [11], [17] and [28] are interested in the sequencing of transactions, taking into account the aspects of synchronization, parallelism and choice between transactions; to obtain this goal, Petri's model, already used in various fields such as in modelling of complex systems [27] and detection of system failures [24] was also applied here.

The same perspective was followed in our research work but, at the same time, we tried to make the connection between the modelling of transactions and the modelling of data.

Remark: our study was limited to a centralized information system whose data are stored and managed by the same software, which can be either database software or ad hoc software.

1. DATA SEMANTICS

We use Codd's relational model [4] and present below only its concepts which are useful to our paper and which have been described in [7] and [9].

1.1 Basic concepts

An attribute is the name of a certain type of data. To each attribut is associated a domain which is a non empty set of objects relative to the attribute.

If a is an object of attribute A, one conventionally denotes $a \in A$.

A relation R is a set of attributes $X = \{A_1 \ldots A_n\}$ and a predicate defined over X; this predicate, denoted $\|R(A_1 \ldots A_n)\|$, where attributes $A_1 \ldots A_n$ act as variables, expresses the intension which connects those attributes. R designates the name of the relation.

If $a_1, a_2 \ldots a_n$ designate objects of $A_1, A_2 \ldots A_n$ respectively,
and if $\|R(a_1 \ldots a_n)\|$ = true, then

$x = (a_1, a_2 \ldots a_n)$ is an entity of R. One denotes $x \in R$.

CH1701-2/81/0000/0235$00.75 © 1981 IEEE

12

ON THE OPTIMIZATION OF AN ENTITY-RELATIONSHIP MODEL

HIROTAKA SAKAI
Hitachi Institute of Technology, Hitachi, Ltd.
Tokyo, Japan

The methodology to construct a stable and universal data structure, or a conceptual schema, is required as a basis for the logical design of databases. An entity-relationship model could be used for this purpose. An optimization procedure of this model from information structural view points is formulated based on the dependency structures in attributes, entity sets, and relationship sets. The restructuring processes consisting of two phases are proposed. The model is locally optimized in phase 1 based on the analysis of functional dependencies in attributes. In phase 2, a global optimization is performed analyzing hierarchical and transitive dependencies in relationship sets.

1. INTRODUCTION

The essential problem in the logical design of databases is to construct an optimized data administrator's view of data integrated from various users' views. The integrated logical data structure should be optimized from view points of structural characteristics and usage characteristics of information. The optimization from information structural view points is more essential than that from information usage view points. Because the former provides a stable and universal data structure source, while the latter depends on the usage requirements in actual application environments.

As a practical logical design aid, we shall use Chen's entity-relationship model (E-R model) to represent a data structure, and formulate an optimization methodology of this model from information structural view points.

2. THE ENTIRY-RELATIONSHIP MODEL

2.1 The formalization of the E-R model description

The E-R model was proposed by P.P. Chen. The detailed definition of the model and the step-by-step approach to construct it are described in [1] and [2]. We shall first try to formalize the description of the model as a basis for the discussion to be introduced later on.

(1) The entity set
An entity is a thing which can be distinctly identified. Entities can be classified into different entity types such as PART, PROJECT, and EMPLOYEE. An entity set is a group of entities of the same type. An entity set is described in the form

$$E\ (A_1/V_1,\ A_2/V_2,\ \ldots\ ,\ A_n/V_n)$$

where E is a name of the entity set and each A_i/V_i stands for an attribute-value set pair. An attribute A_i is a function which maps an entity set into a value set V_i. V_i could be a Cartesian product of several value sets.

An entity e which belongs to E is represented as a tuple $(v_1, v_2, \ldots, v_n)$ with $v_i = A_i(e)$. This form of an entity is called an entity relation.

Let $\{A_1/V_1, \ldots, A_k/V_k\}$ be a subset of $\{A_1/V_1, \ldots, A_n/V_n\}$. The set of attributes $X = \{A_1, \ldots, A_k\}$ is called a key of an entity set E, if X is a minimal set of attributes which gives a one-to-one mapping from E into the Cartesian product $V_1 \times \ldots \times V_k$. From the set of keys of an entity set E, we arbitrarily select one as the primary key of E and write PK(E). An attribute which does not appear in any key is called a nonkey attribute of an entiry set.

For an abbreviation of an entity set description, we write simply $E\ (\underline{A_1, \ldots, A_k}, A_{k+1}, \ldots, A_n)$ or $E\ (\underline{X}, Y)$ unless the value sets need explicitly be stated. Here, the primary key X is underlined and Y denotes a set of other attributes.

Example:
EMPLOYEE (EMP-NO/EMP-NO, NAME/FIRST-NAME X LAST-NAME, AGE/NO-OF-YEARS)
PK (EMPLOYEE) = {EMP-NO}
or simply
EMPLOYEE (EMP-NO, NAME, AGE)

(2) The relationship set
Entities are related to each other. Different types of relationships may exist between different types of entities. A relationship set is a set of relationships of the same type. A relationship set is described in the form

$$R\ (K_1/E_1, \ldots, K_n/E_n, A_1/V_1, \ldots, A_m/V_m)$$

where R is a name of the relationship set, K_i/E_i represents a role-entity set pair, and A_j/V_j is an attribute-value set pair. Namely, the entity sets $E_1, \ldots, E_n$ (which are not necessarily distinct) are related to each other constructing a relationship set R, with each E_i having the role K_i in R. Each attribute A_j is a function which maps the Cartesian product $E_1 \times \ldots \times E_n$ into the value set (or the Cartesian product of value sets) V_j.

, relationship r which belongs to R is represented as a tuple $(e_1, \ldots, e_n, v_1, \ldots, v_m)$, where e_i is an entity of E_i and v_j is a value of V_j with $v_j = A_j(e_1, \ldots, e_n)$. This form of a relationship is called a relationship relation. We often use the abbreviated notation $r = (e_1, \ldots, e_n)$ when we are interested in only the entity sets involved. The primary key of R, denoted by PK(R), is represented by the primary keys of the involved entities. That is, $PK(R) = \{PK(E_1), \ldots, PK(E_n)\}$. In the description of a relationship set, we use a simple notation $R(E_1, \ldots, E_n : A_1, \ldots, A_m)$ or $R(G : Y)$, where G is $\{E_1, \ldots, E_n\}$ and Y is a set of nonkey attributes $\{A_1, \ldots, A_m\}$.

Example:
COURSE-GRADE (EXAMINEE/STUDENT, COURSE/COURSE, GRADE/INTEGER)
PK(COURSE-GRADE) = {PK(STUDENT), PK(COURSE)}
or simply
COURSE-GRADE (STUDENT, COURSE : GRADE)

The E-R model is constructed as a set of entity sets and relationship sets.

2.2 The mapping ratio in a relationship set

When a relationship set R is defined on entity sets $E_1, \ldots, E_n$, the mapping ratio of E_i with respect to R is defined as follows. Let d be a characteristic function defined on $E_1 \times \ldots \times E_n$ taking the value 1 or 0, such that $d(e_1, \ldots, e_n) = 1$ if and only if there exists a relationship relation $r = (e_1, \ldots, e_n)$ of R. For an entity e_i of E_i, the mapping ratio of e_i with respect to R is given by

$$m(e_i, R) = \Sigma\, d(e_1, \ldots, e_i, \ldots, e_n)$$

where $\{d(e_1, \ldots, e_n)\}$ are summed up for all tuples $(e_1, \ldots, e_n)$ with the i-th component e_i fixed. The mean, minimum, and maximum mapping ratio of E_i with respect to R are defined as the mean, minimum, and maximum value of $m(e_i, R)$ in E_i, and denoted by $m(E_i, R)$, $\underline{m}(E_i, R)$, and $\bar{m}(E_i, R)$ respectively.

We can generalize this definition to the mapping ratio of a set of entity sets. Let $R(G, H : Z)$ be a relationship set defined on the sets of entity sets $G = \{E_1, \ldots, E_k\}$ and $H = \{F_1, \ldots, F_\ell\}$, and let $e = (e_1, \ldots, e_k)$ and $f = (f_1, \ldots, f_\ell)$ be tuples of entities of G and H respectively. Using the characteristic function d, the mapping ratio of e with respect to R is given by

$$m(e, R) = \Sigma\, d(e, f)$$

where $\{d(e, f)\}$ are summed up for all tuples (e, f) with the component e fixed. The mean, minimum, and maximum mapping ratio of G with respect to R are defined as the mean, minimum, and maximum value of $m(e, R)$ in G, and denoted by $m(G, R)$, $\underline{m}(G, R)$ and $\bar{m}(G, R)$ respectively.

2.3 The E-R diagram

The E-R diagram is a diagrammatic technique for exhibiting entities and relationships ([1]). The E-R diagram is more strictly defined using the mapping ratio. Consider a set of entity sets $G = \{E_1, \ldots, E_n\}$ and a set of relationship sets $S = \{R_1, \ldots, R_m\}$ where each R_j involves G. Let $m_{ij} = m(E_i, R_j)$ be a mapping ratio of E_i with respect to R_j, then

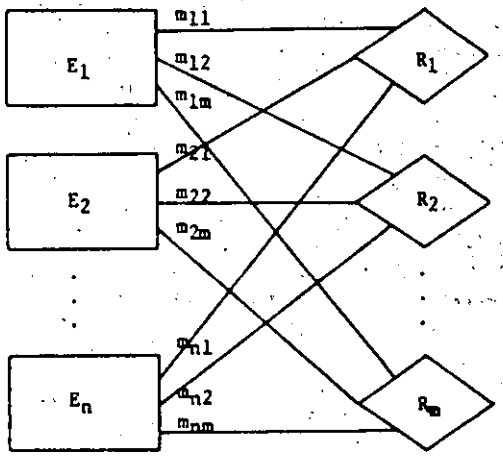


Fig. 1. An E-R diagram.

the E-R diagram of G and S is illustrated as in Fig. 1.

3. THE LOCAL OPTIMIZATION - THE RESTRUCTURING PHASE 1

3.1 The functional dependency in the E-R model

P.P. Chen suggested a top down approach to design the E-R model ([1]) : (1) identify the entity sets and the relationship sets of interest ; (2) identify semantic information in the relationship sets ; (3) define the value sets and attributes ; (4) organize data into entity/relationship relations and decide primary keys.

However, further refinements would be required in order to construct a stable and universal data structure. We shall formulate an optimization procedure of the E-R model from information structural view points. The procedure consists of two phases, the local optimization (the restructuring phase 1) and the global optimization (the restructuring phase 2).

As the first step to the optimization procedure, we shall investigate the functional dependencies embodied in the E-R model.

Let $\{A_1/V_1, \ldots, A_k/V_k\}$ and $\{B_1/W_1, \ldots, B_\ell/W_\ell\}$ be two sets of attribute-value set pairs in the same entity set E or in the same relationship set R defined on the entity sets $E_1, \ldots, E_n$. Let e be any entity of E or tuple of entities $(e_1, \ldots, e_n)$ involved in R. If for a value $(A_1(e), \ldots, A_k(e))$ of $V_1 \times \ldots \times V_k$, there exists at most one value $(B_1(e), \ldots, B_\ell(e))$ of $W_1 \times \ldots \times W_\ell$, then we say the set of attributes $Y = \{B_1, \ldots, B_\ell\}$ is functionally dependent on a set of attributes $X = \{A_1, \ldots, A_k\}$, and write $A_1, \ldots, A_k \rightarrow B_1, \ldots, B_\ell$ or $X \rightarrow Y$. This fact is called a functional dependen-

cy in attributes. We say Y is fully functionally dependent on X, if we have X → Y, and Y is not functionally dependent on any proper subset of X.

From the definition of an entity/relationship set, each nonkey attribute is functionally dependent on the primary key of that entity/relationship set. Since an entity set might have several keys, the nonkey attribute is functionally dependent on any key, and the keys are mutually functionally dependent on each other.

There is another type of functional dependency related to entities in a relationship sets. Let R be a relationship set involving sets of entity sets $G = (E_1, \ldots, E_k)$ and $H = (F_1, \ldots, F_\ell)$. If $\overline{m}(G, R) = 1$ holds, then we can consider a functional dependency G → H. Namely, for any relationship $r = (e, f)$ of R with tuples of entities e and f of G and H respectively, and for the tuple of primary key values of e, there exists at most one tuple of primary key values of f. We say this fact a functional dependency in entities.

3.2 The restructuring phase 1

The restructuring phase 1 is to locally optimize the E-R model so that it satisfies the properties P_1 and P_2.

P_1 : Each nonkey attribute in the entity/relationship set is fully functionally dependent on the primary key.
P_2 : Each nonkey attribute in the entity/relationship set is not functionally dependent on any other nonkey attribute.

If these properties do not hold in the model, we can locally restructure it as follows.

(1) The entity set not satisfying P_1
Let E (X, Y) be an entity set, and let some attributes subset Y' of Y be fully functionally dependent on a primary key attributes subset X' of X. Then decompose E (X, Y) into two entity sets E (X, Y-Y') and E' (X', Y'). Replace E (X, Y) with E (X, Y-Y'), and connect E' to E with a relationship set R (E, E'). This decomposition should repeatedly performed on E until it satisfies P_1. Note that from the minimality of a key, an entity set of the form E (X, Y'') will remain at the end of the decomposition process with original relationship sets involving E left unaffected.

(2) The entity set satisfying P_1 but not P_2
The entity set of this type is also restructured without affecting the original relationship sets involving it. Consider an entity set E (X, Y, Z) having PK(E) = X and sets of nonkey attributes Y and Z. Let Y be functionally dependent only on X, and let Z be fully functionally dependent on some subset Y' of Y. Then decompose E (X, Y, Z) into two entity sets E (X, Y-Y') and E' (Y', Z). Replace E (X, Y, Z) with E (X, Y-Y') and connect E' to E with a relationship set R (E, E'). E (X, Y-Y') satisfies P_1 and P_2, and originally existing relationship sets involving E will be left unaffected. E' (Y', Z) should further be decomposed until all of the decomposed entity sets satisfy P_1 and P_2. Theoretically, this restructuring process requires a formal analysis to find redundant functional dependencies embodied in an entity set. The required entity sets could be restructured from the minimal cover of functional dependencies embodied in an entity set. The minimal cover algorithm is presented in [3], [4], and [5].

(3) The relationship set not satisfying P_1
The similar decomposition as in case (1) can be applied to the relationship set of this type. Consider a relationship set R (G : Y), where a subset Y' of Y is fully functionally dependent on the set of primary keys PK (G') of G' which is a subset of G. Then decompose R (G : Y) into two relationship sets R (G : Y-Y') and R' (G' : Y'). This decomposition should repeatedly performed on R until it satisfies P_1.

(4) The relationship set satisfying P_1 but not P_2
This case will again require the minimal cover algorithm for restructuring the relationship set. We shall only show a decomposition process in a simple case. Let R (G : Y, Z) be a relationship set defined on a set of entity sets G and having sets of attributes Y and Z. Let Y be functionally dependent only on PK(R), and let Z be fully functionally dependent on some subset Y' of Y. Then decompose R (G : Y, Z) into a relationship set R (G : Y-Y') and an entity set E (Y', Z). Replace the original R with R (G : Y-Y') and connect E to G with a relationship set R' (G, E). R (G : Y-Y') satisfies P_1 and P_2. If E does not satisfy P_2, E should be decomposed according to the process described in (2).

4. THE GLOBAL OPTIMIZATION - THE RESTRUCTURING PHASE 2

4.1 The first order hierarchical decomposition in relationship sets

At the completion of the restructuring phase 1, we obtain a locally optimized E-R model in which each entity/relationship set satisfies P_1 and P_2. The next step of the optimization is to decompose a relationship set into simpler forms, and to eliminate redundant relationship sets which are derivable from other relationship sets. For this purpose, we shall introduce the first order hierarchical decomposition and the transitive dependency concepts in relationship sets.

Normalization processes in a relational model has been discussed by many autors ([3], [4], [5]). These methodologies principally aimed to eliminate functional dependencies which are derived transitively from other functional dependencies. However, in the realistic database design process, we have much more cases where there exist many-to-many relationships rather than functional dependencies. Moreover, we often have more than one different relationships between entities whereas the theories thus far developed assume the uniqueness of the functional dependency between any two sets of attributes.

The first order hierarchical decomposition and the transitive dependency in relationship sets could be thought as a useful idea for a practical optimization of the model. We shall first define a projection and a join operation.

(1) A projection of relationship sets
Let R (G : Y) be a relationship set defined on $G = (E_1, \ldots, E_n)$, and let $G' = (E_1, \ldots, E_k)$ be

a subset of G. For a relationship relation $r = (e_1, \ldots, e_n)$ of R where e_i is an entity of E_i, the G' component of r is defined by $r[G'] = (e_1, \ldots, e_k)$. The projection of R on G' is a set of r [G'] for all relationship relation r of R and denoted by R [G']. R[G'] could be thought as a relationship set R(G') defined on G'.

(2) A join of relationship sets
Let R_1 (G_1) and R_2 (G_2) be relationship sets, and $G = G_1 \cap G_2$ be non empty. The join of R_1 and R_2 is a set of tuples which are generated by concatenating the relationship relations r_1 of R_1 and r_2 of R_2 such that r_1 and r_2 have a common G component, and is denoted by $R_1 * R_2$. Namely,

$$R_1 * R_2 = \{(r_1, r_2\,[G_2 - G]) \mid r_1 \in R_1,\ r_2 \in R_2,\ r_1\,[G] = r_2\,[G]\}$$

Delobel introduced the notion of first order hierarchical decomposition in a relational model which allows to decompose relations independently of functional dependencies ([6]). We can apply this concept to the decomposition of relationship sets.

A relationship set R (G, G_1, ... , G_n), where G, G_1, ... , G_n are disjoint sets of entity sets, is said to obey the first order hierarchical decomposition (F O H D) if it is decomposable as follows:

$$R = R\,[G, G_1] * \ldots * R\,[G, G_n]$$

This means that, for each i and j ($i \neq j$), all the tuples of entities $(g_i) \in G_i$ which are related to a tuple of entities $g \in G$ are also related to all the tuples $(g_j) \in G_j$ which are related to the same g.

Under this condition, we can define relationship sets R_1 (G, G_1), ... , R_n (G, G_n) such that each R_i is equal to the projection R [G, G_i], and write R = FOHD (R_1, ... , R_n). Then R can be replaced with (R_1, ... , R_n). In this replacement process, some R_i should be eliminated as redundant, if there already existed a relationship set R' having the projection R' [G, G_i] which is equal to R_i.

For example, consider a relationship set R (CLASS, STUDENT, TEACHER) where all the STUDENTs attending the CLASS are taught by all TEACHERs. We can write R = FOHD (R_1 (CLASS, STUDENT), R_2 (CLASS, TEACHER)) by properly defining R_1 and R_2. If there exists another relationship set R' (CLASS, STUDENT : GRADE) having the projection R' [CLASS, STUDENT] = R_1, then R_1 should be eliminated from the replacement list. Note that FOHD list of a given relationship set is not necessarily unique.

A relationship set including the functional dependency in entity sets is treated as a special case of FOHD. Namely, if the functional dependency $G = G_1$ holds in a relationship set R (G, G_1, G_2), then we have R = R [G, G_1] * R [G, G_2] and R obeys the FOHD condition.

4.2 The transitive dependency in relationship sets

Now another refinement procedure of the E-R model is introduced using the notion of the transitive dependency in relationship sets.

Let R_1(G_1 : Y_1), R_2 (G_2 : Y_2), and R (G : Y) be relationship sets such that $H = G_1 \cap G_2$ is non empty and $G = G_1 \cup G_2 - H$. When the following conditions hold, we say R is transitively dependent on R_1 and R_2 and write R = TD (R_1, R_2).

(a) $R_1\,[H] = R_2\,[H]$

(b) $R\,[G] = (R_1\,[G_1] * R_2\,[G_2])\,[G]$

This means that for every $r_1 \in R_1$ there exists $r_2 \in R_2$ such that $r_1\,[H] = r_2\,[H]$ and the G component of the tuple (r_1 [G_1], r_2 [G_2-H]) is a G component of some $r \in R$, and R consists of only such relationship relations.

If a relationship set R is transitively dependent on relationship sets R_1 and R_2, and has no nonkey attribute, then R could be eliminated because it is derivable from R_1 and R_2.

A relationship set which is derivable from a sequence of relationship sets using the transitive dependency could also be eliminated. Let R_i (G_i : Y_i) (i = 1, 2, ... , n) and R (G : Y) be relationship sets such that each $H_i = G_i \cap G_{i+1}$ (i = 1, 2, ... , n-1) is non empty, and $G = \bigcup G_i - \bigcup H_i$. When the following conditions hold, we say R is transitively dependent on R_1, ... , R_n, and write R = TD (R_1, ... , R_n).

(a) $R_i\,[H_i] = R_{i+1}\,[H_i]$ for i = 1, ... , n-1

(b) $R\,[G] = (R_1\,[G_1] * \ldots * R_n\,[G_n])\,[G]$

In this situation, R could be eliminated if it does not contain any nonkey attribute.

After the relationship sets which obey the FOHD have been decomposed, all redundant relationship sets which are transitively derivable from others and have no nonkey attribute should be eliminated. The set of redundant relationship sets would not uniquely be determined depending on the order of elimination. In this restructuring process, the relationship sets in which functional dependencies in entity sets are embodied should be eliminated as later as possible, because the relationship sets of this type would serve for further refinement of the model in the way to be described.

Before proceeding to the final step, we shall show an example of the elimination process. Consider an E-R model as illustrated in Fig. 2. It is assumed that the restructuring phase 1 and the decomposition process at the begginning of the restructuring phase 2 have already completed. In Fig. 2, let us assume that all relationship sets except for R_4, R_5, and R_6 have no nonkey attribute, and there exist transitive dependencies R_2 = TD (R_4, R_6), R_3 = TD (R_4, R_5), R_6 = TD (R_7, R_8), R_7 = TD (R_8, R_6), and R_8 = TD (R_6, R_7). Then the elimination process is performed as follows.

(i) R_4, R_5, and R_6 should not be eliminated because they have nonkey attributes.

(ii) R_2 and R_3 should be eliminated because they have no nonkey attribute and are derivable from R_4, R_5, and R_6.

(iii) Either R_7 or R_8 could be eliminated by the same reason as above. R_8 embodies a functional dependency COURSE → ROOM, therefore R_7 should be eliminated.

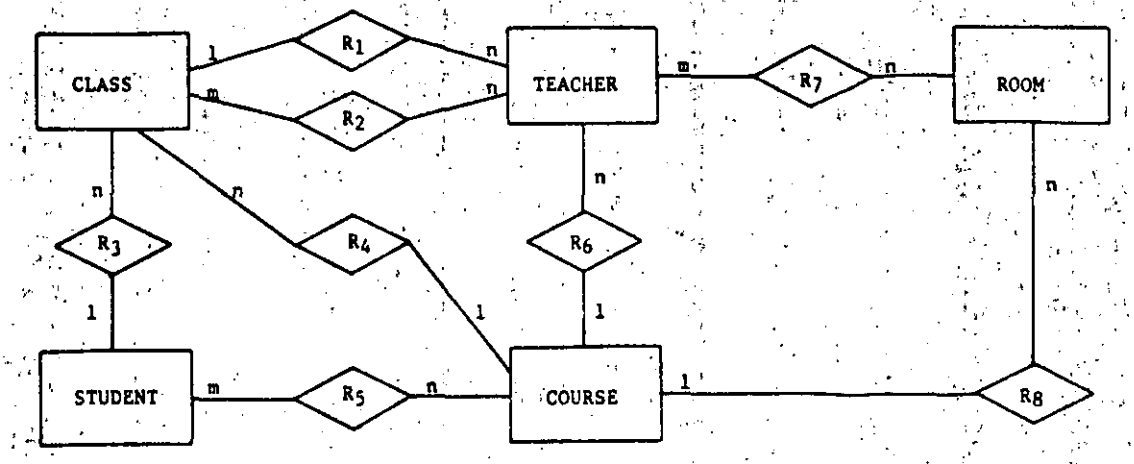


Fig. 2. An E-R diagram including redundant relationship sets.

If there exist relationship sets which embody functional dependencies in entity sets, the further reduction of the number of entity/relationship sets is possible under certain conditions.

The relationship set which is simple but often found in the design is of type R (E, F) which relates an entity set E to another entity set F with $\bar{m}$ (E, R) = 1. If F consists of only its primary key and is involved in no other relationship set than R, then PK(F) could be included in the nonkey attributes of E, and F could be eliminated together with R.

Consider the more general case where R (G, H) relates a set of entity sets G to another set of entity sets H with $\bar{m}$ (G, R) = 1. If each entity set of H consists of only its primary key, and is involved in no other relationship set than R, then R (G, H) and each entity set of H could be replaced with a relationship set R (G : PK(H)) where PK(H) denotes a set of primary keys of entity sets of H.

5. CONCLUSION

The optimization process of an E-R model from information structural view points is summarized as follows.

(1) Local optimization - the restructuring phase 1
For each entity/relationship set, analyze the functional dependencies in nonkey attributes.
Restructure entity/relationship sets so that they satisfy the properties P_1 and P_2.

(2) Global optimization - the restructuring phase 2
First, find the relationship sets which obey the FOHD and decompose them into simpler forms.
Second, analyze the transitive dependencies in relationship sets. Eliminate redundant relationship sets which are derivable from the remaining relationship sets. Finally, analyze the functional dependencies in entity sets embodied in relationship sets. Refine the model so that there exists no such a functional dependency, as far as it is semantically reasonable.

The optimization process mentioned so far is very useful for a logical database design from practical view points. The E-R model thus optimized provides a stable data structure source from which we can derive a final logical structure suitable for data access requirements in actual application environments. We are currently concerned with designing a data dictionary to store and manipulate the model as a logical database design aid.

REFERENCES

[1] Peter P.S. Chen, The entity-relationship model : towards a unified view of data, ACM Transactions on Database Systems, vol. 1, no. 1, March 1976, 9-36.

[2] Peter P.S. Chen, The entity-relationship model : a basis for the enterprise view of data, AFIPS Conference Proceedings, vol. 46, AFIPS Press, 1977, 77-84.

[3] P. A. Bernstein, Synthesizing third normal form relation from functional dependencies, ACM Transactions on Database Systems, vol. 1, no. 4, December 1976, 277-298.

[4] C. Delobel and R.G. Casey, Decomposition of a data base and the theory of Boolean switching functions, IBM Journal of Research and Development, vol. 17, no. 5, September 1973, 374-386.

[5] C.P. Wang and H.H. Wedekind, Segment synthesis in logical data base design, IBM Journal of Research and Development, vol. 19, no. 1, January 1975, 71-77.

[6] C. Delobel and M. Leonard, The decomposition process in a relational model, International Workshop on Data Structures, IRIA, Namur, May 1974.

DISEÑO DE BASES DE DATOS

P E T R I N E T S*

MAYO, 1985

Petri Nets*

JAMES L. PETERSON

Department of Computer Sciences, The University of Texas, Austin, Texas 78712

Over the last decade, the Petri net has gained increased usage and acceptance as a basic model of systems of asynchronous concurrent computation. This paper surveys the basic concepts and uses of Petri nets. The structure of Petri nets, their markings and execution, several examples of Petri net models of computer hardware and software, and research into the analysis of Petri nets are presented, as are the use of the reachability tree and the decidability and complexity of some Petri net problems. Petri net languages, models of computation related to Petri nets, and some extensions and subclasses of the Petri net model are also briefly discussed.

Keywords and Phrases: Petri nets, system models, asynchronous concurrent events

CR Categories: 1.3, 5.29, 8.1

INTRODUCTION

A *Petri net* is an abstract, formal model of information flow. The properties, concepts, and techniques of Petri nets are being developed in a search for natural, simple, and powerful methods for describing and analyzing the flow of information and control in systems, particularly systems that may exhibit asynchronous and concurrent activities. The major use of Petri nets has been the modeling of systems of events in which it is possible for some events to occur concurrently but there are constraints on the concurrence, precedence, or frequency of these occurrences.

Since many readers may be unfamiliar with Petri nets, we first present a very brief and informal introduction to their fundamentals and history. Then we consider several aspects of Petri nets in more detail. We begin, in Section 2, by considering the use of Petri nets for modeling systems of parallel or concurrent activities. Section 3 presents a more formal definition and discussion of the fundamental concepts and notations of Petri nets. Section 4 considers the extensive body of research dealing with the analysis of Petri nets, their advantages, and their limitations. Petri net languages are presented in Section 5. Finally, in Section 6, we consider some of the many variations of Petri nets that have been defined, both as generalizations of Petri nets and as subclasses of the general model; the more general models have certain advantages for modeling, while the more restricted models have certain advantages for analysis.

1. OVERVIEW

Figure 1 shows a simple Petri net. The pictorial representation of a Petri net as a graph used in this illustration is common practice in Petri net research. The Petri net graph models the static properties of a system, much as a flowchart represents the static properties of a computer program.

* This work was supported, in part, by the National Science Foundation, under Grant Number MCS75-16425.

CONTENTS

The graph contains two types of nodes: circles (called *places*) and bars (called *transitions*). These nodes, places and transitions, are connected by directed arcs from places to transitions and from transitions to places. If an arc is directed from node i to node j (either from a place to a transition or a transition to a place), then i is an *input* to j, and j is an *output* of i. In Fig. 1, for example, place p_1 is an input to transition t_2, while places p_2 and p_3 are outputs of transition t_2.

In addition to the static properties represented by the graph, a Petri net has dynamic properties that result from its execution. Assume that the execution of a computer program represented by a flowchart is exhibited by placing a marker on the flowchart to mark the instruction being executed, and that as the execution progresses, the marker moves around the flowchart. Similarly, the execution of a Petri net is controlled by the position and movement of markers (called *tokens*) in the Petri net. Tokens, indicated by black dots, reside in the circles representing the places of the net. A Petri net with tokens is a *marked Petri net*.

The use of the tokens rather resembles a board game. These are the rules: Tokens are moved by the *firing* of the transitions of the net. A transition must be *enabled* in order to fire. (A transition is enabled when all of its input places have a token in them.) The transition fires by removing the enabling tokens from their input places and generating new tokens which are deposited in the output places of the transition. In the marked Petri net of Fig. 2, for example, the transition t_2 is enabled since it has a token in its input place (p_1). Transition t_3, on the other hand, is not enabled since one of its inputs (p_3) does not have a token. If t_2 fires, the marked Petri net of Fig. 3 results. The firing of transition t_2 removes the enabling token from place p_1 and puts tokens in p_2 and p_3, the two outputs of t_2.

The distribution of tokens in a marked Petri net defines the state of the net and is called its *marking*. The marking may change as a result of the firing of transitions. In different markings, different transitions may be enabled. For example, in the marked net of Fig. 3 three transitions are enabled: t_1, t_3, and t_5, none of which were enabled in the marking of Fig.

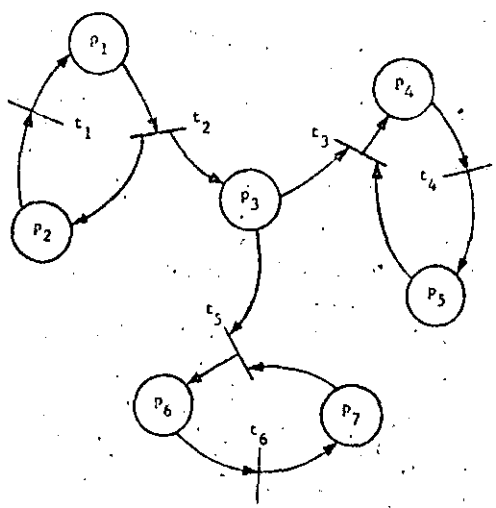


FIGURE 1. A simple graph representation of a Petri net.

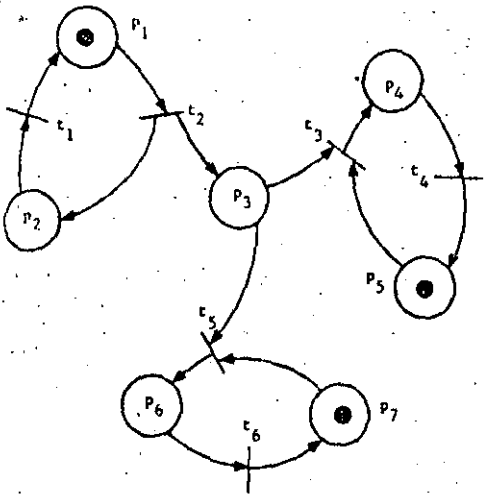


FIGURE 2. A marked Petri net.

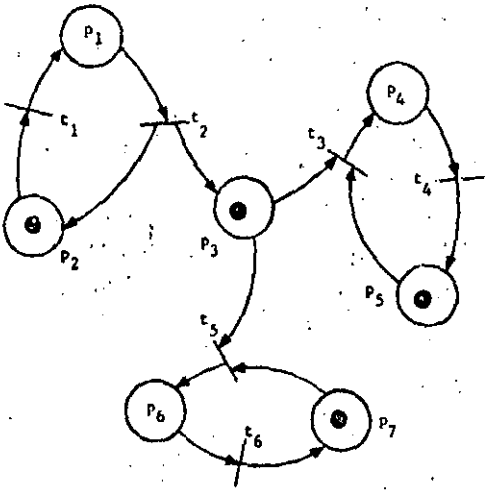


FIGURE 3. The marking resulting from firing transition t_2 in Fig. 2. Note that the token in p_1 was removed and tokens were added to p_2 and p_3.

2. In this situation, we have a choice as to which transition will fire next. Figure 4 shows the three possible resultant markings; from each of these, other markings may then be reached since transition firings may continue as long as there is an enabled transition.

Note that in the marking of Fig. 4a, transitions t_3 and t_5 remain enabled, and transition t_2 is also enabled; if transition t_2 fires, the resulting marking will have two tokens in place p_3. In the marking of Fig. 4b, transition t_1 remains enabled but transition t_5 has been disabled since there is no longer a token in place p_3. In the marking of Fig. 4c, transition t_3 has become disabled. Firing either of transitions t_3 or t_5 disables the other; they are said to be *in conflict*.

This simple, vague, and incomplete example of a Petri net is meant to give a brief introduction to the basic concepts of Petri nets. It also raises some questions for further consideration. For example, the markings of Figs. 3, 4a, 4b, and 4c were generated from the marking of Fig. 2 by firing transitions. Can we characterize the class of markings that may be reached from a given marked Petri net? Can we characterize the class of sequences of transition firings that are possible from a marked Petri net? What interesting properties can Petri nets have and how can these properties be tested for?

Some of these questions are of interest for Petri nets as abstract formal entities. Other questions relate to Petri nets in their function as models of other systems, existing or proposed. For example, the net of Fig. 2 can represent a producer–consumer problem [25] with one producer (places p_1 and p_2) and two consumers (places p_4, p_5 and p_6, p_7). The items produced by the producer are passed to the consumers. This is modeled by place p_3 and the tokens "produced" by transition t_2 and "consumed" by transitions t_3 and t_5. For this interpretation of the net, we may be interested in how far the producer can get ahead of the consumers (the maximum number of tokens in p_3), whether the consumers could get ahead of the producer or consume the same item twice, and so on. The use of Petri nets in modeling is discussed in Section 2.

Although Petri nets are basically very simple, they may be approached and utilized in a large number of ways. Petri nets can be considered as formal automata and investigated either as automata or as generators of formal languages [37, 79]. Questions dealing with the theory of computational complexity have been raised [64, 49]. Petri nets have associations with the study of linear algebra [69], Presburger arithmetic [52], and graph theory. They

are a major model of concurrent systems [6], particularly computer systems. They are of interest in some areas of hardware design, description and construction, software systems, and the interactions between design and implementation.

Because of the breadth of application and depth of research into Petri nets, we can only touch here on many of the results. We refer the interested reader to the original works cited in the Bibliography for the proofs and details of much of the research.

History

The theory of Petri nets has developed from the work of Carl Adam Petri, A. W. Holt, Jack Dennis, and many others. Petri nets originated in the early work of Petri, in Germany, who in his thesis [80], developed a new model of information flow in systems. This model was based on the concepts of asynchronous and concurrent operation by the parts of a system and the realization that relationships between the parts could be represented by a graph, or net.

The ideas of Petri came to the attention of a group of researchers at Applied Data Research, Inc., working on the Information Systems Theory Project [43]. This group, led by Anatol Holt, developed the theory of "systemics" [44] which was concerned with the representation and analysis of systems and their behavior. It was this work which provided the early theory, notation, and representation of Petri nets, and showed how Petri nets could be applied to the modeling and analysis of systems of concurrent processes.

Applied Data Research's associations with Project MAC at MIT, and particularly the Computation Structures Group under the direction of Jack Dennis, introduced the concepts of Petri nets to this latter group. The Computation Structures Group has been a most productive source of research and literature in this field, publishing several PhD theses and numerous reports and memos on Petri nets [73, 32, 34, 84, 38, 28]. Two pertinent conferences have been held by the Computation Structures Group: the Project MAC Conference on Concurrent Systems and Parallel Computation at Woods Hole in 1970, [23] and the Conference on Petri Nets and Related Methods at MIT in 1975.

From the work at Applied Data Research and MIT, the use of Petri nets has spread widely. A large amount of research has been done on both the nature and the application of Petri nets, and their use seems to be expanding. The simplicity and power of Petri nets make them excellent tools for working with asynchronous concurrent systems. Unfortunately much of the work on Petri nets is in the form of theses, dissertations, reports, and memos that are not readily available nor in wide circulation. This paper is an attempt to remedy this situation; it is intended as both a survey and a tutorial on Petri nets.

It should be noted that Petri nets can be viewed in many different ways; we cannot present here all such views. In part due to the difficulty of obtaining literature on Petri nets and the newness of the theory, the terminology, notation, and emphasis have varied widely in research on this subject. This problem is also caused by the power of Petri nets and the resultant diversity of applications.

Petri has expanded upon his original theory, continuing work on the basic concepts of information flow and the structure of concurrent systems. This has resulted in a form of general systems theory called net theory [81, 82] which is related to topology. This research, involving the fundamental nature of information and its control, has stimulated a wealth of further research in Europe, particularly at the Institut für Informationssystemforschung of the Gesellschaft für Mathematik und Datenverarbeitung in Bonn. While starting with the same fundamental concepts as the work in the United States, this work developed in a different direction, evolving into a more general and abstract theory.

Holt also has continued to develop new concepts from the original work on Petri nets, concentrating on the development of tools for the representation and analysis of systems. His work has centered mainly on research into the fundamental aspects of concurrency and conflict in systems with multiple parts.

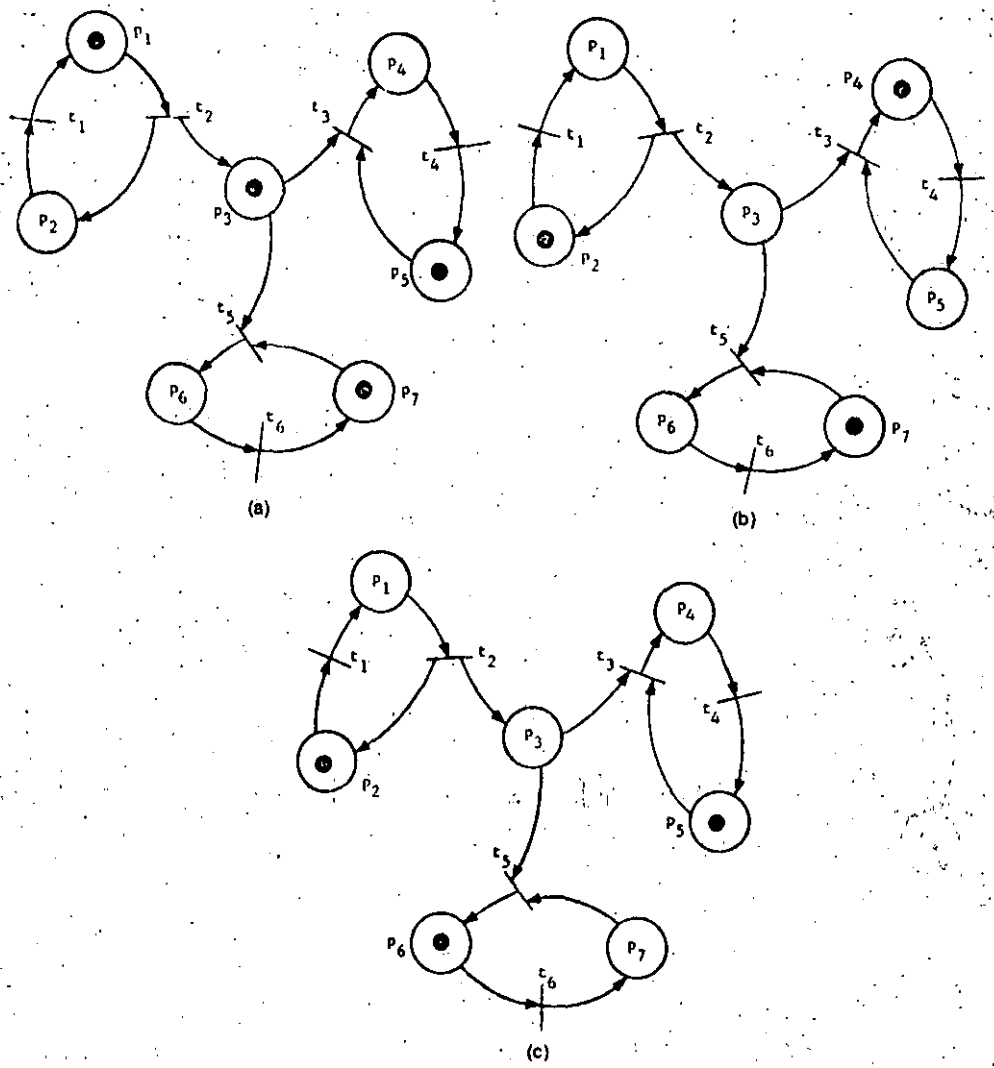


FIGURE 4. Markings resulting from the firing of different transitions in the net of Fig. 3. (a) Result of firing transition t_1; (b) Result of firing transition t_3; (c) Result of firing transition t_5.

In contrast to the work of Petri, Holt, and many European researchers, which emphasizes the fundamental concepts of systems, the work at MIT and many other American research centers concentrates on those mathematical aspects of Petri nets that are more closely related to automata theory. (This paper is written from the latter point of view.) This approach is motivated by a desire to analyze systems by modeling them as Petri nets, and then manipulating the Petri nets to derive properties of the modeled systems. This requires the development of techniques for analyzing Petri nets in order to answer questions similar to those raised earlier (e.g., what markings are reachable in a given Petri net? What sequences of transition firings are possible? etc.). This mechanistic approach is quite different in orientation from the more philosophical approaches of Holt and Petri.

2. MODELING WITH PETRI NETS

In many sciences, a phenomenon is studied by examining not the actual phenomenon itself but rather a *model* of the phe-

nomenon. A model is a representation, often in mathematical terms, of what are felt to be the important features of the object under study. By the manipulation of the representation, it is hoped that new knowledge about the modeled phenomenon, and the model itself, will be obtained without the cost, inconvenience, or danger of manipulating the real phenomenon itself. For example, much work on atomic energy has been done by modeling because of the expense and danger of handling radioactive materials.

Most modeling uses mathematics. The important features of many physical phenomena can be described numerically and the relations between these features described by equations or inequalities. Particularly in physics and chemistry, properties such as mass, momentum, acceleration, position, and forces, are describable by mathematical equations. To successfully utilize the modeling approach, however, requires a knowledge of both the modeled phenomena and the modeling techniques. Thus, mathematics has developed as a science in part because of its usefulness in modeling phenomena in other sciences. For example, the differential calculus was developed in direct response to the need for a means to model continuously changing properties such as position, velocity, and acceleration in physics.

Petri nets are also a modeling tool. They were devised for use in the modeling of a specific class of problems, the class of discrete-event systems with concurrent or parallel events. Petri nets model systems, and particularly two aspects of systems, *events* and *conditions*, and the relationships among them [44]. In this view, in a system, at any given time, certain conditions will hold. The fact that these conditions hold may cause the occurrence of certain events. The occurrence of these events may change the state of the system, causing some of the previous conditions to cease holding, and causing other conditions to begin to hold.

A simple example might be that the simultaneous holding of both the condition 'A card reader is needed' and the condition

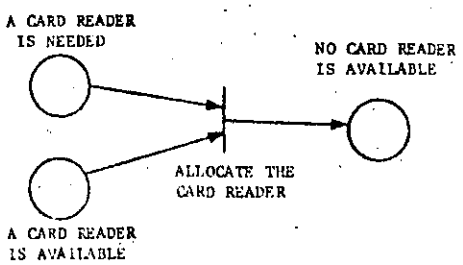


FIGURE 5. A simple model of three conditions and an event.

'A card reader is available' might cause the event 'Allocate the card reader' to occur. The occurrence of this event results in the ceasing of the conditions 'A card reader is needed' and 'A card reader is available', while causing the condition 'No card reader is available' to become true. These events and conditions, and their relationships, may be modeled as in Fig. 5, where we are using places to represent conditions and transitions to represent events. Note that other conditions, such as 'The card reader is allocated', may also hold in the system even though they are not modeled.

More complicated systems may also be modeled in this manner. Consider for example the following description of a computer system:

- Jobs appear and are put on an input list. When the processor is free, and there is a job on the input list, the processor starts to process the job.
- When the job is complete, it is placed on an output list, and if there are more jobs on the input list, the processor continues with another job; otherwise it waits for another job.

This is a very simple system composed of several elements: the processor, the input list, the output list, and the jobs. We can identify several conditions of interest:

- The processor is idle;
- A job is on the input list;
- A job is being processed;
- A job is on the output list;

and several events:

- A new job enters the system;
- Job processing is started;
- Job processing is completed;
- A job leaves the system.

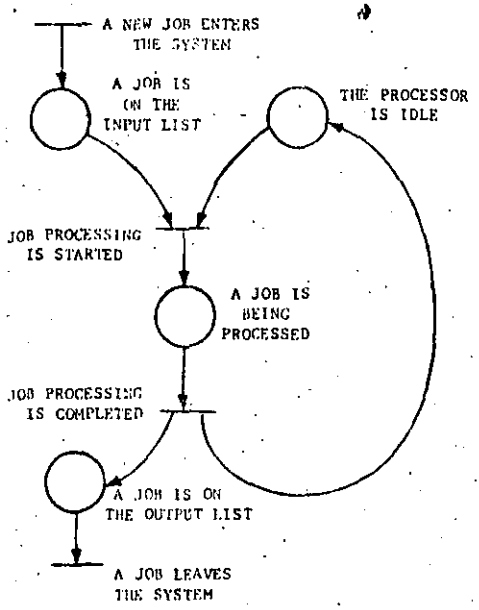


FIGURE 6. Modeling of a simple computer system.

The Petri net of Fig. 6 illustrates the modeling of this system. The "job enters" transition in this illustration is a *source*; the "job leaves" transition is a *sink*.

Properties of Petri Nets Useful in Modeling

The example above illustrates several points about Petri nets and the systems they can model. One is inherent *concurrency* or *parallelism*. There are two main kinds of independent entities in the system: the job and the processor. In the Petri net model, the events which relate solely to one or the other can occur independently; there is no need to synchronize the actions of the jobs and the processor. Thus jobs may enter or leave the system at any time independent of the action of the processor. However, when synchronization is necessary, for instance when both a job and an idle processor must be available for processing to start, the situation is also easily modeled. Thus a Petri net would seem to be ideal for modeling systems of distributed control with multiple processes occurring concurrently.

Another major feature of Petri nets is their asynchronous nature. There is no inherent measure of time or the flow of time in a Petri net. This reflects a philosophy of time which states that the only important property of time, from a logical point of view, is in defining a partial ordering of the occurrence of events. Events take variable amounts of time in real life; the Petri net model reflects this variability by not depending upon a notion of time to control the sequence of events. Therefore, the Petri net structure itself must contain all necessary information to define the possible sequences of events of a modeled system.

Thus, in the net of Fig. 6 the event 'Job processing is completed' must follow the corresponding event 'Job processing is started' because of the structure of the net although no information at all is given or considered concerning the amount of time required to process a job. On the other hand, events which need not be constrained in terms of their relative order of occurrence are not constrained; thus while a job is being processed the event 'A new job enters the system' may occur, before, after, or simultaneously with the occurrence of the event 'Job processing is completed'.

A Petri net, like the system which it models, is viewed as a sequence of discrete events whose order of occurrence is one of possibly many allowed by the basic structure. This leads to a *nondeterminism* in Petri net execution. If at any time more than one transition is enabled, then any of the several enabled transitions may fire. The choice as to which transition fires is made in a nondeterministic manner, i.e., randomly or by forces that are not modeled. This feature of Petri nets reflects the fact that in real-life situations where several things are happening concurrently, the order of occurrence of events is not unique, so that any of a set of sequences may occur. While nondeterminism is advantageous from a modeling point of view, it introduces considerable complexity into the analysis of Petri nets.

To reduce this complexity, one limitation is generally accepted in the modeling of systems by Petri nets. The firing of a transition (occurrence of an event) is considered to be *instantaneous*, i.e., to take

zero time. Since time is a continuous variable, then, the probability of any two or more events happening simultaneously is zero, and two transitions cannot fire simultaneously. The events being modeled are considered *primitive* events. Note that this need cause no problems in the modeling of events. For example, in Fig. 6 the event 'Process a job' was modeled. But since this event is not a primitive one (it takes nonzero time and other events, such as the entering and leaving of the system by other jobs, may occur at the same time), it is decomposed into a beginning and an ending, which are instantaneous events, plus the noninstantaneous occurrence. This is shown in Fig. 7. Since this technique can be used for any nonprimitive event, the modeling power of Petri nets is not reduced.

The nondeterministic and nonsimultaneous firing of transitions in the modeling of concurrent systems takes two forms. One of these is shown in Fig. 8, which depicts "simultaneous" events that may occur in either order. In this situation the two enabled events do not affect each other in any way and the possible sequences of events include some in which one event occurs first and some in which the other occurs first.

The other type of situation, where simultaneity causes difficulties in modeling, is handled by defining events to occur nonsimultaneously. This is illustrated in Fig. 9. Here the two enabled transitions t_j and t_k are in *conflict*. Only one transition can fire, since in so doing it removes the token from p_i and disables the other transition. To accurately model a system using Petri nets requires careful attention to assure that in cases such as the above the Petri net reflects all, and only those, event sequences which are possible in real life.

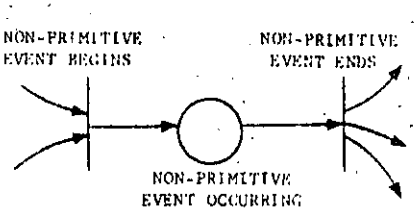


FIGURE 7. Modeling of a nonprimitive event.

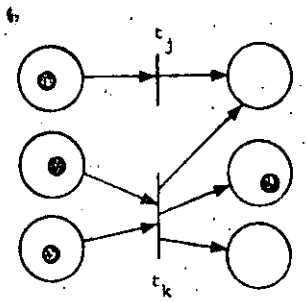


FIGURE 8. Modeling of "simultaneous" events which may occur in either order.

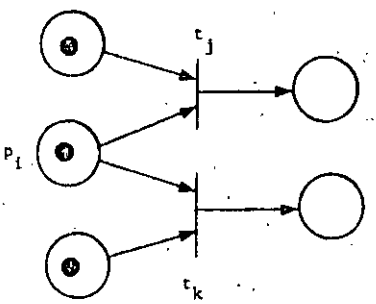


FIGURE 9. Illustration of conflicting transitions. Transitions t_j and t_k conflict since the firing of one will disable the other.

The two concepts illustrated by Figs. 8 and 9, *concurrency* and *conflict*, are basic to an understanding of Petri nets. At the same time, the usefulness of Petri nets as models of information flow derives from the natural way in which they can be used to express and analyze concurrency and conflict.

An important aspect of Petri nets is that they are uninterpreted models. The net of Fig. 6 has been labeled with statements that indicate to the human observer the intent of the model, but these labels do not, in any way, affect the execution of the net. The net of Fig. 10 is identical to that of Fig. 6 in that it has an identical structure. However, no meaning is attached to the places and transitions in this uninterpreted net; we deal only with the abstract properties inherent in the structure of the net. Since we are interested in the properties of Petri nets per se, in this paper we concern ourselves only with uninterpreted Petri nets.

Another valuable feature of Petri nets is their ability to model a system hierarchi-

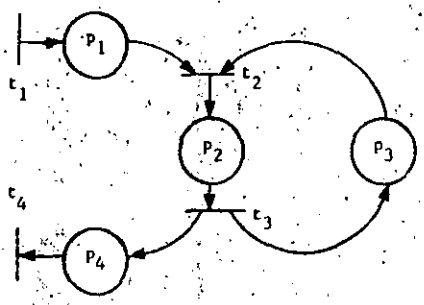


FIGURE 10. An uninterpreted Petri net.

cally. An entire net may be replaced by a single place or transition for modeling at a more abstract level (abstraction) or places and transitions may be replaced by subnets to provide more detailed modeling (refinement). Figure 11 illustrates this hierarchical modeling capability.

Most of the work on Petri nets has been in the investigation of the properties of a given net or class of nets. Little explicit attention has been paid to developing modeling techniques specifically for Petri nets. However, there are certain areas in which Petri nets would seem to be the perfect tool for modeling: those areas in which events occur asynchronously and independently. There are many examples, some of which we present here.

Modeling of Hardware

Large, powerful computer systems often use asynchronous parallel activities in an effort to achieve maximum parallelism and hence increase effective processing speed. In computers such as the CDC 6600 [91] and the IBM 360/91 [4], for example, multiple functional units are provided to perform computations on multiple registers. The control unit of the machine attempts to keep several of these units in operation simultaneously.

However, the introduction of parallelism in this manner must be controlled so that the results of executing the program with and without parallelism are the same. Certain operations in the program will require that the results of previous operations have been successfully computed before the following instructions can proceed. A system which introduces parallelism into a sequential program in such a way as to maintain correct results is called *determinate*. The conditions for maintaining determinancy have been considered by Bernstein [10]. They are the following: For two operations a and b such that a precedes b in the linear precedence of the program, b can be started before a is done if and only if b does not need the result of a as an input and the results of b do not change either the inputs or outputs of a [15].

One method of applying these constraints to the construction of the computer control unit that is to issue instructions is to use a reservation table. An instruction for functional unit u using registers i, j, and k can be issued only if all four of these components are not reserved; if the instruction is issued, all four of them become reserved. If the instruction cannot be issued at this time, the control unit waits until the instruction can be issued before continuing to the next instruction.

This sort of scheme can be modeled as a Petri net. To each functional unit and each register we associate a place. If the unit or register is free, a token will be in the place; if it is not, no token will be in the place. Figure 12 shows a portion of a Petri net which could be used to model the execution of an instruction using unit u and regis-

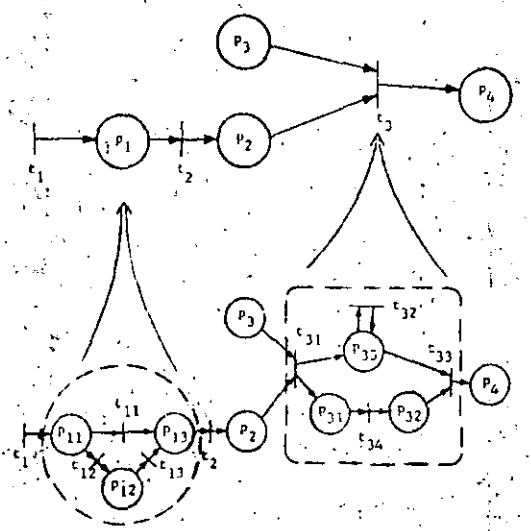


FIGURE 11. Hierarchical modeling in Petri nets by replacing places or transitions by subnets (or vice versa).

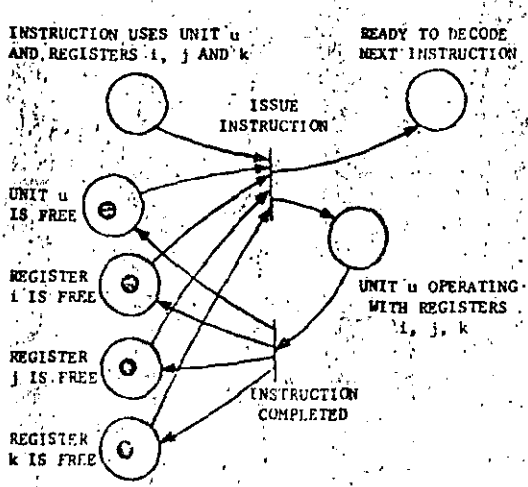


FIGURE 12. A portion of a Petri net modeling a control unit for a computer with multiple registers and multiple functional units.

ters i, j, and k. Modeling the entire control unit would of course require a much larger Petri net.

The scheme described above is a very simple method of introducing parallelism and does not consider, for example, the fact that multiple functional units can use the same register as an input simultaneously. Thus, this scheme may not produce schedules with maximum parallelism [55]. However, there are other schemes which can do so. These (more complicated) schemes can also be modeled by (more complicated) Petri nets. For example, the CPU of a CDC 6600 has been modeled by a Petri net [90]. This model was used to determine how object code should be generated to minimize execution time by maximizing parallelism between the various functional units.

Another approach to the construction of a high-performance computer is the use of pipelines [13]. This technique is useful, particularly for vector and array processing, and is similar to the operation of an assembly line. The pipeline is composed of a number of stages, which may be in execution simultaneously. When stage k finishes, it passes on its results to stage $(k + 1)$ and looks to $(k - 1)$ for new work. If each stage takes t time units and there are n stages, then the complete operation for one operand takes nt time units. However, if the pipe is kept supplied with new operands, it can turn out results at the rate of one every t time units.

As an example, consider the addition of two floating-point numbers. The gross steps involved are:

a) Extract the exponents of the two numbers;
b) Compare the exponents, and interchange if necessary to properly order the larger and smaller of the exponents;
c) Shift the smaller fraction to equalize exponents;
d) Add fractions;
e) Post-normalize;
f) Consider exponent overflow or underflow, and pack the exponent and fraction of the result.

Each of these steps can be performed by a separate computational unit, with a particular operand being passed from unit to unit for the complete addition operation.

The coordination of the different units can be handled in several ways. Typically, the pipeline control is synchronous with the time allowed for each step of the pipe, being some fixed constant time t; every t time units, the result of each unit is shifted down the pipe to become the input for the next unit. However, this can unnecessarily hold up processing, as the time needed may vary from stage to stage and may also vary for different inputs. For example, the post-normalization step in the floating-point addition above may take different amounts of time depending on how long the normalization shift should be and whether it should be to the left or to the right. Thus processing might be speeded up by an asynchronous pipeline in which results from stage k are sent on to stage $(k + 1)$ as soon as stage k is done and stage $(k + 1)$ is free. This scheme can be easily modeled by a Petri net.

Consider an arbitrary stage in the pipeline. Operations at this stage require certain inputs and produce certain outputs. Obviously, there has to be a place to put the inputs and outputs. Typically, this involves registers: the unit uses the values in its input register to produce values in its output register. It must then wait until 1) a new input is available in its input regis-

ter and 2) its output register has been emptied by being copied into the input register of the next stage. Thus the control for the pipeline needs to know when the following conditions hold:

- input register full;
- input register empty;
- output register full;
- output register empty;
- unit busy;
- unit idle;
- copying taking place.

Figure 13 shows the Petri net which models the operation of an asynchronous pipeline of this kind. Other forms of pipeline control units can also be defined in terms of Petri nets.

The above examples show some of the uses to which Petri nets can be put in the modeling of hardware. Petri nets have also been associated with the description of general modular asynchronous systems [22, 75] and macromodules [14]. At Honeywell, Petri nets have been used for investigating the fault-tolerant properties of designs [48].

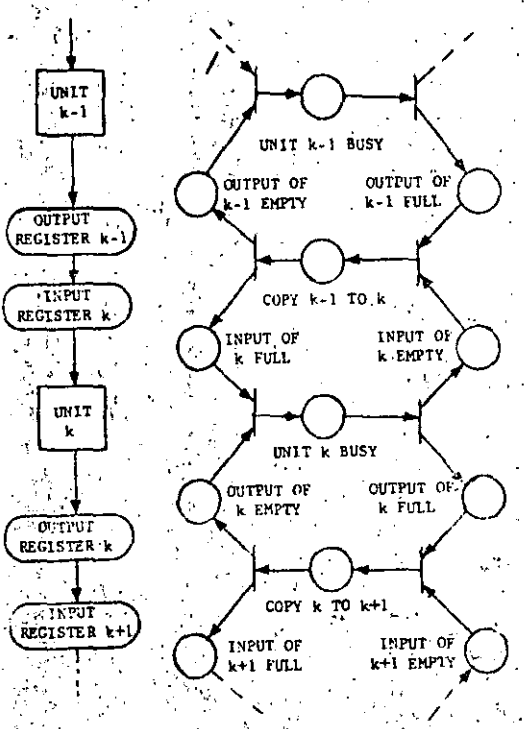
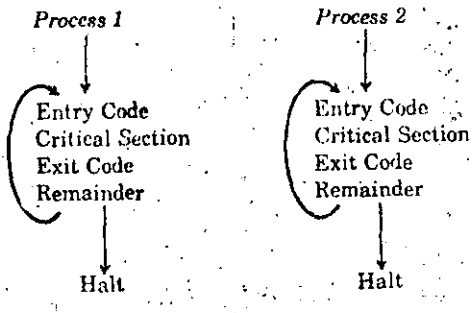


FIGURE 13. Representation of an asynchronous pipelined control unit. The block diagram on the left is modeled by the Petri net on the right.

Modeling of Software

On a more abstract level, Petri nets can also model software concepts [24, 77, 61]. Resource allocation, deadlock, and process coordination in an operating system can be modeled. A process can be modeled by a Petri net in the same way that it can be modeled by a flowchart, and then the interactions between processes can be modeled as additional places, arcs and transitions.

For example, consider the mutual exclusion problem [25]. This is a problem of enforcing coordination of processes in such a way that particular sections of code called critical sections, one in each process, are mutually excluded in time. That is, if Process 1 is executing its critical section, then Process 2 may not begin *its* critical section until Process 1 has left its own critical section. At its highest level of abstraction, the situation is illustrated as follows:



The problem is to define appropriate entry code and exit code to assure mutual exclusion.

The mutual exclusion problem can be easily solved by using P and V operations as defined in [25] for process synchronization and coordination. The P and V operations operate on semaphores, and only P and V instructions may be executed on a semaphore. (A semaphore S is a variable with integer values.) These operations can be defined as follows:

$P(S)$: As soon as $S > 0$, set $S := S - 1$;
$V(S)$: $S := S + 1$.

These are indivisible operations. A process executing a P operation must wait until the semaphore is positive before it can decrement it and continue. A V operation simply adds one to the semaphore (perhaps allowing some other process to execute a P operation). Two processes cannot execute P or V operations on the same semaphore concurrently. For example,

Process 1	*Process 2*
P(mutex);	P(mutex);
"Critical Section";	"Critical Section";
V(mutex);	V(mutex);

is a solution of the mutual exclusion problem diagrammed above using P and V operations, which are primitive, and the semaphore "mutex" which is global to the two processes and has an initial value of one.

P and V operations have been used widely. Systems of processes that use these operations can be modeled by Petri nets (see [74]). A semaphore is modeled by a place; the number of tokens in the place models the value of the semaphore. A V operation on the semaphore places a token in the semaphore; a P operation removes a token (it waits until there is one to remove, if necessary). This is illustrated in Fig. 14. The mutual exclusion problem can then be modeled as shown in Fig. 15, in which the place S models the semaphore. Note that places p_2 and p_4 are mutually excluded.

The modeling of P and V operations by Petri nets has resulted in the discovery and proof of some limitations of these nets. Patil showed that certain process synchronization problems (such as the Cigarette Smokers Problem) cannot be solved by P and V operations [74]. Other work following up this technique of using Petri nets to prove properties of semaphore systems is reported in [57], [3], and [56].

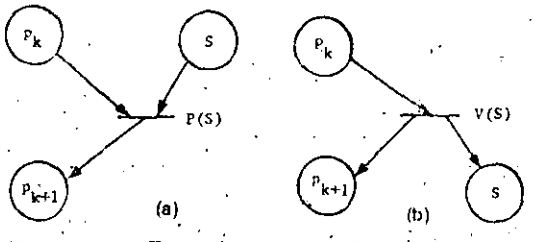


FIGURE 14. Examples of modeling with semaphores. (a) Modeling of a P operation; (b) Modeling of a V operation.

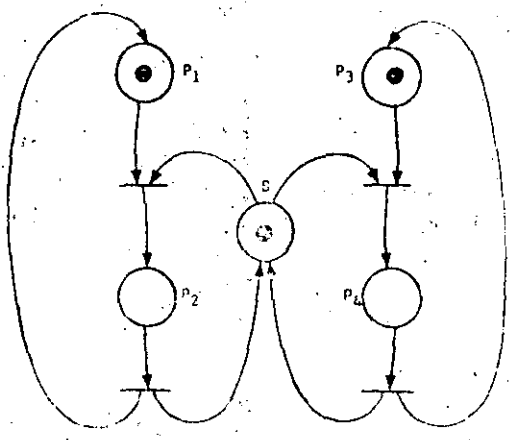


FIGURE 15. A Petri net model of a P/V solution to the mutual exclusion problem.

These simple examples can only provide a slight indication of the modeling power of Petri nets and the many diverse systems that can be modeled by them. Many other areas of study have been mentioned as possible subjects of Petri net modeling, including resource allocation, operating systems [71], queueing networks, traffic control, distributed computer systems, legal systems [65], proofs in mathematics [30], and brain modeling. While much work in developing modeling techniques remains to be done, Petri nets are a very powerful modeling tool that can be applied to a large variety of systems.

3. STRUCTURE OF PETRI NETS

The use of Petri nets for the modeling of concurrent systems requires a careful understanding of the properties of such nets. The development of an appropriate theory has motivated most of the research on Petri nets. This basic theory is presented in the following sections. Since this paper is tutorial in nature, we have tried to limit the formality of the presentation; however, all of the concepts presented here have been rigorously defined and formalized in the literature. The reader who is interested in a more formal treatment should consult the references.

Petri nets are composed of two basic

components: a set of *places*, P, and a set of *transitions*, T. To complete the definition, it is necessary to define the relationship between the places and the transitions. This can be done by specifying two functions connecting transitions to places: I, the *input* function, and O, the *output* function. The input function I defines, for each transition t_j, the set of *input places* for the transition $I(t_j)$. The output function O defines, for each transition t_j, the set of *output places* for the transition $O(t_j)$.

These four items define the *structure* of a Petri net. Places and transitions are the fundamental undefined concepts of Petri net theory; other concepts are defined in terms of these concepts. Formally, a Petri net C is defined as the four-tuple $C = (P, T, I, O)$.

Consider the following example Petri net structure, defined as a four-tuple. Each component of the structure is given:

$$C = (P, T, I, O)$$
$$P = \{p_1, p_2, p_3, p_4, p_5\}$$
$$T = \{t_1, t_2, t_3, t_4\}$$
$$I(t_4) = \{p_1\} \qquad O(t_1) = \{p_2, p_3, p_5\}$$
$$I(t_2) = \{p_2, p_3, p_5\} \qquad O(t_2) = \{p_5\}$$
$$I(t_3) = \{p_3\} \qquad O(t_3) = \{p_4\}$$
$$I(t_4) = \{p_4\} \qquad O(t_4) = \{p_1, p_5\}$$

The Petri Net Graph

Although the definition given above is useful and appropriate for formal work with Petri nets, it is ill-suited for the illustration of many of the concepts of Petri nets in an informal and intuitive, yet concise, manner. For this purpose a different representation of a Petri net is more useful: the *Petri net graph*. Figure 16 shows the Petri net graph corresponding to the structure described above.

A Petri net structure consists of places, transitions, and the input and output functions. In a Petri net graph there are two types of nodes corresponding to the places and transitions of the Petri net structure: a circle represents a place, and a bar represents a transition. The input and output functions are represented by directed arcs from the places to the transitions and from the transitions to the places. An arc is directed from a place p_i to a transition t_j if the place is an input of the transition.

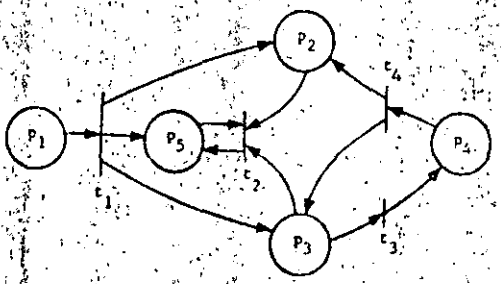


FIGURE 16: A Petri net graph.

Similarly, an arc is directed from a transition t_j to a place p_i if the place is an output of the transition.

A Petri net graph is a *directed* graph since the arcs are directed. In addition, since its nodes can be partitioned into two sets (places and transitions) such that each arc is directed from an element of one set (place or transition) to an element of the other set (transition or place), it is a *bipartite* directed graph.

The correspondence between Petri net graphs and Petri net structures is so natural that in most work they are considered not merely as different representations for the same concept, but rather as the same concept. Thus we refer to either Petri net graphs or Petri net structures as Petri nets. In this paper we give our examples as Petri net graphs, but our discussion and techniques are defined in terms of Petri net structures.

Markings

A marking μ of a Petri net is an assignment of *tokens* to the places in that net. ("Token" is a primitive concept for Petri nets.) Tokens reside in the places of the net. The number and position of tokens in a net may change during its execution. The vector $\mu = (\mu_1, \mu_2, \cdots, \mu_n)$ gives, for each place in the Petri net, the number of tokens in that place. The number of tokens in place p_i is μ_i, $i = 1, \cdots, n$. We may also define a marking function $\mu: P \to N$ from the set of places to the natural numbers, $N = \{0, 1, 2, \cdots\}$. This allows us to use the notation $\mu(p_i)$ to specify the number of tokens in place p_i. For a marking μ, $\mu(p_i) = \mu_i$.

On a Petri net graph, tokens are repre-

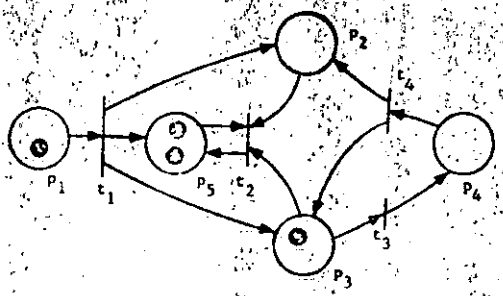


FIGURE 17. A marked Petri net.

sented by small solid dots inside the circles representing the places of the net. Figure 17 is an example of a Petri net graph with a marking. It represents the structure described in the previous section with the marking $\mu = (1, 0, 1, 0, 2)$. A Petri net $C = (P, T, I, O)$ with a marking μ becomes the *marked Petri net*, $M = (P, T, I, O, \mu)$.

Since the number of tokens in a place is unbounded over the set of all markings, there is an infinite number of markings for a Petri net. It is, of course, a denumerable infinity.

Execution Rules for Marked Petri Nets

Having presented the definitions and representations of Petri nets and their markings, we now present the execution rules for marked Petri nets.

A Petri net executes by *firing* transitions. A transition may fire if it is *enabled*. A transition is enabled if each of its input places has at least one token in it. In Figure 17, for example, since the inputs to transition t_2 are places p_2, p_3, and p_5, transition t_2 is enabled if p_2 has at least one token, p_3 has at least one token, and p_5 has at least one token.

A transition fires by removing one token from each of its input places and then depositing one token into each of its output places. Transition t_3 in Fig. 17, with $I(t_3) = \{p_3\}$ and $O(t_3) = \{p_4\}$, is enabled whenever there is at least one token in place p_3. Transition t_3 fires by removing one token from p_3 (its input) and placing one token in p_4 (its output). Extra tokens in p_3 are not affected by firing t_3 although they may enable additional firings of t_3 later. Transition t_2 with $I(t_2) = \{p_2, p_3, p_5\}$ and $O(t_2) = \{p_5\}$ fires by removing one token from each of p_2, p_3, and p_5 and then puts one token in p_5.

Firing a transition will in general change the marking of the Petri net, μ, to a new marking μ'. Note that since only enabled transitions may fire, the number of tokens in each place always remains nonnegative when a transition is fired. Firing a transition can never remove tokens that are not there: if any one of the input places of a transition contains no tokens, then the transition cannot fire. Figure 18 summarizes the possible results of firing a transition. If a place is an input to the transition, one token is removed; if it is an output, one token is added. No net change occurs if the place is neither an input nor an output, or is both an input and an output. In the latter case, it is necessary for a token to be in the input place even though no change in marking occurs for this place.

The State Space of a Petri Net

The *state* of a Petri net is defined by its marking. Thus the firing of a transition represents a change in the state of the net. The *state space* of a Petri net with n places is the set of all markings, i.e., N^n. The

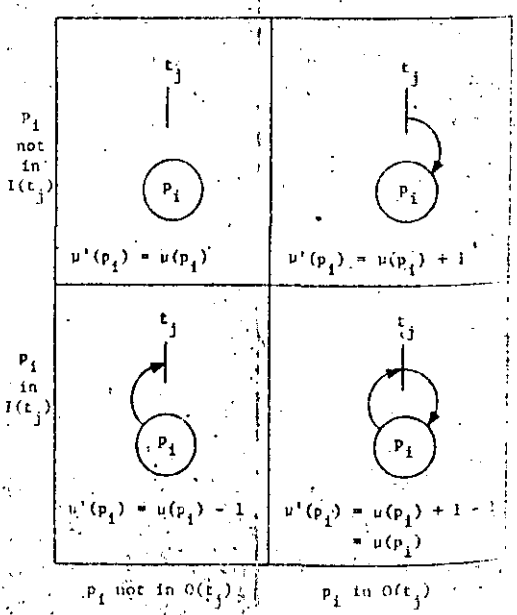


FIGURE 18. The changes in the marking of a place p_i which result from firing a transition t_j.

change in state caused by firing a transition is defined by a partial function δ, called the *next-state function*. Application of this function to a marking μ and a transition t_j yields the value of the marking that results from the firing of transition t_j in marking μ. Since t_j can fire only if it is enabled, $\delta(\mu, t_j)$ is undefined if t_j is not enabled in marking μ. If t_j is enabled, then $\delta(\mu, t_j) = \mu'$, where μ' is the marking that results from removing tokens from the inputs of t_j and adding tokens to the outputs of t_j.

Given a Petri net and an initial marking μ^0, we can execute the Petri net by successive transition firings. Firing a transition t_j in the initial marking produces a new marking $\mu^1 = \delta(\mu^0, t_j)$. In this new marking, we can fire any new enabled transition, say t_k, resulting in a new marking $\mu^2 = \delta(\mu^1, t_k)$. This can continue as long as there is at least one enabled transition in each marking. If we reach a marking in which no transition is enabled, then no transition can fire and the execution of the Petri net must stop.

As an example of Petri net execution, consider the execution of the marked Petri net of Fig. 17. In the marking $\mu^0 = (1, 0, 1, 0, 2)$, two transitions, t_1 and t_3, are enabled. Choosing one arbitrarily, we can fire t_3 producing the marking $\delta(\mu^0, t_3) = (1, 0, 0, 1, 2) = \mu^1$. In this marking, transitions t_1 and t_4 are enabled. Firing t_4 changes the marking to $\delta(\mu^1, t_4) = (1, 1, 1, 0, 2) = \mu^2$. In μ^2, t_1, t_2, and t_3 are enabled. Firing t_1 results in $\mu^3 = \delta(\mu^2, t_1) = (0, 2, 2, 0, 3)$. This process can continue as long as at least one transition is enabled.

Two sequences result from the execution of the Petri net: a sequence of markings $(\mu^0, \mu^1, \mu^2, \cdots)$, and a sequence of transitions $(t_{j(0)}, t_{j(1)}, t_{j(2)}, \cdots)$ such that $\delta(\mu^k, t_{j(k)}) = \mu^{k+1}$ for $k = 0, 1, 2, \cdots$. In the example above the transition sequence was $t_3, t_4, t_1, \cdots$; therefore $j_0 = 3$, $j_1 = 4$, $j_2 = 1, \cdots$. Given the transition sequence and μ^0, we can easily derive the marking sequence for the execution of the Petri net and, except for a few degenerate nets, given the marking sequence we can derive the transition sequence. Both of these sequences thus provide a record of the execution of the net.

The Reachability Set of a Petri Net

From a marking μ, a set of transition firings is possible. The result of firing a transition in a marking μ is a new marking μ'. We say that μ' is *immediately reachable* from μ if we can fire some enabled transition in the marking μ resulting in the marking μ'. A marking μ' is *reachable* from μ if it is immediately reachable from μ or is reachable from any marking which is immediately reachable from μ. We then define the reachability set $R(M)$ for a marked Petri net $M = (P, T, I, O, \mu)$ as the set of all markings which can be reached from μ. This is the reflexive transitive closure of the "immediately reachable" relationship.

The reachability set of a marked Petri net is the set of all states into which the Petri net can enter by any possible execution. Hence many analysis questions deal with properties of the reachability set of a Petri net. (This is discussed in more detail in Section 4.)

Considering a Petri net in terms of states and state changes may obscure some of the more important concepts relating to concurrent systems that can be modeled by Petri nets. One of these is the concept of *local* changes in state, as modeled by transitions. In a complex system composed of independent asynchronously operating subparts, each part can be modeled by a Petri net. The enabling and firing of transitions are then affected by, and in turn affect only, local changes in the marking of the Petri net. Separate parts of the total system may operate independently and concurrently. The view of Petri nets presented here, with a *global* state and a global sequence of transitions, can hide the inherent modularity and concurrency in the Petri net model. However, despite some important objections to this automata-theory-related conception [44], most research in the United States has been based on this approach.

4. ANALYSIS OF PETRI NETS

Why should systems be modeled as Petri nets? Originally, the purpose was mainly *descriptive*. Petri nets with their uniform

and simple execution rules, can be used to describe a system in terms of simple concepts which provide a natural way to depict systems of asynchronous concurrent processes [24]. After a short time, however, it became obvious that another use of Petri nets was to take the description of the system, as a Petri net, and *analyze* it for the presence of desirable or undesirable properties. A body of work is being developed which is aimed at deriving, from a Petri net, properties of the net, and from these, the properties of the system which the net models. Some of the analytic questions that one would like to ask about a Petri net are quite difficult; hence, restricted subclasses of Petri nets have been defined to make analysis easier in specific situations. (This will be discussed in Section 6.)

Following this train of thought, another use of Petri nets would seem to be in the *design* of concurrent systems. One method of design would consist in first creating a design in a traditional representation, for example, a logic circuit or a program. Then the design would be converted into a Petri net and the Petri net analyzed. If no design errors were discovered, the design could then be implemented in the traditional manner. If there were errors, however, it would be necessary to determine how the error which was found in the Petri net representation manifests itself in the original design, modify the design, and repeat the entire process of conversion to a Petri net and analysis. This process could be simplified if the design process could be carried out directly in Petri nets and the resultant Petri net implemented directly. This approach requires that both the necessary design techniques and the methods for implementing Petri net designs, in hardware or software, be developed.

Although some work on design with Petri nets [67] and implementation of Petri nets [73, 27, 75] has been done, it has been limited in scope, presumably because its success hinges on the existence of effective analysis techniques.

Analysis Questions

The first task in developing analysis techniques is to define the types of questions that the analysis procedures are to answer, and the properties to be studied. Obviously, the analysis techniques should be oriented towards the solution of those problems that most need to be solved rather than towards areas that are only of academic interest. We discuss now some of the properties that have been investigated for Petri nets.

One property of Petri nets derives from their original definition in terms of events and conditions. A condition is represented by a place. The fact that the condition holds is indicated by a token in the place. Consider, however, that either a condition holds or it does not hold. Hence, a token should either be present or it should be absent. Also, no more than one token should ever be present in one place at one time, as it seems pointless to have multiple tokens when one is sufficient. Petri nets which are constructed such that no more than one token can ever be in any place of the net at the same time are *safe* nets [44].

Another definition of a safe net is the statement that there is a bound on the number of tokens in any place of the net, and that bound is 1. A natural generalization of this is to allow multiple tokens in a place but only to the extent that there are no more than k tokens in any given place at the same time. Nets in which the number of tokens in any place is bounded by k are called *k-bounded* nets. (Thus a safe net is a 1-bounded net.) If a net is k-bounded for some k but we do not know the value of k, it is simply called *bounded*.

Boundedness is a very important practical property of Petri nets. If we wish to implement a design modeled by a Petri net, then since the capacity of any given hardware is bounded, the Petri net must be bounded if construction is to be possible. In other words, if, for example, places are to be implemented as counters, then since every physically realizable counter can only hold a bounded number, the net must also be bounded.

Another property that might be important is conservation of tokens. If tokens are used to represent resources, then it follows that since resources are neither created nor destroyed, tokens should also

be neither created nor destroyed. A Petri net is *conservative* if the number of tokens in the net is conserved. This implies that each transition in a conservative net is conservative, in the sense that the number of inputs of each firable transition is equal to the number of outputs of that transition. More generally, weights can be defined for each place allowing the number of tokens to change as long as the weighted sum is constant [62].

Notice that the above statement was qualified to restrict it to firable transitions. Consider the Petri net of Fig. 19, which depicts a nonfirable transition. Transitions t_1 and t_2 are conservative. Transition t_3 is not conservative, in the sense that if it ever fired it would decrease the number of tokens by two. However, for any initial marking in which the number of tokens in p_3 is zero, this transition is not firable and hence the number of tokens in the net is conserved. This means, then, that the transition t_3 and place p_3 can be deleted from the net, simplifying it, with no change in its behavior. This would allow a simpler and cheaper implementation.

The notion of transitions that cannot fire seems strange, and we want to be able to identify such transitions. Note that a transition which cannot fire is not simply a transition which is not enabled, but rather a transition which cannot become enabled. A transition is *dead* in a marking if there is no sequence of transition firings that can enable it. A transition is *potentially firable* if there exists some sequence that enables it [35]. A transition of a Petri net is *live* if it is potentially firable in all reachable markings.

The importance of the concepts of liveness and deadness of transitions comes from considerations in the modeling of operating systems. Liveness is tied to the concept of deadlocks and deadlock-freeness [45]. Thus it may be important not only that a transition be firable in a given marking, but that it stay potentially firable in all markings reachable from that marking. If this is not true, then it is possible to reach a state in which the transition is dead, perhaps signifying a possible deadlock.

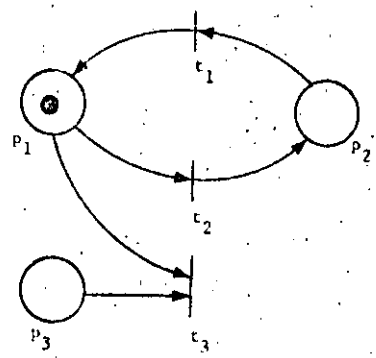


FIGURE 19. A Petri net with a nonfirable transition. Transition t_3 is dead in this marking.

A number of different definitions of liveness have been considered. Commoner [16] defines four subtly different forms of liveness for a transition t_j and a marking μ:

- L_1: If there exists a μ' in $R(M)$ such that $\delta(\mu', t_j)$ is defined (i.e., t_j is potentially firable);
- L_2: If for every positive integer n there exists a transition sequence σ such that $\delta(\mu, \sigma)$ is defined and t_j appears at least n times in σ;
- L_3: If there exists an infinite sequence of transition firings such that $\delta(\mu, \sigma)$ is defined and t_j appears infinitely often in σ;
- L_4: If t_j is live (i.e., potentially firable in all reachable markings).

Note that the implications of the four definitions of liveness are quite different (see [53]). Thus, whenever the property of liveness in a Petri net is discussed, it is important to state carefully the definition being used, since we do not yet have a single commonly accepted definition.

As mentioned earlier, the concept of liveness was developed to deal with deadlock problems in operating systems. Other problems in operating systems can also be posed in terms of Petri nets. The actual statement of these questions depends upon the manner in which the system is modeled. For example, access to a resource may be modeled by a transition or a place. The mutual exclusion problem is to assure that at most one of perhaps several processes tries to access the resource at the same time. Depending on the modeling used, this will be expressed as a question concerning whether or not two transitions

can be enabled simultaneously or whether or not two places may have tokens simultaneously.

Notice, however, that both of these questions can be stated in terms of the reachability of any of a set of undesirable states. In fact, many questions can often be reduced to the *reachability problem*. The reachability problem is simply the following: Given a marked Petri net (with marking μ) and a marking μ', is μ' reachable from μ? This problem is very important to the analysis of Petri nets. It can be considered a special case of the *set reachability problem*, which is to determine if a set of markings, $S = \{\mu^1, \mu^2, \ldots \mu^k\}$, is a subset of the reachability set $R(M)$ of a marked Petri net.

There are many other interesting questions that might be studied with Petri nets. Furthermore, since the questions designers want to ask about their designs depend on the projected use of the designs, there will always be new questions. Thus it is important to develop general techniques that allow new types of questions to be answered. The basis for the importance of the reachability problem is that many questions about the correctness of systems modeled as Petri nets can be translated into instances of this problem. For instance, Hack has shown that the liveness problem (are all transitions live?) is reducible to the reachability problem and that in fact the two problems are equivalent, since reachability is also reducible to liveness [35].

Solution Techniques

While several approaches to the analysis of Petri nets have been considered, almost all work in this area eventually uses one basic technique. This technique involves finding a finite representation for the reachability set of a Petri net, in recognition of the fact that many of the properties of a Petri net are based on properties of its reachability set. The representation used is known as the *reachability tree*. It consists of a tree whose nodes represent markings of the Petri net and whose arcs represent the possible changes in state resulting from the firing of transitions [51, 53].

Notice, however, that the reachability set of a marked Petri net is often infinite. Thus, to form a finite representation of an infinite set we must map many markings into the same node of the tree. This many-to-one mapping is accomplished by collapsing a set of states into a node by ignoring the number of tokens in a place of the net when this number becomes "too large." This is represented by using a special symbol, ω, for the number of tokens in this place.

The symbol ω represents a value which can be arbitrarily large. Because of this we must interpret the operations of addition, subtraction, and comparison as

$$\omega + a = \omega$$
$$\omega - a = \omega$$
$$a < \omega$$

for any natural number a. Thus, ω might be thought of as a symbol for infinity.

Each node in the reachability tree is labeled with a marking; arcs are labeled with transitions. The initial node (root of the reachability tree) is labeled with the initial marking. Given a node x in the tree, additional nodes are added to the tree for all markings that are directly reachable from the marking of the node x. For each transition t_j which is enabled in the marking for node x, a new node with marking $\delta(x, t_j)$ is created, and an arc labeled t_j is directed from the node x to this new node. This process is repeated for all new nodes.

Continuing this process will obviously create the entire state-space. A path from the initial marking (root) to a node in the tree corresponds to an execution sequence. Since the state-space may be infinite, two special steps are taken to define a finite reachability tree. First, if a new marking is generated which is equal to an existing marking on the path from the root node to the new marking, the new (duplicate) marking becomes a terminal node. Since the new marking is equal to the previous marking, all markings reachable from it have already been added to the reachability tree by the earlier identical marking.

Second, if any new marking x is generated which is greater than a marking y on

the path from the root node to the marking x, then those components of marking x which are strictly greater than the corresponding components of marking y are replaced by the symbol ω. Since marking x is greater than marking y, any sequence of transition firings which is possible from marking y is also possible from marking x. In particular, the sequence that transformed marking y into marking x can be repeated indefinitely, each time increasing the number of tokens in those places which have a ω. Thus the number of tokens in these places can be made arbitrarily large.

As an example of this construction, consider the marked Petri net of Fig. 20. We begin with (1, 0, 1, 0) as the root of the tree. In this marking, we have only one enabled transition. Thus we have a new node corresponding to firing t_3, (1, 0, 0, 1) and an arc from (1, 0, 1, 0) to (1, 0, 0, 1). From this marking we can fire t_2, resulting in (1, 1, 1, 0). Now, since (1, 1, 1, 0) $\geq$ (1, 0, 1, 0), we replace the second component by ω. This reflects the fact that we can fire the sequence t_3t_2 an arbitrary number of times and make the number of tokens in place p_2 as large as desired. In the marking (1, ω, 1, 0), two transitions are enabled, t_1 and t_3. Firing these two would give us two new nodes, (1, ω, 0, 0) and (1, ω, 0, 1). The first of these has no successors since $\delta((1, \omega, 0, 0), t_j)$ is undefined for all t_j. The second enables t_2, which fires to give (1, ω, 1, 0) which is identical to an earlier node. Thus, the complete reachability tree is as shown in Fig. 21.

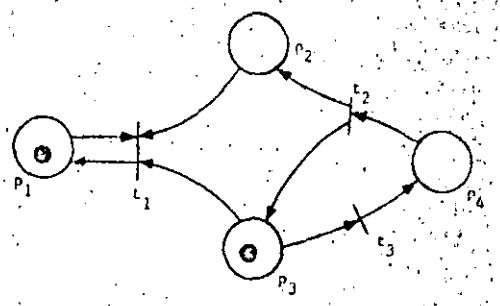


FIGURE 20. A Petri net with marking (1,0,1,0) and infinite reachable state-space.

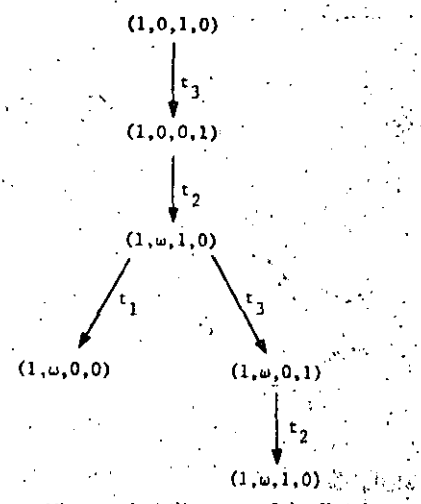


FIGURE 21. The reachability tree of the Petri net of Fig. 19.

Analysis Using the Reachability Tree

How is the reachability tree used for analysis? Let us consider some of the questions raised in the previous section.

On the questions of safeness and boundedness, notice that if a Petri net is k-bounded, then, by definition, no more than k tokens are ever in any place. Thus the possible values for each place are drawn from the set $\{0, 1, \ldots, k\}$ and there are only $(k + 1)^n$ possible reachable markings. Therefore, the reachable state-space is finite.

In the same way, consider a conservative Petri net. If we let k be the number of tokens in the net, then we must always have k tokens in the net. Since there are only a finite number of ways to partition k tokens among n places, we must again have a finite reachability set.

Now consider the reachability tree. If any node in the reachability tree contains the symbol ω, then that component can become arbitrarily large, i.e., it is not bounded. Thus, if the symbol ω is anywhere in the reachability tree, the reachability set is not finite and hence neither bounded nor conservative. On the other hand, if the ω symbol does not occur anywhere in the tree, then the reachability tree *is* the reachability set and both are finite. This means that the reachability set is bounded, and the bound can be established by inspection. Similarly, if the

reachability set is finite, conservation can be determined by inspection. In fact, for a finite reachability set, any analysis question can be solved by inspection.

Other problems can also be solved using the reachability tree. For example, the *coverability problem* can be solved by inspection of the reachability tree [51, 35]. The coverability problem is the following: Given a marked Petri net M and a marking μ, does there exist a marking μ' in $R(M)$ such that $\mu' \geq \mu$? This problem is useful in determining whether violations of mutual exclusion occur in a system, and in testing transitions for liveness (deadlock).

The Reachability Problem

The more general questions of liveness and reachability are not answerable by the reachability tree. Because of the fundamental nature of the reachability problem in the analysis of Petri nets and vector addition systems—an equivalent modeling system [51, 34]—it has been the object of a considerable amount of research. It has been shown that the general reachability problem is equivalent to several special cases such as the zero reachability problem (is the zero vector an element of the reachability set?) [70] and the subset reachability problem (given a nonempty subset of places and a marking μ, is any reachable marking equal to μ for the specified subset of places, with all other places allowed to have any value?) [34]. These problems are equivalent in that if an algorithm can be found to solve any one of them it can be modified to solve any of the others.

Such an algorithm has recently been found [88]. The algorithm is very difficult to follow and depends upon both a search through a finite tree of possible solutions and the recursive solution of reachability problems for lower-dimensional state-spaces. However, regardless of the complexity of the algorithm, its existence shows that the reachability problem is solvable (although possibly at a high cost).

Since the liveness problem is equivalent to the reachability problem [35], it also is solvable.

Unsolvable Problems

Some Petri-net problems are not solvable despite their apparent similarity to the reachability problem. The first such problem studied was the subset problem: given two marked Petri nets, is the reachability set of one net a subset of the reachability set of the other net? Rabin showed this problem to be undecidable [9, 33]. Later Hack showed that the equality problem—given two marked Petri nets, is the reachability set of one net equal to the reachability set of the other net?—is also undecidable [36].

These problems are important for applications in which one might want the Petri nets to be optimized, but the set of reachable markings not to be changed. Unfortunately, it has been shown that both of these problems are *undecidable* in the sense that there exists no general algorithm which can decide, for two arbitrary marked Petri nets, if their reachability sets are equal or one is the subset of the other. This proof is quite complicated. It is based on the construction of a Petri net which (weakly) computes the value of a polynomial in such a way that if the equality problem is decidable then Hilbert's tenth problem is solvable. Since Hilbert's tenth problem is known to be unsolvable, the equality problem is undecidable. This in turn implies the undecidability of the subset problem.

Complexity

While much attention has been focused on the decidability of the reachability problem and similar problems, other aspects of Petri net analysis procedures have also been investigated. One aspect that has recently come under investigation is the computational complexity of the problem. While it is not yet possible to determine the complexity exactly, it is possible to give its lower bounds. Lipton has shown that the reachability problem is exponential time-hard and exponential space-hard [64]. That is, the amount of time and memory space needed to solve the reachability problem must be at least an exponential function of the length of the input descrip-

tion of the Petri net (in the worst case). This is a lower bound; the actual complexity could be much worse. Lipton also shows that the coverability problem has an exponential space lower bound. Rackoff [83] has obtained an algorithm for solving coverability in exponential space, showing this to be a tight lower bound. The complexity of some other problems in Petri nets has also been considered [49].

These complexity analyses are very important in determining the usefulness of Petri nets for the modeling and analysis of systems. The recent discovery that the reachability problem is decidable marked a significant advance in the search for analysis techniques; however, the complexity bounds as well as the large number of places and transitions needed to model even simple systems tend to indicate that, although analysis questions may be decidable using Petri nets, in the worse case the cost of answering even simple questions may make such analysis unfeasible.

5. PETRI NET LANGUAGES

Another area in which Petri nets have been used is the study of formal languages (see [79, 37]). Here Petri nets are used to model the flow of information and control of actions in a system. The firing of a transition models the occurrence of an operation in the modeled system. A Petri net properly models a system if every sequence of actions in the modeled system is possible in the Petri net and every sequence of actions in the Petri net represents a possible sequence in the modeled system.

To represent these concepts, we label the transitions of a Petri net; with each transition we associate a symbol naming the transition. Since there are only a finite number of transitions, we can define a finite alphabet Σ which is the set of all these symbols. A labeling function σ maps transitions to symbols, i.e., $\sigma: T \to \Sigma$. A labeled marked Petri net defines a set of strings over Σ, each string corresponding to a possible execution of the net. The set of all possible strings corresponding to the possible executions of a marked labeled Petri net defines a *Petri net language*.

Several varieties of Petri net languages result from slightly different approaches to defining the languages of a Petri net. One entire group of languages results from the use of different labeling policies, since restrictions on the allowable labeling functions create restricted classes of Petri net languages.

The *free* languages are those obtained when one introduces the requirement that all transition labels be distinct and nonnull, i.e., $\sigma(t_i) \neq \sigma(t_j)$ for $t_i \neq t_j$. This requirement reflects the view that since distinct transitions model distinct events, they should be distinctly labeled.

A more general class of languages results if one allows a more general label function in which many transitions may be labeled with the same symbol. This reflects the view that the same action can result from different circumstances, and hence may be modeled by different transitions. The modeling process may even introduce some "extra" transitions that are necessary for proper token movement but do not correspond to actions of interest in the modeled (real) system.

A third class of labeling functions allows transitions to be labeled with the *null* label λ. A null label is defined as a label which does not show up in the string resulting from an execution of the Petri net.

As an example of the differences of these labeling policies, consider the Petri net of Fig. 22. Let the language under consideration be the set of sequences whose net result is to move the token in place p_1 to place p_4 (i.e., the set of sequences $\{t \in T^* \mid \delta((1, 0, 0, 0), t) = (0, 0, 0, 1)\}$). If we label the transitions with the free labeling

$$\sigma_1(t_1) = a \qquad \sigma_1(t_3) = b$$
$$\sigma_1(t_2) = c \qquad \sigma_1(t_4) = d,$$

then the language is $\{a^n c b^n d \mid n \geq 0\}$. A nonfree but λ-free labeling such as

$$\sigma_2(t_1) = a \qquad \sigma_2(t_3) = b$$
$$\sigma_2(t_2) = a \qquad \sigma_2(t_4) = b$$

yields the language $\{a^n b^n \mid n > 0\}$. If transition t_1 is assigned a null label, then the language which results could be the regular language $\{ab^*c\}$.

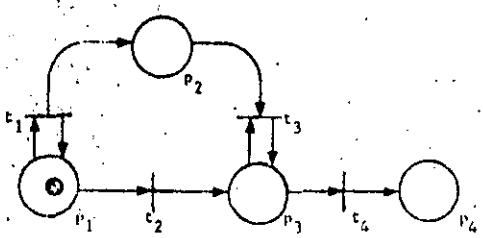


FIGURE 22. A Petri net with marking (1,0,0,0).

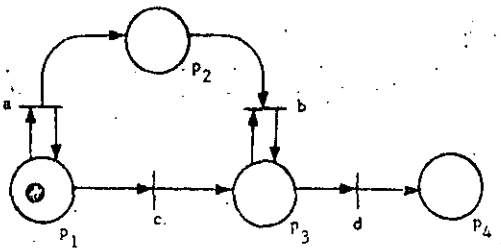


FIGURE 23. Labeled marked Petri net corresponding to the net of Fig. 22.

The class of labeling functions is only one of the determinates of Petri net languages. Another is the definition of the set of final states. Remember that a language is the set of all possible sequences resulting from the execution of a labeled Petri net starting in an initial marking (or one of a finite set of initial markings) and terminating in any element of a set of *final markings*. Different classes of languages correspond to different definitions of the set of final markings.

Four types of Petri net languages have been defined in terms of the definition of final markings [39]:

L-type: The set of final markings is defined by a finite final marking set F;

G-type: Given a finite marking set F, a final marking is any marking which is greater than or equal to any element of F;

T-type: A final marking is any terminal marking (a marking in which no transition is enabled);

P-type: All reachable markings are final markings.

As an example of the differences between these different language classes, consider the Petri net of Fig. 23, which is a labeled version of the net of Fig. 22. (The labeling shown is free, but that is not important.) For a final state set $F = \{(0, 0, 1, 0)\}$, the L-type language is $\{a^n cb^n \mid n \geq 0\}$, the G-type language is $\{a^m cb^n \mid m \geq n \geq 0\}$, the T-type language is $\{a^m cb^n d \mid m \geq n \geq 0\}$, and the P-type language is $\{a^m \mid m \geq 0\} \cup \{a^m cb^n \mid m \geq n \geq 0\} \cup \{a^m cb^n d \mid m \geq n \geq 0\}$.

With three different kinds of labeling functions and four different kinds of final-state sets, twelve different classes of Petri net languages can be defined. Despite their differences, these classes are closely related. Preliminary investigations have shown that a number of containment relationships hold between the classes. For example, since the labeling functions are successively more general, all Petri net languages with free labelings are also Petri net languages with λ-free labelings which in turn are also Petri net languages with arbitrary labelings. It can be shown that all P-type languages are also G-type languages, that all G-type and T-type languages with arbitrary or λ-free labelings are L-type languages with the same type of labeling, and that L-type languages with arbitrary labelings are also T-type languages. These relationships are shown in Fig. 24, in which an arc between classes is used to indicate that one class is contained within the other. It is not known whether other arcs might also exist or which containments are proper.

The L-type and P-type languages have been investigated in greater depth. L-type languages have been shown to be closed under union, intersection, concatenation, concurrency, reversal and λ-free homo-

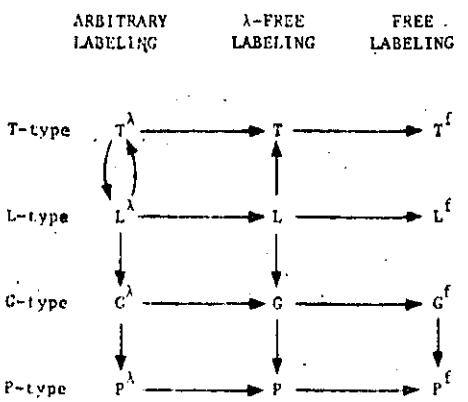


FIGURE 24. Relationships among the different classes of Petri net languages.

morphism [77, 37]. P-type languages are more restrictive but are still closed under union, intersection, concatenation and concurrency [37]. Hack has developed a characterization theorem for L-type languages showing that the class of L-type languages is the smallest of languages that includes a finite language and the complete parenthesis language and is closed under inverse homomorphism, concurrency, intersection and λ-free homomorphism. (The complete parenthesis language is the context-free language over two symbols {(,)} whose strings are properly nested parenthesis strings [37].)

The relation of Petri net languages (of the L-type) and other classes of languages has also been examined. All regular languages are Petri net languages. Some context-free languages are Petri net languages and some Petri net languages are context-free, but neither class includes the other. Their common intersection includes regular languages and bounded context-free languages, among others. Surprisingly, the complement of a free Petri net language is context-free [18]. All λ-free Petri net languages are context-sensitive [77]. These relationships are illustrated in Fig. 25. In this diagram, an arrow between two classes of languages indicates proper containment. Note that Petri net languages appear to be roughly equivalent to context-free languages in complexity (and interest).

The original impetus for studying Petri net languages was to try to settle some of the decidability questions for Petri nets. It has been shown [37] that the membership problem for Petri nets with λ-free or free labelings is decidable, but the inclusion and equivalence problems for P, P^λ, L and L^λ languages are undecidable. Many decidability problems are equivalent to the reachability problem and thus decidable.

A different approach to the study of Petri nets by the use of formal language theory has been considered by Crespi-Reghizzi and Mandrioli [19]. They noticed the similarity between the firing of a transition and the application of a production in a derivation in which places are nonterminals and tokens are separate instances of the nonterminals. The major difference of this approach is the lack of ordering information in the Petri net contained in the sentential form of the derivation. To accommodate it, Crespi-Reghizzi and Mandrioli defined the *commutative grammars*, which are isomorphic to (generalized) Petri nets. In addition, they considered the relationship of Petri nets to matrix, scattered-context, non-terminal-bounded, derivation-bounded, equal-matrix, and Szilard languages. For example, it is not difficult to see that the class L' is the set of Szilard languages of matrix context-free languages [21]. Similar work by Keller considered the class of commutative semi-Thue systems [53]. Keller has also pointed out that λ-free languages are a subset of real-time counter languages [26].

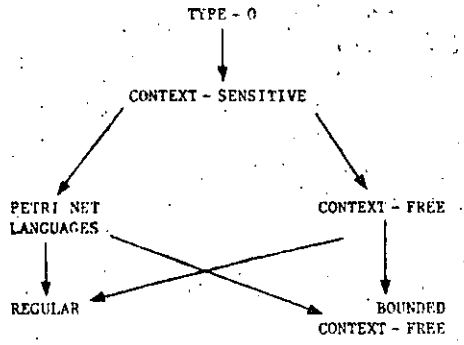


FIGURE 25. Relationships among Petri net languages and the classical language classes.

It should be pointed out that this entire approach to Petri nets and languages may represent an approach from the wrong direction: Petri nets were designed to represent concurrent activity, yet the representation of a Petri net execution by a string forces all activity to be represented serially, incorrectly implying a total ordering between events. Some work has considered other representations of the partial orderings resulting from concurrent activities [43, 85], but further research is needed in this area.

6. EXTENSIONS AND SUBCLASSES

The success of any model is due to two factors: its modeling power and its decision power. Modeling power refers to the abil-

ity to correctly represent the system to be modeled so that the model will be a faithful representation of the modeled system. Decision power refers to the ability to analyze specific versions of the model and determine properties of the modeled system [54].

These two factors generally work at cross purposes. Consider, for example, finite-state systems. Since the set of reachable states is finite, it is possible to answer almost any question about a finite-state model; hence such a model has very high decision power. On the other hand, the class of systems which can be modeled is severely limited, which means that such a model has very low modeling power. Turing machines, by contrast, have good modeling power but, since most general questions are undecidable, have poor decision power. When we increase modeling power (and hence the complexity of the models and the modeled systems), our ability to algorithmically determine the properties of the models is generally decreased.

Petri net models represent an attempt to compromise between these two factors. They have better modeling power than finite-state models while (one hopes) retaining most of the latter's decision power. As a matter of fact, Petri nets were originally defined in answer to the limited modeling power of finite-state models.

Not all researchers have been satisfied with the modeling power of Petri nets, however. It is difficult to model some events or conditions in systems by Petri nets, and it has been shown that the correct modeling of other relatively reasonable systems is impossible [3, 57]. Thus several proposals have been put forth for extending the modeling power of Petri nets.

Extended Petri Nets

One of the first extensions is to remove the constraint that a place may contribute or receive only one token from the firing of a transition. Consider the modeling of chemical reactions. Here a token in a place represents the availability of a certain molecule or atom. Chemical reactions are modeled by transitions and may occur whenever tokens indicate the availability of the reactants. The firing of the transition models the reaction, which consumes inputs (reactants) and produces outputs (products). Notice that a chemical reaction may well require more than one unit of a particular reactant. This is modeled by allowing multiple arcs between transitions and places, signifying the number of tokens needed. Figure 26 illustrates a Petri net model of a reaction that needs three Cl_2 and two P to produce two PCl_3. In order for the transition to fire, at least three Cl_2 and two P must be available. The firing of the transition absorbs these tokens and produces two tokens in its output place.

Petri nets that allow multiple arcs have been called *generalized Petri nets* [34, 54]. Hack has shown that these nets are equivalent to ordinary Petri nets (at most one arc between a place and a transition). Hence although this change may increase the convenience of use, it does not change the fundamental modeling power or decision power of Petri nets. Most researchers thus use generalized Petri nets in their work, often ignoring the distinction between them and what we have defined as Petri nets in this paper.

A more fundamental extension of Petri nets was undertaken by a number of authors [1, 5, 73] in response to difficulties in the modeling of priority systems with Petri nets. This extension involves so-called zero-testing [53]: the introduction of arcs from a place p_i to a transition t_j which allow the transition to fire only if the place

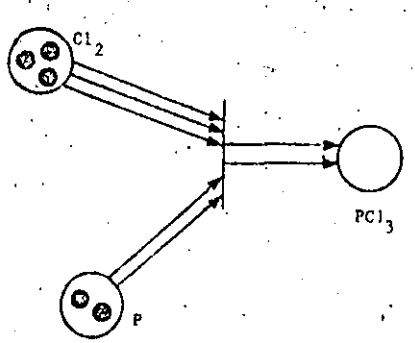


FIGURE 26. Petri net model of a chemical reaction, illustrating the concept of multiple input and output arcs between a transition and a place.

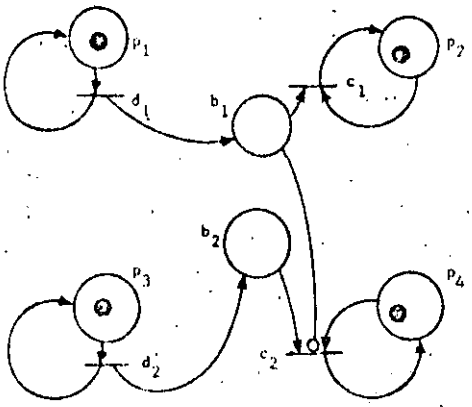


FIGURE 27. An extended Petri net which has no equivalent narrowly defined Petri net.

p_i has zero tokens in it. These special arcs have been drawn in several ways. We represent them as shown in Fig. 27. Note that transition c_2 can fire only if places p_4 and b_2 each have at least one token in them and place b_1 has exactly zero tokens. The arc from b_1 to c_2 is called an *inhibitor* arc; it gives transition c_1 priority over transition c_2.

The addition of inhibitor arcs is a major extension of the concept of Petri nets. Agerwala has shown that Petri nets extended in this manner have the modeling power of a Turing machine and hence can also be used to show that many decision problems are undecidable [1]. Many other extensions of Petri nets including the introduction of priorities between transitions, time bounds on transition firings [66], or constraint sets that prohibit tokens residing simultaneously in two places [73] are equivalent to Petri nets with inhibitor arcs and hence to Turing machines. In terms of modeling power Petri nets seem to be just below Turing machines, so that any significant extension results in Turing-machine equivalence [78].

Subclasses of Petri Nets

It was hoped that the limitations on the modeling power of Petri nets relative to Turing machines would be balanced by a compensating increase in decision power. This appears to be the case, since for Petri nets many decision problems are equivalent to the reachability problem, which has been shown to be decidable. However, research on the complexity of the reachability problem has shown that even though it is decidable, it is very difficult to solve. Thus, from a practical point of view, Petri nets may be too powerful to be analyzed.

The result of this has been the definition of a number of subclasses of Petri nets, in hopes of finding a subclass with (known) decision power and still adequate modeling power for practical purposes. These subclasses are defined by restrictions on their structure intended to improve their analyzability.

Two subclasses are most commonly considered, state machines and marked graphs [44]. State machines are Petri nets which are restricted so that each transition has exactly one input and one output. These nets are obviously conservative and hence finite-state. In fact, they are exactly the class of finite-state machines. This is clearly shown by considering the state graph of a finite-state machine, as in Fig. 28a. The nodes of this graph represent the states of the finite-state machine. An arc from state i to state j labeled x indicates that there is a transition from state i to state j with input x. Note that this state graph is nondeterministic. The graph of Fig. 28a can be converted to an equivalent Petri net by simply making each state a place, and making each arc between two places a transition. This is illustrated in Fig. 28b. Note that this Petri net is conservative. If the state graph had been nondeterministic, then the Petri net would also have this characteristic. Finite-state machines, being finite, have very high decision power, but they are of limited usefulness in modeling systems which are not finite.

Marked graphs, the dual of state machines, have also been studied extensively [17, 44]. A *marked graph* is a Petri net in which each place has exactly one input transition and one output transition. Algorithms are known for showing that a marked graph is live and safe, and for solving the reachability problem for marked graphs. Thus, marked graphs have high decision power. They have limited modeling power, however, since they

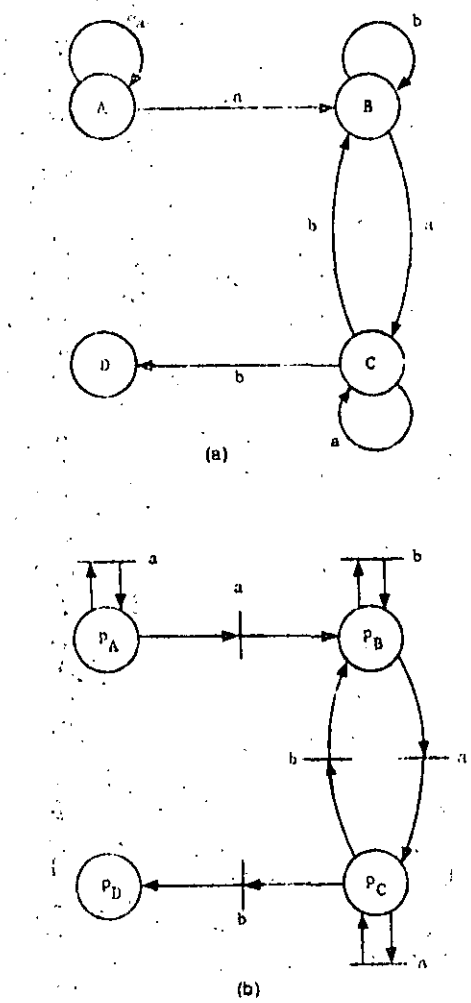


FIGURE 28. Equivalent models of a finite-state machine. (a) State graph model. (b) Petri net model.

are able to model only those systems whose control flow has no branches. In other words, parallel activities can be easily modeled, but not alternative activities.

The problem with modeling data-dependent decisions (branches) in a Petri net is that conflicts may arise, and nets with conflicts seem to be difficult to analyze. Hack has investigated the class of *free-choice Petri nets* [32] in which each arc from a place is either the unique output of the place, or the unique input to a transition. This restriction means that if there is a token in a place then either the token will remain in that place until its unique output transition fires or, if there are multiple outputs for the place, then there is a free choice as to which of the transitions is fired.

Hack and Commoner have shown that liveness and safeness for free-choice Petri nets are decidable and have given necessary and sufficient conditions for these properties. Hack has also shown that free-choice nets can model a class of systems called *production schemata* which are similar to assembly-line systems.

Other subclasses of Petri nets have been defined [32], for example simple Petri nets, but little analysis of them has been done to date. Figure 29 shows allowed and disallowed situations for three subclasses. Landweber and Robertson [58] have studied the classes of conflict-free and persistent Petri nets.

Related Models

Any discussion of Petri nets would be incomplete without a mention of vector addition systems. These systems were defined by Karp and Miller [51] and are equivalent

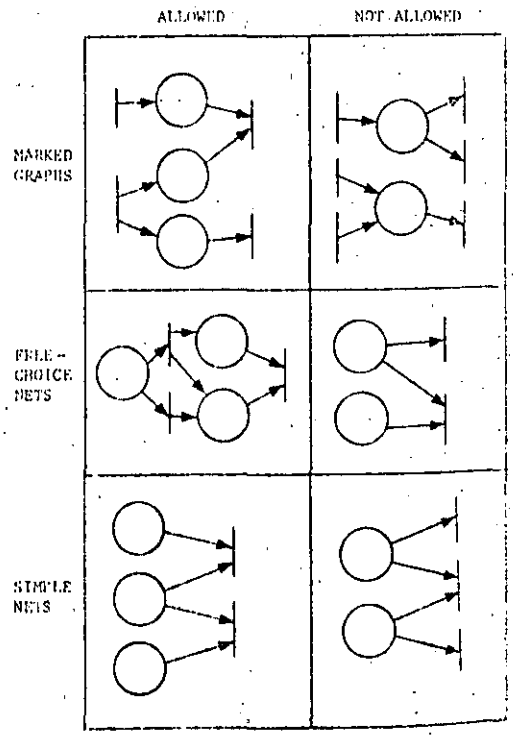


FIGURE 29. Differences between the subclasses of Petri nets.

to Petri nets [34]. A vector addition system is essentially a mathematical formulation, in terms of vectors, of the markings and transitions of a Petri net. Since the mathematical formulation is more convenient for formal manipulation than Petri net graphs, many results are given in terms of vector addition systems, although they apply equally to Petri nets. Vector replacement systems [53] are a related (and equivalent) model based on a generalization of the vector addition systems.

It should also be pointed out that Petri nets are far from the only model of concurrent systems to have been developed. The many other models developed to date include program graphs [87], computation graphs [50], message transmission systems [85], flow graph schemata [89], and complex bilogic directed graphs [31]. Baer has published a survey of some of these models [6]. A comparison of the properties of many of these models [78] has shown that most of them are either subclasses of Petri nets or equivalent to Petri nets. These results have been reinforced by the comparisons of Agerwala [2] which arrive at much the same conclusion concerning the relative modeling power of the various models. The definition of equivalence must be carefully considered, however. Lipton, Snyder, and Zalcstein [63] have compared models using a definition of equivalence considerably different but no less valid than those of Peterson and Bredt or Agerwala, and arrived at important differences in the modeling power of the various models of concurrent systems.

CONCLUSIONS

The Petri net has been defined as a model for systems exhibiting concurrent asynchronous activities. The major factors that might affect its acceptance are concerns regarding the modeling power and decision power of the model. Although Petri nets are not the only models of asynchronous concurrent systems, they are equivalent to or include most other models. In addition they have a certain clearness and cleanness which permits a simple and natural representation of many systems. Thus they have gained increasing acceptance in the last decade, and their use is growing.

A major modeling system must provide more than simply a convenient representation system, however. It must also provide analysis procedures that can be used to determine properties of the modeled system through the model. Some such analysis procedures for Petri nets do exist, allowing the analysis of systems for boundedness, conservation, coverability, and reachability of a marking. However, other properties, such as inclusion or equivalence of two Petri nets, have been shown to be undecidable. Even though problems such as reachability may be decidable, complexity results tend to indicate that these problems may be intractable, requiring too much computational time and space to be practical. Any significant extension of the Petri net model tends to be equivalent to a Turing machine, and hence analysis of these extensions is not possible due to decidability problems. The subclasses which have been examined have good decision properties, but may be too limited for useful modeling. On this topic as on many others relating to Petri nets, much work remains to be done.

ACKNOWLEDGMENTS

I am grateful to R. M. Keller, M. Hack, L. H. Landweber, D. Mandrioli, E. L. Robertson, the referees, and the editors for their suggestions and comments.

BIBLIOGRAPHY

[1] AGERWALA, T. *A complete model for representing the coordination of asynchronous processes*, Hopkins Computer Research Report No. 32, Computer Science Program, Johns Hopkins Univ., Baltimore, Md., July 1974, 58 pp.

[2] AGERWALA, T. *An analysis of controlling agents for asynchronous processes*, Hopkins Computer Research Report No. 35, Computer Science Program, Johns Hopkins Univ., Baltimore, Md., Aug. 1974, 85 pp.

[3] AGERWALA, T.; AND FLYNN M. "Comments on capabilities, limitations and 'correctness' of Petri nets," in *Proc. 1st Annual Symp. Computer Architecture*, G. J. Lipovski, and S. A. Szygenda (Eds.), ACM, N.Y., 1973, pp. 81–86.

[4] ANDERSON, D. W.; SPARACIO, F. J.; AND TOMASULO, R. M. "The IBM System/360 Model 91: Machine philosophy and instruction han-

dling," *IBM J. R. & D.* 11, 1 (Jan. 1967), 8-24.

[5] BAER, J. L. "Modelling for parallel computation: a case study," in *Proc. 1973 Sagamore Computer Conf. Parallel Processing*, Springer-Verlag, N.Y., 1973.

[6] BAER, J. L. "A survey of some theoretical aspects of multiprocessing," *Computing Surveys* 5, 1 (March 1973), 31-80.

[7] BAKER, H. G. *Petri nets and languages*, Computation Structures Group Memo 68, Project MAC, MIT, Cambridge, Mass., May 1972, 6 pp.

[8] BAKER, H. G. "Equivalence problems of Petri nets," MS Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., June 1973, 53 pp.

[9] BAKER, H. G. *Rabin's proof of the undecidability of the reachability set inclusion problem of vector addition systems*, Computation Structures Group Memo 79, Project MAC, MIT, Cambridge, Mass., July 1973, 18 pp.

[10] BERNSTEIN, A. J. "Program analysis for parallel processing," *IEEE Trans. Electronic Comp.* EC-15, (Oct. 1966), 757-762.

[11] BERNSTEIN, P. A. *Description problems in the modeling of asynchronous computer systems*, Tech. Rep. 48, Dept. Computer Science, Univ. Toronto, Toronto, Canada, Jan. 1973.

[12] CERF, V. G. "Multiprocessors, semaphores, and a graph model of computation," PhD Thesis, Computer Science Dept., Univ. Calif., Los Angeles, April 1972.

[13] CHEN, T. C. "Overlap and pipeline processing," in *Introduction to computer architecture*, H. S. Stone (Ed.), Science Research Associates, Chicago, Ill., 1975, pp. 375-431.

[14] CLARK, W. A. "Macromodular computer systems," in *Proc. 1967 Spring Jt. Comp. Conf.*, Thompson Book Co., Washington, D.C., 1967, pp. 335-336.

[15] COFFMAN, E. G.; AND DENNING, P. J. *Operating systems theory*, Ch. 2, Prentice-Hall, Englewood Cliffs, N. J., 1973, pp. 31-82.

[16] COMMONER, F. G. *Deadlocks in Petri nets*, CA-7206-2311, Applied Data Research, Wakefield, Mass., June 1972, 50 pp.

[17] COMMONER, F.; HOLT, A. W.; EVEN, S.; AND PNUELI, A. "Marked directed graphs," *J. Computer and Systems Science* 5, (Oct. 1971), 511-523.

[18] CRESPI-REGHIZZI, S.; AND MANDRIOLI, D. "Properties of firing sequences," presented at *MIT Conf. Petri Nets and Related Methods*, MIT, Cambridge, Mass., July 1975.

[19] CRESPI-REGHIZZI, S.; AND MANDRIOLI, D. *Petri nets and commutative grammars*, Internal Report No. 74-5, Laboratorio di Calcolatori, Instituto di Elettrotecnica ed Elettronica del Politecnico di Milano, Italy, March 1974.

[20] CRESPI-REGHIZZI, S.; AND MANDRIOLI, D. "A decidability theorem for a class of vector-addition systems," *Information Processing Letters* 3, 3 (Jan. 1975), 78-80.

[21] CRESPI-REGHIZZI, S.; AND MANDRIOLI, D. "Petri nets and Szilard languages," *Information and Control* 33, 2 (Feb. 1977), 177-192.

[22] DENNIS, J. B. "Modular asynchronous control structures for a high performance processor," in *Record of the Project MAC Conf. Concurrent and Parallel Computation*, ACM, N. Y., 1970, pp. 55-80.

[23] DENNIS, J. B. (Ed.), *Record of the project MAC conf. concurrent systems and parallel computation*, ACM, N. Y., 1970, 199 pp.

[24] DENNIS, J. B., *Concurrency in software systems*, Computation Structures Group Memo 65-1, Project MAC, MIT, June 1972, 18 pp.; also in *Advanced course in software engineering*, F. L. Bauer (Ed.), Springer-Verlag, Berlin, W. Germany, 1973, pp. 111-127.

[25] DIJKSTRA, E. W., "Cooperating sequential processes," in *Programming languages*, F. Genuys (Ed.), Academic Press, N. Y., 1968, pp. 43-112.

[26] FISCHER, P. C.; MEYER, A. R.; AND ROSENBERG, A. L. "Counter machines and counter languages," *Mathematical Systems Theory* 2, 3 (1954), 265-283.

[27] FURTEK, F. "Modular implementation of Petri nets," MS Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., Sept. 1971.

[28] FURTEK, F. C. "The logic of systems," PhD Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., May 1976; also Tech. Rep. 170, MIT Laboratory Computer Science, June 1976.

[29] GENRICH, H. J. *Einfache nicht-sequentielle Prozesse* (Simple nonsequential processes), Gesellschaft für Mathematik und Datenverarbeitung, Birlinghoven, W. Germany, 1970.

[30] GENRICH, H. J. *The Petri net representation of mathematical knowledge*, GMD-ISF Internal Report 75-06, Institut für Informationssystemforschung, Gesellschaft für Mathematik und Datenverarbeitung, Birlinghoven, W. Germany, 1975.

[31] GOSTELOW, K. P. "Flow of control, resource allocation and the proper termination of programs," PhD Thesis, Computer Science Dept., Univ. Calif., Los Angeles, Dec. 1971, 219 pp.

[32] HACK, M. "Analysis of production schemata by Petri nets," MS Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass.; also MAC TR-94, Project MAC, MIT, Feb. 1972, 119 pp; Errata: Hack, M. *Corrections to 'Analysis of production schemata by Petri nets,'* Computation Structures Note 17, Project MAC, MIT, June 1974, 11 pp.

[33] HACK, M. *A Petri net version of Rabin's undecidability proof for vector addition systems*, Computation Structures Group Memo 94, Project MAC, MIT, Cambridge, Mass., Dec. 1973, 12 pp.

[34] HACK, M. *Decision problems for Petri nets and vector addition systems*, Computation Structures Group Memo 95, Project MAC, MIT, Cambridge, Mass., March 1974; also Technical Memo 59, Project MAC, MIT, March, 1975.

[35] HACK, M. *The recursive equivalence of the reachability problem and the liveness problem for Petri nets and vector addition systems*, Computation Structures Group Memo 107, Project MAC, MIT, Cambridge, Mass., Aug. 1974, 9 pp.; also in *Proc. 15th Annual Symp. Switching and Automata*, IEEE, N. Y., 1974.

[36] HACK, M. *The equality problem for vector addition systems is undecidable*, Computation Structures Group Memo 121, Project MAC, MIT, Cambridge, Mass., April 1975, 32 pp.; also in *Theoretical Computer Science* 2, 1 (June 1976).

[37] HACK, M. *Petri net languages*, Computation Structures Group Memo 124, Project MAC, MIT, Cambridge, Mass., June 1975, 128 pp.; also TR 159, Laboratory Computer Science,

MIT, March 1976.

[38] HACK, M. "Decidability questions for Petri nets," PhD Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., Dec. 1975; also TR-161, Laboratory Computer Science, MIT, June 1976, 194 pp.

[39] HACK, M.; AND PETERSON, J. L. "Petri nets and languages," presented at *MIT Conf. Petri Nets and Related Methods*, MIT, Cambridge, Mass., July 1975.

[40] HANSAL, A.; AND SCHWAB, G. M. *On marked graphs III*, Report LN 25.6.035, IBM Vienna Laboratories, Vienna, Austria, Sept. 1972.

[41] HENHAPL, W. *Firing sequences of marked graphs*, Report LN 25.6.033, IBM Vienna Laboratories, Vienna, Austria, June 1972.

[42] HENHAPL, W. *Firing sequences of marked graphs II*, Report LN 25.6.036, IBM Vienna Laboratories, Vienna, Austria, June 1972.

[43] HOLT, A. W.; SAINT, H.; SHAPIRO, R. M.; AND WARSHALL, S. *Final report of the information system theory project*, Tech. Rep. RADC-TR-68-305, Rome Air Development Center, Griffiss Air Force Base, N. Y.; Sept. 1968.

[44] HOLT, A. W.; AND COMMONER, F. *Events and conditions*, Applied Data Research N.Y., 1970; also in *Record Project MAC Conf. Concurrent Systems and Parallel Computation*, (Chapters I, II, IV, and VI) ACM, N.Y., 1970, pp. 3–52.

[45] HOLT, R. C. "On deadlock in computer systems," PhD Thesis, Dept. Computer Science, Cornell Univ., Ithaca, N.Y., Jan. 1971; also TR 71-91, Dept. Computer Science, Cornell Univ.; and TR CSRG-6, Computer Science Research Group, Univ. Toronto, Toronto, Canada, July 1972.

[46] IZBICKI, H. *On marked graphs*, Report LR 25.6.023, IBM Vienna Laboratories, Vienna, Austria, Sept. 1971.

[47] IZBICKI, H. *On marked graphs II*, Report LN 25.6.029, IBM Vienna Laboratories, Vienna, Austria, Jan. 1972.

[48] JACK, L. "Graphical representation for fault tolerant phenomena," presented at Seminar, Dept. Electrical Engineering, Univ. Texas, Austin, Jan. 1976.

[49] JONES, N. D.; LANDWEBER, L. H.; AND LIEN, Y. E. *Complexity of some problems in Petri nets*, TR-276, Comp. Science Dept., Univ. Wisconsin-Madison, Sept. 1976, 43 pp.; to appear in *Theor. Comp. Sci.*

[50] KARP, R. M.; AND MILLER, R. E. "Properties of a model for parallel computation: determinacy, termination, queueing," *SIAM J. Appl. Math.* 14, 6 (Nov. 1966) 1390–1411.

[51] KARP, R. M.; AND MILLER, R. E. "Parallel program schemata," *J. Computer and Systems Science* 3, 4 (May 1969), 167–195.

[52] KASAMI, T.; TOKURA, N.; AND PETERSON, W. W. "Vector addition systems and synchronization problems of concurrent processes," draft manuscript, 1974.

[53] KELLER, R. M. *Vector replacement systems: a formalism for modelling asynchronous systems*, Tech. Rep. 117, Computer Science Laboratory, Princeton Univ., Princeton, N.J., Dec. 1972; Revised: Jan. 1974, 57 pp.

[54] KELLER, R. M. *Generalized Petri nets as models for system verification*, Tech. Rep. 202, Dept. Electrical Engineering, Princeton Univ., Princeton, N.J., Aug. 1975.

[55] KELLER, R. M. "Look-ahead processors," *Computing Surveys* 7, 4 (Dec. 1975), 177–196.

[56] KELLER, R. M. "Formal verification of parallel programs," *Comm. ACM* 19, 7 (July 1976), 371–384.

[57] KOSARAJU, S. R. *Limitations of Dijkstra's semaphore primitives and Petri nets*, Tech. Rep. 25, Johns Hopkins Univ., Baltimore, Md. May 1973, 5 pp.; also in *Operating Systems Review* 7, 4 (Oct. 1973), 122–126.

[58] LANDWEBER, L. H.; AND ROBERTSON, E. L. *Properties of conflict free and persistent Petri nets*, Tech. Rep. 264, Computer Sciences Dept., Univ. Wisconsin-Madison, Madison, Wisc., Dec. 1975, 30 pp.

[59] LAUER, P. E.; AND CAMPBELL, R. H. *A description of path expressions by Petri nets*, Tech. Rep. 64, Computing Laboratory, Univ. Newcastle Upon Tyne, England, May 1974, 39 pp.

[60] LAUER, P. E. *Path expressions as Petri nets, or Petri nets with fewer tears*, MRM 70, Computing Laboratory, Univ. Newcastle Upon Tyne, England, Jan. 1974, 61 pp.

[61] LAUTENBACH, K.; AND SCHMID, H. A. "Use of Petri nets for proving correctness of concurrent process systems," in *Proc. IFIP Congress 74*, North-Holland Publ. Co., Amsterdam, The Netherlands, 1974, pp. 187–191.

[62] LIEN, Y. E. "Termination properties of generalized Petri nets," *SIAM J. Computing* 5, 2 (June 1976), 251–265.

[63] LIPTON, R. J.; SNYDER, L.; AND ZALCSTEIN, Y. "A comparative study of models of parallel computation," in *Proc. 15th Annual Symp. Switching and Automata*, IEEE, N.Y., 1974, pp. 145–155.

[64] LIPTON, R. "The reachability problem and the boundedness problem for Petri nets are exponential-space hard," presented at *MIT Conf. Petri Nets and Related Methods*, MIT, Cambridge, Mass., July 1975; also TR-62, Dept. Computer Science, Yale Univ., New Haven, Conn., Jan. 1976.

[65] MELDMAN, J. A.; AND HOLT, A. W. "Petri nets and legal systems," *Jurimetrics J.* 12, 2 (Dec. 1971).

[66] MERLIN, P. A. "A study of recoverability of computing systems," PhD. Thesis, Dept. Information and Computer Science, Univ. Calif. Irvine, 1974, 165 pp.

[67] MISUNAS, D. "Petri nets and speed independent design," *Comm. ACM* 16, 8 (Aug. 1973), 474–481.

[68] MILLER, R. E. *A comparison of some theoretical models of parallel computation*, RC 4230, IBM T.J. Watson Research Center, Yorktown Heights, N.Y.; also *IEEE Trans. Comp.* C-22, 8 (Aug. 1973), 710–717.

[69] MURATA, T.; AND CHURCH, R. W. *Analysis of marked graphs and Petri nets by matrix equations*, Research Report MDC 1.1.6, Dept. Information Engineering, Univ. Illinois, Chicago Circle, Nov. 1975, 25 pp.

[70] NASH, B. O. "Reachability problems in vector addition systems," *American Math. Monthly* 80, 3 (March 1973), 292–295.

[71] NOE, J. D. *A Petri net model of the CDC 6400*, Report 71-04-03, Computer Science Dept., Univ. Washington, 1971; also in *Proc. ACM SIGOPS Workshop on System Performance Evaluation*, ACM, N.Y., 1971, pp. 362–378.

[72] NOE, J. D.; AND NUTT, G. J. "Macro E-Nets for representation of parallel systems," *IEEE*

Trans. Comp. C-22, 8 (Aug. 1973), 718–727.

[73] PATIL, S. S. "Coordination of asynchronous events," PhD Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., May 1970; also MAC TR-72, Project MAC, MIT, June 1970, 236 pp.

[74] PATIL, S. S. *Limitations and capabilities of Dijkstra's semaphore primitives for coordination among processes*, Computation Structures Group Memo 57, Project MAC, MIT, Cambridge, Mass., Feb. 1971.

[75] PATIL, S. S. *Circuit implementation of Petri nets*, Computation Structures Group Memo 73, Project MAC, MIT, Cambridge, Mass., Dec. 1972; 14 pp.

[76] PATIL, S. S.; AND DENNIS, J. B. *The description and realization of digital systems*, Computation Structures Group Memo 71, Project MAC, MIT, Cambridge, Mass., Oct. 1972; also in *Proc. Sixth Annual IEEE Computer Society Internatl. Conf. Digest of Papers*, IEEE, N.Y., 1972.

[77] PETERSON, J. L. "Modelling of parallel systems," PhD Thesis, Dept. Electrical Engineering, Stanford Univ., Stanford, Calif., Dec. 1973, 241 pp.

[78] PETERSON, J. L.; AND BREDT, T. H. "A comparison of models of parallel computation," in *Proc. IFIP Congress 74*, North-Holland Publ. Co., Amsterdam, The Netherlands, 1974, pp. 466–470.

[79] PETERSON, J. L. "Computation sequence sets," *J. Computer and System Sciences* 13; 1 (Aug. 1976), 1–24.

[80] PETRI, C. A. "Kommunikation mit Automaten," *Schriften des Rheinisch-Westfälischen Institutes für Instrumentelle Mathematik an der Universitat Bonn*, Heft 2, Bonn, W. Germany 1962; translation: C. F. Greene, Supplement 1 to Tech. Rep. RADC-TR-65-337, Vol. 1, Rome Air Development Center, Griffiss Air Force Base, N.Y., 1965, 89 pp.

[81] PETRI, C. A. "Concepts of net theory," in *Proc. Symp. and Summer School on Mathematical Foundations of Computer Science*, High Tatras; Sept. 3–8, 1973, Math. Inst. Slovak Academy of Science, 1973, pp. 137–146.

[82] PETRI, C. A. *Interpretations of net theory*, Interner Bericht 75-07, Gesellschaft für Mathematik und Datenverarbeitung, Bonn, W. Germany, July 1975, 34 pp.

[83] RACKOFF, C. *The covering and boundedness problems for vector addition systems*, Tech. Rep. 97, Dept. Computer Science, Univ. Toronto, Toronto, Canada, July 1976, 14 pp.

[84] RAMCHANDANI, C. "Analysis of asynchronous concurrent systems by timed Petri nets," PhD Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., 1974; also MAC-TR-120, Project MAC, MIT, Feb. 1974.

[85] RIDDLE, W. E. "The modelling and analysis of supervisory systems," PhD Thesis, Computer Science Dept., Stanford Univ., Stanford, Calif., March 1972, 173 pp.

[86] RIDDLE, W. E. *The equivalence of Petri nets and message transmission models*, SRM 97, Univ. Newcastle Upon Tyne, England, Aug. 1974, 11 pp.

[87] RODRIGUEZ, J. E. "A graph model for parallel computation," PhD Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., Sept. 1957, 120 pp.

[88] SACERDOTE, G.; AND TENNEY, R. L. "The decidability of the reachability problem for vector addition systems," (submitted for publication), Nov. 1976, 8 pp.

[89] SLUTZ, D. R. "The flow graph schemata model of parallel computation," PhD Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., Sept. 1968.

[90] SHAPIRO, R. M.; AND SAINT, H. "A new approach to optimization of sequencing decisions," *Annual Review of Automatic Programming* 6, 5, (1970), 257–288.

[91] THORNTON, J. E. *Design of a computer: the Control Data 6600*, Scott, Foresman and Co., Glenview, Ill., 1970, 181 pp.

[92] TSICHRITZIS, D. *Modular system description*, Tech. Rep. 33, Dept. Computer Science, Univ. Toronto, Toronto, Canada, Oct. 1971, 20 pp.

[93] VAN LEEUWEN, J. "A partial solution to the reachability-problem for vector addition systems," in *Proc. 6th Annual ACM Symp. Theory of Computing*, ACM, N.Y., 1974, pp. 303–309.

DISEÑO DE BASES DE DATOS

THE ANSI/X3SPARC DBMS FRAMEWORK
REPORT OF THE STUDY GROUP ON DATABASE MANAGEMENT SYSTEMS

MAYO, 1985.

Inform. Systems Vol. 3, pp. 173–191
1978 Printed in Great Britain

THE ANSI/X3/SPARC DBMS FRAMEWORK
REPORT OF THE STUDY GROUP ON DATABASE MANAGEMENT SYSTEMS

Edited by DENNIS TSICHRITZIS and ANTHONY KLUG
University of Toronto,
Toronto, Canada

PERGAMON PRESS
OXFORD · NEW YORK · TORONTO
SYDNEY · PARIS · FRANKFURT

U.K. Pergamon Press Ltd., Headington Hill Hall, Oxford OX3 0BW, England
U.S.A. Pergamon Press Inc., Maxwell House, Fairview Park, Elmsford, NY 10523, U.S.A.
CANADA Pergamon of Canada Ltd., 33 Coronet Road, Toronto, Ontario M8Z 2L9, Canada
AUSTRALIA Pergamon Press (Aust.) Pty. Ltd., 19a Boundary Street, Rushcutters Bay, N.S.W. 2011, Australia
FRANCE Pergamon Press SARL, 24 rue de Ecoles, 75240 Paris, Cedex 05, France
WEST GERMANY Pergamon Press GmbH, 6242 Kronberg/Taunus, Pferdstrasse 1, Frankfurt-am-Main, West Germany

Preface
Summary
Introduction
Chapter I—Con
Chapter II—Sys
Chapter III—In
Chapter IV—A
Chapter V—Co

Published in the Journal *Information Systems*, Volume 3, Number 3, and supplied to subscribers as part of their subscription. Also available to non subscribers.

Published previously by American Federation of Information Processing Societies, Inc.

The ideas an
representative

Printed in Great Britain by A. Wheaton & Co. Ltd., Exeter

land
Y 10523.

2L9, Canada
cutters Bay,

France
rankfurt-am-

---

real system or
:, mechanical,

oscribers as part

Inc.

CONTENTS

AFIPS PRESS
210 Summit Avenue
Montvale, NJ 07645, U.S.A.

PREFACE

As a preliminary, it is appropriate to discuss briefly the sequence of events that has led to this report. Among the responsibilities of the Standards Planning and Requirements Committee (SPARC) of the American National Standards Committee on Computers and Information Processing (ANSI/X3) is the generation of recommendations for action by the parent Committee on appropriate areas for the initiation of standard development efforts. For some time SPARC has been aware that database management systems are becoming central elements of information processing systems and that there is less than full agreement in the community on their design. In addition to the existence of a number of implementations of such systems, a list that continues to grow, there are several documents generated from the collective wisdom of some segment of the information processing community which are either proposals for specific systems (CODASYL Data Base Task Group Report, Conference on Data System Languages, ACM New York) or more general statements of requirements (GUIDE-SHARE Data Base Management System Requirements, Joint Guide-Share Data Base Requirements Group, New York). There is a debate in the community on whether existing and proposed implementations meet the indicated requirements, or whether the requirements as drawn are all really necessary. Further, there have been serious questions about the economics of systems meeting all the stated requirements.

In addition to the above considerations there is a continuing argument on the appropriate data model for the user, e.g., the relational, hierarchical and network models. A good answer to the question of which data model to use is "all of the above". A major consequence of the framework described in this report is a mechanism that appears to permit this answer in a meaningful sense. Much of the work has been driven by the desire to accommodate the various requirements, statements and differing view-points.

In the autumn of 1972, responding to the clearly perceived need to rationalize the growing confusion, SPARC took formal action to initiate investigation of the subject of database management systems in the context of potential standardization. Consistent with usual practice when confronted with a complex subject, SPARC established an *ad hoc* Study Group on Data Base Management Systems. This Study Group was convened with a charge to investigate the subject of database management systems with the objective of determining which, *if any*, aspects of such systems are *at present* suitable candidates for the development of American National Standards. The "if any" qualification is important because a negative response is just as meaningful as a positive response in a standards context. Standards at the wrong time could easily restrain technological advance. The "at present" qualification is equally significant, indicating the continuing need for review as the requirements, technologies and economics change over time. The official Scope and Program of work of the Study Group at the time it was formed follows.

SUMMARY

The ANSI/X3/SPARC Study Group on Database Management Systems is chartered to investigate the potential for standardization in the area of database management systems. A necessary first step has been the development of a set of requirements for effective database management systems. These requirements have emerged in the form of a generalized framework for the description of database management systems. No existing or proposed implementation of a database management system completely satisfies these requirements nor comprises all of the concepts involved. A necessary preliminary to any discussion of standards is, therefore, an explanation of this framework. This report provides such an explanation.

In the course of the early discussions of the Study Group, it emerged that what any standardization should treat is *interfaces*. There is potential disaster and little merit in developing standards that specify *how* components are to work. What is proper for standards specification is how the components are meshed; in other words, the specification of interfaces. With this motion in mind, a generalized framework for a database management system has been developed that highlights the interfaces and the kind of information and data passing across them. Figure 1 is a simplified schematic view of this framework. The complete diagram of the system framework is given in Fig. 2.

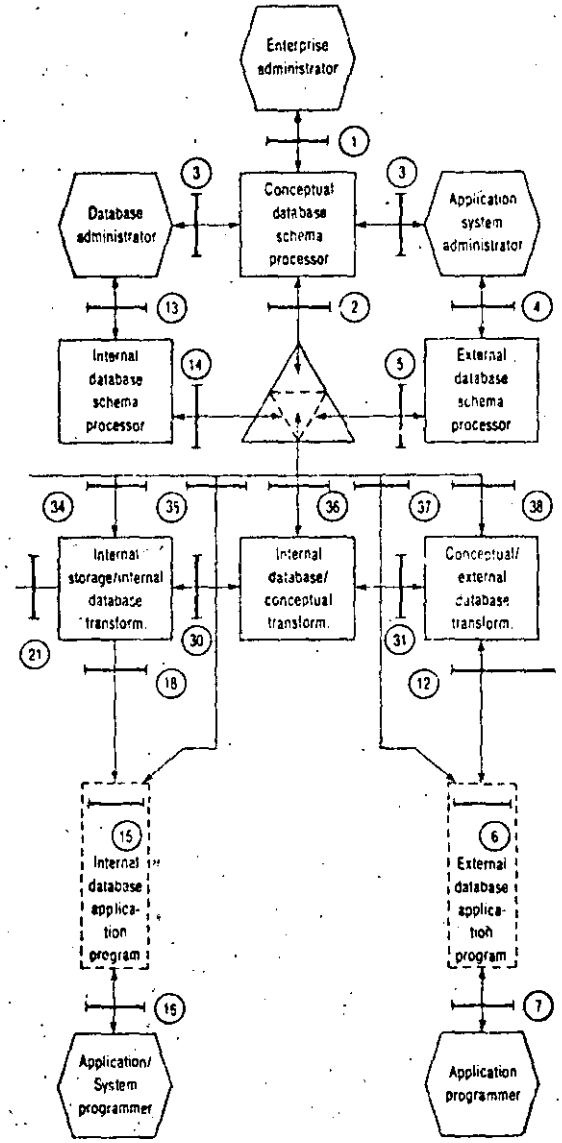


Fig. 1. Partial schematic.

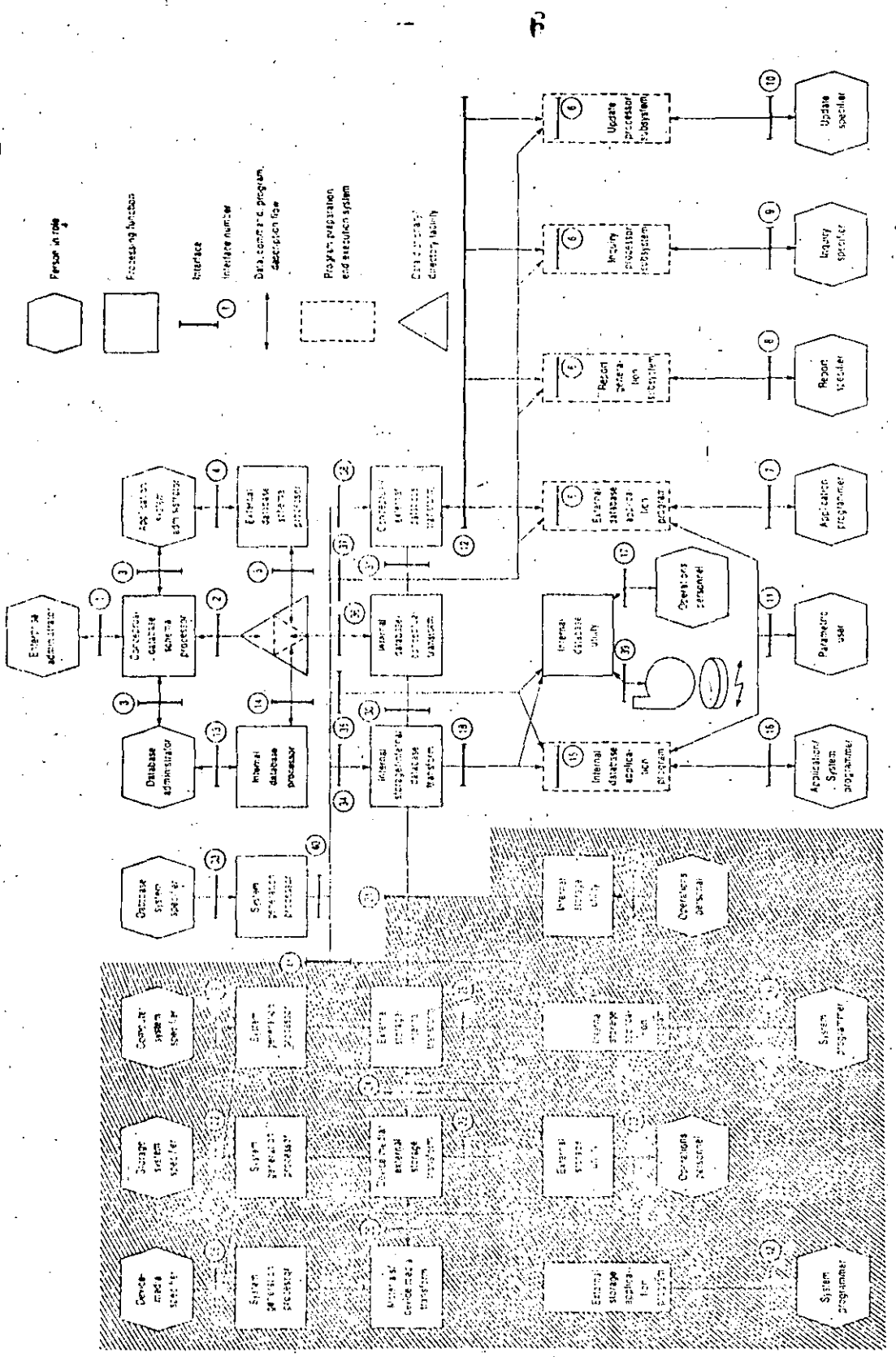


Fig. 2. System schematic.

The hexagonal boxes in Fig. 1 depict people in specific roles. The rectangular boxes represent processing functions. The lines represent flow of data, control information, programs and data descriptions. The dashed boxes represent program preparation and execution subsystems (including compilation and interpretation functions). Finally, the solid bars represent identified interfaces, the ultimate subject matter of the Study Group's deliberations. These interfaces are numbered rather than conventionally named for simplicity of discussion and to avoid confusion. Figure 1 is a simplified diagram. It is consistent with the detailed diagram of Fig. 2 discussed in Chapter III.

It should be noted that, except for the human-system interfaces, the technological nature of an interface is not determined; it could be hardware, software, firmware or some mixture. The diagrams specify function, not implementation. The implementation of the system is not prescribed, only the requirements that must be satisfied. For example, a box may correspond to a compiler, to a process or to several modules of various types.

Some processes and interfaces are omitted on this simplified version of the diagram. For instance, the various ways that system programmers and machine operators can use the system to make *ad hoc* repairs are not present. The bypasses of the system mechanism which are asserted to promote efficiency but may impact data independence, integrity and security are not present. The entire structure of physical mapping of data onto specific storage media is absent. Much of it will be dictated by the hardware implementations and, as such, is of little concern to the current investigation. The principal elements of the Study Group's view of a database management system are displayed. In particular, the multiple schema approach, reflecting the new element introduced by this work, is emphasized.

The lower right hand side of the diagram, the hexagon labelled "application programmer," the dashed rectangle labelled "application program subsystem" and the two interfaces labelled "7" and "12" comprise the entire activity of preparing and executing an application program. This structure may be viewed as replicated into a variety of subsystems, all interfacing with the database management system through interface 12, differing only in the nature of the language used by the programmer to communicate across the human-machine interface 7. This language may be a conventional procedural language such as COBOL, ALGOL or PL/I, special languages like report generators, inquiry languages or update specifiers, or some potentially new type of procedure or problem oriented language. All data passes between the application program subsystem and the database system via interface 12.

The lower left hand side of the diagram, the hexagon labelled "system programmer", the dashed rectangle labelled "system program subsystem" and the two interfaces labelled "16" and "18" comprise the entire means available to the system programmer when it is necessary to bypass the ordinary mode of access to the system. Routine system maintenance and modification will mainly occur through this subsystem. It should be noted that the installation option of permitting application programmers to operate across these interfaces is clearly available, recognizing the tradeoffs in data independence, integrity and security.

Current database management systems envision a two level organization: the data as seen by the system and the data as seen by the programmer. A plethora of confusing terminology has been employed to distinguish between these views. The Study Group has chosen to employ the neutral terms "internal" and "external" to make this distinction. In addition, the Study Group has taken note of the reality of a third level, which is called the "conceptual". It represents the enterprise's description of the information as modelled in the database. This description is that which is informally invoked when there is a dispute between the user and the programmer over exactly what is meant by program specifications. The Study Group contends that in the database world this description *must* be made explicit and, in fact, made known to the database management system. The proposed mechanism for doing this is the *conceptual schema*. It must be emphasized that the conceptual schema is described explicity in machine readable form in some well defined and potentially standardizable language. The other two views of data, internal and external, must necessarily be consistent with the view expressed by the conceptual schema.

This report emphasizes the three levels corresponding to the external, conceptual and internal views of the database. This is not to imply that a particular system should be strictly restricted to three levels. However, it is felt that these three levels exist and are distinct.

Four human roles are given special attention in this report: the enterprise administrator, the database administrator, the application administrator, and the application programmer. Notice that these are *roles* as opposed to *individuals*. The same individual may function in different roles and one role may involve several individuals simultaneously. It is critical, however, that there be only one enterprise administrator and one database administrator (viewed as roles) while there may be several application administrators and several application programmers. There can be several external schemas, each representing a different view of the data, provided each is consistent with the single conceptual schema. By extension there can be several application programmers, not necessarily working on the same program, using the same external schema.

Each administrator is responsible for providing to the system a particular view of the necessary data, the relevant relationships among that data, the rules and controls pertinent to its use, and the mappings between this view and other views. The central view is that of the enterprise administrator who provides the conceptual schema.

The enterprise administrator defines the conceptual schema and, to the extent possible and practicable, validates it. The conceptual schema contains the definitions of entities and their properties and relationships. No entities or properties may be referenced in the database unless they are defined in this schema. Relationships between these entities will be defined. In addition, constraints on the entities and relationships will be defined. By defining the persons with access to the database management system as entities of interest, it is possible to directly model the rules of access and thus provide the necessary access control at the level of the conceptual schema. Access may be further limited at other levels.

The database administrator (our definition of this role is somewhat at variance with conventional conceptions of the task) is responsible for defining the internal schema. This schema relates to the performance strategies employed by the database management system. Whether the data is actually stored flat, hierarchically, in a network, inverted or otherwise, including any meaningful combination, is specified in the internal schema. Media and device related performance strategies are not directly specified in the internal schema. The "internal syntax" of the data values will also be found in the internal schema: such items as the radix for numeric values, coding schemes used, units of measure, etc. Access paths and the connectivity between data representatives will be defined. All of these must be consistent with and mappable to the conceptual schema, which, therefore, must be available for display to the database administrator. The internal schema processor (see Fig. 1) provides a mechanical check on this consistency. Within the limits imposed by this requirement of consistency with the conceptual schema the database administrator is free to alter the internal schema in any way appropriate to optimization of the data base management system operation. Indeed, by use of suitable interpreters it may be possible to reorganize the internal view of the database dynamically while normal operations continue. In view of the massive size of some databases, and in view of the need for some databases to operate continuously, this is an essential requirement.

An application administrator provides the multiple external schemas which define the application views of the data. Each application will employ the description of data in terms of a suitable data structure as provided by an external schema. It is envisioned that each general application area will have its own application administrator who provides the appropriate schemas for that area. These are the only data descriptions (schemas) seen by an application program and provide the only avenue of data name resolution. All external name resolution, whether performed at compile time, program invocation time, or module execution time is done across interfaces 7, 12 and 31 using the appropriate external schema defined across interface 5 and accessed across interfaces 37 and 38.

As with the internal schema, an external schema must be consistent with and mappable to the conceptual schema. However, one external schema may be mapped to a combination of others. In that case, the external schema processor may only validate a new external schema against some existing external schemas known to be consistent with the conceptual schema.

The triangle in Figs. 1 and 2 represents the data dictionary/directory facility. This facility provides, at a minimum, a repository for schema and mapping definitions. Some existing data dictionary products provide the capability for preparation and checking of data descriptions. These products incorporate some of the functions of the three schema processors shown in Fig.

1. In this sense, interfaces 1, 3, 4 and 13 may be interpreted as a set of functions performed at the dictionary interface.

After the appropriate schemas are defined, the system dynamics become quite straightforward and differ little from those of current systems. The application programmer (or report specifier, inquiry specifier, etc.) does his job in the usual way, using the external schema provided, both explicitly and implicitly, as his set of data declarations, providing procedural description across interface 7 and invoking compilation, generation or other relevant processes through the application program subsystem. Upon entry to execution mode, requests for data are passed across interface 12 to the database management system. Although it may seem that the system runs interpretively and explicitly passes through each transformation function to the left of interface 12, the framework does not demand this. Any part of the process may be compiled before execution. By processing the schemas and mappings, a compiler may generate code which transforms external requests at interface 12 directly into requests at some lower level, e.g. at the internal or even storage level. The internal schema could be thought of as representing storage as an infinite, linear address space. It will be necessary to remap this model of storage onto hardware constructs such as tracks, cylinders and the like. A request is then passed across interface 21 and may go through other transformations until actual data is obtained and the process reversed. This brief description has been outlined in terms of obtaining data but, of course, storage of data proceeds in a similar fashion.

This report discusses in detail the framework summarized above. Chapter I introduces the basic organization and the associated concepts. Chapter II discusses several aspects of the dynamics involved in using this framework. Chapter III discusses the interfaces in greater detail. Chapter IV outlines the administration roles. Finally, Chapter V offers some concluding remarks.

INTRODUCTION 10

X3/SPARC STUDY GROUP ON DATABASE SYSTEMS (X3 PROJECT 226)

Scope

Review existing and proposed Database Systems, published reports on Database Systems requirements and other material on Database Systems. Develop proposals (SPARC/90) on those areas which appear suitable, or unsuitable, at present for development of either American National Standards or Guidelines relating to Database Systems.

Program of work

1. Define overall structure of an information system in order to identify those portions which are within the scope of Database Systems.
2. Review the activities and decisions to date by X3 and SPARC on the subject of Database Systems.
3. Establish and maintain liaison with—and solicit input from—appropriate other groups.
   - 3.1 Study the expressed requirements for Database Systems.
   - 3.2 Study representative systems and approaches.
4. Define the structure and component parts of Database Systems.
5. Iterate on above as required to determine one of the following eventual outcomes.
   - 5.1 Identify a component of database systems which is appropriate for standardization or guideline activity. Produce a SPARC/90 report justifying this disposition.
   - 5.2 Identify a component of database systems which is as present inappropriate for standardization or guideline activity. Produce a SPARC/90 report justifying this disposition.
   - 5.3 Identify a component for which there is some other rationale for inaction such as the requirement or problem definition is insufficient to make a determination regarding standardization or guidelines. Recommend further study, if appropriate.

The eventual result of the deliberations of this Study Group will be a series of reports in a specified format (SPARC 1974), identifying potentially standardizable elements of database management systems and recommending whether or not there is a need, technological feasibility and economic justification for the initiation of a standards development project in the area.

It is appropriate to provide a list of the past and present members of the Study Group and their affiliations to indicate the breadth of representation. It is worth noting the extent to which the user community is participating in this effort, a rare event in data processing standardization.

Bachman, C. W.	Honeywell
Cohn, L.	IBM Corporation
Florance, W. E.	Eastman Kodak Company
Harris, S.	Sperry Univac
Kent, W.	IBM Corporation
Kirshenbaum, F.	Equitable Life
Kunecke, H.	Boeing Computer Services
Lavin, M.	Sperry Univac
Mairet, C. E.	Deere and Company
Scott, E. D.	NCR
Sibley, E. H.	University of Maryland
Smith, D. M.	Exxon Corporation
Steel, T. B., Jr.	Equitable Life
Turner, J. A.	Columbia University
Weeks, R.	CINCOM
Yormark, B.	The RAND Corporation

The initial tasks of the Study Group were the difficult ones of understanding and coming to respect the varying points of view and developing a vocabulary that was consistent and mutually comprehensible. It is not clear whether this last task has yet been fully accomplished, although considerable closure has been attained.

Another early task for the Study Group was to determine exactly what should constitute database management from our perspective. For this purpose, information systems were considered to consist of five basic components:

1. Messages
2. Records
3. Procedures
4. Resources
5. Processes

Given these components, the question was clearly which of them are involved in a database management system. The Study Group decided that all real input/output, card in and out, printer output, terminal input and output, and data being transferred between processes would be considered messages and be gathered under a discipline "message management". Furthermore, it was not part of database management. The Study Group decided that all the activities which go into the preparation, compiling, testing, cataloging of a program, such that it would be available to be executed, would be gathered under a discipline called "procedure management", and it was not part of database management. The Study Group decided that all the memory allocation problems, swapping, dispatching and tape and disk drive assignments had to do with the physical resources of the computer and would be gathered under a discipline called "resources management", and it was not part of database management. The Study Group decided all of the aspects of local variables, working storage, instruction counters, had to do with the state of a process and should be gathered under a discipline called "process management" and it was not part of database management.

So the Study Group took records, fields, files, etc., and the descriptions for all of these, and all the indices, mapping techniques, access methods, file organizations

and end user languages, and gathered them under a discipline called "database management". They constituted the database systems which would be studied. The resulting framework reflects this decision in that its components do not require a fixed interpretation as resources, messages, and/or processes.

The Study Group produced an interim report in 1975. The interim report was mainly a working document to be used by the members of the group. As such, it contained much redundancy and some inconsistencies. In addition, the structure and writing style was informal. Although intended as an internal working document, the interim report was widely circulated both informally and as an ACM SIGMOD publication (FDT bulletin Vol. 7, No. 2, 1975). The interim report received much attention and generated some confusion. It was obvious that people were reading the "interim" report and judging it as a "final" report. To assist in eliminating confusion and to bring out the main points of the interim report, the Study Group decided to edit the interim report into a shorter and more cohesive version. It was felt that an outside group could do a faster and more independent job than the members of the Study Group, and members of the Data Base Group at the University of Toronto were asked to do the editing. This report is the result of their editing. It supercedes the interim report and represents the current thinking of the Study Group as it evolved since the publication of the interim report. It is hoped that the reader will find that this report is easier to read and that it retains the main ideas of the interim report.

CHAPTER 1—CONCEPTS

Objectives of database management systems

Data is recognized as an important resource in any organization. The main objective of a database management system is to treat data as a manageable corporate resource. A database management system helps to increase data utilization and to integrate smoothly the data access and processing function with the rest of the organization. A database management system should ease data access by many classes of users. It should enhance data security. It should provide data integrity. The data integration provided by a database management system assures greater data consistency and control of redundancy. Finally, a database management system should provide the ability to evolve by emphasizing data independence. Data independence allows the users to add new applications, to follow changes in the enterprise, to correct design errors and to tune for better performance and response.

The framework proposed in this report addresses the issues arising from the database management system objectives. Many of the issues are related to the concept of data independence. This goal will, therefore, be further elaborated.

Data independence

Data independence insulates a user from the adverse effects of the evolution of the database environment. Some of the factors that tend to interfere include a change to the representation, formatting, organization, data model or location of the stored data, other users expressing requirements upon the same stored data that may be not congruent, other users sharing the same stored data concurrently or other users accidentally or intentionally damaging the database.

Data independence does not include the capability of a database management system to automatically cope with changes in the program's algorithm, changes to the program's view of the stored data, or (accidental or intentional) unavailability of stored data. Programs are subject to impact of changes internal to their logic or internal to their view of stored data.

Data independence is not a property of a database management system that merely provides alternative views of the same stored data. Data independence is the property of a database management system that provides these views and preserves them during the evolution of the data environment.

The necessity for data independence cannot be avoided by attempting to establish and maintain compatibility, that is, to ensure that all changes and uses are "compatible with each other". Data independence is not a discipline; it is a flexibility.

The necessity for data independence cannot be avoided by choosing the right way to organize the stored data such that it never has to change. Change is inevitable. Data independence is not the capability to avoid change; it is the capability to reduce the trauma of change.

A database management system that provides data independence ensures that applications can continue to run, perhaps at reduced performance, if the stored data is reorganized to accord other applications higher performance. Such a database management system does not prevent one from rewriting and retuning the old application to improve its performance at such a time as is economically justified.

It is imperative to conceive of database management systems which will provide the maximum degree of data independence. While the interim report of the Study Group covered a number of subjects besides data independence, the group feels its major contribution is in the field of data independence, and the rest of the report will concentrate on it.

Concepts relating to databases

A *database* contains data about a selected part of the real world. This part of the real world is called an *enterprise*. It generally represents information about a group of people, artifacts, ideas, events and processes organized to support collective goals. These real world concepts may be expressed in terms of any one of a number of formal schemes. The schemes generally involve precise definitions of such notions as entity, property and relationship. For the purposes of this report, however, the informal English usage of these terms is sufficient.

Databases and data processing correspond to the representation and manipulation of symbols about the real world. There are several realms of interest within databases. Three of these realms have special

significance in this report. These realms are: *external*, including a simplified model of the real world as seen by one or more applications; *conceptual*, including the limited model of the real world maintained for all applications of the enterprise; and *internal*, including a model of the data maintained for the representation of this limited model of the real world.

Within the realm of the enterprise there exists a number of *applications* or application systems. An application is a part of the enterprise organized to accomplish a specific subgoal in pursuit of the enterprise goal. For example, a business enterprise may have among its applications payroll, marketing and research and development.

The external realm contains any number of *external views* of the database, each of which is a collection of objects representing the entities, properties and relationships of interest to a specific application. Each external view of the database is associated with an *external schema* describing the objects in that external view of the database. The object which models an entity may, for example, be a logical record in COBOL, an owner or member of a data-structure-set, a tuple in a relation, a line in a form or whatever is most appropriate for a specific use. In addition, different views of (part of) the enterprise may be provided by the various external views of the database. For example, a class-roll application may view several students enrolled in each class, while a student-schedule application may view each student as enrolled in several classes.

An object that is local to an external view of the database need not be represented by internal data or correspond to an object described in the conceptual schema. Such local objects may, for example, represent temporary storage needed in the course of a computation, or they may contain raw input which the application will operate upon before the actual database updates.

In the conceptual realm there is the *conceptual view* of the database. This is a collection of objects representing the entities, properties and relationships of interest in the enterprise. The objects are described in a description language. The description of the objects according to this formalism is called the *conceptual schema*. The objects in the conceptual view of the database directly model entities. Some examples of things which might serve as conceptual objects are records, irreducible relations, nodes in a graph structure, etc. This report does not propose a particular formalism as exceptionally appropriate to express the conceptual view of the database.

The purposes of the conceptual realm are the following. It should provide a description of the information of interest to the enterprise. It should provide a stable platform to which internal and external schemas may be bound. It should permit additional external schemas to be defined or existing ones to be modified or augmented, without impact on the internal level. It should allow modifications to the internal schema to be invisible at the external level. It should provide a mechanism of control over the content and use of the database.

The internal realm contains the *internal view* of the database. This database is a collection of objects which are related to the objects in the conceptual view of the database. It is described by an *internal schema*. The internal realm is oriented towards the most efficient use of the computing facility, consistent with the processing requirements of the enterprise.

The internal data model should be architecturally sufficient for reflecting current storage technologies, and for serving as a valid abstraction between those technologies and the conceptual realm. Different internal data models are of course possible, reflecting similar or different technological, economic, configuration, implementation or packaging considerations.

It is reasonable to assume that entities and conceptual objects representing facts about entities may have a longer life than the technologies upon which hardware and software implementations are based. Therefore, the internal model should be able to change to reflect changing technologies. In this way the system may remain most economical. However, the conceptual schema is not committed to such technologies.

Correspondence of the levels

The correspondences between the objects in the external, conceptual and internal realms are established through mappings and transforms. Mappings bind descriptors in one schema to those in another. The binding may be between an external schema and more encompassing external schemas, an external schema and the conceptual schema, the conceptual and internal schemas or between an external schema and the internal schema. Transforms, which are processing functions (see Figs. 1 and 2), use the information provided by the mappings to translate external-oriented requests to internal-oriented requests either directly or indirectly via the conceptual schema.

The placement of the conceptual schema between an external schema and the internal schema is necessary to provide the level of indirection essential to data independence. Omitting the conceptual schema and binding the names and characteristics of objects described in an external schema (objects known to an application) directly to the names and properties of objects described in the internal schema has a detrimental effect on data independence. Without the assistance of the indirection provided by the conceptual schema it becomes awkward to write applications that can survive inevitable variations in the characteristics of the stored data.

The objects in an external view of the database (e.g. external records, external fields) are materialized on demand and they cease to exist when they are no longer of interest to the application. For clarity of presentation, this report addresses an external view of the database as though a population of external objects does exist.

Materializing intuitively implies the abstraction of an external object from stored objects. In this report, however, *materializing* is also intended to imply the reverse process. This report mainly covers materializing between the internal and external views of the database.

The objects in the conceptual view of the database need not be materialized. However, the understanding of the role played by the conceptual schema is enhanced if

one assumes that objects seen by an application are mapped to objects in the conceptual view of the database and that objects in the conceptual view of the database are mapped to objects in the internal view of the database.

The objects in the internal view of the database exist as abstractions from physical storage as managed by the storage subsystem. Depending upon how the storage subsystem is managed and what kinds of interfaces it exposes, the internal view of the database, and hence the conceptual and external views of the database, may or may not represent a specific population of data. The storage subsystem might permit the internal view of the database to be bound to different populations of stored data, in which case the schemas of the different levels serve as templates and processing controls. If this option is not provided, then the internal view of the database represents a real population of data. This aspect of the storage subsystem affects the significance of a number of database management system functions such as binding, security, integrity and administration roles, but the detailed relationships are beyond the scope of this report.

The conceptual schema may contain descriptors of objects that may not be represented by internal data. This may be true for instance, during the development or augmentation of the conceptual schema before defining and collecting internal data.

Depending upon the flexibility of the database management system, external objects need not correspond one-to-one to internal objects. For example, the data in an external record may be materialized from concatenations of portions of different internal records.

CHAPTER II—SYSTEM DYNAMICS

The preceding chapter has provided a general description of the framework. This section provides more detail by stepping through several levels and describing the interactions among components at each level. It provides a basis for the descriptions of the interfaces which follow in Chapter III. Although the following paragraphs present the preparation of schemas and mappings as occurring in a fixed and disjoint sequence, in practice these activities may occur iteratively, in parallel, or in a different order.

Data dictionary/directory

Throughout this section reference is made to the data dictionary/directory facility represented by the triangle in Figs. 1 and 2. The data dictionary/directory is a meta-database: the repository of information about the database. At a minimum, it contains schema and mapping definitions. It may also include usage statistics of various database objects and object types; access and security declarations; control structures for restart, recovery, accounting and auditing; descriptions of users; and textual statements relative to the preceding items.

Interfaces 2, 5, 14 and 34–38 may be interpreted as a set of storage and retrieval functions across a single interface. Some existing data dictionary products provide the capability for preparation and checking of data descriptions. These products incorporate some of the functions of the three schema processors shown in Figs. 1 and 2. In this sense, interfaces 1, 3, 4 and 13 may be interpreted as a set of functions performed at the dictionary interface. Such data dictionary functions might also be included in a more comprehensive facility for tracking system objects and their interdependencies.

Preparation of the conceptual schema

Prior to undertaking the development and creation of a database, it is necessary to have an understanding of the environment which that database is to serve. Consequently, a very important step is the characterization or synthesis of the information needs within an enterprise. This includes determining what information flows through the enterprise and how it is to be used. This systems synthesis function, in the context of the many applications utilizing the database, is one function of the *enterprise administrator*. The enterprise administrator serves as the focal point for identification of information use in an enterprise. This function has the responsibility for identifying what information is to be managed, and what security, integrity and availability considerations are to be applied to this information and, in addition, describing the relationships among information objects. By performing these functions the enterprise administrator is, in effect, describing some of elements which would be stored in a data dictionary/directory. Hence, the data dictionary/directory can be viewed as a repository for a "first level" description of the conceptual schema.

Once the enterprise administrator understands the information needs of the organization and has documented the uses, flow and accessibility of the information, a conceptual schema is prepared (in a "descriptive" language) to serve as the information model of the enterprise and as a control point for further database development.

The conceptual schema is passed to the conceptual schema processor for encoding into a computerized form. The conceptual schema processor performs syntactical and consistency checks of the conceptual schema. It may store the conceptual schema description in the data dictionary/directory.

Preparation of internal schema

The *database administrator* has the responsibility for specifying an internal description (i.e. the internal schema) of the information represented by the conceptual schema. To perform this task effectively, the database administrator must determine usage requirements for, and source of data for the applications competing for this resource. The internal schema must reflect certain facts about the environment in which the database is to operate. Among these are questions of total system performance, timelines of response and expected system load—all in a varying environment of concurrent access. In addition, if programs are to be written at the internal level, security and integrity rules consistent with the specifications in the conceptual schema must be specified in the internal schema.

Furthermore, the internal schema addresses implementation issues regarding access methods and techniques for representing relationships. Resolution of these issues provides the database administrator with the ingredients for specifying the internal schema. The internal schema processor converts the schema into a form which can be used by the database management system. The internal schema may be stored in the data dictionary/directory by the internal schema processor.

Preparation of external schemas

The various application programs utilizing an organization's database are under the control of *application administrators*. Together with the enterprise administrator, the application administrators determine the objects of interest for each specific class of applications, and they introduce various data structures or data models through which these objects are presented. Each external schema may be used by one or more application programs. In designing the external schema for a specific application, an application administrator consults with the other administrators to determine what data is required and whether it is available. It is the responsibility of the application administrator to interface with the application (or systems) programmers to assist them in preparing programs using a particular external schema. When preparing an external schema, an application administrator may either use a specific programming language or a descriptive language which is programming language independent. To enable application programmers to define the data description portion of their programs, the external schema is transmitted to them, perhaps indirectly through an external schema formatter for convenience to the recipient. The external schema processor may store the external schema in the data dictionary/directory, making it available for application programs.

It is important to iterate that different applications may have different external views of the same database. Each of these views may result in the creation of an external schema relevant to that application.

Preparation of mappings

To establish a correspondence between the conceptual schema and the internal schema, the database administrator specifies a mapping. Similarly, the application administrator establishes correspondences between external schemas and the conceptual schema. In addition, if an external schema is derived from other external schemas, the application administrator will provide the necessary mappings between these external schemas. Mappings between the internal schema and external schemas are also allowed, but only if they can also be defined as the product of mappings passing through the conceptual schema.

The mapping specifies the correspondence between objects at different levels of schema. The types of correspondences include name association, rules for reformatting information including data-type conversion, subsetting, reordering, composition, truncation, encryption, etc., and rules for materializing derived data. The means for identifying and selecting individual occurrences of objects may be specified. Additionally, mappings provide the correspondence between security and integrity rules in the respective levels of schema. Mappings may be prepared separately from schemas, perhaps using a level-specific descriptive mapping language, and processed by mapping processors independently from schemas. Alternatively, mappings could be prepared together with the schemas using a common definition language with consistency checking. The processed mapping can be stored in the data dictionary/directory.

Mappings are a vehicle for achieving data independence by allowing changes at a given level of the framework to be absorbed without affecting other levels. For example, if the database administrator retunes the database by changing the internal schema, the mapping between the conceptual and internal schemas is changed but the conceptual schema remains the same. If an application view (external schema) changes, the conceptual schema remains the same; it is sufficient to change only the mapping to the conceptual schema.

Application program preparation and execution

At these stages a number of actions take place. The application programmer codes the program. The source application program is processed by one or more program development functions (e.g. a compiler). The application program is bound to the internal schema via mappings through the conceptual schema.

The external schema can be bound to the application program at different times. Dynamic external schemas could be allowed by delaying this binding until execution time. The external schema is made known in a suitable form to the application programmer and/or the program development function by the external schema formatter (part of the external schema processor).

The interaction of an application program with the database is effected via a sequence of bindings of the external schema through the conceptual schema to the internal schema (and from there into the storage subsystem). The later these bindings are made, the more interpretive the system tends to be. There is a range of times at which these bindings can take place and also a variation in the amount of processing done with the mappings which are made known. If early binding is used, a module may be produced which, at execution time, directly generates requests to the internal view of the database. Changes to an external schema will necessitate reprocessing of an early bound application program.

In the case of late binding, the mappings are made known at program initiation (JOB) time. The mappings may be used for linking to appropriate system routines or they may be input to system code generators. A similar late binding may also occur during application program execution by a type of OPEN command. There is more flexibility here since there may be a whole series of OPEN (and CLOSE) commands issued. If the internal schema changes between runs, a late bound program will not have to be reprocessed.

With dynamic binding the mappings could change or be reanalyzed for each request to the database. This

would, for example, allow dynamic reorganization of the internal view of the database without any effect on the application program.

There is a range of possibilities in the implementation of mappings and transforms. This range allows different tradeoffs for performance and functional flexibility. At one end of the range, an implementation may require that a conceptual object be congruent to an internal object, and that an external object be congruent to a conceptual object. In this case, an external object is materialized directly (one-to-one) from an internal object. Greater flexibity is possible if an implementation permits conceptual objects to correspond to several internal objects and permits external objects to be extracted from several conceptual objects. Processors can be developed which can analyze mappings statically and can generate a transform which materializes external objects directly from internal objects. Alternatively, an implementation can analyze the mappings dynamically with materialization algorithms locally optimized to reflect the varying characteristics of internal objects. At the most flexible end of the range, an implementation can analyze mappings statically or dynamically, with the system determining which is the optimum access. The concepts and definitions in this report are applicable to the entire range of implementations.

Conclusions

Data independence is related to system dynamics in two significant ways: by the complexity of the mappings that can be accommodated, and by the dynamics of the changes that can be accommodated.

Increasing the complexity allowed in mappings increases the data independence of the system. For example, when mappings allow corresponding internal and external data items to have different formats, one format may be altered without changing the other. If internal records do not need to be in one-to-one correspondence with external records, then attribute migration may occur at one level without occurring at the other level.

The dynamics of change involves such things as permitting partial reorganizations while stored data remains in use, permitting modification of external schemas (while the internal schema remains fixed), permitting modification of external schemas concurrent with modification of internal schema.

If all applications are designed to work on predetermined stored data, and all characteristics of this stored data are always known in advance, then static binding, for example, by a compiler, is satisfactory. However, if any application is designed to work generically (on alternate stored data), or if any of the characteristics of this stored data may vary, then the application must be reprocessed or there must be late or dynamic binding. This capability is very sensitive to the allocation of system functions.

A database management system must support a variety of users including report specifiers, inquiry specifiers, database update specifiers, parametric users (e.g. bank tellers) and operations clerks (i.e. parametric users of systems utilities). Each user communicates with the system through application programs executing at various levels of interface to the system.

The enterprise administrator must revise the conceptual schema to include the new uses or revised views of information within the enterprise. More frequently, however, the information usage of the enterprise as synthesized in the conceptual schema will remain relatively stable while the internal and external environments require change to absorb new hardware and/or software systems, to re-optimize or restructure the database for efficiency purposes, to integrate new programs (or entire application systems), reformat or reload the database. While the conceptual schema is sensitive to business policies, diversification, mergers, new interests and other dynamics of the enterprise, it is likely to remain more stable than either the internal or external schemas; hence, internal and external schemas are prepared and mapped against the relatively stable conceptual schema rather than against each other, in order to insulate one from the other.

Although the proposed framework does not specifically resolve the problems of a changing environment, the conceptual schema and its surrounding framework are designed to accommodate change more easily.

CHAPTER III—INTERFACES

The proposed framework is depicted in the system schematic of Figs. 1 and 2. This section will outline the interfaces and their purposes. Interfaces at the left of the figure, which belong to the storage subsystem, will not be discussed.

INTERFACE 1: Conceptual Schema Description—Source Format

The source format of the conceptual schema description is the interface by which the enterprise administrator makes known to the database management system his declarations of the conceptual schema. The conceptual schema description includes specification of the conceptual objects and their properties and relationships, operations permitted upon these objects, consistency, integrity, security, recovery and administrative matters. Validation of these declarations by the conceptual schema processor includes syntax checking and checking for self-consistency and consistency with other descriptors. In addition, maintenance operations are available through this interface such as inserting or deleting conceptual descriptors.

INTERFACE 2: Conceptual Schema Description—Object Format

The object format of the conceptual schema description is the interface by which the edited conceptual schema is made known to the rest of the database management system via the data dictionary.

INTERFACE 3: Conceptual Schema Description—Display Format

The display format of the conceptual schema is prepared for (selective) distribution to and for reference by the personnel both inside and outside the data

processing organization, subject to the constraints of security (need to know, right to know). It may be used by these individuals to determine the availability of conceptual data and the characteristics of conceptual data. It is the most comprehensive statement of any of the models that include data about the enterprise. The conceptual schema may be displayed to the enterprise administrator for maintenance purposes, and it may be displayed to an application administrator in order to map an external schema.

INTERFACE 4: External Schema Description—Source Format

The source format of an external schema description is the interface by which an application administrator makes known to the database management system his declarations of an external schema. It includes the specifications of the external objects, the associations and structures in which they are related, operations permitted upon these objects, and administrative matters. In addition, maintenance operations such as inserting, modifying or deleting an external descriptor are available across this interface. This interface is also used to define mappings of an external schema to the conceptual schema or to other external schemas. Validation of these declarations includes syntax checking, checking for self-consistency and consistency with other descriptors, and, if a mapping is defined, checking that these objects can be bound to objects declared in the conceptual schema.

INTERFACE 5: External Schema Description—Object Format

The object format of an external schema description is the interface by which an edited external schema and external/conceptual mapping are made known to the rest of the database management system via the data dictionary.

INTERFACE 6: External Schema Description—Host Language Format

The host language format of an external schema description is the interface through which an External Schema Formatter makes an external schema available for use in writing or processing an application program. An external schema formatter may exist for each host language, e.g., COBOL, FORTRAN, PL/I, or end user language.

This interface provides an application programmer with a version of an external schema compatible with the programming language being used. Through this interface the end user is provided with a version of an external schema in a format tailored to his application, vocation or skill level, and syntactically compatible with his programming language. This interface presents to a program development function an external schema in a format syntactically acceptable for compilation, translation or interpretation.

INTERFACE 7: External Data Manipulation Language—Source Format

The source format of an external data manipulation language is the interface by which an application programmer specifies the selection and manipulation statements within the application program on external data objects that are defined in an external schema.

INTERFACES 8, 9, 10, 11: End User Facilities

In the area of end user facilities a great deal of imaginative development of languages can be expected in the next decade. Interfaces 8, 9, 10 and 11 collectively represent some of the language families which can be expected.

The report specification language (interface 8) is the interface by which the end user as a report specifier describes a particular report to be prepared. The report specifier expects a relatively large volume of highly formatted output and is concerned with external object selection, summarization of certain quantities, formatting of output, etc.

The inquiry specification language (interface 9) is the interface by which the end user as an inquiry specifier describes a particular inquiry to be answered. The inquiry specifier is concerned with external object selection and expects a relatively small volume of output. Formatting could be left to the system's discretion.

The update specification language (interface 10) is the interface by which the end user as an update specifier describes simple updates to the database. The updates would be monitored to prevent damage to the database.

A parametric interface (interface 11) is created by an application programmer to interact with an individual, such as an engineer, a reservations clerk or a doctor. These individuals are expected to be skilled in their own field, and familiar with their own practices and terminology. A parametric interface will therefore be designed to interact with the use in accordance with these skills, practices and terminology. In this sense, a parametric interface is the easiest of the end user facilities to use, and it is expected to be the interface most frequently used. Standards applicable to this interface would be the result of application standardization rather than database standardization.

INTERFACE 12: External Data Manipulation Language—System Format

This is the interface by which an external data manipulation language is expressed in a form independent of any host language. This interface may have several forms depending upon the facilities provided by the system. It may, for example, be in the form of execution time subroutine calls to system procedures, or it may be effected as compilers of application programs.

INTERFACE 13: Internal Schema Description—Source Format

The source format of the internal schema description is the interface by which the database administrator makes known to the database management system the declarations of the internal schema and the internal to

conceptual mapping. It includes specifications of the internal objects, indices, pointers and other implementation mechanisms, other parameters affecting (optimizing) the economics of internal data storage, integrity, security, recovery, and administrative matters. Historically, these parameters were essentially fixed by the designers of the database management system; few if any options were available to the database administrator. Validation of the declarations for the internal data description includes syntax checking for self-consistency with other descriptors, and checking for consistency with the conceptual schema. In addition, this interface is used for maintenance operations on the internal schema such as adding or deleting an access mechanism.

INTERFACE 14: Internal Schema Description—Object Format

The object format of the internal data description is the interface by which the edited internal schema and internal to conceptual mapping are made known to the rest of the database management system via the data dictionary.

INTERFACE 15: Internal Schema Description—Display Format

The display format of the internal data description is the interface by which the internal schema is communicated to persons authorized to see it. The internal schema may be displayed to the database administrator so that he can maintain it; it may be displayed to a systems or utility programmer, so that he can manipulate internal data objects. The internal data description can be used by a host language processor during program preparation (compilation, interpretation, etc.) of internal data manipulation statements.

INTERFACE 16: Internal Data Manipulation Language—Source Format

The source format of the internal data manipulation language is the interface by which a programmer specifies access and manipulative statements on internal data objects that are defined in the internal schema.

INTERFACE 17: Internal Data Utilities—Control Language

The control language for the internal data utilities is the interface by which a site operator specifies operations upon objects that are components of the internal view of the database and are described in the internal schema. The functions performed include reorganization of internal data, the transformation of data into interchange format, and copying upon an interchange volume. There can also be facilities for emergency use to reconstruct internal data when normal recovery and restart facilities cannot be used.

INTERFACE 18: Internal Data Manipulation Language—Object Format

The object format of the internal data manipulation software is the interface by which an executing system program accesses and manipulates internal data objects that are defined in the internal schema. This interface is similar to interface 30.

INTERFACES 19–29: These are part of the storage subsystem and will not be described.

INTERFACE 30: Internal Data Manipulation Language—System format

The system format of the internal data manipulation language is the interface by which an application program's requests for data manipulation are represented as operations on internal data. Most likely, this interface would be used at program execution time, in the form, for example, of subroutine calls to the storage to internal database transform.

INTERFACE 31: Conceptual Data Manipulation Language—System Format

This is the interface by which interactions are made with the conceptual view of the database. This interface may take the form of execution time subroutine calls, or it may be "traversed" by system code generators before execution.

INTERFACE 32: This is part of the storage subsystem and will not be described.

INTERFACE 33: Database Management System Specification Language—Source Format

Through this interface the specifier (vendor) of the database management system makes known to the system the names and definitions of data object types available in each realm (external, conceptual, internal). For example, it would define tools of data storage organization, such as chains, record arrays, pointer arrays, secondary indices, etc.

INTERFACES 34, 35, 36, 37, 38: Data Dictionary/Directory Interfaces

These interfaces are between the data dictionary/directory and the various transform modules and program preparation and execution subsystems. They transmit the schemas and mappings to these functions and thereby allow application program DML to be compiled, interpreted or processed in some other fashion.

INTERFACE 39: Database Transportability Interface

Through this interface, a database may be copied onto an interchange volume in some interchange format.

INTERFACE 40: Database Management System Specification Language—Object Format

This is the internal form of interface 33.

INTERFACES 41–42: These are part of the storage subsystem and will not be described.

CHAPTER IV—ADMINISTRATION ROLES

There are three main administration roles proposed as part of the framework: the enterprise administrator, the

application administrator and the database administrator. All three should be regarded as roles and not necessarily as distinct persons.

Enterprise administrator

The enterprise administrator, in consultation with line organizations, as well as with the application analysts and designers, identifies the overall requirements for existing, proposed, or potential applications. To some extent this function is similar to that of an application administrator. If an application administrator has been appointed to represent management and operations for a specific application, then the enterprise administrator would negotiate with him. The enterprise administrator determines: (i) the totality of the information required by the enterprise; (ii) the best way of collecting the data and the best canonical way of viewing the data; and (iii) the integrity and security rules associated with the data. The enterprise administrator determines from the application administrator how the external data will appear. It is not necessary that all applications be required to view the external data in identical representations; however, it is advantageous to negotiate away trivial and arbitrary differences.

Having gathered the collection of required views the enterprise administrator extrapolates to a more generalized conceptual schema. This conceptual schema incorporates all the conceptual objects. The enterprise administrator anticipates additional application needs, and provides the required stability for application construction.

The process of obtaining the conceptual schema has been described in a "define the application's needs first" manner. In fact, the definition of the conceptual schema may proceed in a "define the components of the conceptual schema first" manner. Neither process is inherently more error prone nor more subject to modification than the other.

The enterprise administrator protects the database against abuse and misuse. Two separate, distinct functions can be identified: integrity and security. The mechanisms for performing these functions may have much in common. Integrity is protecting the database from authorized users. Security is protecting the database from unauthorized users. Security is necessary for privacy. Privacy itself is not the subject of this report. It deals with the concern of people for the privacy of information about themselves. It is a social and political, rather than a technical issue.

Integrity requires that the conceptual schema be current, consistent, and correct. Preservation of integrity includes editing and validating overt changes to the internal data and maintaining consistency of redundant or derived internal data. It also implies resolving conflicts to ensure that no change inadvertently counters a previous change or that a partially completed modification is not inadvertently displayed.

The enterprise administrator, in consultation with line organizations, as well as with the application analysts and designers, documents the editing, validation and consistency rules that govern the database. Correctness in this context means that the external objects that can be materialized from the internal objects satisfy the rules specified by the enterprise administrator.

Security implies that external objects are made available for access or modification only to those individuals, locations, applications and programs that are properly authorized. The enterprise administrator, in consultation with line organizations, determines what the authorization requirements should be in order to see the conceptual schema. The authorization may be on behalf of an individual online, an individual or organization submitting the application, etc.

It is necessary to differentiate between control and ownership. The internal data may be owned by some department or application. The application-oriented requirements, rules for maintenance and rules for its use may be proposed by its owner. The internal data may be entrusted to the enterprise administrator for control over its contents and usage. The enterprise administrator may even exercise control over internal data he is not authorized to see.

The enterprise administrator interacts with the system, using the system as a tool for enterprise administration. The enterprise administrator may use the conceptual schema processor in interactive manner. The enterprise administrator can enter specifications for definition of the conceptual schema and for protection and control of the database through the conceptual model.

Application administrator

An application administrator, in consultation with line organizations, as well as with the application analysts and designers, identifies the overall requirements for the external data required for the proposed application(s).

Having a collection of the required external views, an application administrator defines or augments the definition of an external schema. Each external object can be defined in terms of an existing conceptual schema definition. The external object can be defined by explicitly defining all the components, by extracting from an already defined external object(s) or by extracting from a conceptual schema definition. The preceding process exemplifies defining an external schema. Similar processes may augment, modify or delete an external schema, or any of the definitions it contains.

An application administrator exercises control over his applications by defining specific constraints for external objects and by defining specific structures for these external objects. He defines external object descriptors that may be copied into a source program. He also may prescribe the bindings of the external object to specific conceptual schema objects. He may prepare the binding control statements that may be included with a program.

An application administrator negotiates with the person responsible for security for authorization and authentication of individuals and applications within his scope.

The application administrator interacts with the system, using the system as a tool for application administration. An application administrator may use the external schema processor in an interactive manner. An

application administrator can enter specifications for definition of the external schema, for control of the external objects and for binding to the conceptual schema objects.

Database administrator

The database administrator, in consultation with the enterprise administrator, identifies the overall requirements for, and availability of, internal data required for an existing, proposed, or potential application. The enterprise administrator has determined the conceptual objects, relationships and structures defined in the conceptual schema that must be derivable from the internal data.

Having been instructed in the collection of definitions in the conceptual schema, and the global economics of use of external and internal data in the enterprise, the database administrator defines a complement of internal fields, internal records, etc. He determines the tradeoff between copying and converting existing internal data. He decides on redundant copies of internal data to increase responsiveness and maintain consistency between the copies. He also specifies the procedures that exercise control on his behalf. The preceding process exemplifies creating a definition to the internal schema. Similar processes modify or delete definitions.

Having defined the new internal objects, next to be defined is the mapping from conceptual objects defined in the conceptual schema. The mapping of internal objects to conceptual objects should take into account, for example, the distribution of internal fields into internal records, and of redundancies and dependencies among the internal fields defined in the internal schema.

If a needed internal object does not exist, the database administrator establishes the definitions for that object. The database administrator constructs a mapping from a conceptual object to internal object(s). He defines the selection of internal object values that correspond to an added conceptual object. He defines the aggregating algorithms for viewing conceptual objects in terms of internal objects.

The database administrator exercises control over the economics of the database system. The database administrator provides a formal, precise definition of the internal data, how it is represented, how it is organized, how it is stored, how it is accessed. He establishes the quality of services, levels of service, and cost-service-performance tradeoffs. He manipulates, reorganizes, and refines the internal view of the database, to achieve optimum performance under current priorities and demands. He directs the database management system to monitor and examine use of the internal data, to ensure that the policies, procedures, and controls are effective and achieve the enterprise's goals. Obviously, the database administrator is constrained against discarding internal data still required for operational or performance reasons.

The database administrator may examine the conceptual schema, the internal schema, and the conceptual to internal mapping. The database administrator can enter specifications for definition of the internal schema, for mapping from the conceptual schema, and for control of the internal data. The internal database schema processor checks the syntax and semantics of the input for self-consistency and for incremental consistency with the already-specified portions of the internal schema and mapping. It analyzes and possibly simulates the effect of the additional specifications and reports to the database administrator.

In conclusion, the rationale for the different administration roles are summarized. First, allocating the external data description and control functions to the *application administrators* allows some changes to be instituted and enforced without the continuous and usually prohibitive expense of modifying source programs. Establishing control over policies in the hands of the *enterprise administrator* helps ensure that they reflect the requirements of the enterprise as a whole, and not parochial, shortsighted, or uninformed interests. For instance, an application group probably will not understand the information economics of the entire enterprise, especially applications it is not familiar with. Their needs will be best served by their application administrator. Finally, isolating the optimization, measurement, and management of internal data in the hands of the database administrator ensures the appropriate technical skills for the task.

CHAPTER V—CONCLUDING REMARKS

This report establishes a basic framework for database management systems. While much effort and progress has been made, there is still more to be accomplished. Significant areas which remain to be studied within the context of this framework by the Study Group include:

- distributed database (potential impact),
- security,
- recovery,
- integrity,
- elaboration of data dictionaries (especially within the context of distributed systems),
- concepts and constructs at each level,
- manipulative functions at the conceptual schema,
- mappings,
- external and internal schema interfaces

The study is neither complete nor all encompassing. Many items remain unclear and much work remains to be done. The Study Group invites both the user community and the research and development community to respond both to the ideas presented in this report and the challenging new directions that are opened by this report.

DISEÑO DE BASES DE DATOS

DATABASE ABSTRACTIONS: AGGREGATION

MAYO, 1985.

Data: Abstraction, Definition, and Structure

B. Wegbreit Editor

Database Abstractions: Aggregation

John Miles Smith and Diane C.P.Smith
University of Utah

Aggregation is introduced as an abstraction which is important in conceptualizing the real world. Aggregation transforms a relationship between objects into a higher-level object. A new data type, called aggregate, is developed which, under certain criteria of "well-definedness," specifies aggregation abstractions. Relational databases defined as collections of aggregates are structured as a hierarchy of n-ary relations. To maintain well-definedness, update operations on such databases must preserve two invariants. Well-defined relations are distinct from relations in third normal form. It is shown that these notions are complementary and both are important in database design. A top-down methodology for database design is described which separates decisions concerning aggregate structure from decisions concerning key identification. It is suggested that aggregate types, and other types which support real-world abstractions without introducing implementation detail, should be incorporated into programming languages.

Key Words and Phrases: data abstraction, relational database, data type, aggregation, database design, data structure, knowledge representation, data definition language

CR Categories: 3.65, 3.69, 3.79, 4.29, 4.33, 4.34

This work was partly supported by the National Science Foundation under grant MCS75-09903. Authors' address: Computer Science Department, University of Utah, Salt Lake City, Utah 84112.

A version of this paper was presented at the SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure, Salt Lake City, Utah, March 22–24, 1976.

A database is a dynamic model of a system—perhaps a physical system such as a car-body or an organizational system such as a city or an industry. The model must provide for:

(i) a *concise*, *understandable* representation of the state of the system at any time;
(ii) *efficient* and *high-level* access to information, and
(iii) *simple* and *consistent* update as the system changes state.

In [1–3] Codd developed relational models and showed their advantages of relatively concise representation, high-level information access and simple updating. The questions of the understandability, efficiency, and consistency of relational models have not yet been fully resolved. As developed by Codd, a relational model consists of a flat (nonhierarchic) collection of *n*-ary relations. As Schmid [8] points out, there is often difficulty in understanding how a collection of relations is actually modeling the underlying system. Furthermore Schmid discovers that it is difficult to formulate general rules to guarantee the consistency of the collection of relations as they are updated.

In this paper our principal objective is to increase the understandability of relational models by the imposition of additional semantic structure. This structure is the minimum necessary to support one kind of *abstraction* that people seem to use in conceptualizing the real world. By abstraction we mean the suppression of all details about some object (or activity) except for those relevant to the understanding of some phenomena of interest. The particular kind of abstraction under consideration transforms a relationship between several named objects into a (higher-level) named object. We call this kind of abstraction "aggregation." We introduce the notion of a "well-defined" relation as a relation which represents an aggregation abstraction.

This notion of "well-definedness" provides a framework within which to consider the consistency of relational models. We show that two properties of relational models must remain invariant during update operations. These invariant properties must be satisfied by *all* relational models (which support aggregation) irrespective of the qualities of the system being modeled. In general, a model of a particular real-world system will require that additional invariants be maintained. However, these latter invariants are dependent on the peculiar nature of the real-world system.

Well-defined relations are distinct from relations in "third normal form" (3NF) [4]. We show that some well-defined relations are not 3NF relations and vice versa. Essentially, well-definedness is a stronger semantic notion than 3NF; however, 3NF places stronger conditions on the extraction of redundant information. We suggest that these notions are complementary and that they are both important in database design.

We introduce a new data type (the "aggregate"

type) for specifying the structure of relational models. The aggregate type is explicitly designed to support aggregation abstractions and is in keeping with the principles of data abstraction laid down by Hoare [6]. Models designed by using aggregates consist of a hierarchy of n-ary relations—not a flat collection of n-ary relations. We develop a top-down methodology for the design of such models. In this methodology we minimize the number of details with which a designer must contend at one time. A principal feature of the methodology is the separation of decisions concerning model structure from decisions concerning 3NF and key identification. This separation simplifies both activities.

In Section 2 we introduce the notion of aggregation and develop a type definition facility suitable for supporting these abstractions. We use a syntax in the style of PASCAL [12] for type definitions. Section 3 develops the idea of well-defined relations and discusses their properties which must remain invariant during update operations. Well-defined relations are compared to 3NF relations and several conclusions are drawn about database design. In Section 4 a methodology for database design is developed and applied in a nontrivial example. Section 5 is a conclusion.

In addition to Codd and Hoare, this work has been influenced by the writings of Senko [9] and Liskov [7].

2. The Notion of Aggregation

It is well known that the real world can be usefully modeled in terms of two types of primitives: entities and relations. Entities correspond to real-world objects which can be regarded as indivisible for modeling purpose. Relations are a formalization of relevant real-world relationships in which the entities participate. This viewpoint has strongly influenced the development of both the database [5] and AI [13] fields.

In this paper we propose a stronger method for real-world modeling based on a single type of primitive: aggregation. An aggregation is an abstraction which allows a relationship between named objects to be thought of as a (higher-level) named object. For example, a certain relationship between a person, a hotel, a room, and a date may be abstracted as the aggregate object "reservation." The name "reservation" may be used without bringing to mind all the details of the underlying relationship.

Aggregation is commonly used in natural language to simplify the expression of complex relationships. Consider, for example, the following relationship:

> Pupil P received a grade G in a *class* of the *course* numbered $C\#$ with CH credit hours and description D taught by instructor I during semester S in room R.

This could be thought of as an 8-ary relationship among the objects named by P, G, $C\#$, CH, D, I, S, and R. However, it is not at all clear what this relationship means—or more precisely, how this relationship can be abstracted as a named object.

Closer scrutiny reveals that we have actually used aggregation twice in the statement of the relationship. The noun "course" is an aggregation of a relationship among $C\#$, CH, and D. The noun "class" is an aggregation of a relationship among a course (say CO), I, , and R. The stated relationship can thus be simplified to a 3-ary relationship among a class (say CL), P, and G. This relationship can itself be abstracted as the aggregate object "enrollment."

Aggregations can be defined using Hoare's structures. For example, we can define "enrollment" as a hierarchy of three record types:

```
type enrollment = record
                    P: pupil;
                    CL: class;
                    G: grade
                  end
type class = record
               CO: course;
               S: semester;
               I: instructor;
               R: room
             end
type course = record
                C#: number;
                CH: credit-hours;
                D: description
              end
```

Notice that in each definition the component objects are *attributes* of the object defined. For example, in the definition of "enrollment" we have: P is the pupil who is enrolled, CL is the class in which he is enrolled, an G is the grade he received as a result of enrollment. Since "enrollment" references "class," we say "enrollment" is an aggregate object at a higher level of abstraction than "class." Similarly, "class" is at a higher level of abstraction than "course." As we move down the abstraction hierarchy, we encounter more and more details which ultimately pertain to "enrollment."

Unfortunately, there are several problems with this structuring which are particularly acute in a database environment. These problems are concerned with the naming and accessing of objects and also with the representation of objects at lower implementation levels.

Let's assume a university context in which there are many instances of courses, classes, and enrollments. If we create an instance E of enrollment (as defined above), we obtain a hierarchic object which contains a class and a course. Now a particular database user may only be interested in courses, which are independent objects in their own right. However, under the present scheme, the user can only access courses via enrollments and classes—for example, by writing "$E.CL.CO$". To access courses directly, he would need to create separate instances of "course." Unfortunately this would cause course representations to be duplicated. Such duplication occurs anyway whenever the same course participates in several classes and/or enrollments. Not only does this waste storage space at the

Fig. 1. A snapshot of collections courses, classes and enrollments.

enrollments:

pupil	class-key	grade
Brown	306 F76	B+
Smith	419 S76	C
Cohen	306 S76	A-
Cohen	419 S76	B
Smith	306 F76	B

classes:

course-key	semester	instructor	room
306	F76	Tom	BH210
306	S76	Mary	BH210
419	S76	Mary	MH105
241	F76	Jack	SB325

courses:

number	credit-hour	description
241	3	241-descrip
306	4	306-descrip
419	3	419-descrip
511	4	511-descrip

representation level, but it also introduces problems of possible inconsistency.

These problems can be partially alleviated if pointers are allowed. However, pointers introduce another set of problems. Pointers are objects which have no real-world analog and serve to dramatically increase the complexity of database interactions. Ideally we would like a naming (accessing) structure which reflects natural language. We would like each abstract object to be uniformly accessible independent of its level of abstraction. We would like a one-to-one correspondence between abstract objects and their representation in the database.

In natural language, if we want to reference an object we supply a sufficient number of its attributes to identify it uniquely. For example, a course number is normally sufficient to identify a course, a course together with a semester is normally sufficient to identify a class, and a class together with a pupil can identify an enrollment. In general, several different combinations of attributes can be used to uniquely identify an abstract object.

This observation suggests that whenever we define an abstract object we should specify one subset of its attributes which can be used as a unique identifier. For example, we may define "course" as:

```
type course = aggregate [C#]
                C#: number;
                CH: credit-hours;
                D: description
              end
```

We use "aggregate," rather than the weaker term "record," to indicate that the definition is intended to support an aggregation abstraction. The square brackets are used to delimit the selectors for a set of identifying attributes. We call this set of identifying attributes the *key* of the aggregate.

To define "class" we reference "course" values. However, we want these references to range over keys of *actual* courses. The range of *legal* keys (would be specified by the type definition for "number") may be much larger than the range of *actual* keys for courses existing at some time. We must therefore specify the set[1] which is to contain the actual courses:

```
var courses: collection of course.
```

We assume that "courses" is maintained by insert and deletion operations, so that it contains all (relevant) courses at all times.

We can now define "class" as follows:

```
type class = aggregate [C#, S]
               C#: key courses;
               S: semester;
               I: instructor;
               R: room
             end
```

This definition states that the C# component of "class" can take on as a value any key of a course currently included in the collection "courses."

We can now declare a set which is to contain classes of interest at all times:

```
var classes: collection of class.
```

Objects of type "enrollment" can now be defined:

```
type enrollment = aggregate [P, (C#, S)]
                    P: pupil;
                    (C#, S): key classes;
                    G: grade
                  end
```

Notice that the component which references class keys has the "composite" selector (C#, S). Since a pupil and a class are both required to identify an enrollment, the enrollment key is [P, (C#, S)]. Finally, we can declare the set which is to contain all enrollments of interest:

```
var enrollments: collection of enrollment.
```

Figure 1 illustrates how the collections courses, classes, and enrollments may appear at some time. Notice that every course key in "classes" occurs in "courses," though not every course key in "courses" occurs in "classes." We end this section by considering the semantic conditions necessary for an aggregate type to be "well-defined."

Suppose we define an aggregate type T with component types $T1, \ldots, Tn$ and key I. To be meaningful and each Ti must be natural-language nouns, and the definition should express an aggregation abstraction about the real world. This requires the definition to satisfy two conditions:

(i) Each real-world instance of T must determine a unique instance of each Ti.

(ii) Two distinct real-world instances of T must not determine the same instances of Ti for all Ti whose selectors are in I.

[1] We use the term "collection," rather than "set," in type definitions to avoid confusion with the very specific PASCAL notion of set.

Fig. 2. Syntax summary of aggregate definitions.

```
type A = aggregate [selectorlist]
           s1: {key} T1;
           ...
           sn: {key} Tn
         end
```

where (i) *Ti* is either a collection identifier (in which case "**key**" must appear) or a type identifier (in which case "**key**" must not appear) (ii) selectorlist is a sequence of *si*'s separated by commas.

The first condition guarantees that each *Ti* is an attribute of *T*. The second condition ensures that two real-world instances of *T* can be distinguished by the values of their key attributes. We say that an aggregate is *well-defined* if its definition satisfies both these semantic conditions.

The aggregates course, class, and enrollment are all well defined within the usual university context. As an example of a definition that does not meet the first condition, consider:

```
type class = aggregate [C#, S]
               C#: key courses;
               P: pupil;
               S: semester
             end
```

Notice that in the real world a specific class does not (normally) determine a unique pupil in attendance.

As an example of a definition which fails the second condition, consider:

```
type course = aggregate [CH, E]
                CH: credit-hours;
                E: max-enrollment
              end
```

Notice that in the real world several distinct courses may have the same credit hours and maximum enrollment. If we are really interested in courses per se then we should be particularly concerned about attributes which distinguish courses as individuals. The absence of such attributes indicates we are interested instead in some (obscure) relationship between the credit hours of courses and their maximum enrollment. This relationship should be abstracted as an aggregate other than "course."

In Figure 2, the syntax for an aggregate type definition is summarized.

3. Well-Defined Relations

In the previous section we introduced aggregate and selection types as a method for structuring world objects. A collection of aggregate objects is an *n*-ary relation in the sense of Codd [1]. However, we shall now show that some *n*-ary relations are not collections of *well-defined* aggregates. We shall say that a relation is "well-defined" if it is a collection of well-defined aggregates. We shall use the terms "aggregate" and "tuple" interchangeably. In this section we first consider invariant properties of well-defined relations and suggest restrictions on update operations to preserve these properties. Second, we compare the notion of a well-defined relation with that of a relation in third normal form [4].

Codd [3] has proposed a set of high-level operators for transforming relations into relations. These are used to retrieve information from a relational database. Operators are also necessary to insert, delete, and modify tuples in a relation. Such update operators are:

insert(*r*, *t*):	enter tuple *t* into relation *r*.
delete(*r*, *k*):	remove tuple with key *k* from relation *r*.
modify(*r*, *k*, *t*):	modify tuple with key *k* in relation *r* in accordance with tuple *t*.

In practice, it is often necessary to allow tuples to have some components with "blank" values.

At any time, a set of well-defined relations will satisfy the following properties: (i) all tuples in a relation will have unique keys, and (ii) if the type of component *s* of tuple *t* in relation *r* is *key r'* then *t.s* is the key of some tuple in relation *r'*. We call these two conditions *relational invariants*.

Essentially, the relational invariants constrain a relation to represent a collection of aggregate objects. If we want to support abstraction, these invariants should be satisfied by *every* relation in a world-model irrespective of the particular kind of aggregate collection.

To maintain the relational invariants, certain restrictions must be placed on update operations. A conservative set of restrictions is shown in Figure 3. Under these restrictions update operations will preserve the relational invariants; however, many correct (relative to the world being modeled) update operations will not be allowed. For example, a tuple cannot be deleted from a relation until all tuples referencing the tuple are themselves deleted.

To maintain the relational invariants *and* allow all correct update operations to be performed, a scheme which automatically propagates update operations through the set of relations is necessary. For example, deleting a tuple causes all tuples which reference it to be themselves deleted. Such a scheme is investigated in [10]. The disadvantage of this latter approach is that pollution caused by incorrect input data is rapidly and invisibly propagated throughout the database.

One criterion which should be considered for the implementation of relations is that the update restrictions can be checked efficiently at the time update operations are performed. This requires that the following tuples be easily accessible from a tuple *t*: (i) the tuples which *t* references and (ii) the tuples which reference *t*. This requirement can be met by implementing relations using "owner-coupled" sets [11].[2]

We now contrast our notion of a "well-defined"

[2] In the terminology of [11], a collection *T* of aggregates of type *A* (see Figure 2) has the following owner-coupled set implementation: (i) declare a record type for *T* and also for each collection *Ti*, (ii) declare a set type (with owner *Ti* and member *T*) for each collection *Ti*.

relation with Codd's notion of a "third normal form" (3NF) relation. Both these notions offer criteria for what is or is not an acceptable relation definition.

We first define 3NF. From the meaning of a key, it follows that there is a functional correspondence between the key components of a tuple and the remaining components. In other words, for each key (declared or undeclared) of a relation, every component in a tuple is "functionally dependent" on the key components. A relation is in 3NF if these are the *only* functional dependencies between its components.

Consider the relation "classes" defined in Section 2. This relation may or may not be in 3NF, depending on the university context. For example, suppose there is a regulation that an instructor must always teach all his courses in the same room. This regulation implies that the component *R* is functionally dependent on *I* in the relation "classes." Alternatively, suppose there is a regulation that an instructor always teaches one and the same course. In this case the component *C#* is functionally dependent on *I* in the relation "classes." Since *I* (alone) is not a key of "classes," the existence of either of these two functional dependencies prevents "classes" from being a 3NF relation.

When a relation is not in 3NF, the presence of "embedded" aggregates is indicated. For example, if *R* is functionally dependent on *I* in "class," this indicates that the components "instructor" and "room" together form an independent aggregate (with key *I*) which is embedded in "class." An embedded aggregate which corresponds to a useful abstraction should be removed from its "host" and declared as an independent aggregate. A reference to this aggregate must be included in the host. An embedded aggregate which is retained in its host may cause duplication of information. This in turn requires special care to ensure that update operations maintain consistency. If all embedded aggregates are removed, relations will be transformed to 3NF.

When a hierarchy of well-defined relations is being transformed to 3NF, it is sufficient to move functionally dependent components down the hierarchy from one relation to another. New relations will only be necessary when components are moved down to the leaves of the hierarchy. This movement of components does not alter the *nature* of existing abstractions; however, it may force changes to the declared key of some relations. For example, given the functional dependency of *R* on *I*, we can remove the component "room" from "class" and include it as a component of "instructor." This movement does not change the abstraction "class" or "instructor," nor does the key of "class" need to be changed. Given the functional dependency of *C#* on *I*, we can remove the component "room" from "class" and include it as a component of "instructor." This movement does not change the abstraction "class"; however, its key must be changed to *[I, C#]*.

We have seen that some well-defined relations are not 3NF relations. We now show that the converse is also true. Consider the following relationship:

Pupil *P* has *enrolled* in a class in room *R* and has *climbed* mountain *M*.

We can declare the following relation to represent this relationship:

```
var X: collection of aggregate [P, R; M]
                    P: pupil;
                    R: room;
                    M: mountain
                  end
```

This relation is in 3NF—it has no functional dependencies.[3]

The relation "*X*" is not well defined. There is no English noun which abstracts the stated relationship to a higher-level aggregate object. In other words, within the context of the relationship there is no object type (which we can name in natural language) whose instances determine unique values for pupil, room, and mountain. The stated relationship would have to be represented by using two well-defined aggregates, "enrollment" as before, and "climb" as below:

```
type climb = aggregate [P, M]
               P: pupil;
               M: mountain
             end
```

By restricting the names for collections and aggregate types to natural language nouns, we are able to distinguish aggregations which represent abstractions from aggregations which are merely a spurious grouping of distinct concepts. In his formalism for relational models, Codd does not include natural language names for objects as an integral part of his formalism. As a result, 3NF is insensitive to many semantic aspects of aggregation. Even if 3NF is strengthened to exclude more spurious aggregates, it seems unlikely that any simple syntactic constructs will be able to fully capture the richness of noun formation in natural language. Moreover, given an aggregate type definition it is relatively easy to determine whether it is well-defined. On the other hand, it is often quite difficult to check if a relation is in 3NF.

There are several conclusions which can be drawn from the previous discussion:

(i) The criteria for a well-defined relation are

Fig. 3. Update restrictions to preserve the relational invariants.

Update Operation	Restriction
insert(r, t)	i) the key of t contains no blanks and is unique within r. ii) all tuples referenced (via their keys) by t exist.
delete(r, k)	i) the tuple with key k is not referenced by a tuple in another relation.
modify(r, k, t)	i) the components forming the key k are not modified. ii) all tuples referenced by t exist.

[3] We exclude the trivial functional dependencies in which components determine themselves—these are always present.

portant in the initial formulation of aggregate abstractions in a world model.

(ii) The criteria for 3NF are important for detecting embedded aggregates in this initial formulation.

(iii) The transformation of a hierarchy of well-defined relations to 3NF can be accomplished while preserving the same aggregate names.

(iv) This transformation should be made *before* keys are selected to avoid subsequent key modification. These conclusions are utilized in the next section.

4. Database Design – An Example

In this section we apply our ideas about aggregation abstractions to the problem of database design. At the outset, the designer must interact with the user to determine the objects and relationships which must be represented in the database. This determination is usually made in a bottom-up development starting with the most primitive objects and activities with which the user is concerned. The difficulty with a bottom-up development is that the designer must immediately immerse himself in a mass of detail. One reason that a top-down development is not usually followed is the initial difficulty in recognizing the abstractions which constitute the top level of the model. Nevertheless a top-down development has the distinct advantage of allowing the designer to isolate himself from all details except those relevant to the decision at hand.

A database is usually a model of some very familiar real-world system. If the user requires a database, he has presumably interacted with the actual system long enough to acquire a sophisticated conception of it. In acquiring this conception, he must have abstracted various relationships as named objects in order to articulate phenomena of interest in the system. These named objects will occur at various levels of abstraction. The user may not have abstracted the relationships between the highest-level named objects. Since such relationships do not have to be thought of as a whole, there is no strong reason to make these abstractions. However, these relationships cannot be very complex, or the user would not be able to articulate them effectively. The designer should therefore be able to abstract these relationships as named objects in one or two additional levels. On this basis, we conjecture that the top level for a system model is at most one or two levels higher than the most abstract objects which can be named by the user.

Once the designer has determined the names of the user's most abstract objects, he should go one or two levels up to abstract their relationships, and then move down to begin a top-down development. Thereafter, the designer will be principally concerned with decomposing objects into relationships between simpler objects. During this decomposition activity, the designer is likely to factor out the same object from different parts of the decomposition hierarchy. This indicates the presence of an object which participates in several relationships. Finally, when the designer reaches objects whose details are of no interest to the user, he has located the bottom level of the model. It may well happen that the designer will ultimately discover better abstractions for structuring the model than those employed by the user. Nevertheless, we contend that one or two levels above the user's most abstract named objects constitutes a valid point of departure for a top-down development.

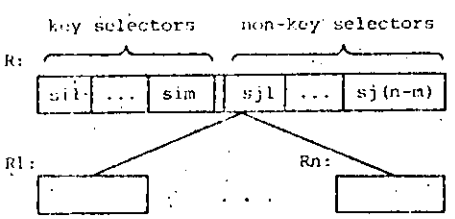
During this top-down development, the designer should postpone decisions about keys as these are irrelevant in determining the abstract structure of the model. Once the abstract structure is completed, it may be checked against third normal form requirements. When necessary, attributes may be moved down the hierarchy to transform each relation into third normal form. Finally, selector names and keys can be determined during an upward pass through the hierarchy.

In practice mistakes and misconceptions will prevent a purely top-down development, such as we described, from occurring. Nevertheless, it is a sound practice to seek a development which proceeds as far as possible in a top-down fashion. We shall give an idealized example of a top-down development to indicate how abstractions are utilized.

To avoid a lengthy textual development, we use the following graphical notation to abbreviate a relation declaration. A declaration:

```
var R: collection of aggregate [si1, . . . , sim]
                si1: {key} R1;
                . . .
                sin: {key} Rn
              end
```

will be represented by:



We assume that a database is required which models instructor development within a national university system. In particular, information about the committee assignments and teaching activities of instructors is necessary.

The first design step is to determine which are the most abstract objects concerned with an instructor's committee assignments and teaching activities. As far as committee assignments are concerned, we have the abstract objects "committee" and "instructor." There is no generally used term which refers to a higher abstraction involving committee assignments. As far as teaching activities are concerned, the most abstract objects we have are those of "class" and "pupil." The

Fig. 4. Design step 1.

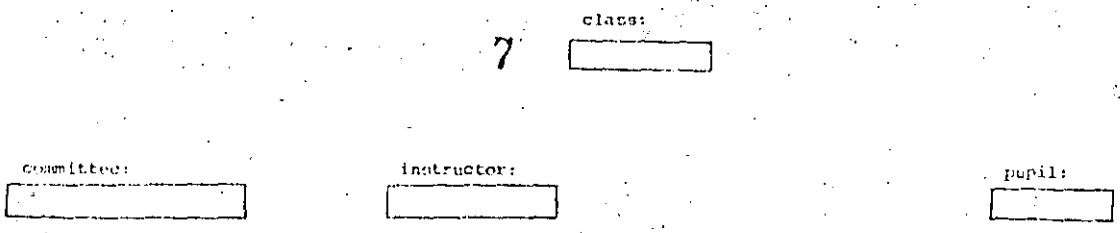


Fig. 5. Design step 2.

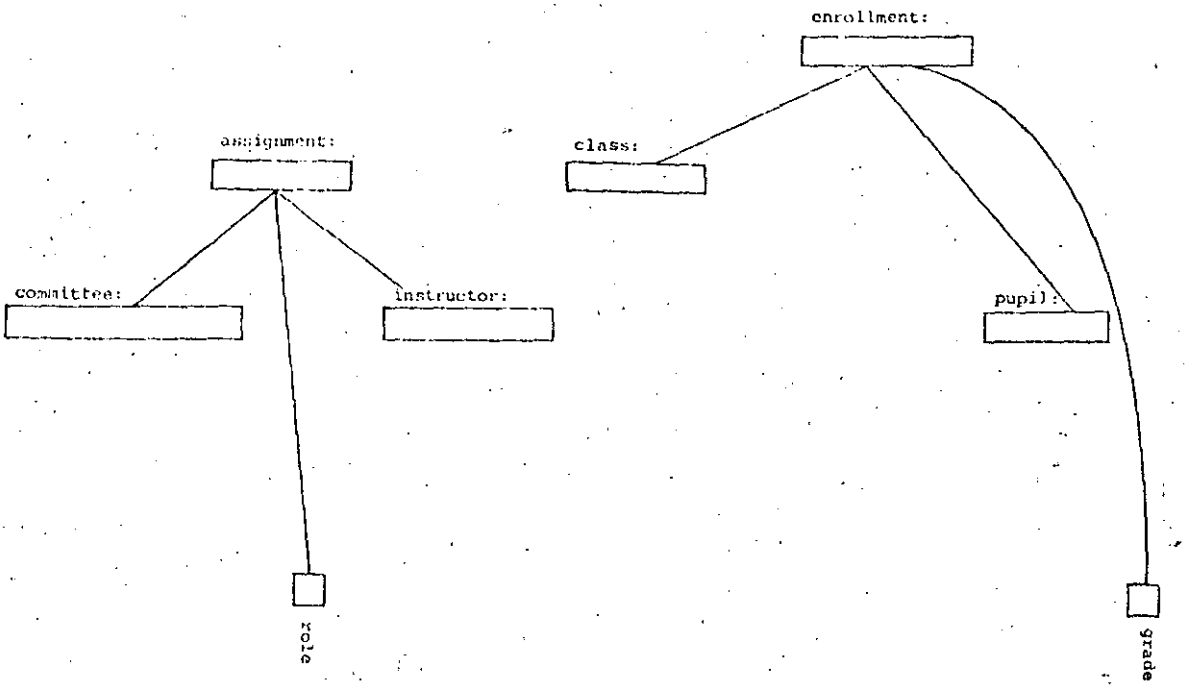


Fig. 6. Design step 3.

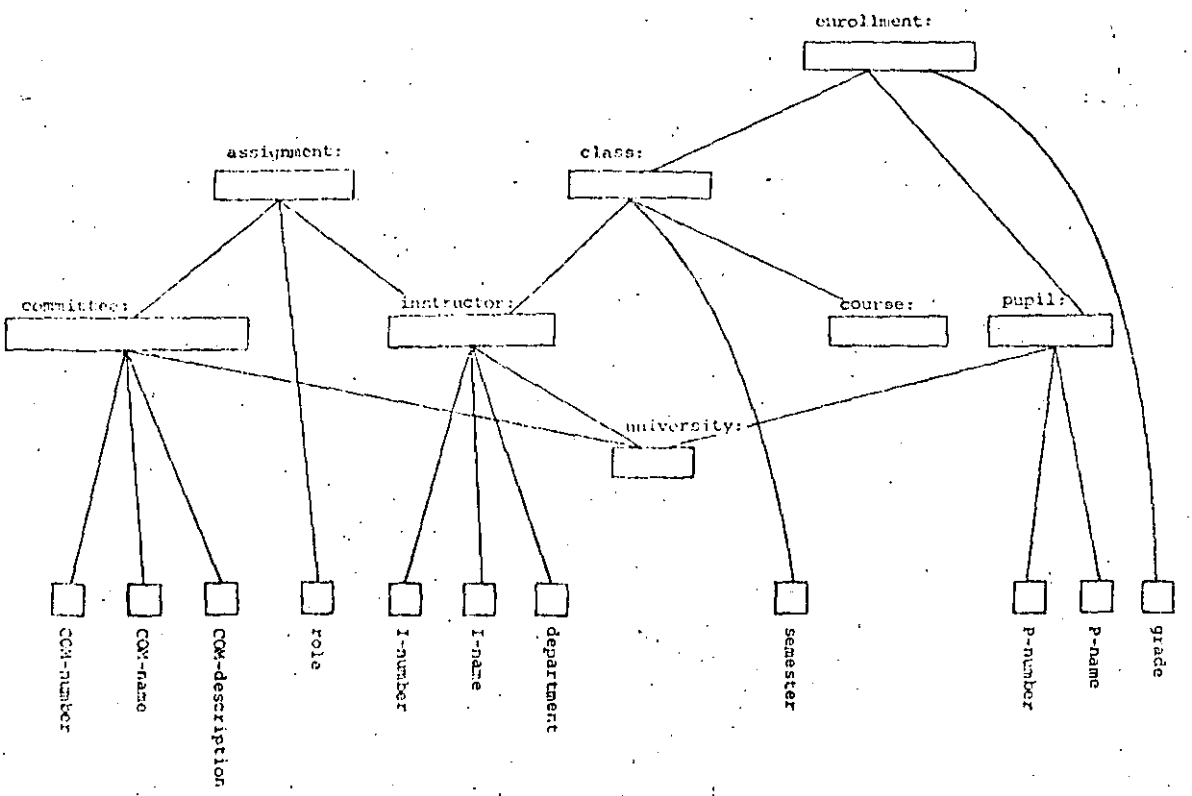


Fig. 7. Design step 4.

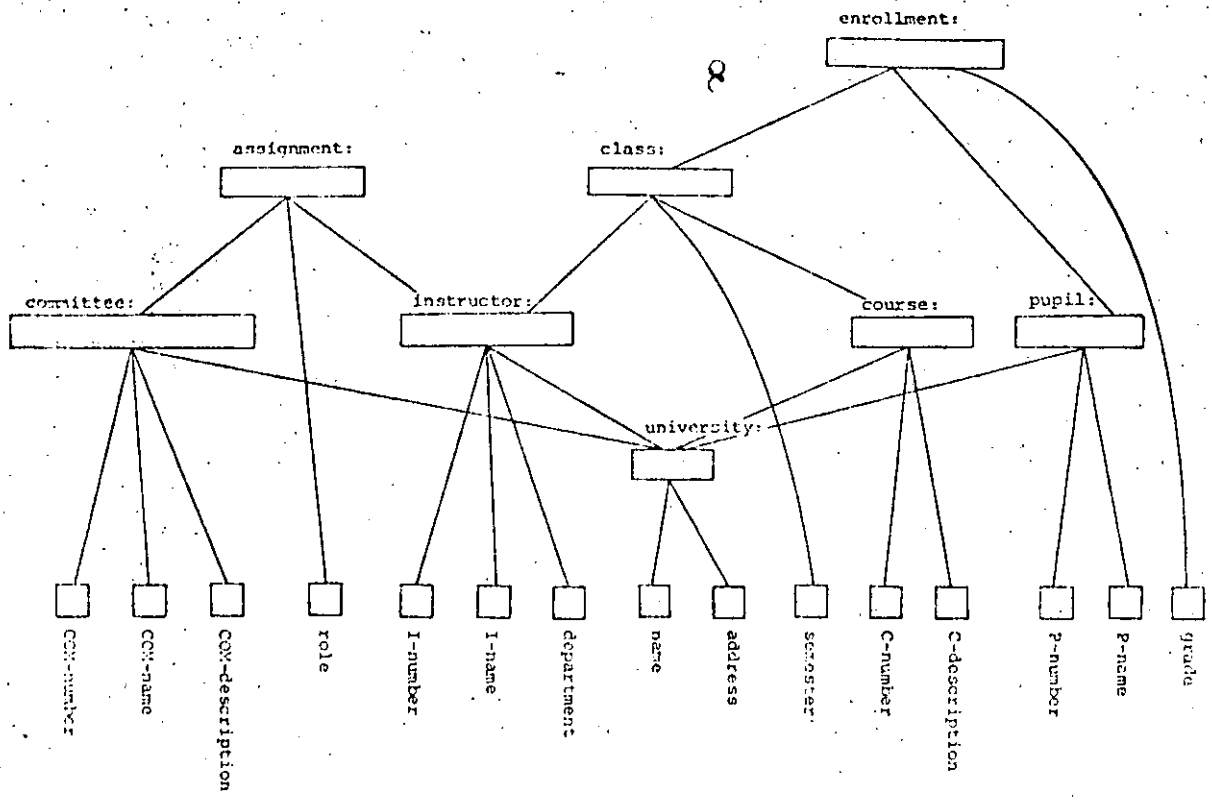
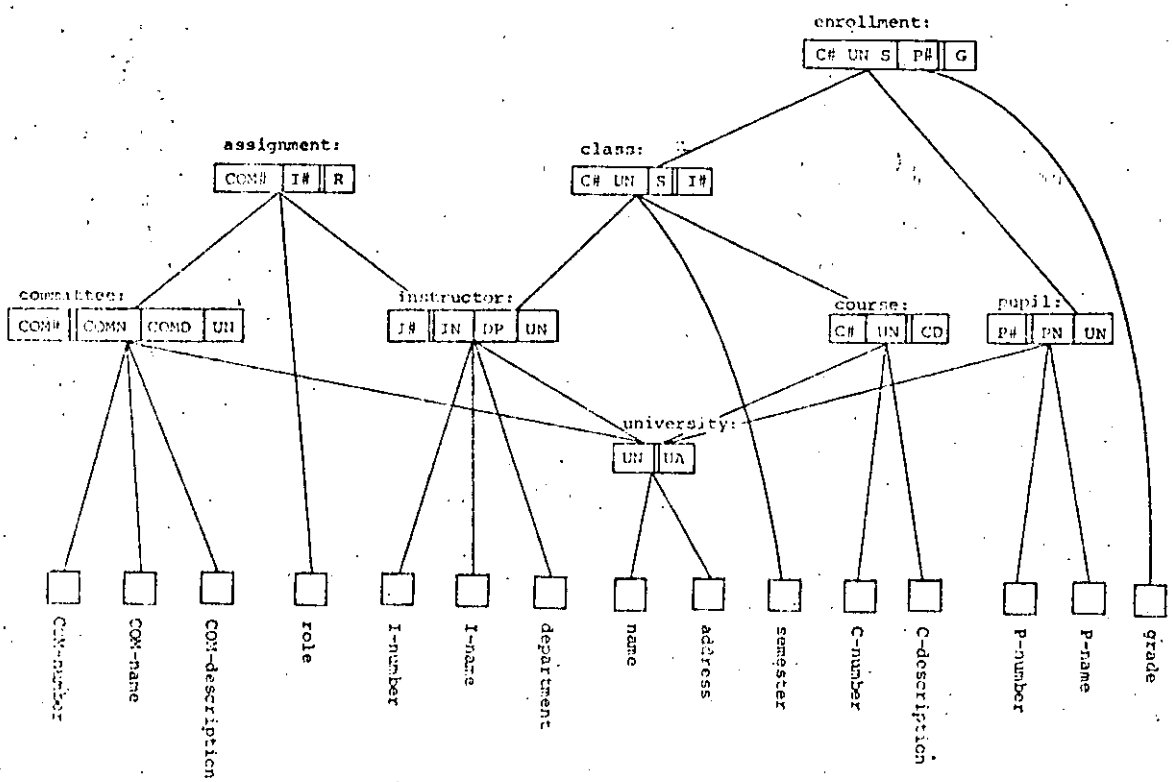


Fig. 8. A model for a national university system.



model consists initially of these four abstract objects as shown in Figure 4..

We next determine the relationships of interest in which these objects participate. We are interested in the assignment of instructors to committees and the enrollment of pupils in classes..We abstract these two relationships as objects and include any additional interesting attributes of these objects. We are interested in the grade resulting from an enrollment and the role played by the instructor in the assignment. The model now appears as shown in Figure 5. We are not concerned with selector names and keys at this stage.

We decide next that "role" and "grade" are primitives since their details are of no interest to the user. We proceed to decompose "committee," "instructor," "class," and "pupil" as relationships between lower-level objects. In doing so we introduce the objects "course" and "university" in addition to many new primitive objects. The model now appears as shown in Figure 6. Notice that the objects "instructor" and "university" participate in several relationships. For example, an instructor teaches a course during a semester, and an instructor is assigned to a committee in a certain role. We finally terminate this top-down development by decomposing "university" and "course" into relationships over primitive objects. The model then appears as the hierarchy of abstract objects shown in Figure 7.

It is at this time that we should check the model for conformity with third normal form. If a particular relation is not in third normal form, then we can move down all appropriate attributes to the next level of the hierarchy. For example, if "instructor" functionally determines "course" in the "class" relation, then we can move down the attribute "course" to the "instructor" relation.

We must now assign selector names and keys to each object. This can be done by an upward pass through the object hierarchy. We first assign selector names to the lowest-level nonprimitive objects and determine their keys. In determining keys it is often necessary to use auxiliary information about the system being modeled. For example, we must know that a university is uniquely determined relative to the national system by its name. We must know that committee numbers have been classified nationally and are therefore unique throughout the system; however, course numbers are unique only within a given university. The keys for committee and course number are thus chosen as [*COM#*] and [*C#, UN*], respectively. The information that a pupil may only register at one university and that an instructor may only be employed at one university allows us to use [*P#*] and [*I#*] as the keys for "pupil" and "instructor," respectively. We can now move up the hierarchy, assigning keys as we go.

The final model appears in Figure 8. The model is comprised of many details; however, by employing abstraction appropriately, we have been able to reduce to a minimum the number of details that the designer must keep in his mind at any one time. A very attractive feature of this approach is that we can separate in time the activity of decomposing abstract objects from the activity of determining the keys of these objects.

5. Conclusion

Aggregation is only one kind of abstraction that is useful in understanding the real world. Another kind is generalization in which a set of "similar" objects is abstracted as a single generic object. In [10] we discuss a method for structuring relational databases which supports both aggregation and generalization.

It is important to distinguish between abstractions, such as aggregation and generalization, which are useful in thinking about the real world and abstractions which are useful in the implementation of data structures. "Real-world" abstractions hide details the user wants to ignore *momentarily* in thinking about the real world. "Implementation" abstractions hide details the user *never* wants to see about data structure implementation. The successive real-world abstractions employed in a relational hierarchy are orthogonal to the successive implementation abstractions necessary to represent a relation as a complex pointer-connected secondary storage data structure.

At present, programming languages do not contain adequate primitives to support real-world abstractions. Constructs such as arrays and files are primarily implementation abstractions of random-access storage and sequential-access storage. The record type of PASCAL serves a dual role: to express real-world abstractions and also as an implementation mechanism. It would seem important to add new "real-world" types (such as aggregate and collection) to programming languages and simultaneously to distinguish "real-world" types from "implementation" types.

Acknowledgements. We are grateful to R.W. Taylor, C.A.R. Hoare and N. Wirth for their insightful comments on earlier versions of this paper. We also acknowledge the useful criticisms of the referees.

References

1. Codd, E.F. A relational model of data for large shared data banks. *Comm. ACM 13*, 6 (June 1970), 377–387.
2. Codd, E.F. Further normalization of the data base relational model. In *Courant Computer Science Symposia 6: Data Base Systems*, Prentice-Hall, Englewood Cliffs, N.J., May 1971, pp. 33–64.
3. Codd, E.F. Relational completeness of data base sublanguages. In *Courant Computer Science Symposia 6: Data Base Systems*, Prentice-Hall, Englewood Cliffs, N.J., May 1971, pp. 65–98.
4. Codd, E.F. Recent investigations in relational data base systems. Information Processing 74, North-Holland Pub. Co., Amsterdam, 1974, pp. 1017–1021.
5. Fry, J.P., and Sibley, E.H. Evolution of data-base management systems. *Computing Surveys 8*, 1 (March 1976), 7–42.
6. Hoare, C.A.R. Notes on data structuring. In *APIC Studies in Data Processing No. 8: Structured Programming*, Academic Press, New York, 1972, pp. 83–174.
7. Liskov, B., and Zilles, S. Programming with abstract data types Proc. Symp. on Very High Level Languages, Santa Monica, Calif., March 1974, pp. 50–59.
8. Schmid, H.A., and Swenson, J.R. On the semantics of the relational data model. Proc. 1975 SIGMOD Conf., San Jose, Calif., May 1975, pp. 211–223.
9. Senko, M.E., Altman E.B., Astrahan, M.M., and Fehder, P.L. Data structures and accessing in data-base systems. *IBM Syst. J. 1*, 1 (1973), 30–93.
10. Smith, J.M., and Smith, D.C.P. Database abstractions: aggregation and generalization. *ACM Trans. on Database Syst. 2*, 2 (June 1977), 105–133.
11. Taylor, R.W., and Frank, R.L. CODASYL Data-base management systems: *Computing Surveys 8*, 1 (March 1976), 67–103.
12. Wirth, N. The programming language PASCAL. *Acta Informatica 1*, 1 (1971), 35–63.
13. Woods, W.A. What's in a link: foundations for semantic networks. In *Representation and Understanding*, Academic Press, New York, 1975, pp. 35–82.

DISEÑO DE BASES DE DATOS

EXPRESS: A DATA EXTRACTION, PROCESSING AND RESTRUCTURIN SYSTEM

MAYO, 1985.

EXPRESS: A Data EXtraction, Processing, and REStructuring System

N. C. SHU, B. C. HOUSEL,
R. W. TAYLOR, S. P. GHOSH, and V. Y. LUM
IBM Research Laboratory

EXPRESS is an experimental prototype data translation system which can access a wide variety of data and restructure it for new uses. The system is driven by two very high level nonprocedural languages: DEFINE for data description and CONVERT for data restructuring. Program generation and cooperating process techniques are used to achieve efficient operation.

This paper describes the design and implementation of EXPRESS. DEFINE and CONVERT are summarized and the implementation architecture presented.

The DEFINE description is compiled into a customized PL/1 program for accessing source data. The restructuring specified in CONVERT is compiled into a set of customized PL/1 procedures to derive multiple target files from multiple input files. Job steps and job control statements are generated automatically. During execution, the generated procedures run under control of a process supervisor, which coordinates buffer management and handles file allocation, deallocation, and all input/output requests.

The architecture of EXPRESS allows efficiency in execution by avoiding unnecessary secondary storage references while at the same time allowing the individual procedures to be independent of each other. Its modular structure permits the system to be extended or transferred to another environment easily.

Key Words and Phrases: data translation, data conversion, file conversion, data restructuring, program generation, very high level languages, data description languages, data manipulation languages
CR Categories: 4.12, 4.33, 4.41

1. INTRODUCTION

Data conversion is a problem every user faces when moving data to a database environment. The problem is one of cost. The data to be moved to the new system will typically exist in a variety of formats and file structures. To move the data, a number of conversion programs are required (one paper reports a manual conversion effort requiring 100 COBOL programs for restructuring of 29 files [26]). These programs are nontrivial because they involve accessing many files, extracting portions of them, and restructuring the data to conform with the new database design. But the cost of developing these programs usually cannot be amortized either over years of use or over many installations. In addition there is the problem of validating the correctness of the conversion, as in the development of any system

Authors' address: IBM Research Laboratory, 5600 Cottle Road, San Jose, CA 95193.

of programs. Errors in the data conversion process will affect every usage of the data.

In recent years several approaches to this problem have been proposed [4–6, 12–14, 18, 19, 22–25]. Several prototype implementations exist [1–3, 15–17, 20]. Among the approaches proposed is the methodology given in the papers by Housel, Lum, and Shu [7–9, 12, 21]. Although the method in those papers seemed attractive, there are questions remaining to be answered. For example, is the approach efficient? Will users accept a tool requiring additional training? After all, users know how to do data conversion, though it may be difficult and tedious. What additional resources, storage space, computer, etc., does it take to use such a system? Although these issues can be argued, the only way to gain some concrete answers is to build a system to determine the merits or demerits of a generalized approach. This paper describes the architecture and implementation of such a system, named EXPRESS, a data EXtraction, Processing, and REStructuring System.

EXPRESS has several new features not included in other previous prototype implementations. It can access and restructure data from multiple files and produce multiple restructured files in a single run. The restructuring language CONVERT allows very general restructurings to be specified without resorting to procedures written in a traditional programming language. The accessing and restructuring steps can be separated if desired. This allows a systematic treatment of data errors and also allows parts of the package to run on different machines. On the other hand, the accessing and restructuring steps can be combined allowing the package to be used as a data extraction tool.

The model for EXPRESS is well defined in the original paper [12]. However, to make this paper self-contained, Section 2 will describe briefly the model as implemented along with some motivations for the architecture.

Users of EXPRESS interact with the system via two languages—DEFINE for describing data structures of source and target files and CONVERT for specifying the transformations of data from one or more files to a set of files for a target system. These languages are described in [8, 9, 21]. Since the understanding of this paper depends heavily on knowing the semantics of the languages, Section 3 will give a brief summary of them.

Section 4, which is the main part of this paper, is broken down into four major subsections. Section 4.1 deals with the process of accessing the source data and changing it into a system-independent form. Section 4.2 discusses the preprocessing required prior to the actual execution of the restructuring process. Section 4.3 discusses the run-time system. Section 4.4 describes additional aspects of preprocessing that are necessary but difficult to explain prior to the discussion of the run-time system.

Performance and usability have been major concerns in the design and implementation of EXPRESS.

First, let us discuss the issue of performance. Since the system may be processing a voluminous amount of data, inefficiency will be magnified. One major step taken to avoid this effect is the compilation of DEFINE and CONVERT statements into programs customized for the particular tasks. (A comparison of compilation with interpretation in one data translator is given in [1].) For example, a DEFINE de-

scription of a file is compiled into a procedure that can read a file with that description. In like manner a CONVERT specification is compiled into a customized restructuring program. PL/1 was chosen as the target language for the DEFINE and CONVERT compilers because of its ability to deal with pointer-based structures and exceptional conditions and its relatively wide availability.

Another major part of achieving good performance is the reduction of I/O activities. From the user's viewpoint, each CONVERT operation requires one or more real files as input and generates a real file. Actually this does not happen. Through the concept of pipelining, to be described later, the system generates real intermediate files only when necessary. A module, called the Analyzer, was built to analyze the CONVERT program to determine when it is necessary to create intermediate files.

Other features designed for good performance are the use of concurrency in the execution of CONVERT operations and the generation of efficient code.

EXPRESS is believed to be user friendly. The languages themselves, being high level, provide the user with a concise and nonprocedural way of specifying the job to be done. In addition we have built into the system several features which enhance usability. First, the system allocates buffers and generates job control statements automatically. Second, the PL/1 code generated by the DEFINE and CONVERT compilers is quite readable. This allows the user to follow or modify the generated code easily if he so desires (e.g. for special error checking). This topic is discussed in Section 4. Third, the system has "user hooks." For example, a user can intercept data for modification during reading.

EXPRESS is operational in a laboratory environment. We plan to test the application of the system in a real database conversion.

2. THE DATA TRANSLATION PROCESS

The translation process in EXPRESS can be broken into three major steps, consisting of the Read step, the Restructuring step, and the Load step, as shown in Figures 1–3. First, we briefly describe the three steps, and then we discuss some of the issues involved which led to this approach.

The Read step. The function of the Reader is to access the source data stored in the source system, perform extensive error checking and data editing, and transform each physical source file into an internal form. These files will subsequently be used as an input to the Restructuring step. The term "source system" in our context is defined as the combination of hardware and software systems required to support the source data being accessed. As shown in Figure 1, the Reader consists of two main components: a source system-dependent component called the "Physical Reader" (denoted by P) and a system-independent component called the "Generated Reader" (denoted by GEN).

The role of the Physical Reader is to serve as an interface between the source system and the Generated Reader. This component uses the source system access methods (e.g. DL/1, QSAM, etc.) to access the source files and filter out all system-dependent data.

The role of the Generated Reader is to read the source data, break it down into data item instances, aggregate the item instances into group instances, and connect the group instances appropriately to form a hierarchically structured record in

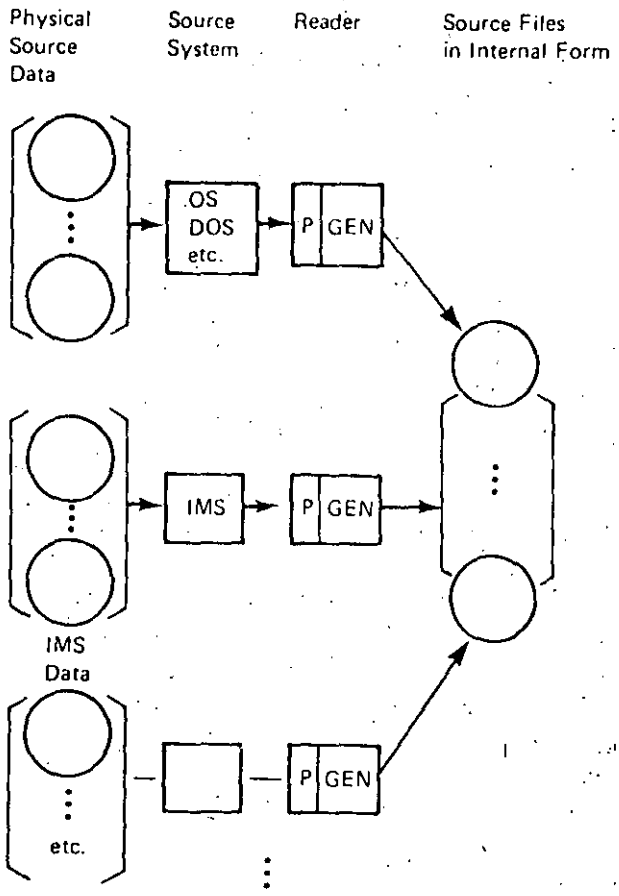


Fig. 1. Read step

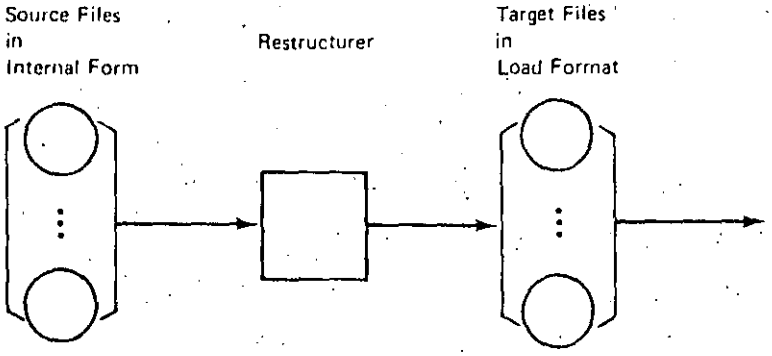


Fig. 2. Restructuring step

internal form. In order to decompose and check the source data a description of the file's structure and valid instances must be given. The DEFINE data description language is used for this purpose. A brief summary of DEFINE is given in Section 3.1.

The Restructuring step. The Restructuring step derives a set of target files from

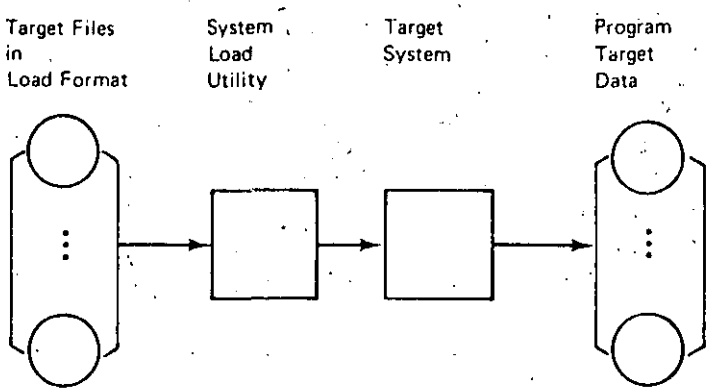


Fig. 3. Load step.

the source files as specified by the restructuring language CONVERT. CONVERT is summarized in Section 3.2. All the actual restructuring is performed with files in internal form. However, when a record is to be written to a target file, it is converted by a Writer program into the required load format of the target system.

The Load step. This step (Figure 3) simply involves taking advantage of the target system facilities to load the target files produced above into the target databases. The EXPRESS system is not concerned with this function.

We now discuss the relevant issues in the process described above. First, we implied that all user views of source files conform to a hierarchical representation. This approach was adopted for several reasons. Since the preponderance of data is hierarchical (e.g. COBOL files), this is a convenient form in many cases. Furthermore, networks can be expressed as a family of hierarchies; relational data structures can be considered as degenerate hierarchies. A full discussion of this issue appears in [12].

Another issue is: Is it necessary to separate the Read step from the Restructuring step? The answer is that it is not necessary, but highly desirable. One reason is portability because it is likely that the Read step will be executed on one machine and the Restructuring step on another. Another reason for this separation is to allow the Reader to perform error checking and editing so that users can correct or modify their data at an early stage of the translation process. In discussions with people who have done data conversion, we found that a 40–60-percent error rate was not uncommon. Finally, by processing the entire source file, the Reader can collect statistics. These statistics (e.g. number of instances) can be used for the allocation of files and buffers in the Restructuring step.

In summary EXPRESS takes full advantage of existing facilities and localizes all system dependencies. Its architecture allows the restructuring of data regardless of origin or destination.

3. EXPRESS SPECIFICATION LANGUAGES

In earlier papers [8, 9, 21] the EXPRESS specification languages DEFINE and CONVERT have been described at length. A full discussion of these languages is beyond the scope of this paper. A brief description of each language is given here.

3.1 DEFINE

DEFINE is a nonprocedural data description language used to describe the user's view of the source as it is passed from the Physical Reader to the Generated Reader. It is also used to describe the target file in load format.

As explained in [12], a hierarchic model of data is adopted for each individual file involved in a translation. Thus DEFINE is oriented toward describing a single hierarchical record, referred to as a section in this paper. (For comparison, the same concept is called a database record in IMS [10]). A *section* is thus defined to be one root instance with all its descendant instances. A *file* is a series of sections. Within a section, there are still many possibilities for optional, variable, and self-describing data. Also, the section may span a number of physical records in the source system. DEFINE gains its power by handling a wide variety of these cases without having to resort to procedures.

There is one FILE DESCRIPTION for each file to be read by the Reader and also one for each file to be produced by the Writer. In our examples we focus on the features used in describing a single file.

Consider the data stream and its structure represented by the hierarchic type tree in Figure 4(a) (spaces have been introduced for readability and would not appear in the actual data). This data structure represents a department "master record" with two types of "detail records"—employees and projects—intermixed arbitrarily following each "master record." The number of employees is given by the first integer (3 in this example); the number of projects is given by the second integer. Data streams like this are common in COBOL applications which use record descriptions and the REDEFINES option. Normally, a COBOL program would "know" that the first record was a DEPARTMENT record. It would read it and save the two count fields NEMP and NPROJ. An iteration would then be performed. On each iteration, a "detail" record would be read, its TYPE field would be examined, and the appropriate processing would be done for this record type. The important point to note is that, from a data description viewpoint, the hierarchical structure description of the record does not appear in the declarations but is embedded in the procedural logic.

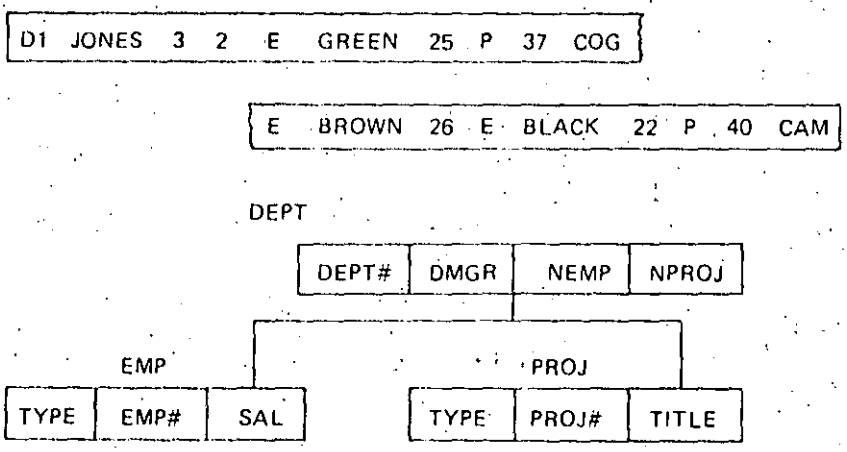


Fig. 4(a). Hierarchical file with self-describing data

```
FILE DEPT-EMP-PROJ:
     OCCURS FROM 1 TIMES,
     FOLLOWED BY EOF;

  GROUP DEPT-REC:
    DEPT#:   CHAR(PIC IS 'A9');;
    DMGR:    CHAR(20);;
    NEMP:    BINARY(15);;
    NPROJ:   BINARY(15);;

    GROUP EMP-PROJ:
      OCCURS NEMP+NPROJ TIMES;
      TYPE:   CHAR(1);;
      GROUPCASE EP:
        GROUP EMP(TYPE = 'E'):
          EMP#:      CHAR(20);;
          SAL:       DECIMAL(8);;
        END EMP:

        GROUP PROJ(TYPE = 'P'):
          PROJ#:   CHAR(4);;
          TITLE:   CHAR(20);;
        END PROJ;
      END EMP-PROJ;
  END DEPT-REC;
END;
```

Fig. 4(b). DEFINE description of self-describing data

The DEFINE description of this data stream is given in Figure 4(b). The hierarchic structure is made explicit by "embedding" the group EMP-PROJ within the group DEPT-REC. The repetition control is described by the OCCURS clause. The GROUPCASE statement describes the conditional nature of the data stream on each iteration. On each iteration, the TYPE item will be isolated by the Reader. On this basis, one or the other (or neither when there is an error) of the conditional groups is selected and the data stream processed according to that description.

Conditionally appearing items or groups are handled similarly to the EMP or PROJ groups in Figure 4. A predicate is associated with the item or group. This predicate is evaluated (possibly by using look-ahead techniques to find the relevant operands). If the predicate evaluates to *true*, the item or group is processed; otherwise it does not appear in the data stream.

DEFINE also has a wide variety of constructs for describing repetition. One of the legal repetition control clauses is:

```
OCCURS FROM 10 TO 20 TIMES,
       PRECEDED BY '(',
       INFIXED BY ',',
       FOLLOWED BY ')';
```

This description will cause the Reader to use information in the data stream to delimit the repetition. All three clauses are optional to accommodate the various possibilities found in real data. The Reader will check that the data conforms to the repetition bounds.

A common case in the arrangement of data in a hierarchy is to have the instances belonging to the same section appearing in top to bottom, left to right order, with each group carrying a "type code." This case is discussed in more detail in Section 4.1. In DEFINE, a group in this hierarchy could be described with

```
OCCURS FROM 0 TIMES;
        with each instance
        PRECEDED BY HEX '03';
```

As another example,

```
OCCURS FROM 1 TIMES
        UNTIL It = 0;
```

describes data dependent repetition. The Reader would continue parsing instances of these groups until the condition became true.

In addition to describing the structure of the data stream, DEFINE has many facilities for stating validity checks that should be enforced. Value ranges and sets of legal values may be declared. For example,

```
I1: BINARY(15); VALUE IS > 10 AND < 30;;
I2: CHAR(5); VALUE IS IN ('NUT','BOLT');;
```

Furthermore, the language has a powerful user-extendible picture facility. In addition to the normal alphabetic (A), digit (9), and alphanumeric (X) pictures, a user can define character literals that must exist in certain positions. Thus

```
'99"Q"AA"7"99'
```

would specify two digits, the letter Q, two alphabetics, the digit 7, and two more digits. Sets of characters can also be declared as in

```
PICTURE CODE D is '-' OR '/';
```

This would allow dates written in various ways to be specified as

```
99D99D99
```

Variable length strings can also be handled:

```
PICTURE CODE Q IS '9(1:3)';
```

means the class of digits of length 1 to 3. Omission of the upper bound implies indefinite repetition. Thus if

```
PICTURE CODE R IS ' ','999';
```

then the picture

```
QR(0:)
```

describes an indefinite length digit string of the form

```
9
99
999
9,999
etc.
```

In addition to the picture facility, DEFINE has considerable editing capability.

The TRANSLATE clause allows data values to be changed on the basis of a table look-up function:

```
TRANSLATE ('EXY','EXPY','EXWAY') TO 'EXPWAY';
TRANSLATE ('      ') TO 0;
```

It is also easy to change the type or length of an item in either the Reader or the Writer DEFINE specification.

Of course users will want to insert their own error detection and correction procedures, and these can be accommodated, as will become clear in Section 4.1. The user procedures will be called either unconditionally or on some detected condition.

To summarize, DEFINE is a nonprocedural data description language oriented toward hierarchies. It makes explicit the hierarchical group and item structures of the source data, which may have been procedurally or implicitly defined in the source system. DEFINE is similar in organization and data structure concepts to existing data description facilities. Thus it should not be difficult to learn. However, DEFINE is more powerful than common data description languages because of its ability to deal with common cases of self-defining, variable, and optional data. Its extensive error checking capabilities allow the Reader procedure to perform many checks so that users may "clean" their data before restructuring begins.

3.2 CONVERT

CONVERT is a very high level language designed to operate on hierarchical data. It contains a total of nine operations in the format of "operator (operand)" (Figure 5). Conceptually from a user's viewpoint, every operation starts with one or more real files and generates a real file. Thus every CONVERT statement involves an operator with an assignment operation in the format of

```
Target-file = Operation;
```

However, CONVERT operations can be nested. In such cases the assignment is implicit. The system assigns a file name to the result of a nested operation.

The structure of the target file can be declared either on the left-hand side of the assignment or in a separate declaration. In either case the structure on the left must be consistent with that of the operation on the right. EXPRESS can also derive the structure of files when no explicit declaration is given.

For convenience a CONVERT user may view his hierarchical data represented in a tabular format called a *Form*. Figure 6 shows the correspondence between a type tree or hierarchy graph (representing the structure of a file) and a Form. The type tree corresponds to the heading of the table, referred to as a *Form Schema* Repeating groups are enclosed in parentheses in the Form Schema. The data instances together with the type tree make up the Form. Furthermore, note that subtrees are named but not every level (i.e. node) of a tree is named. More detailed description of Forms can be found in articles [8, 12, 21] cited in the References. When no confusion exists, the terms Form and file will be used interchangeably.

Let us now proceed to describe three of the CONVERT operators, SLICE, SELECT, and GRAFT, to illustrate the language. The following conventions will be used: ( ) to denote an optional clause, / to denote alternatives, capital letter "F" to de-

1. Assignment
2. SLICE (c1, . . . , cn FROM F)
3. CONSOLIDATE (F)
4. SELECT (<Expr1, . . .> FROM F <WHERE selection criteria>)
5. GRAFT (F1 ONTO F2 <BEFORE/AFTER f>
   WHERE match criteria <, ELSE F1/F2/F1;F2 PREVAIL >)
6. MERGE (F1, F2)
7. ELIMDUP (F)
8. SORT (F BY f1 <ASC/DES>, . . . , fn <ASC/DES>)
9. Built-in functions:
   SUM, MAX, MIN, COUNT, AVG

Notations:

< >	optional specification
/	alternative
___	default
F	Form name
f	field name
c	component (field or subtree) name

Fig. 5. CONVERT operations

note a Form, small letter "f" to denote a field, and small letter "c" to denote a component (i.e. field or subtree). Keywords will be capitalized.

(1) SLICE (c1, ... , cn FROM F)

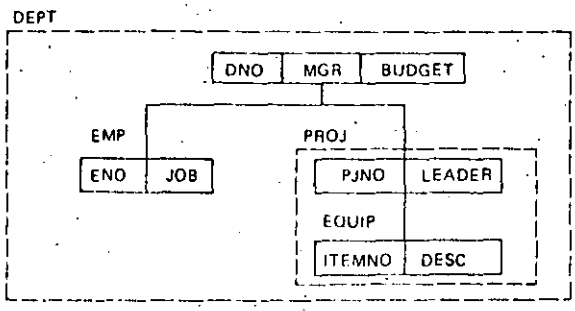
The objective of this operation is to transform hierarchical data into a flat file. Consequently the components c1, ... , cn must all be contained in the same hierarchical path. The operation can stop at any level. However, no level can be skipped. Using the DEPT Form in Figure 6, the operation

D = SLICE (DNO,MGR,PJNO,LEADER,DESC FROM DEPT);

will produce a result like that shown in Figure 7.

(2) SELECT (<Expr 1, ... , Exprn> FROM F <WHERE selection criteria>)

The SELECT operator selects the part of a Form which meets the selection criteria. The operation may be quite simple, as in the case of a projection of a flat file, or it may be quite complex, involving the selection of different components based on the content of the fields. In general an Expr can be a component, a built-in function, or arithmetic operation, or a CASE (to be described later). A selection



Hierarchy graph or Type tree

(DEPT)								
DNO	MGR	BUDGET	(EMP)		(PROJ)			
			ENO	JOB	PJNO	LEADER	(EQUIP)	
							ITEMNO	DESC
D1	DOE	40000	19	ENG	J6	RAE	221	COMPUTER
			41	SEC	J8	MEE	46	SCOPE
			52	TECH			317	LASER
			77	ENG	J11	FAR	271	COMPUTER
D4	SO	20000	60	CHEM	J9	LA	47	MICROSCOPE

Fig. 6. A Form representation for DEPT
Note. A repeating group is represented by ( ).

(D)				
DNO	MGR	PJNO	LEADER	DESC
D1	DOE	J6	RAE	COMPUTER
D1	DOE	J8	MEE	SCOPE
D1	DOE	J8	MEE	LASER
D1	DOE	J11	FAR	COMPUTER
D4	SO	J9	LA	MICROSCOPE

Fig. 7. Example of a SLICE operation

criterion is one or more logical factors connected by ANDs. Each logical factor is a series of comparison expressions separated by ORs. Each comparison expression has the form "operand OP operand," where OP is one of the elements in (EQ,GT,LT,NE,GE,LE) and operand is a field or an arithmetic expression in the source Form being processed. Thus, for example, the CONVERT statement

```
T = SELECT (DNO,MGR,PROJ(PJNO,LEADER) FROM DEPT
           WHERE DESC EQ 'COMPUTER');
```

for the DEPT Form in Figure 6 will produce the result shown in Figure 8(a), and

(a)

(T)			
DNO	MGR	(PROJ)	
		PJNO	LEADER
D1	DOE	J6	RAE
		J11	FAR

(b)

(T)	
DNO	BUDGET
D1	50000
D4	30000

Fig. 8. Examples of SELECT operations

the statement

```
T = SELECT (DNO,BUDGET = BUDGET + 10000 FROM DEPT
            WHERE (DESC EQ 'COMPUTER'
            OR DESC EQ 'MICROSCOPE'));
```

will result in a Form like that in Figure 8(b). Note that the selection process for this operation resembles scratching out items not wanted. In the first SELECT (Figure 8(a)), for example, project J8 does not satisfy our criteria. All instances of this subtree therefore have been scratched and discarded even though D1 has been selected. If no instance of DESC survived in the scratching process, its parent will be scratched, and we proceed to examine the next level of its ancestor. Thus J9 becomes scratched and so does the parent of J9 (i.e. D4). As a result, D4 does not appear in the target Form. This process is referred to as the *scratch algorithm*.

A more complex operation for the SELECT operator is the inclusion of the CASE clause for testing the content of a field prior to making a processing decision. To illustrate, let us consider some examples. Suppose that, using the Form D (shown in Figure 7), we wish to create a file with a field A which contains manager (MGR) if the item description is "COMPUTER," but the project leader (LEADER) if the description is not. Furthermore, we wish to retain department number (DNO) and description (DESC) information in the target file. In CONVERT it becomes

```
T(DNO,A,DESC) = SELECT (DNO,
                        CASE(A = MGR     WHEN DESC EQ 'COMPUTER',
                                 LEADER OTHERWISE),
                        DESC
                        FROM D);
```

The result of this statement is shown in Figure 9(a).

When a selection criterion exists in the SELECT operator with a CASE clause, the selection criterion will be evaluated prior to the CASE operation. Thus

```
T(DNO,A,DESC) = SELECT (DNO,
                        CASE(A = MGR WHEN DESC EQ 'COMPUTER',
                                 LEADER OTHERWISE),
                        DESC
                        FROM D WHERE DNO NE 'D1');
```

will result in a Form without department D1 like that shown in Figure 9(b).

(3) GRAFT (F1 ONTO F2 <BEFORE/AFTER f> WHERE match criteria
<,ELSE F1/F2/F1,F2 PREVAIL>)

(a)

(T)		
DNO	A	DESC
D1	DOE	COMPUTER
D1	MEE	SCOPE
D1	MEE	LASER
D1	DOE	COMPUTER
D4	LA	MICROSCOPE

(b)

(T)		
DNO	A	DESC
D4	LA	MICROSCOPE

Fig. 9. Examples of SELECT operations involving CASE expressions

GRAFT is for the joining of two trees to form a larger tree. By using the BEFORE/AFTER option, a tree F1 can be grafted to another tree F2 at a position immediately to the left or right of a field f in F2. If the BEFORE/AFTER is not given, F1 will be joined to F2 at the rightmost position of F2.

Furthermore, the match criteria is a conjunction of equal comparisons in the format of

```
g1 EQ f1 <AND g2 EQ f2 ...>
```

where g1, ... are fields in F1 and f1, ... fields in F2. If the match criteria are satisfied, the instances will be joined. All fields in both F1 and F2 will be included in the target Form. However, only one field in each match will be in the resulting structure, e.g. either f1 or g1 will appear in the target but not both. As an example, suppose that the DSUP Form with department and supplier information as shown in Figure 10(a) is to be joined to the DEPT Form in Figure 6 to create a file with structure shown in the hierarchy graph of Figure 10(b). The CONVERT statement for this will be

```
T = GRAFT (DSUP ONTO DEPT BEFORE PJNO
           WHERE DEPT.DNO EQ DSUP.DNO);
```

The result Form, T, is shown in Figure 10(c).

If the PREVAIL clause is specified, an instance will be produced even when the field values do not match. Thus the statement

```
T = GRAFT (DSUP ONTO DEPT BEFORE PJNO
           WHERE DEPT.DNO EQ DSUP.DNO,
           ELSE DEPT PREVAIL);
```

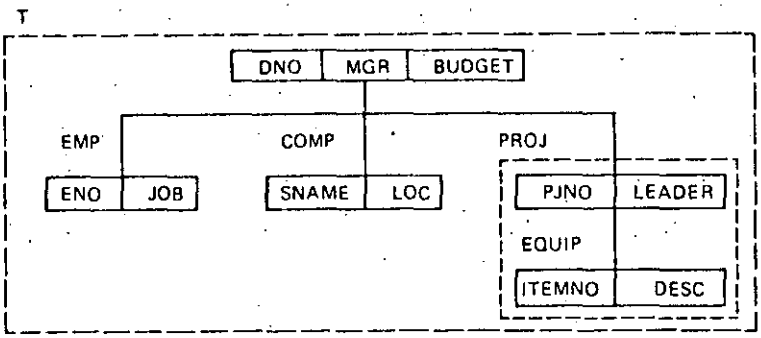
will produce a resulting Form like that shown in Figure 10(d).

Although many issues are not discussed here, the above discussion is sufficient to illustrate the essentials of the language. The operators not discussed are self-explanatory, except perhaps for CONSOLIDATE, which is the inverse of SLICE and is used to construct a tree from a flat file. For further understanding of the language, the reader is referred to [8, 21].

(a)

(DSUP)		
DNO	(COMP)	
	SNAME	LOC
D1	ACME	SF
	EMCO	MV
D5	DELTA	LA
	OXA	NY

(b)



(c)

(T)										
DNO	MGR	BUDGET	(EMP)		(COMP)		(PROJ)			
			ENO	JOB	SNAME	LOC	PJNO	LEADER	(EQUIP)	
									ITEMNO	DESC
D1	DOE	40000	19	ENG	ACME	SF	J6	RAE	221	COMPUTER
			41	SEC	EMCO	MV	J8	MEE	46	SCOPE
			52	TECH					317	LASER
			77	ENG			J11	FAR	271	COMPUTER

(d)

(T)										
DNO	MGR	BUDGET	(EMP)		(COMP)		(PROJ)			
			ENO	JOB	SNAME	LOC	PJNO	LEADER	(EQUIP)	
									ITEMO	DESC
D1	DOE	40000	19	ENG	ACME	SF	J6	RAE	221	COMPUTER
			41	SEC	EMCO	MV	J8	MEE	46	SCOPE
			52	TECH					317	LASER
			77	ENG			J11	FAR	271	COMPUTER
D4	SO	20000	60	CHEM	---	---	J9	LA	47	MICROSCOPE

Fig. 10. Examples of GRAFT operations

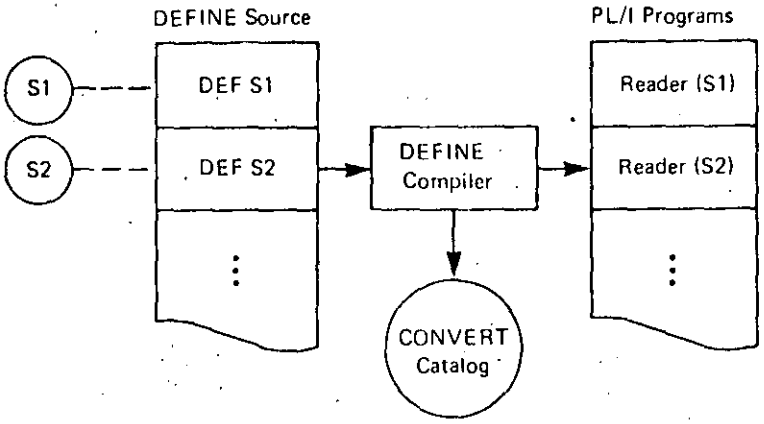


Fig. 11. Reader (DEFINE) compile phase

4. EXPRESS ARCHITECTURE

This section describes the architecture for implementing the Read and Restructuring steps. The function of the EXPRESS system is to translate and execute the specification languages DEFINE and CONVERT described in Section 3. EXPRESS can be divided into two basic phases: a *compile-time* phase and a *run-time* phase. Before going into the detailed architecture, we shall give an overview of what happens in each of these phases for the DEFINE and CONVERT languages. The general architecture of the DEFINE compile-time phase is given in Figure 11.

Corresponding to each source file (e.g. S1), there is a DEFINE description which is compiled by the DEFINE compiler into a customized PL/1 program (e.g. Reader(S1)). In addition, the DEFINE compiler puts information into a file called the CONVERT catalog. This catalog is used to hold common tables and statistics (e.g. descriptions of the internal forms) used by various EXPRESS compile-time and run-time components.

The general architecture of the CONVERT compile-time system is shown in Figure 12. Here, for each CONVERT operation in the CONVERT source program, the CONVERT compiler generates a customized PL/1 procedure called a CONVERT *Operation Procedure* (COP). Entries are put into the CONVERT catalog (e.g. the structure of the derived Forms). A component referred to as the Analyzer is invoked. The Analyzer reads the CONVERT catalog and generates an *Execution Schedule*, which is a file containing job control language statements necessary to execute the COPs.

The DEFINE *run-time* phase is pictured in Figure 13. At this time each Reader program, generated at compile time, is executed in order to transform one source file to its corresponding internal form representation. During Reader execution, each instance of a type tree (i.e. section) described by DEFINE is mapped to a (variable length) record in the internal form. These records represent the hierarchy with byte offset pointers. The COPs use these pointers to efficiently process the sections. The Reader procedure also collects statistics on the source file and puts them into the CONVERT catalog and produces an error report for erroneous data.

The CONVERT *run-time* phase (shown in Figure 14) consists of executing the COPs

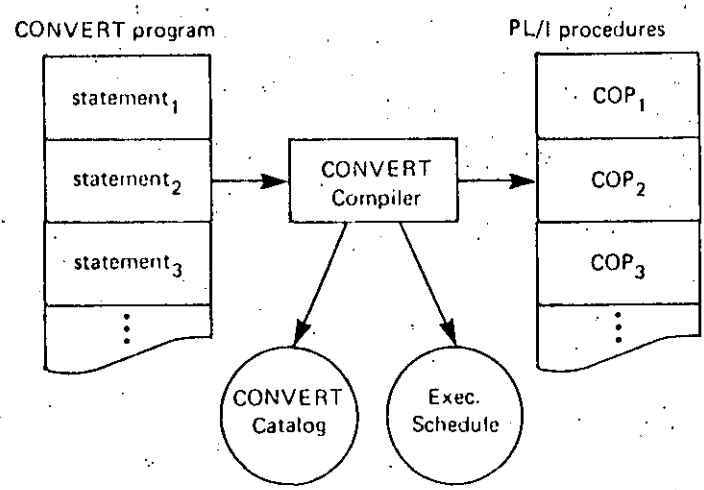


Fig. 12. Restructurer (CONVERT) compile-time phase

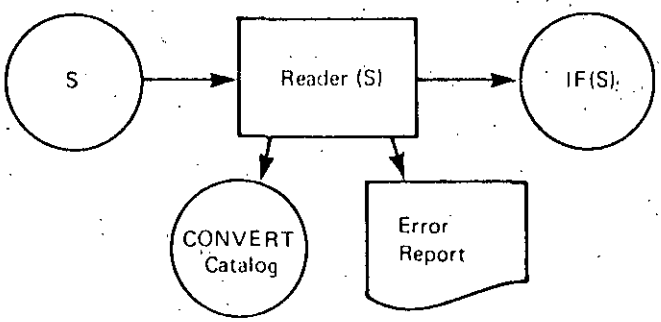


Fig. 13. Reader (DEFINE) run-time phase

generated by the CONVERT compiler. Each COP sequentially reads its input files (in internal form) a section at a time and performs the restructuring to construct one output section. This procedure is repeated until the inputs are exhausted. To allow several COPs to run concurrently, we introduce the notion of "pipelining." We define a CONVERT operator as *pipeable* if it has the ability to process an input section independent of other sections to be processed. A *pipeline* is a series of pipeable operators. The effect of pipelining can be illustrated by considering the simple CONVERT program:

```
FORM S(A,B(C,D(E)));
I = SLICE (A,C,E FROM S);
T = SELECT (ALL FROM I WHERE A LT 10);
```

This can be depicted as a *Process Graph* (Figure 15).

The operators SLICE and SELECT define a pipeline. Every time SLICE puts out a section to I, SELECT can be invoked to process I. The resulting target file T would be the same as if SLICE generated the entire file I before the SELECT was initiated.

Subsections 4.1–4.4 describe in detail the compile-time and run-time phases of the Read step (DEFINE) and the Restructuring step (CONVERT).

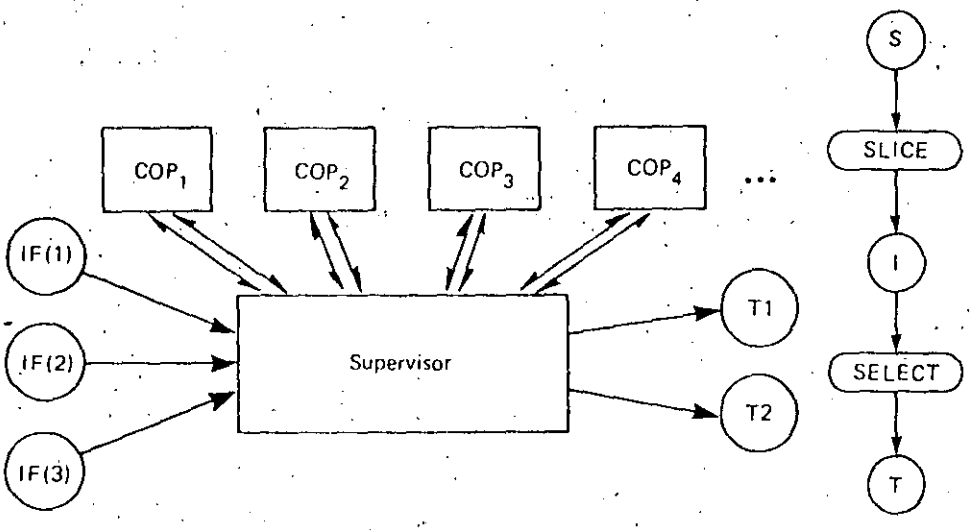


Fig. 14. (above) Restructurer (CONVERT) run-time phase
Fig. 15. (Right) Example of a process graph

4.1 Reader Architecture

The Reader's main task is to access the data in their source format, check them, and construct the internal representation of each section. To understand the Reader, it is convenient first to explain the behavior of the (generated) Reader program during execution. We then outline the major concepts of the DEFINE compiler, which generates the Reader. We also discuss the problem of handling detected errors in such a way that users can manage the sometimes voluminous error reports that "dirty" data can produce.

As discussed in Section 2, the reading process is broken into two parts. One part, called the Physical Reader, is responsible for accessing data by using the access methods of the source system. The other part, called the Generated Reader, is responsible for all other parts of the reading process.

Figure 16 outlines the data and control flow between the Generated and Physical Readers during execution. The Generated Reader is the main program. It calls the Physical Reader to deliver to its buffer (INBUF) the "next" item instance in the byte stream that was described in DEFINE. It is then the responsibility of the Physical Reader to materialize this data request. In practice this is not a difficult task; it generally involves returning the "next" bytes in an internal buffer and refilling this buffer when necessary with a call to the source system access methods.

The Physical Reader is system dependent. Because of this, it has the knowledge of appropriate calling conventions within the source system to obtain more data. Also, the Physical Reader can remove from the data stream any information which is inappropriate for the user view as described by DEFINE. For example, volume header and trailer records, checkpoint records occurring at random intervals, and record marks inserted by the system when spanning variable length records would all be burdensome to describe and might require too great a level of expertise from the DEFINE user.

Generally there are eight entry points in the Physical Reader. These subroutines provide the ability to return data that are "next" on the basis of length, delimiter,

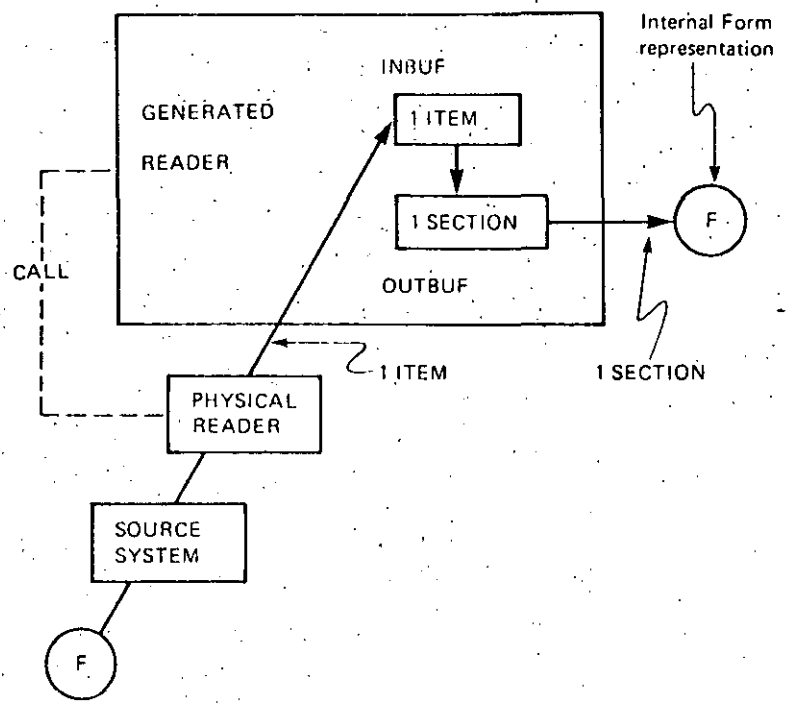


Fig. 16. Generated and Physical Reader

or pattern criteria. Look-ahead facilities are provided (for cases such as record type in column 80), and Boolean valued functions exist for indicating an end of (physical) record or end of file condition. The DEFINE compiler determines during program generation which of these entry points to use on the basis of the DEFINE description.

To move the Reader to a new environment or to access unusual data, only the Physical Reader needs reprogramming. This is generally at most a few hundred statements of code.

Once one data item is delivered to the Generated Reader's buffer, a template (PL/1 based variable) generated at compile time is laid over it. Statements are then executed to remove delimiters, check validity, and translate the data according to any TRANSLATE clauses. A user escape is also possible here. An assignment statement then moves the data (possibly with type conversion) to the appropriate point in the internal representation.

If during the checking process an error is detected, a message is written to the error file. We defer the details of error handling until later in this section.

The DEFINE compiler has responsibility for generating the Reader. From the discussion above, it can be seen that this involves such problems as:

(a) Deciding if an item or group exists conditionally or unconditionally in the data stream.

(b) Generating the proper control structure for conditional or repeating groups or items.

(c) Discovering how to isolate an item and generating the appropriate call to the Physical Reader.

(d) Generating code to check and translate each item.

(e) Generating error message statements.

(f) Generating declarations for the internal structures and the byte stream (coming from the Physical Reader).

In addition code is generated to collect statistics needed by the analyzer.

To illustrate the kind of code generated for an item, consider the description

```
ITEM I1: CHAR(PIC IS 'AA9(3)');;
```

The compiler would determine from the picture that the item has an implied length of five characters. It would generate a call to the Physical Reader to return the next five characters to the Generated Reader input buffer. Having generated a PL/1 BASED declaration for I1, the compiler would generate code to check conformance with the associated picture. This involves scanning the picture to find substrings which can be treated as a group (like AA) and using the PL/1 SUBSTR (substring) function to isolate portions of the item. The PL/1 VERIFY function is then used to check appropriate character sets for each substring. Failure of any check would trigger an error message. Finally, an assignment statement is generated to move the data to an element having a similar name within the internal form hierarchic record that is being built.

We now turn to the generation of code for optional and repeating structures. Figure 17 gives a diagram of hierarchical subroutine calls of the DEFINE compiler. This is a simplified view of the entire process, but is sufficient to illustrate the major points. Each node represents a subroutine (or collection of subroutines) which accomplishes the function described. Each arc from a higher level node to a lower level node represents a call to the lower level routine, which will return when all of its steps are finished.

The basic strategy is to walk the model of the hierarchic record, as given in the DEFINE description, generating the appropriate code and declarations for each item or group encountered. Thus the parsing portion of the compiler builds tables representing the hierarchic record structure together with all attributes that were given. The generation portion of the compiler then walks these tables. As is shown in Figure 17, for each item or group in the DEFINE description, a master routine classifies it into one of four categories—conditionally appearing item, unconditional item, conditionally appearing group, or unconditional group. If the component (i.e. item or group) is conditional, an IF...THEN statement is generated which (at run time) evaluates the associated condition. Thus the code to parse the component will be executed only when the component is present. The Reader generator then treats the component as if it appeared unconditionally and calls lower level routines (PARSE_ITEM or PARSE_GROUP) which generate code to parse the item or group. If a group repeats, the characteristics of its repetition (delimiters, occurs depending on, etc.) are examined, and the appropriate loop invariant is selected for use in a DO WHILE iteration. The group is then treated as nonrepeating and a call is made to parse a single instance. Having reduced the more complex cases to generating code to parse one instance of a group, the master routine is called recursively for each group component immediately contained within the group. Thus the generation of nested iteration and conditional statements (reflecting repeating groups within repeating groups, etc.) is a consequence

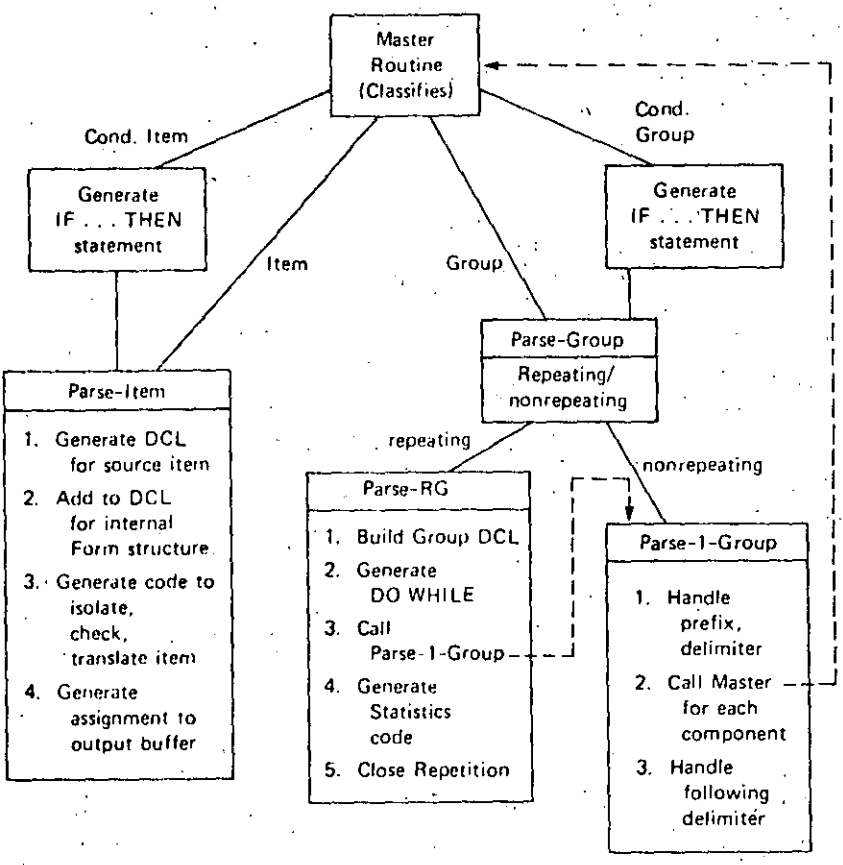
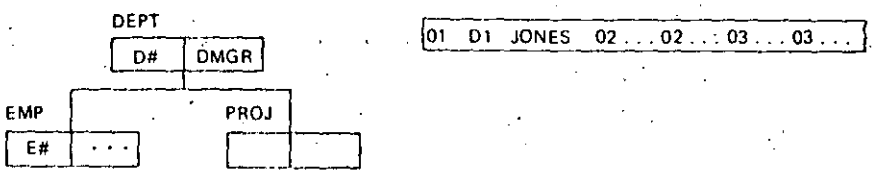


Fig. 17. Subroutine calls of the DEFINE compiler

of the recursion and subroutine calling sequence structure. The generated code, however, is not recursive.

As an example of the generated DO WHILE structure, consider the DEFINE clause associated with an IMS structure like that shown in Figure 18. Here each instance of a subtree is optional, and its existence is indicated by a "segment code" in the first byte of the group. The generated control structure has the form shown in Figure 19. The function NCHARLEN is a Physical Reader function which "looks ahead" 8 bits. Given that a match is found, code is executed to allocate a new subtree in the internal buffer. A count of the number of loop iterations is also kept, as this can be used upon loop exit to compare with a running maximum of the maximum number of children under a given parent. This is one of the statistics needed by the Analyzer.

We now turn to the problem of data errors encountered by the Reader. It is generally recognized that data errors will frequently occur, especially when trying to coordinate data from diverse sources or installations. We use the term "error" to refer to a failure to conform to a validity check given in DEFINE. Thus "errors" may result from a new enterprise-wide definition of data, as well as other reasons.




```
GROUP DEPT :
        OCCURS FROM 1 TIMES;
                FOLLOWED BY EOF;
                PRECEDED BY HEX '01';
          .
          .
          .
        GROUP EMP:
                OCCURS FROM 0 TIMES;
                PRECEDED BY HEX '02';
          .
          .
          .
        END EMP;
        GROUP PROJ:
                OCCURS FROM 0 TIMES;
                PRECEDED BY HEX '03';
          .
          .
          .
        END PROJ;
END DEPT;
```

Fig. 18. An IMS structure

In any case the detection, correction, and management of error reports is an integral part of data translation.

As discussed earlier in Section 3.1, DEFINE has a broad capability for error detections. Thus much of the error checking and reporting is done in the Reader, though checks that extend across hierarchical records would normally be handled in CONVERT. Extended picture codes, range and value checks, and user-provided procedures can all be used to detect errors. The TRANSLATE clause and user procedures can be used to correct errors. In any case when an error is detected, an error message is written to an error report file. This error report file is in internal form with the structure shown in Figure 20. The advantage of this approach is that CONVERT can be used to process the error files as well as to specify the restructuring process. Frequently it is necessary to select, sort, or gather statistics about errors. It is convenient to be able to distribute error reports by originating source (frequently part of the key) or to examine errors of a certain type on a certain item to detect possible causes of error. Since CONVERT offers extensive data manipulation capabilities, it seems natural to use it for the error files as well.

There are other kinds of errors which are more serious in terms of proper Reader execution. For example, if the number of repetitions of a repeating group is given by a count item, and that count item is incorrect, then the Reader program can get "out of phase" relative to the data stream. Whether this will be a serious prob-

```
DEPT_COUNT = 0;
DO WHILE (^EOF);
   DEPT_COUNT = DEPT_COUNT + 1;
      .
      .   code for processing D# and DMGR
      .
      EMP_COUNT = 0;
      DO WHILE (NCHARLEN(8) = '00000010'B);
         EMP_COUNT = EMP_COUNT + 1;
            .
            .   code for EMP subtree
            .
      END;
            .
            .   statistics code
            .
      PROJ_COUNT = 0;
      DO WHILE (NCHARLEN(8) = '00000011'B);
         PROJ_COUNT = PROJ_COUNT + 1;
            .
            .   code for PROJ subtree
            .
      END;
            .
            .   statistics code
            .
   /* WRITE ONE SECTION IN INTERNAL FORM */
      .
      .
      .
END;
/* WRITE FINAL STATISTICS REPORT */
   .
   .
   .
END;   /* of Generated READER */
```

Fig. 19. Generated iteration structure for IMS

lem in practice will not be known until EXPRESS is used in a "live" environment. If the program which processed the file in the source environment relied upon this count, then it too would have had problems; thus it seems the count item should be reliable. But that is not guaranteed. The approach adopted at present is for the Reader to try to continue processing until a maximum error count is exceeded, but a more cautious approach to certain types of errors may be necessary.

Currently the DEFINE compiler does not support all legal descriptions in the DEFINE language. However, common CONOL and IMS structures are supported, and several test cases have been run. It has been found that the generated PL/1 program to read the file contains at least ten times as many statements as the DEFINE description. These figures vary somewhat depending on the amount and complexity of the validity checks and on whether the data is optional. Nonethe-

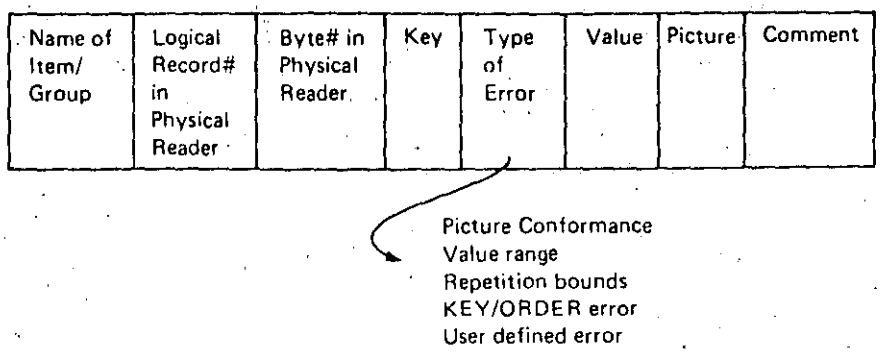


Fig. 20. Reader error report file

less, it is at least a guideline concerning the level of effort necessary to accomplish the first stage of the data translation process. The generated code is easy to read. It is pretty-printed, is commented, preserves user names, and uses only IF... THEN...ELSE, DO...WHILE, and sequential control structures. Generating clear code was considered important since users may want to augment the generated program with extra statements prior to running the Reader.

4.2 The CONVERT Compiler

The CONVERT compiler produces COPs and an execution schedule by using the CONVERT program and information from the CONVERT Catalog. There are three major components in the compiler: (1) a Parser and Semantic Checker, (2) a Program Generator, and (3) an Analyzer. The compiler architecture is shown in Figure 21. In this section we discuss the first two components, alluding to the Analyzer only when it is necessary to make the concepts clear. The Analyzer is discussed in detail in Section 4.4.

The Parser will first accept the Reader-produced Form specification statement and the user-written CONVERT program. An example of a Form specification statement is shown in Figure 22(a). In essence it is a description of the Form schema shown in Figure 22(b) or the hierarchy graph shown in Figure 22(c) with additional information on KEYs, ordering of instances, the maximum section length, and occurrence frequencies. The Parser encodes this information in the CONVERT Catalog to be used by the Analyzer and the Program Generator later. An example of a user-written CONVERT program is shown in the Appendix. The Parser will unnest any nested CONVERT statements, make sure that all the CONVERT statements are syntactically correct, and encode these statements into internal table entries.

Based on the CONVERT Catalog and the encoded CONVERT statements, the Semantic Checker now performs its functions. They include the following:

(1) Target Form derivation—For each CONVERT statement, the target Form specification is derived according to the CONVERT operation involved. For instance, given the DEPT Form described in Figure 22(a),

```
T1 = SELECT (DNO,EMP FROM DEPT)
```

will cause the derivation of T1's entry in the CONVERT Catalog, namely, the en-

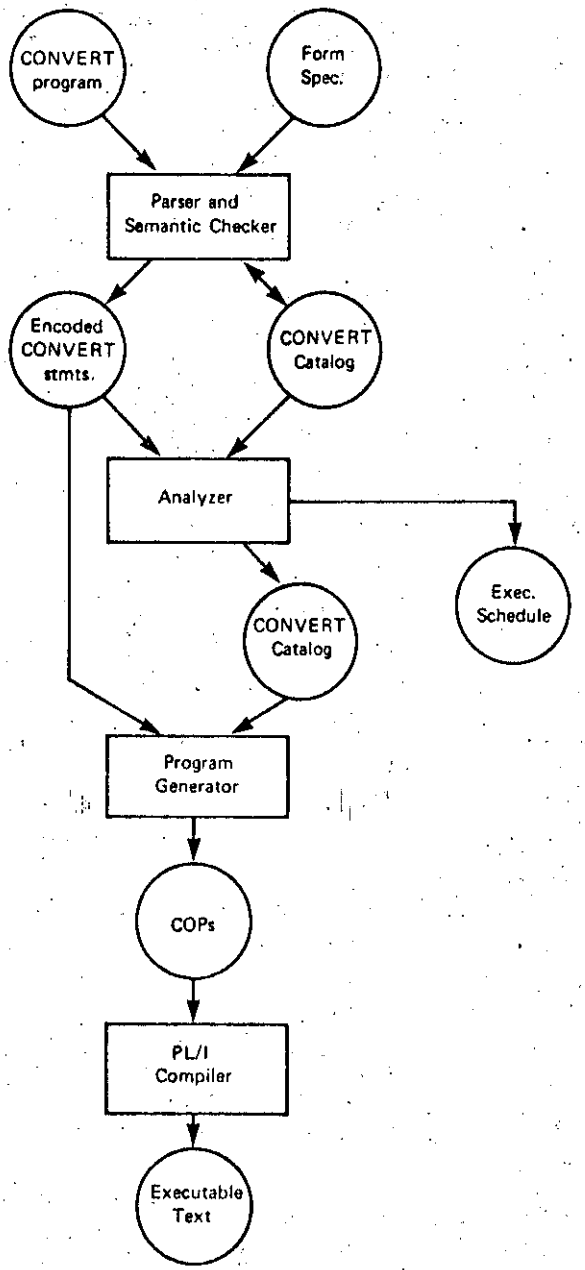


Fig. 21. The CONVERT compiler architecture

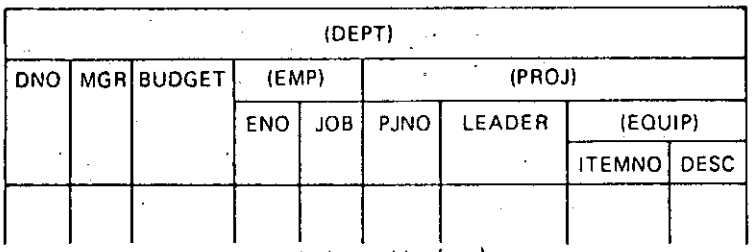
coded representation of

```
FORM T1(DNO CH(3),
        EMP(ENO CH(3), JOB CH(2))(OCCURS 30,7 TIMES)):
        KEY IS (DNO), ORDERED ON (DNO), OCCURS 8 TIMES,
        SECLEN IS 200;
```

(a). A Form Specification Statement

```
FORM DEPT ( DNO  CH(3),
            MGR  CH(3),
            BUDGET BIN,
            EMP  ( ENO  CH(3),
                   JOB  CH(2),
                   (OCCURS 30,7 TIMES),
            PROJ ( PJNO CH(2),
                   LEADER CH(3),
                   EQUIP  ( ITEMNO BIN,
                            DESC CH(9) )
                            (OCCURS 25,2 TIMES) )
                   (OCCURS 15,3 TIMES,
                    KEY IS (PJNO), ORDERED ON (PJNO) ) ):
            KEY IS (DNO), ORDERED ON (DNO),
            OCCURS 8 TIMES, SECLEN IS 200;
```

(b). A Form Schema



Note: A repeating group is denoted by ( ).

(c). A Hierarchy Graph for DEPT

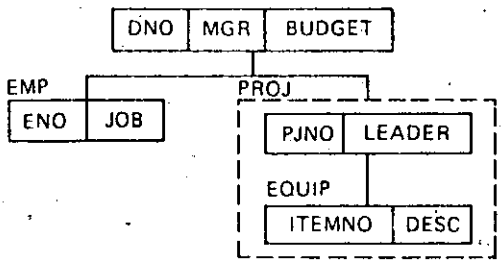


Fig. 22. Description of a Form

(2) Consistency check—For instance, even though the user is not required to provide any Form specification statements, he is not discouraged from providing them in his CONVERT program. If he has provided a Form specification statement, the Semantic Checker makes sure that the derived Form specification is isomorphic to the user specified one.

(3) Legality check—For instance, the components specified for a SLICE operation must be on a single hierarchical path. For example, SLICE (DNO,ENO,PJNO FROM DEPT) would be illegal.

Thus, at the end of the semantic Checking phase, the encoded CONVERT statements have passed the legality and consistency tests, and the CONVERT Catalog is almost complete. In the next phase, the Analyzer performs an "order" analysis

over all the CONVERT and Form specification statements. It may impose SORT operations to allow pipelined operation and thus change the CONVERT Catalog.

Nearly all CONVERT operators can be pipelined. It is obvious that SLICE, SELECT, MERGE, and aggregate functions (SUM, COUNT, etc.) can all be placed in pipelines. It is also easy to see that when the input is in sorted order, ELIMDUP is pipeable. On the other extreme, SORT requires the entire input file to be available before it can produce any output. Thus a SORT operation will break a pipeline. After investigating several implementation strategies for CONSOLIDATE and GRAFT, we concluded that when certain sort order requirements are enforced, they too can be placed in pipelines. In the case of CONSOLIDATE we require the input to be sorted according to the KEY fields of the target. In the case of GRAFT we require matching fields at the root level of both input Forms, sorted in the same order, with the match fields containing the KEY in at least one input Form.

When the CONVERT Catalog is complete, we are ready for program generation. For each unnested CONVERT statement, we generate a PL/1 procedure to carry out the necessary processing. The detailed procedure generated for each of the CONVERT statements could be quite complicated. The complexity arises for the following reasons:

(1) Data structures could be complex. The generated program must be able to traverse and build nested repeating groups of many levels, each having a variable number of instances.

(2) Many decisions must be embodied in the generated program. The control flow for the scratch algorithm and CASE assignment involve many decisions. A GRAFT procedure must be able to decide when one or the other or both input files must be advanced.

(3) The output structure of a CONVERT procedure can be drastically different from its input. Programs which support SLICE, CONSOLIDATE, GRAFT, and SELECT extract instances of subtrees or portions of nodes in building the output section. Even when the output structure remains the same as the input, the scratch algorithm can produce an output section that is quite different from the input.

(4) Data volume is expected to be large. Inefficient algorithms could severely degrade the running time.

Although we have no control over the complexity of the processing required, we do have control over the quality of the generated program. Modularity, efficiency, and readability are certainly some desirable qualities for the programs. Readability has been discussed in Section 4.1. We now discuss modularity and efficiency in turn.

In our context modularity is the ability to generate or modify a CONVERT procedure independent of the process graph. This allows users to modify the generated procedures or add their own without being concerned with other procedures. It also allows the compiler for each CONVERT operator to be developed independently. Modularity is achieved by having each procedure get its inputs and produce its output through a set of I/O interface routines. The procedure is unaware of whether its inputs are produced by a previous procedure in a pipeline or from a real file.

In order to produce efficient programs, we designed a "best" program model for each of the different CONVERT operators. For each individual CONVERT statement,

we then take the program model and fit it with compile time decisions to come up with a customized program.

To illustrate the optimizing strategies, let us take SELECT as an example. Briefly the function of a SELECT operation is to compose target sections from the source component instances which have survived the scratch algorithm. The goal is to generate a program that will perform necessary functions with minimum scanning and movement of data. The following are some of the decisions that the Program Generator can make:

(1) If the selection criteria involves only top level fields and the target structure is the same as the source, then a source section is "passed" to the target in its entirety when the selection criteria are met. In "passing" the data, instead of actually moving data from source to target buffer, the generated program simply tells the I/O interface to swap the buffers.

(2) When the above cannot be done, the component instances are moved to the target area as the scratch algorithm is performed. This requires only one pass over the data. (An alternative would be to mark the failed component instances on the first pass and move the surviving instances to the target area on the second pass.)

(3) Selection criteria are decomposed by the Program Generator into a set of conditions and recorded in a table. Each of these conditions is associated with an object of scratch. (An object of scratch is the instance subtree to be scratched if a field comparison test fails.) During the course of generating code to move data, this table is consulted so that relevent tests on these conditions can be placed at strategic points (prior to moving of the respective object of scratch).

(4) Even though both fields in a field comparison must be on a single hierarchical path or branch, the collection of fields specified in the selection criteria may span many branches. If any one of these branches did not survive the scratch algorithm at level 2 (i.e. the level beneath the root level), propagation of the scratch to the root level will cause the elimination of the section altogether. Thus a section will be scratched as soon as any one of these branches fails to survive.

(5) Codes generated to move a repeating group instance are dependent on whether the target group structure is the same as the source group structure. If they are the same, the instance will be moved as a group. Otherwise, components within the group are moved individually.

Thus, for each user-written CONVERT statement, we are able to produce a customized program. Take the DEPT Form (Figure 22), for example.

```
T1 = SELECT (FROM DEPT WHERE BUDGET GT '100');
```

will produce the following program:

```
/* PROCESS LOOP FOR T1 */
   DO WHILE (not end of file);
      CALL GET (DEPT);
      IF BUDGET > '100'
         THEN CALL BUFFER_SWAP (T1,DEPT);
   END; /* END OF PROCESS LOOP */
```

On the other hand,

```
T2 = SELECT (DNO,EMP,PROJ FROM DEPT
             WHERE JOB GT '07' AND PJNO GT 'J3'
             AND DNO LT 'D6');
```

will produce a drastically different program:

```
   /* PROCESS LOOP FOR T2*/
GETINPUT:
   DO WHILE (not end of file);
         CALL GET (DEPT);
         IF DNO < 'D6'
            THEN DO; T2.DNO = DEPT.DNO;
                     /* PROCESS EMP GROUP */
                     NEMP = 0; */ COUNT EMP IN TARGET */
                     Set cursor to first DEPT.EMP instance;
                     DO WHILE (not end of DEPT.EMP);
                        IF JOB > '07'
                           THEN DO; T2.EMP = DEPT.EMP;
                                    NEMP = NEMP + 1;
                                END;
                        Set cursor to next DEPT.EMP instance;
                     END;
                     IF NEMP = 0 THEN GO TO GETINPUT;
                     /* PROCESS PROJ GROUP */
                     NPROJ = 0; /* COUNT PROJ IN TARGET */
                     Set cursor to first DEPT.PROJ instance;
                     DO WHILE (not end of DEPT.PROJ);
                        IF PJNO > 'J3'
                           THEN DO; T2.PROJ = DEPT.PROJ;
                                    NPROJ = NPROJ + 1;
                                END;
                        Set cursor to next DEPT.PROJ;
                     END;
                     IF NPROJ = 0 THEN GO TO GETINPUT;
                     /* PUT OUT TARGET SECTION */
                     CALL PUT (T2);
                  END; /* END OF THEN GROUP FOR DNO < 'D6' */
   END; /* END OF PROCESS LOOP */
```

Note that "GO TO GETINPUT" is generated. In cases where there are more than two branches in the hierarchy, this gives a more readable program than nested IF...THEN...ELSE statements.

No experiment has been done to compare generated and handwritten programs. However, we have studied the generated programs produced for many diverse examples of CONVERT statements and believe them to be similar to manually written programs that accomplish the same operations.

In summary the CONVERT compiler will compile each CONVERT statement into a PL/1 program which is modular, readable, and efficient. The complexity of the generated program is dependent on the CONVERT statement specified as well as the data structures involved. To give an indication of the power of CONVERT, Table I summarizes the number of total, declarative, and imperative PL/1 statements required for each of the seventy CONVERT statements that we have compiled and tested. Of these compilations, the total number of PL/1 statements for a CONVERT statement ranges from 44 to 216, while the number of imperative statements ranges from 18 to 175. When a complicated process can be specified in a simple yet powerful language like CONVERT, the benefit obtained from the ease of debugging may be far more important than code expansion ratios.

Table I. Summary of expansion from a CONVERT statement to PL/1 statements

CONVERT Stmt. Type		PL/I Statements TOTAL	DCL	Imperative	# Programs Compiled
SELECT	smallest	44	26	18	
	largest	134	41	93	30
GRAFT	smallest	77	30	47	
	largest	216	41	175	15
SLICE	smallest	55	24	31	
	largest	80	28	52	10
CONSOLIDATE	smallest	62	28	34	
	largest	166	31	135	10
ELIMDUP		48	19	29	3
MERGE		45	24	21	2

4.3 The CONVERT Run-Time System

The function of the CONVERT run-time system is to provide the framework for executing the COP's generated by the CONVERT compiler. First, however, let us examine the issues involved. A CONVERT program can be viewed as a "process graph." For example, Figure 23 gives the process graph for the example given in the Appendix. As mentioned in Section 3.2, the CONVERT user views each CONVERT operation as restructuring one or more input files to produce one output file. A simple implementation would be to execute each operation as a separate job step according to the precedence constraints of the process graph. However, from an efficiency standpoint this method is unsatisfactory because it requires the creation of an intermediate file at each step. Therefore we introduced the concept of pipelining.

To implement pipelining, the pipeable operators can be viewed as a set of cooperating sequential processes. Each intermediate file in this set becomes a shared buffer in main memory during execution. A simple example of pipelining is shown in Figure 24. Each time the SLICE process produces a section in its output buffer I, its execution is suspended and execution of the SELECT process is started. Similarly, when the SELECT has completed processing of the section in I, it is suspended and the SLICE process is resumed, etc. This cooperation is possible because all the CONVERT operations except SORT are pipeable. We refer to a set of these cooperating processes as a *schedulable unit*. It will be shown later that generally not all COP's for a process graph can be included in one schedulable unit. Although the previous example is trivial, in general the configuration of data flow in a schedulable unit may be a complex graph, since a file may be an input to any number of operations and some operations (MERGE,GRAFT) require multiple inputs. The main function of the CONVERT run-time system is to synchronize the data and control flow among the different processes in the schedulable unit.

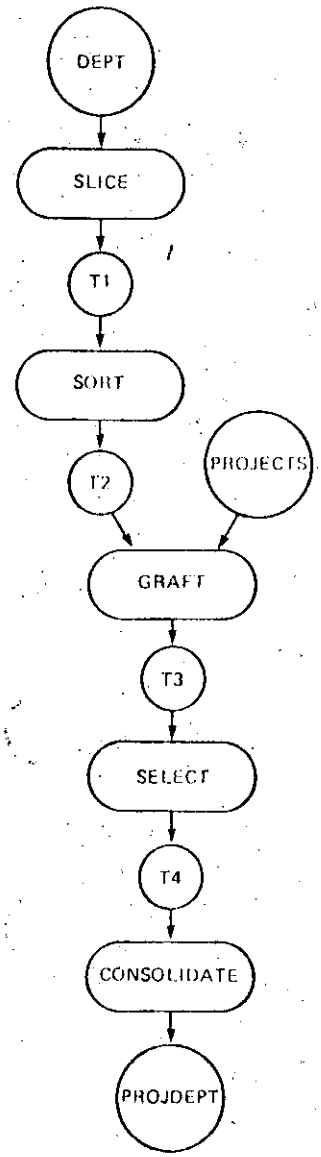


Fig. 23. Process graph for the example given in the Appendix

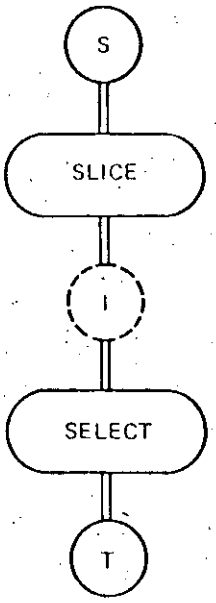


Fig. 24. Example of pipelining

The basic CONVERT run-time system architecture was shown in Figure 14. Each COP becomes a CONVERT operation process. They communicate via a shared reentrant supervisor. The COPs interact with the supervisor via a set of special I/O commands. These commands are similar to those normally found in sequential access methods (e.g. OPEN, READ, WRITE, etc.) except for some special protocols for accommodating buffer sharing and buffer swapping. If the supervisor is called and the request can be satisfied (e.g. the input buffer has been filled), then control is returned to the calling COP with return status. Otherwise, the request

is queued, the calling COP suspended, and another COP whose request has been completed is resumed. (At least one COP will always be ready to run; see Section 4.4.) If the I/O request is to a real file, the supervisor initiates a real I/O request. This architecture allows COPs to be generated independent of the process graph configuration, and it allows execution of the COPs to be event driven. A "user written" COP can easily be incorporated into the system as one node in the process graph.

Efficiency Considerations. A primary consideration in the design of the CONVERT run-time system is the storage and data transfer of sections (records) in main memory. Since a section is one instance of a complete hierarchy, it may be thousands of bytes long. Thus it is beneficial to minimize the number of section buffers and the movement of sections within main memory. Two strategies have been implemented to achieve these goals: shared buffer processing and buffer swapping. With shared buffer processing the COPs process data directly in the shared buffers instead of a local work area. Thus, when a COP read command has been completed, the COP is given access to the shared input buffer. Since many COPs may process the same input buffer, the supervisor issues a "read-share" lock on the input buffer for each COP which is processing its contents. Thus the current contents of the buffer are preserved until all read-share locks have been released. For output processing each COP issues a request to the supervisor for exclusive access to the output buffer. Control is returned when all read-share locks on the output buffer (if any) have been released. Subsequently the COP builds the output section directly in the output buffer and then calls the supervisor committing it to the other cooperating processes.

The buffer swapping strategy involves swapping an input buffer with the output buffer in cases where the contents of the input buffer are to be moved without change to the output buffer. One example of this is the MERGE operation. To swap buffers the supervisor swaps the input and output buffers by exchanging the buffer pointers in the relevant I/O control blocks. Buffer swapping is subject to the constraints that (a) the input buffer is not being shared by another COP and (b) the input and output buffers are the same size.

Another consideration is the overhead of process switching (suspending and resuming). Since a cooperating process will be suspended nearly every time it needs to read or write a section, a large number of process switches will occur. Thus, if the switching time is large compared with the mean time between switches, the throughput of the system may be severely decreased. By implementing our own queueing and dispatching mechanisms, the CONVERT run-time system can switch processes with overhead comparable to a procedure call.

4.4 The Analyzer

Earlier we stated that the Analyzer performs a global analysis on the CONVERT program. More specifically the Analyzer (a) performs order analysis on the CONVERT operations, (b) analyzes for deadlock, (c) partitions the process graph into a family of schedulable units, (d) computes buffer and secondary storage allocations, and (e) generates an execution schedule for executing the restructuring step. Each of these functions is discussed below.

Order Analysis. In Section 4.2 it was stated that the input files for certain operations (i.e. GRAFT, ELIMDUP, CONSOLIDATE) are expected to be in proper order. However, it is often quite burdensome for a user to keep track of the order of files in the process graph in order to determine if he must code SORT operations. One of the advanced functions of the Analyzer is to relieve the user of this task. Given the ordering of the source files (including "random"), the Analyzer begins by examining the operations which require the source files as inputs. For each order dependent operation, its ordering requirements are compared with the given order of the inputs. If the given orderings are not *compatible* with the ordering requirements of the operator, the necessary SORT operations are inserted to achieve compatibility. Next, the operation is analyzed to compute the output order. This process is transitively applied throughout the process graph until the target files are reached. Much could be said here concerning optimization techniques for minimizing the number of sorts required, but we defer this to another paper. To illustrate this process, consider the following example.

```
FORM DEPT(DNO,PROJ(PJNO,PD)):
        ORDERED BY (DNO), KEY IS (DNO);
FORM PROJECT(PJNO,PCOST):
        ORDERED BY (PJNO), KEY IS (PJNO);
T1 = SLICE (DNO,PJNO,PD FROM DEPT);
T2 = GRAFT (T1 ONTO PROJECT
                WHERE T1.PJNO EQ PROJECT.PJNO);
```

SLICE is not an order dependent operation, but it is an order preserving operation if the order fields along the hierarchical path are selected. In this case DNO is the only one, and therefore T1 is marked as being ordered by DNO. In GRAFT we note that the match field is PJNO; thus both inputs must be ordered on PJNO in the same direction. It is ascertained that T1 must be sorted on PJNO since the given order is on DNO. The Analyzer updates the CONVERT catalog to reflect the equivalent of

```
T1 = SORT (SLICE (DNO,PJNO,PD FROM DEPT) BY PJNO);
```

Deadlock Analysis. An interesting situation which can arise is that of deadlock. In this section we describe the deadlock problem in the context of EXPRESS, outline two solutions, and discuss the rationale for the solution we chose to implement in the first version of our system.

As an illustration of how deadlock can occur, consider the simple three-operation program in Figure 25. First, SELECT reads in the first section, filling the DEPT buffer. The test (BUDG > 10) fails and SELECT attempts to read the second section but is suspended because SLICE has not read the first one. Now SLICE reads the DEPT buffer (resulting in a read-share lock on DEPT on behalf of SLICE) and writes "D1 E1" to the T2 buffer, thus suspending SLICE. Next GRAFT attempts to read T1, but T1 is empty; so GRAFT is also suspended. SELECT cannot be resumed because a read-share lock is held on the DEPT buffer by SLICE, and thus all three operations are suspended. A necessary condition for deadlock to occur is the existence of a cycle in the undirected process graph (or subgraph). One sufficient condition for breaking deadlock is that a real file exists on any two

(DEP)		
DNO	BUDGET	(EMP) E#
D1	10	E1 E2
D2	20	E3 E4

```
T1 = SELECT ( DNO, BUDGET FROM DEP
              WHERE BUDGET GT 10 ) ;
T2 = SLICE  ( DNO, ENO FROM DEP ) ;
T3 = GRAFT  ( T2 ONTO T1 WHERE T2.DNO EQ T1.DNO ) ;
```

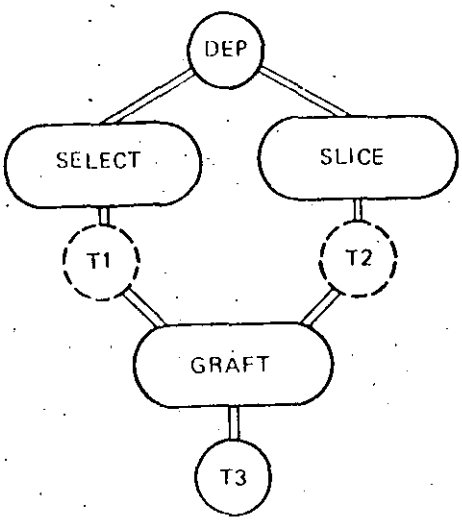


Fig. 25. Example of a program where deadlock can occur

separate directed paths in a cycle. This causes the cycle to be partitioned into separate schedulable units, thus breaking the contention among the COPs. The occurrence of real files in the cycles may result naturally from SORT operations. Alternatively, if a *cycle input file* (i.e. one with two immediate successors in the cycle such as DEPT in Figure 25) is already a real file, deadlock can easily be avoided. In this case the cycle input file may be considered as two files because the CONVERT run-time system can OPEN it twice, thus allowing independent data flow along two directed paths in the cycle. If none of the above conditions hold, then the compile time phase must take some action to avoid the possibility of deadlock at run time.

At least two alternatives are possible. One is to compile more intelligence into the COPs. For example, if the SELECT operation in Figure 25 were to always

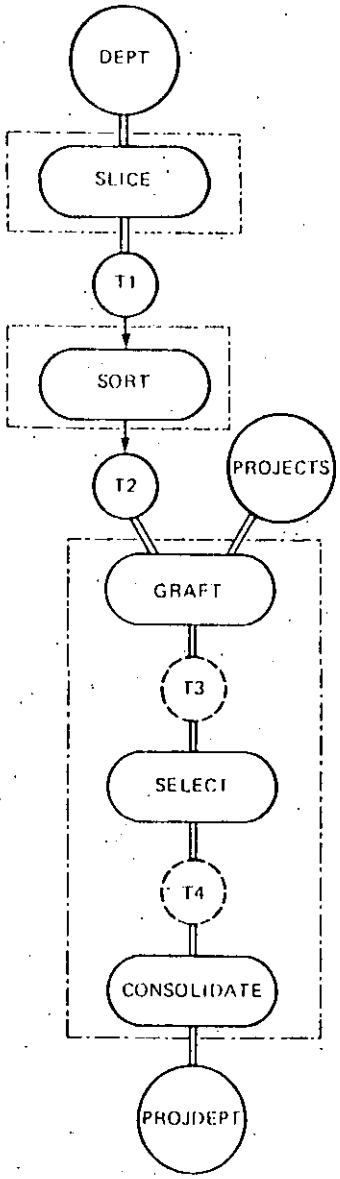


Fig. 26. Example of the partitioning of a process graph

generate an output section in T1 with a flag indicating whether or not it was valid, then the GRAFT operation could, upon completion of a read of T1, test the flag and discard the invalid sections. This would ensure that the data flow in the "pipes" would never be blocked, thus preventing deadlock. Another alternative is for the Analyzer to replace shared buffers with real files to force a schedulable unit partitioning as mentioned earlier. With this approach, at most one additional external file must be introduced per cycle because we can always choose the cycle input file to be a real file.

The trade-off between the two alternatives is I/O verses CPU processing. Additional CPU processing with the flag approach may arise from passing invalid sections down a long pipeline. In the current implementation we have chosen the "real file" approach for several reasons. From a practical standpoint the real file approach is easier to implement. Also it can be shown to be superior in some cases, and it is not expected to cause unreasonable inefficiencies in any case. We expect that deadlock will be extremely rare in practice, and therefore we feel that the additional implementation effort is not warranted at this time. We shall re-evaluate our approach as more understanding about this problem is gained from usage of the EXPRESS package.

Partitioning the Process Graph. Earlier we stated that not all operations in a process graph can, in general, be included in one schedulable unit. Each schedulable unit requires that (a) all input and output files of the schedulable unit be real and (b) all intermediate files within a schedulable unit be shared buffers. Thus, in analyzing a process graph, if an intermediate "file" is required to be real, then it defines a partitioning of the process graph. There are two reasons why an intermediate file must be real; it is either an input or an output of a SORT operation, or it was forced to be real by the Analyzer to break a deadlock cycle. SORT operations are not executed by the CONVERT run-time system but are compiled by the Analyzer as separate job steps. An example of the partitioning is shown in Figure 26.

Space Allocation. The problem here is to compute the buffer lengths and the parameters for secondary storage allocation. Since this analysis is done at compile time, we must guarantee that all buffers will be sufficiently large to hold the maximum size section. At the same time, for the sake of efficient storage utilization our estimates should not be unnecessarily large. It is possible to compute reasonable estimates in all cases except CONSOLIDATE, which is discussed later. The statistics supplied by the Reader for the source files is the basis for computing the space allocation parameters for all other files. As in the order analysis, we begin with the source files and transitively compute the space requirements down the process graph until all target files have been reached.

To illustrate how space parameters are computed, consider the following program:

```
FORM DEPT(
            DNO CH(3),
            MGR CH(6),
            EMP(
                ENO CH(6),
                SAL CH(8),
                TITLE CH(18))(OCCURS 150 TIMES)
            ) (OCCURS 15 TIMES);
T = SLICE (DNO,ENO,TITLE FROM DEPT);
```

In the above example the total number of DEPT sections (i.e. 15) and the total number of EMP subsections (i.e. 150) in the DEPT file have been supplied by the Reader. The section length of T is the sum of the SLICE fields (27 characters). The total number of sections in T is 150, which is the number of subsections of the most descendant subtree (EMP) which contains SLICE fields.

Computation of the output parameters for CONSOLIDATE cannot be performed at compile time. Recall that CONSOLIDATE produces one section from "n" input sections. The difficulty is that the value of "n" is data dependent, and thus a practical upper bound cannot be established for the output buffer length. To handle this case an initial estimate is given (system default or user supplied). At run time the CONSOLIDATE COPs perform overflow tests and call the supervisor to dynamically allocate a larger buffer when necessary. Of course this action may trigger subsequent reallocations for buffers of successor operations in the process graph. When buffers are enlarged dynamically, the CONVERT run-time catalog is updated so that schedulable units which execute later can allocate their buffers correctly.

Execution Schedule File Generation: At this point all the information has been derived for generating the job control language (JCL) statements for executing the restructuring step. One of the basic objectives of this function is to analyze the process graph to produce an execution schedule. For example, in Figure 27 the following execution schedules are possible: ⟨SU1,SU3,SU2,SU4⟩, ⟨SU1,SU2,SU3, SU4⟩, and ⟨SU3,SU1,SU2,SU4⟩. The question is: Which one should be selected? One objective in this decision is to reuse secondary storage space as soon as possible (minimize file idle time). After each SU has executed, all temporary files which are no longer needed are deleted. In the example under discussion it is less efficient to execute SU3 first since I4 will be idle while SU2 is executing. An in-depth discussion of scheduling strategies pertinent to this framework is found in [11].

The Analyzer execution schedule component generates the JCL for all file allocation and deallocation as well as the "execute" commands for invoking the CONVERT run-time system and the system SORT package. These procedures are straightforward and will not be discussed here.

5. CONCLUSIONS

In this paper we have presented a design and implementation of the EXPRESS system for data extraction, processing, and restructuring. The system is based on the approach given in an earlier paper [12].

Two factors have been of prime concern: (1) efficiency and performance and (2) designing a user-friendly system. Performance is achieved by compiling the DEFINE and CONVERT languages into tailored PL/1 programs and by using "pipelining" to minimize I/O activities. At the same time, user friendliness is achieved by having the compilers generate readable code, by having the system handle buffer allocation and the generation of job control statements, and by having built-in "user hooks" in the system.

The compilers for EXPRESS are complex; for example, one CONVERT statement may generate as many as 200 PL/1 statements. The quality of the code, on the other hand, is maintained.

The architecture of the system permits even further possible optimization. The Analyzer component of the CONVERT compiler may be expanded to include other intelligence. For example, one can perform a global analysis on the selection criteria and the matching conditions and then arrange the execution sequence of the operations so that (1) unwanted instances can be filtered out as early as possible (thus

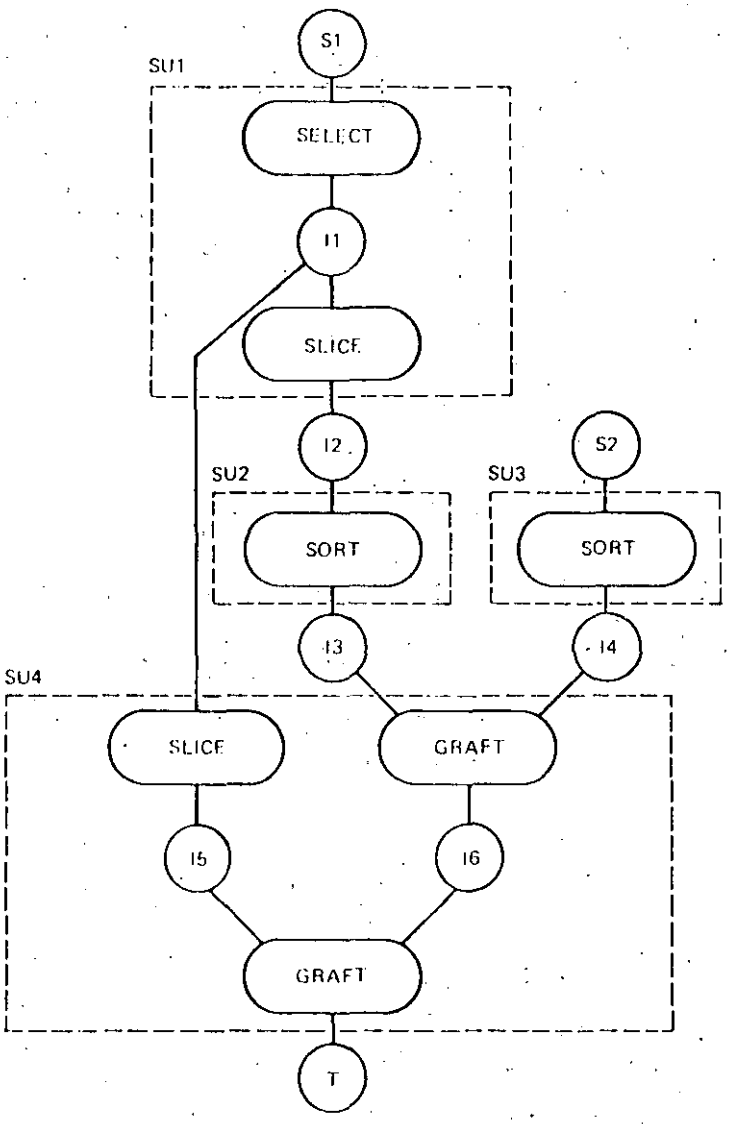


Fig. 27. Example of a process graph where several execution schedules are possible

reducing the number of instances to be processed by subsequent operations) and (2) unwanted fields can be trimmed off as early as possible (thus reducing the width of the Forms to be processed by subsequent operations). One may also move a SORT operation up or down the process graph in order to sort smaller files. Other possibilities involve improving the CONVERT program. For instance, if a SLICE is followed by a SELECT operation which merely permutes the fields, why not specify the desired positioning in the SLICE operation and eliminate the SELECT? Take another example. Is it possible to replace two SELECT operations with a CASE assignment in a single SELECT in order to accomplish the goal? These are but a few of the many improvements that we can conceive. However,

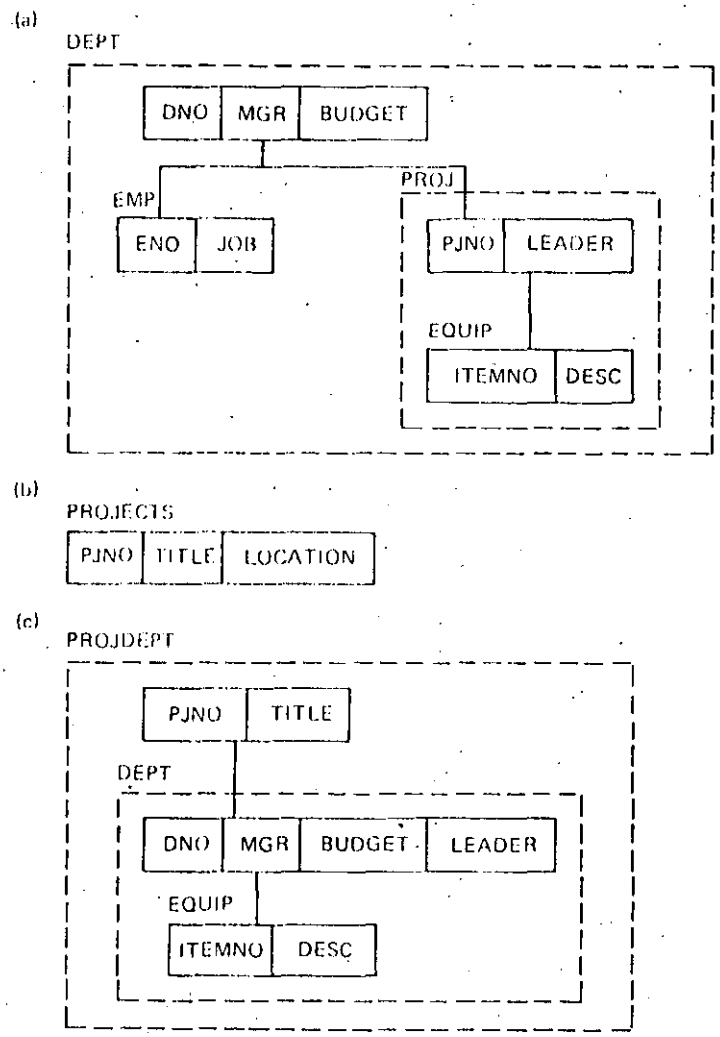


Fig. 28. Hierarchy graphs of the source and target files

even though these kinds of optimizations are interesting, we are not sure of their benefit in a data restructuring environment. Furthermore, users could be taught to apply these considerations when they write the programs in the first place. Therefore we defer these kinds of global optimization to further research.

Now that the system has been implemented, our next step is to verify its usability with real-life applications. An obvious application is database conversion. In practice database conversion is not a "one-shot" process. Rather, application systems and their data are moved gradually as the application programs are rewritten. Another use of EXPRESS is for the extraction of data from a centralized database to support applications requiring a subset of the data in a different "view." For example, EXPRESS could be used to translate data from an operational database on IMS [10] to a planning database used with APL. These exrtaction programs may be incorporated into the routine operation of an enterprise. Thus we feel the

productivity gains offered by EXPRESS, which can be at least a factor of 2 [26], would be of continuing benefit.

APPENDIX. AN EXAMPLE OF A CONVERT PROGRAM

As an example of a CONVERT program, let us consider a company having two source files, DEPT and PROJECTS. The Form schema of the DEPT Form is shown in Figure 28(a). The PROJECTS Form is a flat table containing three fields: PJNO (the project number), TITLE, and LOCATION (Figure 28(b)). In this company more than one department can work on a project, and a department may be responsible for a number of projects in different locations, although a project has only one location. The application is to create a PROJDEPT file for projects located in "SJ" having the hierarchy structure as shown in Figure 28(c).

The following is a CONVERT program which accomplishes the restructuring.

```
FORM DEPT(DNO CH(3),
          MGR CH(3),
          BUDGET BIN,
          EMP(ENO CH(3),JOB CH(2))(OCCURS 30,7 TIMES),
          PROJ(PJNO CH(2);
               LEADER CH(3),
               EQUIP(ITEMNO BIN,
                     DESC CH(9))(OCCURS 25,2 TIMES))
              (OCCURS 15,3 TIMES,
               KEY IS (PJNO), ORDERED ON (PJNO))):
          KEY IS (DNO), ORDERED ON (DNO), OCCURS 8 TIMES,
          SECLEN IS 200;
FORM PROJECTS(PJNO CH(2),TITLE CH(9),LOCATION CH(2)):
              KEY IS (PJNO), SECTION IS 13, OCCURS 6 TIMES,
              ORDERED ON (PJNO), DISP IS (SOURCE);
FORM T4: ORDERED ON (PJNO,DNO);
FORM PROJDEPT(PJNO,TITLE
              DEPT(DNO,MGR,BUDGET,LEADER,
                   EQUIP(ITEMNO,DESC))
                  (KEY IS (DNO))):
              KEY IS (PJNO), DISPOSITION IS (REAL,KEEP);
T1 = SLICE (DNO,MGR,BUDGET,PROJ FROM DEPT);
T2 = SORT (T1 BY PJNO,DNO);
T3 = GRAFT (T2 ONTO PROJECTS
            WHERE T2.PJNO EQ PROJECTS.PJNO);
T4 = SELECT (FROM T3 WHERE LOCATION EQ 'SJ');
PROJDEPT = CONSOLIDATE (T4);
END;
```

Briefly, the first step is to create a flat table (T1) from the right-hand branch (i.e. the branch containing project information) of the DEPT file. The second step is to sort T1 so that we can join it with PROJECTS in the third step (which produces T3). We then select from T3 those projects located at "SJ". The final step is to build a hierarchical Form (PROJDEPT) from the flat table T3.

Note that since the GRAFT operation in step 3 requires the matching of T2 and PROJECTS on PJNO, the sort order required for these two Forms is on PJNO only. However, the final CONSOLIDATE step requires its input to be sorted on the output KEY fields, namely PJNO and DNO. Therefore we carry the sorting

of T2 on (PJNO, DNO) instead of merely on PJNO to avoid another sorting prior to the CONSOLIDATE operation.

REFERENCES

1. Bakkom, D.E., and Behymer, J.A. Implementation of a prototype generalized file translator. Proc. 1975 ACM SIGMOD Int. Conf. on Management of Data, San Jose, Calif., 1975, pp. 99–110.
2. Birss, E.W., and Fry, J.P. Generalized software for translating data. Proc. AFIPS 1976 NCC, AFIPS Press, Montvale, N.J., pp. 889-899.
3. Deppe, M., Lewis, K., and Swartwout, D. Operational software for restructuring network data bases. Working Paper DT 3.2, Data Translation Proj., U. of Michigan, Ann Arbor, Mich., 1976.
4. Fry, J.P., Frank, R.L., and Hershey, E.S. III. A developmental model for data translation. Proc. 1972 SIGFIDET Workshop on Data Description, Access and Control, Denver, Colo., pp. 77–105.
5. Fry, J.P., Smith, D.P., and Taylor, R.W. An approach to stored data definition and translation. ACM 1972 SIGFIDET Workshop on Data Description, Access and Control, Denver, Colo., pp. 13–55.
6. Fry, J.P., Smith, D.C.P., Taylor, R.W., Frank, R.L., Lum, V.Y., Behymer, J.A., and Shneiderman, B. Stored-data description and data translation: A model and language. *Inform. Syst. 2*, 3 (1977), 95–160.
7. Housel, B.C., Lum, V.Y., and Shu, N.C. Architecture to An Interactive Migration System (AIMS). Proc. ACM SIGMOD Workshop on Data Description, Access and Control, Ann Arbor, Mich., May 1974, pp. 157–169.
8. Housel, B.C., and Shu, N.C. A high-level data manipulation language for hierarchical data structures. Proc. Conf. on Data Abstraction, Definition and Structure, Salt Lake City, Utah, March 1976, pp. 155–168.
9. Housel, B.C., Smith, D.P., Shu, N.C., and Lum, V.Y. DEFINE—A nonprocedural data description language for *defining information easily*. Proc. ACM Pacific 75, San Francisco, Calif., April 1975, pp. 62–70.
10. IBM Corp. Information Management System, General Information Manual. IBM Pub. No. GH20-1260, IBM Corp., White Plains, N.Y., 1975.
11. Loomis, M.E. Resource-constrained scheduling of tasks with precedence relationships. Res. Rep. No. RJ 1746, IBM Res. Lab., San Jose, Calif., 1976.
12. Lum, V.Y., Shu, N.C., and Housel, B.C. A general methodology for data conversion and restructuring. *IBM J. Res. and Develop. 20*, 5 (1976), 483–497.
13. Merten, A.G., and Fry, J.P. A data description language approach to file translation. Proc. ACM SIGMOD Workshop on Data Description, Access and Control, Ann Arbor, Mich., May 1974, pp. 191–205.
14. Navathe, S.B., and Fry, J.P. Restructuring for large databases: Three levels of abstraction. *ACM Trans. Database Syst. 1*, 2 (June 1976), 138–158.
15. Ramirez, J.A. *Automatic Generation of Data Conversion Programs Using a Data Description Language (DDL)*, *Vols. I, II*. U. of Pennsylvania, Philadelphia, Pa., May 1973.
16. Ramirez, J.A., Rin, N.A., and Prawes, N.S. Automatic generation of data conversion programs using a data description language. Proc. ACM SIGMOD Workshop on Data Description, Access and Control, Ann Arbor, Mich., May 1974, pp. 207–225.
17. Schneider, G.M., and Desautels, E.J. Design of a file translation language for networks. *Inform. Syst. 1*, 1 (Jan. 1975), 23–31.
18. Shneiderman, B., and Shapiro, S.C. Towards a theory of encoded data structures and data translation. *Int. J. Comptr. Inform. Sci. 5*, 1 (1976), 33-43.
19. Shoshani, A. A logical-level approach to database conversion. Proc. 1975 ACM SIGMOD Conf. Management of Data, San Jose, Calif., pp. 112–122.
20. Shoshani, A., and Brandon, K. On the implementation of a logical data base converter. Proc. Int. Conf. on Very Large Data Bases, Framingham, Mass., Sept. 1975, pp. 529–531.
21. Shu, N.C., Housel, B.C., and Lum, V.Y. CONVERT: A high level translation definition language for data conversion. *Comm. ACM 18*, 10 (Oct. 1975), 557–567.

22. Sibley, E.H., and Taylor, R.W. A data definition and mapping language. *Comm. ACM 16*, 12 (Dec. 1973), 750-759.
23. Smith, D.P. An approach to data description and conversion. Ph.D. Diss., U. of Pennsylvania, Philadelphia, Pa., 1971.
24. Smith, D.P. A method for data translation using the stored data definition and translation task group languages. ACM 1972 SIGFIDET Workshop on Data Description, Access and Control, Denver, Colo., pp. 107-124.
25. Taylor, R.W. Generalized data base management system data structures and their mapping to physical storages. Ph.D. Diss., U. of Michigan, Ann Arbor, Mich., 1971.
26. Winters, E.W., and Dickey, A.F. A business application of data translation. Proc. 1976 ACM SIGMOD Conf., Washington, D.C., June 1976, pp. 189-196.

Received January 1977; revised February 1977

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

DISEÑO DE BASES DE DATOS

A COMPUTER-AIDED METHODOLOGY FOR CONCEPTUAL DATA-BASE DESIGN

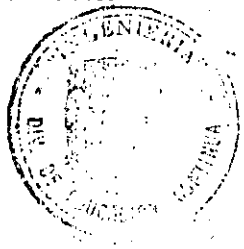
MAYO, 1985

Palacio de Mineria Calle de Tacuba 5 primer piso Deleg. Cuauhtemoc 06000 México, D.F. Tel.: 521-40-20 Apdo. Postal M-2285

Inform. Systems Vol. 7, No. 3, pp. 265–280, 1982
Printed in Great Britain

0306-4379/82/030265-16$03.00/0

A COMPUTER-AIDED METHODOLOGY FOR CONCEPTUAL DATA-BASE DESIGN



C. BATINI and M. LENZERINI
Istituto di Automatica, via Eudossiana 18, Roma, Italy

and

G. SANTUCCI
Selenia S.p.A., Italy

(*Received* 29 *October* 1980; *in revised form* 25 *June* 1981)

Abstract—In this paper we describe an integrated system to aid data-base conceptual design. In this system, incremental enrichments of the conceptual description of user requirements both with a top-down and a bottom-up discipline are allowed.

Such enrichments involve both data and transactions. Data enrichments can be expressed by a disciplined set of design primitives. For every primitive any inconsistencies arising are described and a set of scenarios to eliminate them is suggested.

Transaction enrichments are expressed in terms of declarations which define these transactions at various conceptual refinement levels. Such declarations can be compared to the partial conceptual schema obtained by data enrichments and possible incompleteness and inconsistency situations arising are described.

Future developments of the research program are also outlined.

1. INTRODUCTION

In recent years, various methodologies for data base design have been developed (see [7, 8] for a complete bibliography). In [8] Data Base Design is seen as dealing with four separate areas:

(1) Corporate requirement analysis.
(2) Information analysis and definition.
(3) Implementation design.
(4) Physical design.

This paper concerns methodologies in the second area, i.e. Information Analysis and Definition, where the information acquired from Corporate Requirement Analysis is formally described.

It is commonly agreed that the model by which such formal descriptions are expressed is completely independent of the DBMS used in the implementation of the application. On the other hand, different points of view exist in the literature concerning the process by which such formal descriptions be built.

The different methods proposed can be divided into two distinct classes; for reasons to be explained later we have called them Top-Down Methods (TD methods) and Bottom-Up Methods (BU methods).

Furthermore, it is our opinion that two different meanings can be attached to what we have called TD methods and BU methods.

(a) *In the first meaning*, BU methods refers to those methods that proceed to the formal representation of the global view of the enterprise view or *conceptual schema* (CS) by means of *two* steps (see Fig. 1):

(1) View modelling
(2) View integration

The first step results in a formal representation of each application or *user view* (in the following *user schema* (US)) of interest to the enterprise.

During the second step the different user schemata are integrated into one conceptual schema of the data base. By TD method we mean a method where the conceptual schema is initially produced and only subsequently are the various user schemata derived (see Fig. 2).

In BU methods the user schemata are seen as intermediate analysis step. During view integration *conflicts* among user schemata must be solved so as to gain a common global formal description of the reality.

The process that generates the CS is influenced by the organization of the enterprise into groups of users.

Methods where a *partial skeleton CS* is incrementally enriched by aggregation of the different USs to obtain the completely refined CS can be seen as a compromise between BU and TD methods.

(b) *The second meaning* refers to those methods in which the creation of CS and/or USs is influenced by the

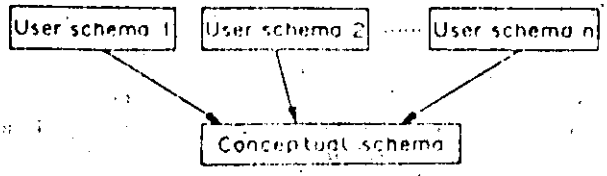


Fig. 1

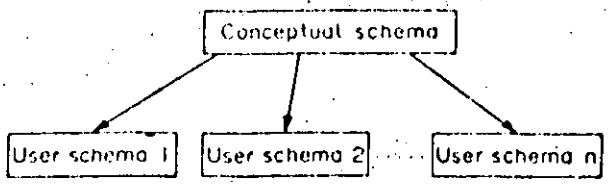


Fig. 2.

organization of concepts according to their *abstraction level.*

The schemata are obtained by a process of *object enrichment* (where by object we mean any concept primitive in the model chosen to represent the view) which can be seen as disciplined by a set of *enrichment primitives.*

The objects of interest to the enterprise may often be categorized according to various abstraction hierarchies. It is commonly agreed in the literature that such abstraction hierarchies be explicitly incorporated into the definition of the model used to represent reality.

In BU methods [1, 3, 9] the initial objects chosen to enrich the schema are *atomic, non decomposable objects.* Subsequently, higher level objects are found and explicitly added, as the abstraction hierarchies are ascended (see Fig. 3).

In TD methods [1–3] the process of schema generation is seen as being composed of a set of refinement steps for the representation of real world details in the schema, so that to every refinement corresponds the substitution of an object by an object structure refining it (see Fig. 4).

In Fig. 5 we try to visualize the interaction between the two different meanings. The cube represents the globality of concepts seen by the various users, coordinate z represents the level of abstraction of these concepts, while partitions over plan β represent the different users which, in consequence of their specific level in the organization, correspond to different abstraction levels with respect to the objects they are able to represent.

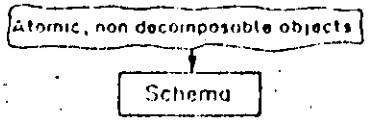


Fig. 3.

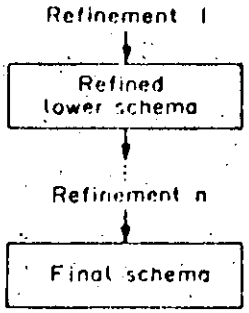


Fig. 4.

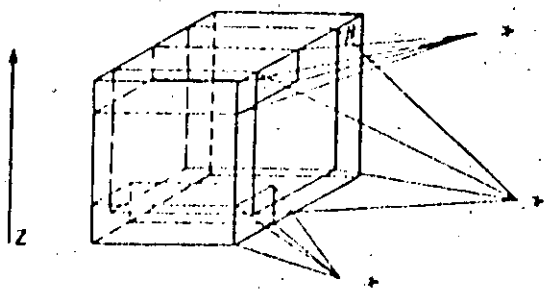


Fig. 5.

It is our opinion that a computer aided methodology for conceptual data base design must enable the environment "user schemata-conceptual schema" to be incrementally enriched according to all of the above described design strategies.

The aim of this paper is to describe the fragment of such a methodology that enables the incremental enrichment of a single schema by refinement and aggregation of objects (coordinate z in Fig. 5).

The allowed enrichments are disciplined in a set of design commands. The semantics of such commands includes:

the transformation operated on the schema by the command;

the possible inconsistencies between the previous representation of the reality expressed in the schema and the new knowledge represented in the primitive.

The paper is organized as follows. In Section 2 we describe the data model referred to in the paper. In Section 3 we give a general description of the design language. In Section 4 we show a detailed example of design session using the system. In Section 5 we sketch further research and developments.

2. THE MODEL

The model referred to in the following is an extension of the Entity-Relationship Model (ERM) proposed by Chen in [3]. A brief description of the model follows.

We distinguish three different types of abstract objects: entities, associations and attributes.

Every abstract object is represented and uniquely identified in the model by an *object name* (entity name, association name, attribute name); it may moreover have both an associated set of synonyms and an explicative text in natural language.

Entities represent those classes of objects in the real

world involved in the application†. An elementary object within a class will be referred to as an *occurrence of an entity*.

Associations represent classes of logical relationships among abstract objects. We allow the existence of associations among entities, associations among entities and associations, associations among associations. An elementary association within a class will be referred to as an *occurrence of an association*.

Attributes represent the properties of entities or associations; an attribute is a mathematical function: the domain is the set of occurrences of an entity or an association, the codomain is a set of values. An elementary value assumed by an attribute will be referred to as an *occurrence of an attribute*.

That set of attributes of an entity that uniquely identifies each occurrence of such an entity is the *key* of the entity. If several keys exist, one of them is chosen as the *primary key* of the entity.

The concept of primary key may be extended to associations: for an association we define as the primary key the union of primary keys of the related entities or associations.

Weak entities are those entities whose existence and unique identification depend on the existence and identification of other entities (*strong entities*): the primary key of a weak entity includes the primary key of the corresponding strong entity.

We now describe those abstraction hierarchies that can be defined in the model.

Such hierarchies hold for entities as well as for associations; in the following we define them for entities only.

The first type of hierarchy is the *subset relationship* (inversely *superset relationship*): an entity $e1$ is a subset of another entity $e2$ if every occurrence of the entity $e1$ is also an occurrence of the entity $e2$.

The second type of hierarchy is *generalization*.

An entity e is a generalization of entities $e1, \ldots, en$ if:

(1) Every entity $ei (1 \leq i \leq n)$ is a subset of the entity e;

(2) Every occurrence of e is an occurrence of a unique entity ei.

The partition over the occurrence of the entity e established by generalization may be seen as induced by a property of e. This property may be explicitly represented in the model by an attribute of the entity e (in this case the property will be referred to as *underlying attribute* [6]) or it may not be represented (in this case we simply name it *property*).

Both in subset relationship and in generalization, attributes of entities at the upper level of a hierarchy are also attributes of the entities at lower level of the same hierarchy. Entities at lower level will generally have additional attributes with respect to the entity at the upper level: in the model we explicitly represent only such attributes.

An ER schema will be represented in the following by means of the symbols of Table 1. In Appendix 1 the syntax of the corresponding DDL appears.

3. GENERAL DESCRIPTION OF THE DESIGN LANGUAGE

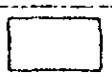
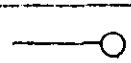
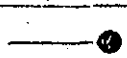
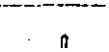
3.1 *Introduction*

The main purpose of the methodology is to help the designer to build the conceptual schema by successive enrichments. Such enrichments can be seen as new knowledge obtained by the designer about the application: such new knowledge may concern both data and transactions. The allowed enrichments are disciplined in a set of primitives, the commands of the *design language*.

Suppose that a given set of commands has been used to create a "partial" conceptual schema S. When a new command (concerning either data or transactions) is applied to S, the consistency of the corresponding trans-

† We therefore use the term "entity" (and in the following "association") as a synonym of the term "entity set" ("relationship set") in [5].

Table 1. The diagrammatic representation

OBJECT	REPRESENTATION
Entity	
Association	
Attribute	
Key attribute	
Generalization relationship with underlying attribute	
Subset relationship	

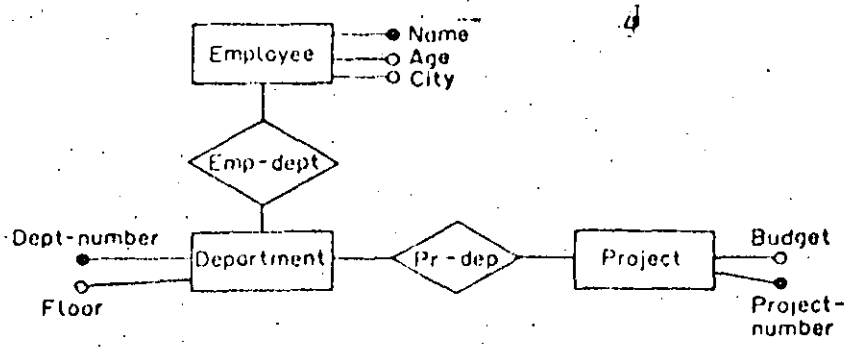


Fig. 6.

formation is checked with respect to the pre-existing schema. In general the types of inconsistencies that can be revealed during the conceptual design depend on:

(1) The formalism to express data.

(2) The formalism to express transactions.

(3) The allowed transformations.

We show now informally an example of inconsistency that may arise during the design.

Example 1

Assume that the schema in Fig. 6 has been previously created. Suppose now the designer creates a new entity City with attributes City-name, State, Zip-Code. An inconsistency is now detected by the system, because the object City has two different types in the schema.

The system aids the designer to solve inconsistencies, providing several scenarios for each type of them. Such scenarios suggest the possible *interpretations of the inconsistency*. To each interpretation a consequent solution corresponds. The designer can accept one of the scenarios or reject all of them. In the first case the corresponding solution is automatically performed. In the second case the command that generated the inconsistency is cancelled.

Example 2

We refer to the inconsistency described in Example 1. There are two interpretations corresponding to the inconsistency:

(a) The new entity and the attribute are homonimous: the corresponding solution is to change one of the names.

(b) The attribute represents a logical link between the entity Employee and the new entity: in this case the solution is to delete the attribute and represent the logical link by means of an association between the two entities.

In order to describe more in detail the features of the system, we now discuss separately the commands that operate on data (Section 3.2) and the commands that operate on transactions (Section 3.3).

3.2 *Data specification commands*

The data specification commands may be classified into two classes:

(1) Bottom-up commands;

(2) Top-down commands.

Bottom-up commands are used for:

(a) Creating and deleting objects (entities, associations, subset and generalization relationships).

Top-down commands are used for:

(b) Expanding objects, i.e. refining them into sets of objects that inherit their logical links.

(c) Giving a structure to entities and associations by defining their attributes, identifiers, cardinalities.

When a top-down command is executed, the expanded object is deleted from the partial schema; trace of the expanded object only remains in the *design history*, i.e. the set of partial schemata obtained during the design.

When a command is applied, the system checks the consistency of the corresponding transformation. We distinguish between *explicit* and *potential inconsistencies*.

Explicit inconsistencies are those that can be automatically detected by an inspection of the schema; a general classification of explicit inconsistencies the system may check is the following:

(1) Naming inconsistencies, arising when two objects of the same type have the same name.

(2) Type inconsistencies, arising when two objects with different type have the same name.

(3) Subset and generalization relationship inconsistencies, when the structure of such relationships in the schema does not agree with their semantics (see in Fig. 6(a) an example of such inconsistency: E is a subset of two entities whose instances are mutually exclusive).

(4) History inconsistencies, arising when a top-down command has been applied and at least one object produced by the expansion appears previously in the design history.

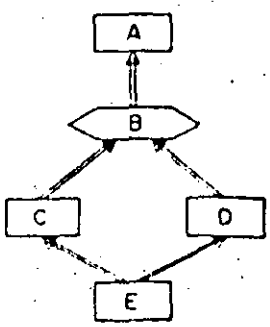


Fig. 6(a).

With regard to potential inconsistencies, they correspond to situations in which *indications* exist of possible:

(1) Synonimies;
(2) Hidden hierarchies;
(3) Redundancies.

Example 3

Suppose the designer defines an association City-State between the entities City and State of the schema described in Fig. 7. A cycle is introduced in the schema: a warning message is printed in this case by the system so that the designer can detect the existence of a redundancy.

A side effect of the application of a command concerns the update of the transactions already defined during the design and referring objects involved in the transformation. The designer can update such transactions with a suitable transaction specification command (see later): this command allows an interactive session to begin, in which the design of the transaction is aided by the system.

In [10], the syntax and semantics of data specification commands are formally described. The semantics of primitives includes the following aspects:

(1) Description of the transformation operated on the schema by the primitive.

(2) Description of possible inconsistencies.

(3) Description of the scenarios suggested to solve the inconsistencies: such scenarios, in [10], are expressed by showing the program in the design language producing them.

The set of suggested scenarios do not constitute an exhaustive list: they can be accepted or rejected by the designer. In the latter case he must propose his own solution.

In the following we use for the object the diagrammatic representation of Table 1, extended with a new symbol:

that represents *concepts*, i.e. objects that are non terminal and that must be translated into terminal objects (entities, associations, attributes) during the design.

We now give for each primitive the syntax, diagrammatic representation and semantics; we also give some examples of possible inconsistencies introduced by the primitive as well as the scenarios suggested to remove them.

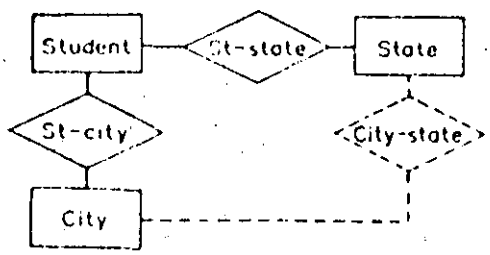


Fig. 7.

(1) *Create a non terminal concept (CREANC)*

Syntax: CREANC: (*id.*).

The diagrammatic representation is: $\emptyset \Rightarrow$  where the symbol $\emptyset$ denotes an absence of input operands.

Semantics: this primitive creates a concept (i.e. a non terminal object) whose name is *id.*.

Notice that primitive CREANC may be considered a top-down primitive when it is the first command in the program (formally it can be seen as the axiom of a context-free grammar); it is a bottom-up primitive when it is applied more than once in the project.

Example

Suppose the primitive

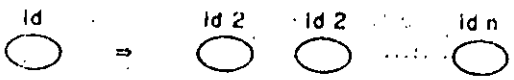
CREANC: COURSE.

has been applied to the partial schema in Fig. 8.

The creation of concept COURSE introduces a type inconsistency because, after its application, two different objects with the same name would exist in the partial schema: in fact COURSE already exists in the schema as a key attribute of the entity EXAMINATION. The system is able to show a scenario removing such inconsistency; the schema suggested by the scenario is shown in Fig. 9. If the designer accepts the suggested solution, the entity EXAMINATION becomes a weak entity; the system will automatically enrich the primary key of the entity EXAMINATION when the designer specifies the primary key of the corresponding strong entity COURSE.

(2) *Expand a non-terminal concept (EXPNC)*

Syntax: EXPNC: (*id.*) $\Rightarrow$ (*id.*), $[(id.)]_0^n$.



Semantics: the concept whose name is *id* is expanded into *n* concepts which inherit logical links of *id*. The primitive EXPNC may be classified as a top-down rule.

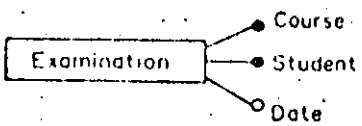


Fig. 8.

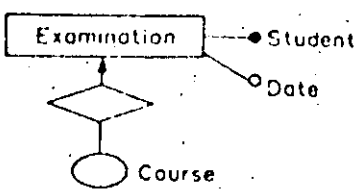


Fig. 9.

Example

Suppose the primitive:

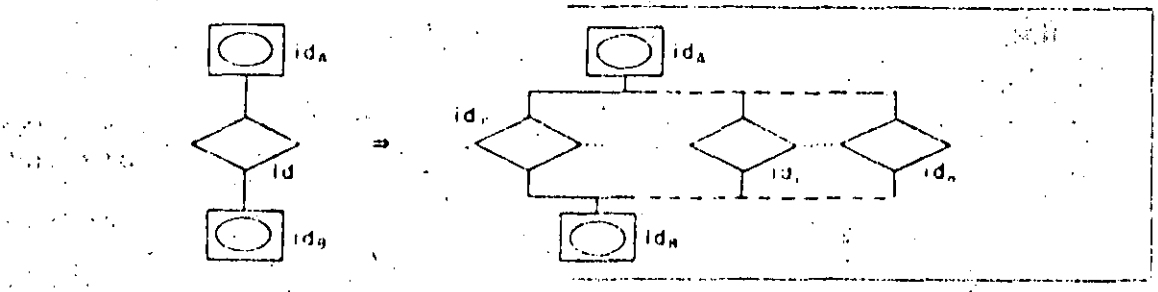
EXPNC: PROJECT-INFO ⇒
INFO ⇒ PROJECT, DEPARTMENT, PROJ-LEADER.

is applied to the partial schema in Fig. 10:

The expansion of the concept PROJECT-INFO gives rise to a hystory inconsistency: in particular the primitive is not consistently appliable to the schema in Fig. 10, because in the original schema an association exists between the two concepts PROJECT-INFO and DEPARTMENT, that are implicitly declared at two different abstraction levels by the primitive EXPNC.

(3) *Expand an association (EXPASS)*

Syntax: EXPASS: $(id_a) \Rightarrow (id_a), \{id_n\}_0^n$.



Symbol ⊠ indicates an object that can be either an entity or a concept.

This rule expands an association between m objects into n associations between the same objects; it may be classified as a top-down rule (in the diagrammatic representation $m = 2$).

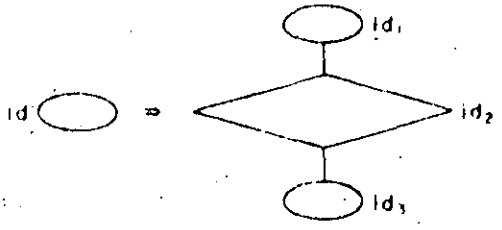
The new associations $id_1, \ldots, id_n$ inherit logical links of id.

(4) *Expand a concept into an association (ENTASS)*

This primitive is a top-down rule; two different syntax forms may be defined for it:

Form 1:

ENTASS: $(id_a) \Rightarrow \{+\}_0^1 (id_a)\{,\{+\}_0^1 (id_{an})[(\text{card})]_0^1\}_2^n$



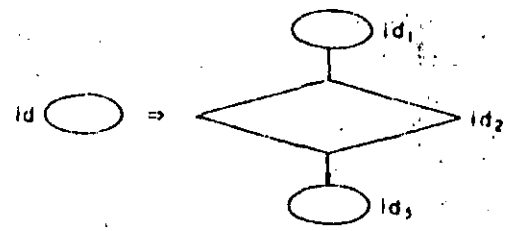
This rule expands a concept into an association between n objects ($n = 2$ in the diagrammatic representation).

In the syntax form, (id_a) represents the name of the association and (id_{an}) the name of the object related by the association; the objects preceded by symbol "+"; inherit the connections of concept (id_a) (*link points* in the following). If symbol "+" does not appear, all objects (id_{an}) inherit such connections.

Finally, (card) represents the cardinality of the object in the association.

Form 2: ENTASS:

$(id_a) \Rightarrow (id_a)\{,\{+\}_0^1 (id_{an})[(\text{card})]_1^n, \{+\}_0^1 *_1(id_{an+1})\}$:



In this case a concept is expanded into an association with one (and only one) weak concept (or entity).

Notice that associations with m–n concepts, each depending on the remaining n concepts, can be represented by m–n associations with only one weak concept.

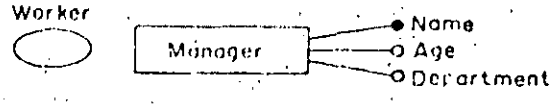
The symbol "+" has the same semantics as in form 1. The symbol "*" marks the weak concept (or entity).

Example

Suppose the following primitive:

ENTASS: WORKER ⇒ EMPL-DEPT, EMPLOYEE, DEPARTMENT.

has been applied to the partial schema:



The expansion of the concept WORKER introduces a type inconsistency: in fact, the schema obtained by the application of the primitive would not be consistent because two different objects (a concept and a non-key attribute of entity MANAGER) identified by the same name would exist in the partial schema.

The system suggests a solution of the conflict, by showing the following schema: (see next page)

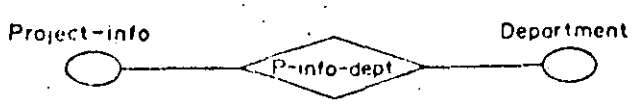
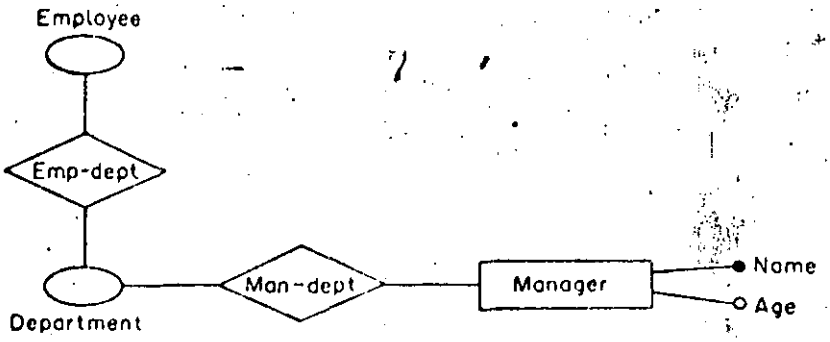


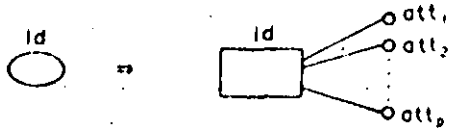
Fig. 10.



(5) *Structuralize an entity* (STRUEN)

Syntax: STRUEN:

$(id_e) \Rightarrow (id_{en}) [, (id_{en})]_m$" # $\{(id_{en}) [, (id_{en})]_n$"$\}_m$"



By this rule a concept is changed into an entity (inheriting the name of the object) and p attributes of the entity; from the syntax we have the (id_{en}) fields on the left of symbol # representing the names of primary key attributes, and the (id_{en}) fields on the right representing the names of remaining attributes. This primitive may be classified as a top-down rule.

Example:

Suppose the following primitive:

STRUEN: EMPLOYEE -> NAME, SURNAME # DEPARTMENT

has been applied to the partial schema:

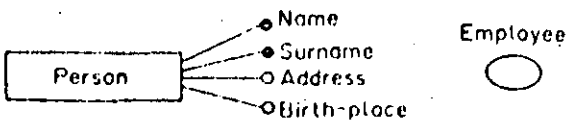


Fig. 11.

The application of the primitive introduces a naming inconsistency, because the attributes NAME and SURNAME form the key of the entity PERSON, already existing in the schema. In this case the scenarios suggested by the system are shown in Fig. 11(a, b).

When the designer chooses the first scenary the conflict is solved.

(6) *Create an abstraction hierarchy* (CREABS)

Syntax: two different forms may occur.

The first syntax form concerns the creation of a superset relationship:

CREABS: (id_{en+1}) SUPERSET OF (id_{en}) [, (id_{en})]$_n$".

The second syntax form concerns the creation of a generalization:

CREABS: (id_{en+1}) GENERALIZATION OF (id_{en}) [, (id_{en})]$_n$" ON (inducing object).

(inducing object) = UNDERLYING ATTRIBUTE (id_{en})/PROPERTY (id_e)

This rule enables a hierarchy to be created between objects (concepts, entities, associations); in syntax forms, (id_{en+1}) represents the object at the upper level; (id_{en}) represents an object at the lower level.

Every concept involved in the hierarchy will, after the application of this rule, be considered a terminal object.

We assume that at least one object must exist in the schema before the application of the rule; it may be considered a top-down or bottom-up primitive, depending on the existence in the schema of the objects involved in it.

When creating a generalization hierarchy, the form UNDERLYING ATTRIBUTE will be used if the inducing object is explicitly represented in the schema as an

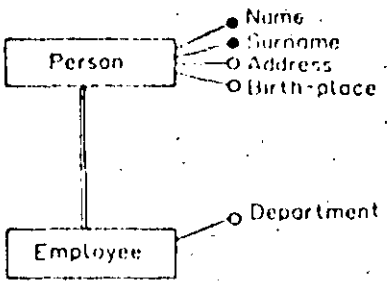


Fig. 11(a)

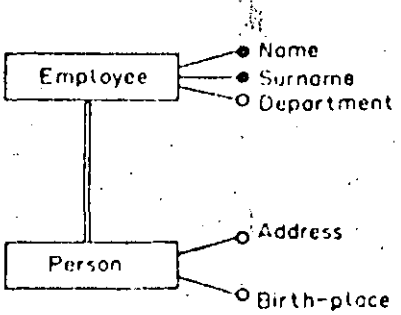


Fig. 11(b)

attribute; in the syntax form, (id_{an}) indicates such an attribute.

The form PROPERTY will be used if the inducing object is external to the schema; in the syntax form, (id_p) indicates such an external property.

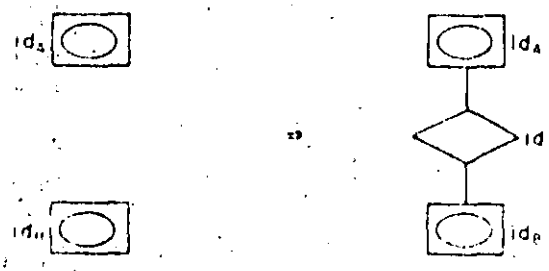
(7) *Create an association* (*CREASS*)

This primitive may be considered a bottom-up rule; two different syntax form are defined for it:

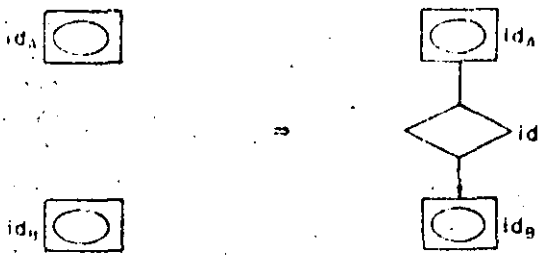
Form 1: CREASS: (id_a) [, (id_{an}) $[((\text{card}))]_n$']$_2$".

In this form the primitive enables an association to be created between n objects ($n = 2$ in the diagrammatic representation). For each object, a cardinality may be defined.

In the syntax, (id_a) represents the name of the association and (id_{an}) the name of the objects related by the association.



Form 2: CREASS: $(id_a)[,(id_{an})]_1$", " (id_{a1}).



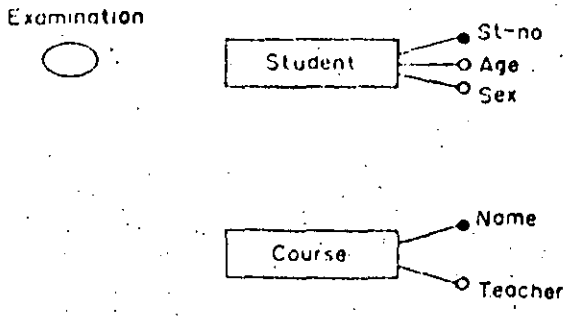
In this case an association with one (and only one) weak entity (or concept) is created.

Example

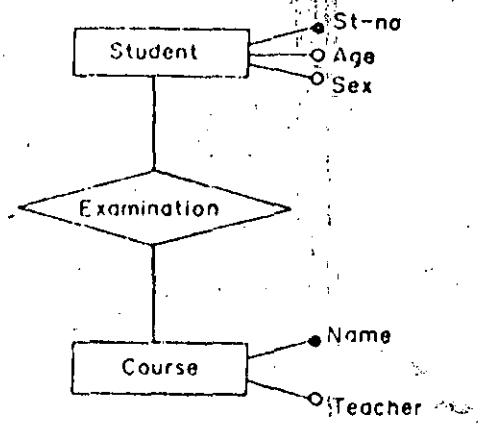
Suppose the primitive:

CREASS: EXAMINATION, STUDENT, COURSE.

has been applied to the schema:



The application of the primitive would create an inconsistency in that it would introduce into the schema an association with the same name as a pre-existing concept (type inconsistency). In this case the proposed scenario gives rise to the following schema fragment:



Example

Suppose the primitive:

CREASS: PERS-REG, PERSON(n), REGION(M).

is applied to the partial schema in Fig. 12 to obtain the schema in Fig. 13.

Before the application of the primitive an association n: 1 was implicitly defined between the entities PER-

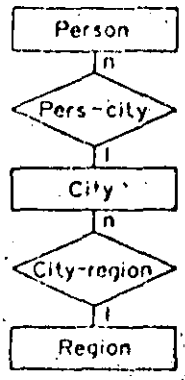


Fig. 12.

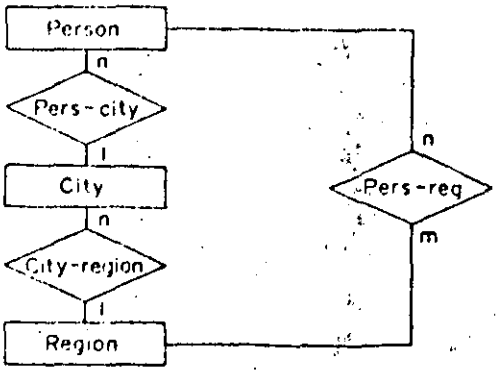


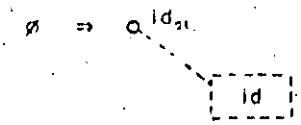
Fig. 13.

SON and REGION: such an association is the transitive closure of the associations PERS-CITY and CITY-REGION.

The creation of the $m:n$ association PERS-REG between PERSON and REGION introduces an inconsistency if the association has the same semantics of the pre-existing relationship. In this case the system prints a warning message and shows the alternative paths created: the designer himself will verify the consistency of the new schema.

In any case, after every activation of the primitive CREASS, if the new association connects two entities already implicitly or explicitly connected, the system informs the designer of the potential pair of paths with the same semantic meaning. If the designer wants to declare the same semantic meaning for two paths, he must use the primitive SYN (see later).

(8) *Create an attribute (CREATT)*
Syntax: CREATT: (id_{at}) [IN]$_0^1$[INKEY]$_0^1$ (id_{at}).



This is the bottom-up rule that enables attributes to be inserted into entities and associations. In the syntax (id_{at}) represents the name of the entity or association where it must be inserted.

If the IN option is used, the attribute (id_{at}) is inserted into the non-key part of (id_{at}). If the INKEY option is used, the attribute is inserted into the key part.

Example
Suppose the partial schema is:

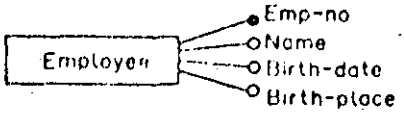


Fig. 13(a).

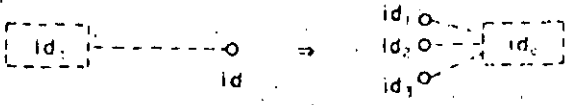
where the attributes NAME, BIRTH-PLACE and BIRTH-DATE were obtained by expansion of the attribute ANAGRAPHICAL-DATA.

If the primitive:

CREATT: ANAGRAPHICAL-DATA IN EMPLOYEE

is applied to the partial schema in Fig. 13(a), a history inconsistency arises: in fact, if the primitive where accepted, two different objects identified by the same name (ANAGRAPHICAL-DATA) would exist in the history.

(9) *Expand an attribute (EXPATT)*
Syntax: EXPATT: $(id_{at}) \Rightarrow (id_{at})\{,(id_{at})\}_0^n$.



This is a top-down rule. An attribute of an entity or an association is expanded into n attributes of the same object.

If the attribute id is a (non) key attribute, then $id_1, \ldots, id_n$ are (non) key attributes.

We now describe two primitives that we may call *documentation primitives*: they allow to declare a set of synonyms and a natural language text for the objects of the schema (and the history).

(10) *Create a textual documentation (TEXT)*
Syntax: TEXT FOR (id): (text in natural language).

A text is declared in natural language for the object (id).

(11) *Define a set of synonyms (SYN)*
Two different syntax forms are defined for this primitive.

Form 1: SYN OF (id): $(id_s)\{,(id_s)\}_0^n$.

In this case the primitive declares a set of synonyms for a given object (id).

When a set of synonyms is created for an object, inconsistencies involving the name of the object must be extended to the name of the synonyms.

Form 2: SYN OF PATH (path 1): (path 2).

where (path 1) and (path 2) are lists of association names.

In this case the primitive declares that two paths in the conceptual schema have the same semantic meaning.

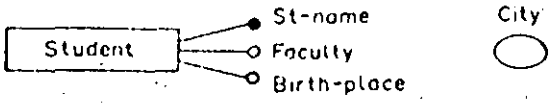
Example
We refer to the partial schema depicted in Fig. 13. With the following primitive

SYN OF PATH PERS-REG: PERS-CITY → CITY-REGION.

the design declares that the path formed by the association PERS-REG has the same meaning as the path: PERSON → PERS-CITY → CITY → CITY-REGION → REGION.

In this case an inconsistency arises, because the first path creates a $m:n$ relationship between PERSON and REGION, the second path creates a $n:1$ relationship between the same entities.

Example
Suppose the partial schema is:



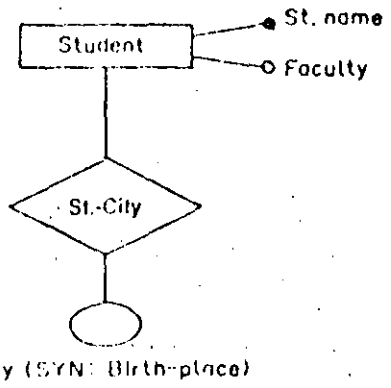
A synonym of CITY is now declared:

SYN OF CITY: BIRTH-PLACE.

The preceding primitive, if activated, would create a conflict in that the same object (called CITY or BIRTH-

PLACE) would have two different roles in the schema: the first consistency condition would not be satisfied.

In this case, the scenario proposed is the following:



(12) *Delete an object (DELETE)*

Syntax: DELETE: (*id*).

The object whose name is (*id*) is deleted.

As a consequence the following are deleted:

—all objects derived from it;

—all associations where (*id*) is involved;

—all the synonyms of (*id*);

—all objects, in the hierarchies, at the lower levels of (*id*);

—all objects weak with respect to (*id*).

3.3 *Transaction specification commands*

There are the goals of managing transactions in the conceptual design phase of data base design (see also [12]):

—Identifying deficiencies in the conceptual schema, i.e. finding additional schema constructs that are needed to execute the operations expressed in user requirements, so to gain completeness of the schema with respect to process requirements;

—Providing formal specification for detailed application development;

—Determining relative access frequencies along functional paths; this information is useful for logical and physical design phase.

We assume that the transaction is initially described by a free text in natural language. Furthermore, we assume that the objects (attributes, entities, associations) of the schema occurring in the free text are always referred to by names and not by values. Any object referred to by values can be named by means of a mental abstraction process. For instance the following text:

"Print out the Students that attended Chemistry in 1978"

has to be transformed in the text:

"Print out the Students that attended a certain Course in a certain Year"

Several different levels of refinement can be defined during the formalization of transactions in the conceptual design step (see Fig. 14).

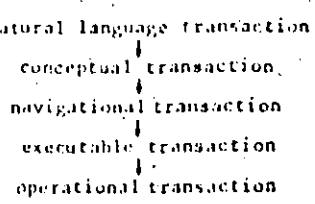


Fig. 14.

By *conceptual transaction* we mean a specification of the transaction in which only the involved subschema is declared, with no concern to procedural aspects. By *navigational transaction* we mean a specification in which the access path is specified. By *executable transaction* we mean a specification in which all the procedural aspects necessary for its execution are declared. By *operational transaction* we mean a specification in which all the information concerning the frequency of use of the transaction and of the objects are declared.

In Fig. 15 the correspondence between goals and levels of refinement suitable for reaching such goals is shown.

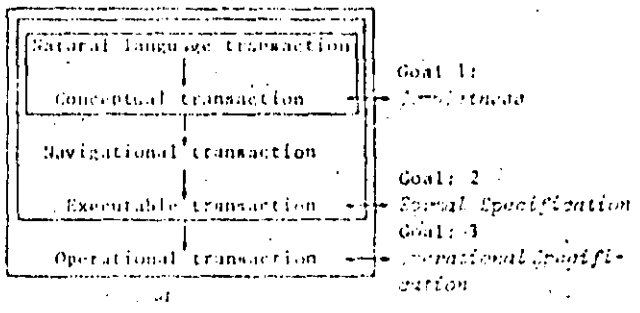


Fig. 15.

In this paper we are mainly concerned with the first of the three goals expressed in Fig. 15, i.e. *completeness*. We need therefore to define a conceptual transaction definition language (CTDL) to express formally the information content of the transaction, with no concern, as we said, to procedural aspects. Such language must allow also partial specification of the subschema involved in the transaction. In other words, the designer is allowed to specify, in the transaction declaration, objects without expressing the path connecting them and/or their properties (type, cardinality, etc). The complete syntax of CTDL is shown in Appendix 3.

In the methodology we propose the designer can match the transaction with the partial schema by means of transaction specification commands expressed in Table 2.

The MATCH TRANSACTION primitive allows *creating* and matching *new transactions* or *matching suspended transactions*.

Assume for instance the schema in Fig. 16 has been previously declared.

Consider now the transaction:

"Print all students of a given teacher".

Table 2. Transaction specification commands.

SYNTAX	SEMANTICS
MATCH TRANSACTION : <trans. name> {<transaction declaration>};	When transaction declaration is present, a new transaction is created and matched with the current data schema; when it is missing, a suspended transaction is matched.
T-DELETE : <trans name>	The transaction is deleted.

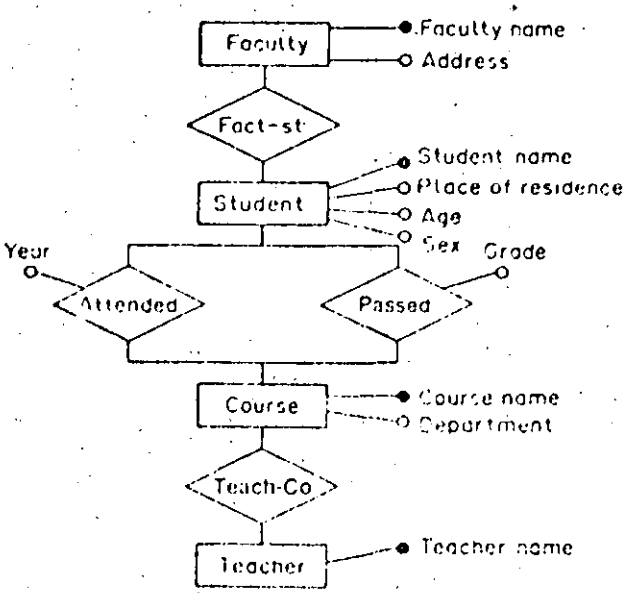


Fig. 16.

The designer can express the transaction with the following partial declaration:

TRANSACTION DECLARATION
TEACHER: EN
STUDENT: EN.

In this case the reason for partial specification is the intrinsic ambiguity of the natural language sentence. If the designer chooses a partial specification, an interactive session begins during which the system checks if such partial specification matches the schema and helps the designer to complete the specification; when a complete specification is given only the checking is performed.

If inconsistencies arise, then scenarios are shown suggesting possible solutions; the designer can choose either to suspend the transaction or to amend the data schema and/or the transaction itself. If the data schema is modified, then (similarly to what happens for data specification commands) transactions referring to objects involved in the modification can be updated or suspended. See the following session for detailed examples of the methodology.

4. AN EXAMPLE OF DESIGN SESSION

In this section we describe an example of design session using the above described commands. From interviews in a department a schema is created by means of the following dialogue (dialogues are enclosed within boxes, symbol '→' precedes the commands of the designer and symbols '***' precede the messages of the system):

```
→CREANC: department-schema.
→ENTASS: department-schema ==⇒ dept-teacher,
         department, teacher-schema.
```

A non terminal concept is created (schema 1 of Fig. 17) and expanded into an association between two new non terminal concepts (schema 2).

```
→ENTASS: teacher-schema ==⇒ teacher-course,
         → teacher, course.
```

The non terminal concept teacher-schema is expanded into an association; the new concept teacher is declared as link point: only this concept inherits the logical links of the refined concept teacher-schema (schema 3).

12

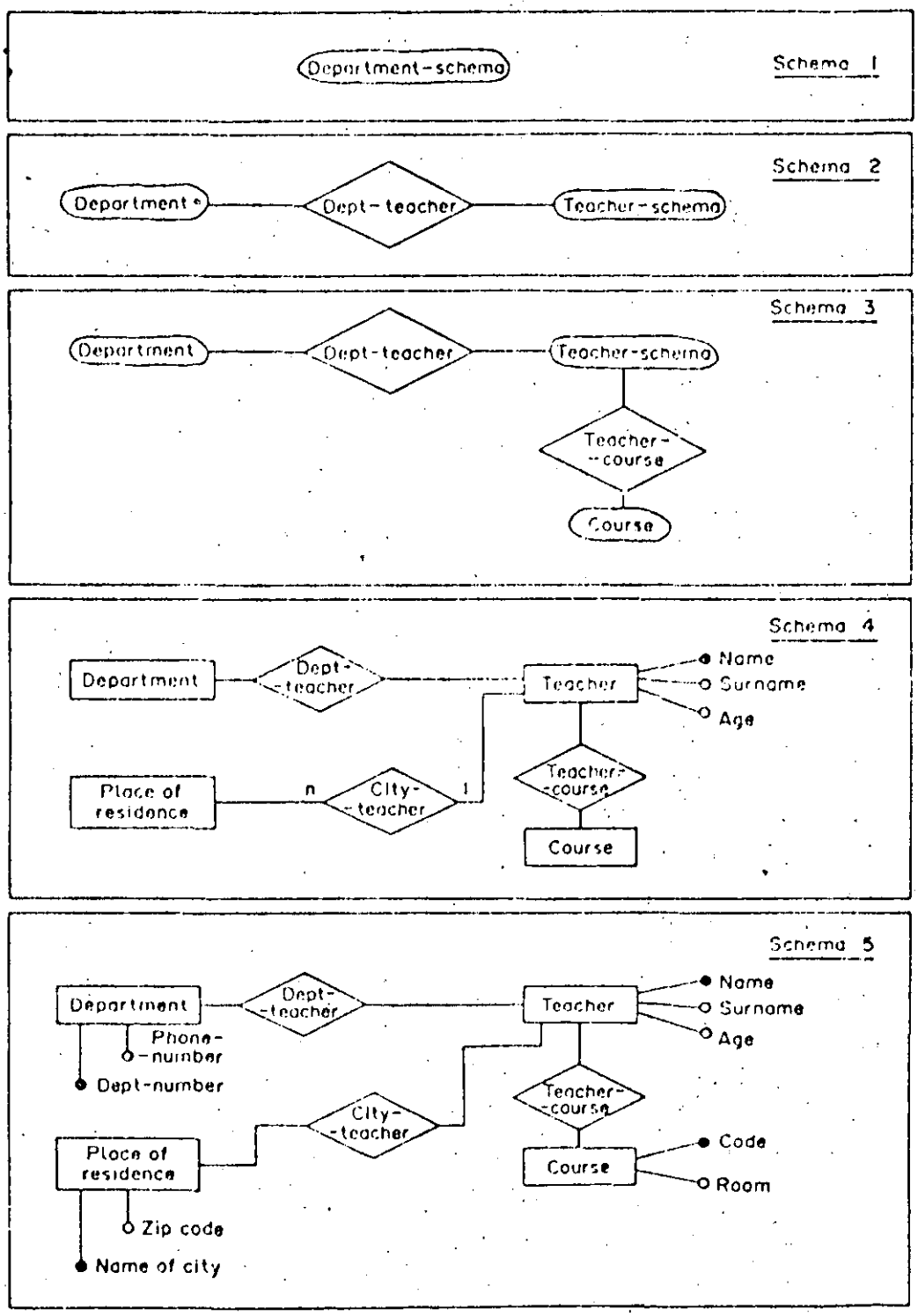

Schema 1
Department-schema
Schema 2
Department
Dept-teacher
Teacher-schema
Schema 3
Department
Dept-teacher
Teacher-schema
Teacher-course
Course
Schema 4
Department
Dept-teacher
Teacher
Name
Surname
Age
Place of residence
n
City-teacher
1
Teacher-course
Course
Schema 5
Department
Dept-teacher
Teacher
Name
Surname
Age
Phone-number
Dept-number
City-teacher
Teacher-course
Place of residence
Course
Code
Room
Zip code
Name of city

Fig. 17.

```
> MATCH TRANSACTION course-list
    course : EN;
    teacher-course : AS; teacher, course;
    teacher : EN;
    code : KAT; teacher : EN.
```

A transaction is declared, whose natural language text is: "Display all the courses given by a teacher with a certain code"

```
***INCONSISTENCY: THE ATTRIBUTES OF THE ENTITY teacher ARE NOT YET DEFINED.
> HELP.
***SCENARIOS:
  1. THE ATTRIBUTE code FOR THE ENTITY teacher MUST BE DEFINED.
  2. THE TRANSACTION MUST BE SUSPENDED.
> 2.
```

An inconsistency is detected by the system. The designer asks for the corresponding scenarios and chooses the second one: the transaction is suspended.

An entity and an association are now created; the type of the association is declared: it relates n instances of the entity teacher with one instance of the entity place-of-residence (schema 4). Then the attributes of the entity teacher are defined by the command STRUEN (symbol "#" divides key attributes (on the left) from the others (on the right)). Now a new inconsistency is detected by the system: the designer accepts the solution suggested by the second scenario and the attribute department is deleted by the system. The non terminal concept teacher is changed into an entity with key attribute code: notice that the inconsistency between the transaction course-list and the schema has been resolved by the last command and the system prints the corresponding message.

The temporary schema is now completed by the following commands:

```
> CREANC : place-of-residence.
> CREASS : city-teacher, teacher (n), place-of-residence (1).
> STRUEN : teacher ==> code # name, surname, age; department.
***INCONSISTENCY : department HAS BEEN PREVIOUSLY DEFINED AS AN ENTITY.
> HELP.
***SCENARIOS:
  1. THE ATTRIBUTE department AND THE ENTITY department ARE HOMONYMOUS: A NEW NAME MUST BE DEFINED FOR THE ATTRIBUTE.
  2. THE ATTRIBUTE department REPRESENTS THE SAME LOGICAL LINK AS ASSOCIATION dept-teacher: THE ATTRIBUTE MUST BE DELETED.
  3. THE ATTRIBUTE department REPRESENTS A NEW LOGICAL LINK BETWEEN THE ENTITIES teacher AND department: A NEW ASSOCIATION MUST BE CREATED AND THE ATTRIBUTE MUST BE DELETED.
> 2.
***THE TRANSACTION course-list MATCHES WITH THE SCHEMA.
```

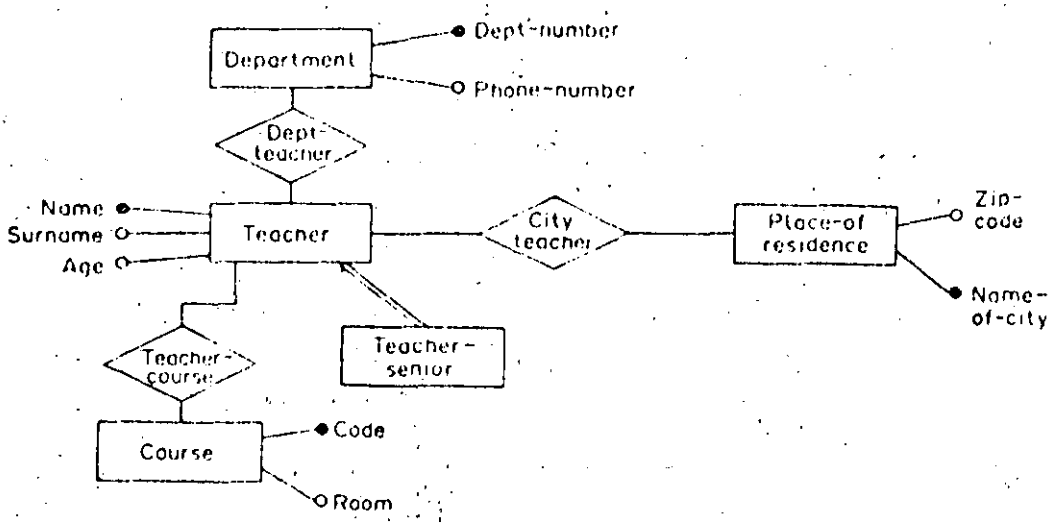


Fig. 18.

lowing commands (schema 5).

```
:- STRUEN: course ===> course-code # room.
:- STRUEN: place of-residence ===> name-of-city # zip-code.
:- STRUEN: department ===> dept-number # phone-number.
:- EXPASS: teacher-course ===> teaches, has-taught.
***THE TRANSACTION course list IS SUSPENDED.THE ASSOCIATION teacher-course HAS BEEN EXPANDED.
```

The association teacher-course is expanded into two new associations: the transaction course-list is now inconsistent with the schema and the system suspends it.

Now the designer amends the transaction by means of the following declaration:

```
:- MATCH CONCEPTUAL TRANSACTION course-list
    course: EN;
    teaches: AS;
    code: KAT; teacher: EN.
```

Suppose now the following transaction must be checked over the schema:

"Given the department number, print name, surname and age of teachers with more than 10 years of seniority, and the codes of the courses in which they have been involved".

```
:- MATCH CONCEPTUAL TRANSACTION
    old-teacher
    dept-number: AT; department: EN;
    name, surname, age: AT; senior-teacher: EN;
    code: AT; course: EN;
    has-taught: AS.

    INCONSISTENCIES:
  1. THE ENTITY senior-teacher HAS NOT BEEN
     DEFINED IN THE SCHEMA.
  2. name, surname, age ARE ATTRIBUTES OF
     THE ENTITY teacher.

:- CREABS: teacher SUPERSET OF senior-teacher.
:- MATCH CONCEPTUAL TRANSACTION
    old-teacher.
```

First the transaction is defined: two inconsistencies arise. Such inconsistencies are solved by creating a subset relationship between the entities senior-teacher and teacher.

The final schema appears in Fig. 18.

5. FURTHER RESEARCH AND DEVELOPMENTS

The design language described in Sections 3 and 4 enables the designer to incrementally specify a schema and a set of conceptual transactions over the schema.

In the future we aim at extending our formal approach to all of the representation levels introduced in Section 3.3 for the specification of the transactions during the conceptual design.

Furthermore, we aim at introducing in the language a new command that allows the integration of user schemata. See [11] for a first version of such an extension.

REFERENCES

[1] C. Batini and M. Lenzerini: Metodi top-down e metodi bottom-up per la derivazione di views, Tech. Rep. DATAID-RT-79-(C)CCA-1, Istituto di Automatica-Universita di Roma, Italy.

[2] C. Batini and G. Santucci: Top-Down design in the Entity Relationship model. *Int. Conf. on the E-R Approach to System Analysis and Design*, Los Angeles, Dec. 1979.

[3] G. Bracchi, S. Ceri, C. Baldissera and G. Pelagatti, Interactive specification of user views in DB Design-VLDB 79.

[4] S. Ceri, G. Pelagatti and G. Bracchi: A Structured methodology for the design of static and dynamic aspects of data base applications, Politecnico di Milano, Italy.

[5] P. C. Chen: The Entity-Relationship model, toward a unified view of data, TODBS 1976.

[6] G. Shulmei and P. Sheuermann: Multiple views, and abstraction with an extended entity—relationship model. *J. Comput. Language*, 1980.

[7] S. B. Yao, S. Navathe and J. Weldon: An integrated approach to logical DB design. *Symposium on DB Design*, New York 1978.

[8] 1978 New Orleans Data Base Design Workshop Report IBM Report RJ2554 (33154) 7/13/79

[9] Data Dictionary System working party. *J. Development*, March 1979.

[10] C. Batini, M. Lenzerini and G. Santucci: A computer aided methodology for conceptual data base design: extended description. Tech. Rep. DATAID-RT-80-CSCCA-1, Istituto di Automatica, Universita di Roma, Italy.

[11] C. Batini and M. Lenzerini: INCOD: A system for interactive conceptual data base design. *Proc. IEEE 18th Design Automation Conf.*, Nashville, 1981.

APPENDIX I

```
⟨object definition⟩::= ⟨entity definition⟩|⟨association definition⟩|⟨concept definition⟩
⟨entity definition⟩::= ENTITY ⟨object name⟩, ⟨entity clause⟩
⟨entity clause⟩::= ⟨primary entity clause⟩|⟨hierarchical entity clause⟩
⟨primary entity clause⟩::= ⟨strong primary entity clause⟩|⟨weak primary entity clause⟩
⟨strong primary entity clause⟩::= ⟨key clause⟩[,⟨non key clause⟩]0¹
⟨weak primary entity clause⟩::= ⟨weak clause⟩, ⟨key clause⟩[,⟨non key clause⟩]0¹
                                ⟨weak clause⟩, ⟨non key clause⟩
⟨hierarchical entity clause⟩::= ⟨hierarchy clause⟩[, ⟨hierarchy clause⟩]0"[, ⟨weak clause⟩]0¹
                                [,⟨non key clause⟩]0¹
```

⟨weak clause⟩ ::= WEAK IN ASSOCIATION ⟨identifier⟩ {, WEAK IN ASSOCIATION ⟨identifier⟩}$_0^n$
⟨hierarchy clause⟩ ::= ⟨ISA external clause⟩/⟨ISA internal clause⟩/⟨subset clause⟩
⟨ISA internal clause⟩ ::= ⟨ISA identifier⟩, UNDERLYING ATTRIBUTE IS ⟨identifier⟩
⟨ISA external clause⟩ ::= ⟨ISA identifier⟩, PROPERTY IS ⟨property description⟩
⟨subset clause⟩ ::= SUBSET OF ⟨identifier⟩
⟨key clause⟩ ::= KEY IS ⟨object name⟩ {, ⟨object name⟩}$_0^n$
⟨non key clause⟩ ::= NON KEY IS ⟨object name⟩ {, ⟨object name⟩}$_0^n$}$_0^n$
⟨association definition⟩ ::= ASSOCIATION ⟨object name⟩ {, ⟨hierarchy clause⟩}$_0^n$,
⟨association clause⟩ {, ⟨non key clause⟩}$_0^n$
⟨association clause⟩ ::= ASSOCIATED OBJECTS ARE ⟨identifier⟩ (⟨card⟩) {, ⟨identifier⟩ (⟨card⟩)}$_1^n$
⟨concept definition⟩ ::= CONCEPT ⟨object name⟩ {, ⟨weak clause⟩}$_0^n$ {, ⟨hierarchy clause⟩}$_0^n$
⟨hierarchy concept clause⟩ ::= ⟨ISA external clause⟩/⟨subset clause⟩
⟨object name⟩ ::= ⟨identifier⟩ {, SYNONYM IS ⟨identifier⟩ {, ⟨identifier⟩}$_0^n$}$_0^n$ {, TEXT IS ⟨text⟩}$_0^n$
⟨identifier⟩ ::= *character string*
⟨text⟩ ::= *free text*
⟨property description⟩ ::= *free text*
⟨card⟩ ::= 1/N/M

APPENDIX 2

Design Language—BNF

⟨primitive⟩ ::= ⟨creanc⟩/⟨expnc⟩/⟨expass⟩/⟨entass⟩/⟨strucn⟩/⟨strucs⟩/
⟨creass⟩/⟨creabs⟩/⟨creatt⟩/⟨expatt⟩/⟨delete⟩/⟨text⟩/⟨syn⟩

⟨creanc⟩ ::= CREANC: id_c
⟨expnc⟩ ::= EXPNC: $id_c \Rightarrow id_c$ {, id_c}$_0^n$
⟨expass⟩ ::= EXPASS: $(id_a) \Rightarrow id_c$ {, (id_a)}$_0^n$
⟨entass⟩ ::= ENTASS: $id_e \Rightarrow$ ⟨en. form 1⟩;ENTASS: $id_e \Rightarrow$ ⟨en. form 2⟩
⟨strucn⟩ ::= STRUCN: $id_{ass} \Rightarrow id_{ass}$ {, id_{ass}}$_0^n$ ≠ {id_{ass} {, id_{ass}}$_0^n$}$_0^n$
⟨strucs⟩ ::= STRUCS: $id_{ass} \supseteq id_{ass}$ {, id_{ass}}$_0^n$
⟨creass⟩ ::= CREASS: ⟨cr. form 1⟩;CREASS ⟨cr. form 2⟩
⟨creabs⟩ ::= CREABS: id_{ass} SUPERSET OF id_{ass} {, id_{ass}}$_0^n$
CREABS: id_{ass} GENERALIZATION OF id_{ass} {, id_{ass}}$_0^n$ ON ⟨property⟩
⟨creatt⟩ ::= CREATT: id_{ass} (IN/,INKEY)$_0^n$ id_{ass}
⟨expatt⟩ ::= EXPATT: $id_{ass} \Rightarrow id_{ass}$ {, id_{ass}}$_0^n$
⟨delete⟩ ::= DELETE: id
⟨property⟩ ::= UNDERLYING ATTRIBUTE id_{ass} /EXTERNAL PROPERTY id_{ass}
⟨text⟩ ::= TEXT FOR id: ⟨text in natural language⟩
⟨syn⟩ ::= SYN OF id: id {, id}$_0^n$
⟨cr. form 1⟩ ::= (id_{ass}) {(⟨card⟩)}$_0^n$}$_2^n$
⟨cr. form 2⟩ ::= id_{ass} {, id_{ass}}$_0^n$, id_{ass}
⟨en. form 1⟩ ::= {•}$_0^1$ id_e {, {•}$_0^1$ (id_{ass}) {(⟨card⟩)}$_0^n$}$_2^n$
⟨en. form 2⟩ ::= (id_{ass}) {, {•}$_0^1$ (id_{ass})}$_1^n$, {•} id_e
⟨card⟩ ::= 1/N/M
(id) ::= $(id_{ass})/(id_{ass})$
(id_{ass}) ::= $(id_c)/(id_c)/(id_c)$
(id_{ep}) ::= ⟨external property identifier⟩
(id_{ass}) ::= $(id_c)/(id_c)$
(id_{ass}) ::= $(id_c)/(id_c)$
(id_c) ::= ⟨concept identifier⟩
(id_c) ::= ⟨association identifier⟩
(id_e) ::= ⟨entity identifier⟩
(id_{at}) ::= ⟨attribute identifier⟩

APPENDIX 3

Transaction declaration language—BNF

A transaction declaration is expressed as

TRANSACTION DECLARATION
⟨object declaration⟩
.
.
.
⟨object declaration⟩
END OF DECLARATION

An object declaration is defined by the following grammar:
⟨object declaration⟩ ::= ⟨name of object⟩/⟨type declaration⟩
⟨type declaration⟩ ::= ⟨name of object⟩::⟨type⟩

16

⟨type⟩: = ⟨simple type⟩/⟨composed type⟩
⟨simple type⟩: = AT/ENTITY/RELATIONSHIP
⟨composed type⟩: = ⟨attribute composed type⟩/⟨association composed type⟩
⟨attribute composed type⟩: = ⟨attribute type⟩: ⟨attribute's operands declaration⟩
⟨attribute type⟩: = KATTRIBUTE/PKATTRIBUTE/NKATTRIBUTE.
⟨attribute's operands declaration⟩: = ⟨name of object⟩: ENTITY/⟨name of object⟩: ASSOCIATION
[? , ⟨name of object⟩: ENTITY, $card[i_m, i_M]$]$_2^n$
⟨association composed type⟩: = ⟨name of object⟩ ASSOCIATION: ⟨name of object⟩: ENTITY [, $card[i_m, i_M]$]$_2^n$
[? : ⟨name of object⟩: ENTITY [, $card[i_m, i_M]$]$_2^n$
⟨name of object⟩: = *character string*.

In [10] the semantics of the transaction declaration language is described.

DISEÑO DE BASES DE DATOS

FINAL REPORT OF THE ANSI/X3/SPARC DBS-SG RELATIONAL DATABASE TASK GROUP

MAYO, 1985

Final Report of the ANSI/X3/SPARC DBS-SG Relational Database Task Group

Edited by

Michael L. Brodie and Joachim W. Schmidt

September 1981

Doc. No. SPARC-81-690

The Relational Database Task Group had the following members at the time that this report and the associated SD-3 was forwarded to the DBS-SG:

Michael L. Brodie.....University of Maryland (co-chairman) and Computer Corporation of America
Joachim W. Schmidt.....University of Hamburg (co-chairman) and Computer Corporation of America
Alex J. Arthur.....International Business Machines Corporation
Margret Ball.....Intel Systems Corporation and Burroughs Corporation
Charlie Bontempo (Alternate for Arthur).IBM Systems Research Institute
Harrison R. Burris.....TRW Inc.
Beth Driver.....PRC Information Sciences Co. and Technology Service Corporation
Roy Hammond.....Statistics Canada
Peter Hitchcock.....University of Victoria and British Columbia Systems Corporation
Kate Kinsley.....University of Central Florida
Nancy McDonald.....University of South Florida
Alain Pirotte.....Philips Research Laboratory, Brussels, and Computer Corporation of America
Daniel R. Ries.....Lawrence Livermore Laboratory and Computer Corporation of America
Edgar H. Sibley.....Alpha Omega Group, Inc.

Acknowledgments

Several RTG members gratefully acknowledge the support of their participation by research agencies:

Michael L. Brodie.....Army Research Office and National Bureau of Standards
Peter Hitchcock.National Scientific and Engineering Research Council of Canada
Joachim W. Schmidt.....Federal Ministry of Research and Technology of W. Germany
Edgar H. Sibley.....Environmental Protection Agency and National Bureau of Standards

The RTG thanks the Computer Corporation of America for its generous support of production of this report. Special thanks are due to David Darcy who carefully formatted the entire document.

Others who contributed to the work of the Relational Database Task Group are:

Kjell Bratbergsengen.....Sperry Univac and University of Trondheim
Edgar F. Codd.....International Business Machines Corporation
Chris Date.....International Business Machines Corporation
Wolfgang Dotzek.....University of Hamburg
Jim Driscoll.....University of Central Florida
Oris D. Friesen.....Honeywell Information Systems
K. Grammel.....University of Central Florida
Andrew Hutt.....International Computers Limited
H. Randall Johnson.....Boeing Computer Services
J.Z. Kornatowski.....University of Toronto
William Kent.....International Business Machines Corporation
Winfried Lamersdorf.....University of Hamburg
Michael Lacroix.....Philips Research Laboratory, Brussels
Jim Larson.....Sperry Univac and Olivetti, Italy
Manuel Mall.....University of Hamburg
John McNally.....University of South Florida
T. William Olle.....T. Wm. Olle, Assoc.
Ken Paris.....Peat, Marwick, Mitchell & Company
Dzenan Ridjanovic.....University of Maryland
C.M. Robertson.....University of Toronto
Michael Stonebraker.....University of California, Berkeley
Tom Wilson.....Sperry Univac

Preface

In May 1979, the Relational Task Group (RTG) was chartered to investigate the justifiability of proposing to ANSI/X3 that a project be initiated to develop a relational standard. Having concluded that such a proposal is justifiable, the RTG drafted a document entitled "Proposal for Standardizing Interfaces to Relational Database Management Systems" to be submitted to ANSI/X3. The RTG proposes that the functionality of the interfaces to Relational Database Management Systems (RDBMS) be standardized.

The current report is intended to support the proposal for a relational standard. It documents the results of the RTG's investigations which consisted of three tasks:

1. Identify the fundamental concepts of the Relational Data Model (RDM).
2. Characterize the features of existing and potential RDBMSs to determine the interface functions.
3. Investigate the role of the RDM and RDBMS in a DBMS architectural framework such as the ANSI/X3/SPARC prototypical architecture, and in a coherent family of DBMS standards.

The first two tasks were tractable. The results of the first task are found in Chapter 2. The results of the second task are found in Chapter 3*. These results form the technical basis for the proposal.

The third task addressed open research problems. Chapter 4 documents the problems by identifying issues and known alternatives. Chapter 5 summarizes the results and proposes guidelines for developing a relational standard.

---

*The results are based on detailed feature analyses of twelve individual RDBMSs. These twelve RTG working documents will be published in Brodie, M.L., and Schmidt, J.W. (Eds.), "Relational Database Management Systems: Analysis and Comparisons," Springer-Verlag, Heidelberg/New York, in press.

CONTENTS

For more than a decade the relational database model (RDM) has received considerable attention in the research community and is now receiving attention by database practitioners. The RDM's simplicity and formality continue to inspire a large amount of research and development. Many practical and theoretical advantages have been demonstrated or claimed, e.g., high level relational languages have strongly influenced the development of database query languages and theoretical results have provided database design aids.

Over the past ten years the relational approach to databases has become a dominant factor in database education. It is used for database design, analysis, and query formulation. Currently there are over 70 DBMSs that are said to support relational concepts and languages. Extensive interest in these systems indicates that they soon will come into widespread use.

Although they are similar, the 70 RDBMSs do not provide the same "relational" functionality. For these and other reasons, the ANSI/X3/SPARC Database Systems Study Group (DBS-SG) established the RTG to investigate the justifiability of proposing to ANSI/X3 that a relational standard be developed. This chapter outlines the motivations, charter, and products of the RTG.

1.1 Motivations for a Relational Standard

Three motives led to the investigation of a relational standard. One motive concerned timing. Since its introduction over a decade ago [CODD70], the core RDM concepts have provided a stable basis for database research and development. For some years, the relational approach has been widely taught and understood in both commercial and academic settings. Currently, significant interest is being expressed in the DBMS marketplace. Over 70 DBMSs, languages, and hardware devices said to be based on the RDM have been developed, many of which are or soon will be commercially available. Keen user interest has resulted in many projections of their widespread use in the near future. The effectiveness of relational technology has been demonstrated for a wide range of user and application needs.

A second motive is the desire for alternative approaches to data description, access, manipulation, and integrity control. This was recognized at the outset of the CODASYL approach to databases. More recently, the desire for coexisting, heterogeneous databases, motivated (in part) the ANSI/X3/SPARC prototypical architecture [TSIC78]. To date, only the CODASYL approach has been considered for standards development. The desire for alternatives can be seen in the variety of DBMS types (e.g., CODASYL, relational, hierarchic, primitive, hybrid) in today's DBMS marketplace. In September of 1980, the National Bureau of Standards made a firm commitment to support three primary alternatives (i.e., CODASYL, relational, and hierarchical). The appropriateness of an alternative depends not only on the nature of the application requirements and user needs but also on matters of taste and style. Since the relationship between the alternatives (e.g., one a subset of another) is an open research topic, they must currently be viewed as distinct.

A third motive involved the claims made, and in some cases demonstrated, for the inherent benefits of the relational approach. Four of these claims are: The RDM and its related languages are simple and uniform and are based on a small number of well understood concepts. The RDM permits a high degree of data independence -- freedom of relational schemata and languages from representation details. The RDM is based on a sound theoretical foundation which permits the analysis of languages and applications, and provides a basis for further research. The RDM provides high level, nonprocedural qualification primitives (e.g., the relational calculus) for the design of high level query and manipulation languages.

1.2 RTG Charter

At its May 29, 1979 meeting in Boston, the ANSI/X3/SPARC Database System Study Group (DBS-SG) unanimously approved the establishment of and charter for the Relational Database Task Group (RTG). The charter reads as follows:

"The Relational Database Task Group (RTG) will establish whether there is sufficient justification for proposing to ANSI/X3 that a standards project be initiated for the relational approach to databases. To do so the RTG will identify or establish:

1. Relational Data Model (RDM) concepts and terminology including an RDM definition;
2. aspects of the RDM and of Relational DBMSs (RDBMSs) that might be appropriate for standards development;
3. issues concerning the role of the RDM and RDBMS features in a DBMS architecture such as the ANSI/X3/SPARC prototypical architecture;
4. the relationship of existing and potential software standards to potential relational standards;
5. other guidelines for standards development in this area, as appropriate."

1.3 RTG Products

The RTG had its first meeting, under the chairmanship of Michael L. Brodie, on July 22-23, 1979 together with DBS-SG in Minneapolis. At that time, it was proposed that the RTG respond to its charter by producing two documents for the DBS-SG to be considered for presentation to SPARC.

The primary product of RTG will be a recommendation for or against the initiation of an ANSI/X3 standards project for the relational approach to databases. The format of the recommendation will follow the one defined in the ANSI/X3/SD-3 document. The resulting document will be referred to as "the SD-3". Typically, if X3 approves an SD-3, it establishes a technical committee responsible for the proposed standards project.

The RTG's main objective for the SD-3 was to provide sufficient and precise information to aid

the DBS-SG, X3, and SPARC in deciding whether to initiate a relational standards project. This involved a clear description of the nature of the standard (i.e., definitions of the RDM and features of an RDBMS) and a thorough evaluation of the state of the art. The SD-3 should also provide adequate guidance for a subsequent technical committee by outlining an unambiguous scope and effective program of work. The SD-3 should not overly constrain the technical committee's work, e.g., issues and alternatives should be raised but not resolved. The SD-3 has been completed and is entitled "Proposal for Standardizing Interfaces to Relational Database Management Systems".

The current document, "The Relational Database Task Group Final Report", has three purposes: it provides technical support for the SD-3; it records the results of the RTG's work; and it provides a base document for subsequent technical committee(s).

During its two year existence, the RTG consisted, on the average, of 18 people. Each of its 13 meetings was attended by at least 12 RTG members.

The RTG established liaison with the following groups and countries:

ANSI/X3 (U.S.A.): H2, H4, J1, J3.1, J4, DBS-SG
CODASYL (U.S.A.): DDLC, FDBLC
National Bureau of Standards (U.S.A.)
International Standards Organization: TC97/SC5/WG3, TC97/SC5/GTS
Canadian Standards Association (Canada)
British Standards Institute (Great Britain)
British Computer Society (Great Britain): DDSWP, DBORG, DDAWG
Deutsches Institut fuer Normung, DIN (Germany)
Gesellschaft fuer Mathematik und Datenverarbeitung: GMD (Germany)
Association Francaise de Normalisation, AFNOR (France)
Bureau d'Orientation de la Normalisation en Informatique (France)
Institute National de Recherche en Informatique et en Automatique, INRIA (France)
Consigla Nazionale delle Ricerche (Italy)
Finland
Norway
Sweden

2. THE RELATIONAL DATA MODEL (RDM)

From their initial formulation to their precise definition and their final realization in a Relational Database Management System, the concepts that constitute the RDM are defined in different ways. There are three distinct steps in this progression:

- The Relational Data Model is the collection of basic concepts that underlies the relational approach to data definition, manipulation, query, and integrity.
- A Relational Data Model Definition provides a notation (syntax) to identify the concepts of the RDM and an unambiguous specification of the meaning (semantics) associated with this notation.
- Relational Database Management Systems (RDBMS) are software systems that provide interfaces with the functionality specified by an RDM definition. Languages (syntax and semantics) at these interfaces are tailored to meet user's needs.

The purpose and relative importance of the steps are clear. RDM concepts are fundamental. An RDM definition is one of many possible expressions of the RDM concepts. A RDM definition is needed to guide the specification and implementation of the relational features of an RDBMS.

This chapter outlines issues in determining the constituent concepts of the RDM and in formulating definitions of RDM concepts. References to the literature are used to illustrate a variety of RDM definitions. The basic concepts are discussed and some relational terminology is introduced.

2.1 Inherent Definitional Issues

This section addresses two central definition issues: first, which concepts should be included in a basic RDM definition and which might be considered as extensions; second, what is the relationship between the purpose of a definition and its form. Issues such as precision, level of detail, and formality of RDM definitions are also raised.

2.1.1 RDM Core Concepts and Extensions

Of the many concepts associated with the RDM, which of them are fundamental, and which could be defined in terms of (or as extensions to) fundamental concepts? For the purpose of a DBMS standard, a major criterion for distinguishing core RDM concepts from extensions concerns the state of the art. The core should contain those concepts needed to account for current RDBMSs as opposed to other DBMSs. Further, the core concepts should be demonstrably practical. Hence, some desirable and useful concepts that are prohibitively hard to implement should remain, for the moment, extensions.

There are other criteria for distinguishing core concepts. Fundamental concepts need not be minimal. They should provide a complete basis for data definition, altering, querying, and integrity of relational databases. They should also provide a basis for distinguishing the RDM from other data models as well as a consistent basis for extending the RDM (e.g., so that a standard based on the fundamental concepts could evolve naturally to take advantage of new research results).

Concepts could be excluded from a basic RDM definition if they:
- conflict with fundamental concepts, e.g., duplicate tuples, row ordering;
- concern representation issues, e.g., representation of relations, detailed syntax of DDLs and DMLs;
- are nonrelational in nature, e.g., library functions, security;
- concern the use of the RDM, e.g., dependency and normalization theory for database design and data integrity;
- concern implementation or language issues, e.g., create and drop relation; store, name, or maintain derived relation values (hence dynamic and static derived relations are excluded);
- concern open research issues, e.g., null values, surrogates.

The RTG considers the following concepts to be fundamental to the RDM. These concepts form the basis of the RDM description given in Section 2.3.

Basic Background:
set theory, propositional logic, first order predicate calculus;

Structure Concepts:
relation, relation schema, relation value, tuple, attribute, domain, relational database, relational database schema;

Operators:
(a) Retrieval and Derivation:
(the relational algebra operations are given; however, the relational calculus could also be used)
Cartesian product, union, intersection, set difference, selection, projection, join, division;
(b) Altering:
insert, delete, replace;

Integrity Concept:
Key;

Examples of concepts excluded by the RTG from the basic RDM definition are:

Structure Concepts:
derived relation (static and dynamic), derived relation scheme, null values;

Operators:
(a) Retrieval and Derivation:
library functions, derivation over null values;

(b) Altering:
altering concerned with null values.

Integrity Concepts: candidate key, referential integrity (foreign key);

2.1.2 Definition Issues

The RDM concepts can be formulated in many substantially equivalent definitions. Formulations may vary in: terminology, representation, notation (formalism), level of detail, precision, and degree of formality. Ideally, each definition taker advantage of the possible variations to meet the particular needs of the intended audience, to emphasize certain RDM aspects, or to express a

particular perception of the RDM.

Possible audiences for an RDM definition differ substantially in interests, needs, and backgrounds. Typical audiences include: DBMS designers and implementors, application development language designers and implementors, applications designers and implementors, application users, students, and researchers. A different RDM definition may be appropriate for each audience and its intended purpose for the definition. For DBMS standards, the most important use of one or more RDM definitions would be as a basis for specifying a relational standard. The definitions must be sufficiently precise, formal, and detailed to permit analysis for consistency and completeness of the standard and for conformity of a particular DBMS with the standard.

Terminology

There is no single generally accepted terminology for DBMSs or for discussing the RDM. While the existence of synonyms is often cited as a terminology problem, the RTG has found pseudo-synonyms (e.g., words that may refer to similar objects but whose definitions differ, such as "record", "tuple", and "row"), and words that have multiple meanings (e.g., "key") to pose even greater problems. The use of a definition and term depends on one's perspective and the topic at hand; for example, "attribute", "data item", and "field" may be used in discussions of different issues.

The relational terminology used throughout this report is given above in the list of fundamental RDM concepts (Section 2.1.1). The terms "table", "row", "column", etc. (see Section 2.2), are not used because they are less precise and are intended for a wider audience.

Representation

The RDM can be represented in many ways. Hence, the RDM should not be biased towards a particular representation. Relations can be perceived or represented in a variety of ways, e.g., sets of structured tuples, n-ary predicates on elementary values, arrays of tuples, or as n-ary functions (in general multi-valued) from elementary values to elementary values. Relational operators can be expressed in several forms, e.g., in relational algebra, tuple-oriented relational calculus, domain-oriented relational calculus, or in n-ary functions.

Only certain combinations of structure representation and operator form are appropriate, e.g., sets with algebra, tuple or domain calculus with predicates, functions for both, and sets and calculus. Some languages mix both sets and arrays, and calculus and predicates.

Notations

Many notations can be used to define the RDM. English can be used for informal definitions and many formalisms can be used for a formal definition, e.g., set theory [CODD70], extended set theory [CHILDS77], Positional Set Notation [HARD80], Meta-IV of the Vienna Development Method [BJOR78], the Vienna Definition Language [LUCAS71], Floyd-Hoare axiomatics [HOARE72], functional specifications [MILLS79], algebraic specifications [GUTT78], initial algebras [GOGU77], and graph theory [FURT78] (see also Section 2.3).

Level of Detail

In comparison with RDBMS implementors, users of relational query languages and users of applications implemented on an RDBMS need to know only relatively few RDM details. For example, the RDM description in Section 2.3 may be adequate for users but inadequate for RDBMS implementors.

Precision

A definition at any level of detail would be precise if there were no question as to the understanding of the details given and the scope of what is described. In addition, a precise definition would remain consistent as details were added. For example, the footnotes in Section 2.3 concerning first order predicate logic as a basis for selection criteria in the relational calculus are precise but not detailed.

Formality

Informal definitions are appropriate for education, end user guidance, and some aspects of database design. Formal definitions facilitate analysis for consistency and completeness of the definition itself, specification of relational languages and systems, some aspects of database design, conformity of a DBMS implementation with a standard.

This report contains several examples of descriptions of the RDM. Section 2.2 lists several others. Table 2.1 compares the descriptions in this report based on the above criteria.

TABLE 2.1 COMPARISON OF SEVERAL RDM DESCRIPTIONS

RDM description	Section 2.2 explicit	Feature Catalogue implicit	Appendix A implicit	Appendix B explicit
purpose	essence of RDM, how RDM concepts go together, education	(implicit) basis for features of an RDBMS	functions constituting RDBMS interface	formal definition for implementation
audience	introduction for those knowledgeable in databases, e.g., students, DBSSG	RDBMS comparers, designers, and implementors	those concerned with DBMS standards, e.g., DBSSG, ANSI/X3	RDBMS designers and implementors, theoreticians
terminology	domain, relation,...	domain, relation,...	domain, relation,...	domain, relation,...
representation	sets; algebra and tuple calculus	sets/predicates/arrays; algebra/tuple calculus	sets; algebra and tuple calculus	sets; tuple calculus and algebra
notation	English	English	functions and English	VDM
Level of detail	essence and high level details	more detail than 2.2	little detail	complete detail
Precision	precise	imprecise	imprecise	precise
Formality	informal	informal	informal	formal

2.2 Examples of RDM definitions

Although there is considerable relational literature, relatively few papers give a complete definition of relational concepts. In the absence of a basic reference document and universally agreed upon concepts, different perceptions of the RDM have been developed, together with specific terminology, assumptions, and concerns, and often a particular data manipulation language. Some of these issues are discussed in Section 2.1.2.

The task of explicitly analyzing the variety of definitions, concepts, and languages as different perceptions of one relational model remains to be done. This section is a contribution to that effort. A number of papers are listed that have contributed significantly to the establishment of some relational concepts, or that provide typical perception of the RDM.

Beeri C., Bernstein P., Goodman N., "A Sophisticate's Introduction to Database Normalization Theory". VLDB Berlin, 1978.

This paper is a tutorial on a number of topics concerning relational theory: dependency constraints (such as functional and multivalued dependencies), normal forms, and relational schema design. It summarizes results and mentions a number of open issues (e.g., the importance of the universal relation assumption).

Bjørner D., "Formalization of Database Models". In: Abstract Software Specifications, Springer Lecture Notes in Computer Science 86, 1980.

Using the Vienna Definition Method (VDM) of denotational semantics definition, this paper gives formal descriptions of relations, the relational algebra, and the tuple relational calculus.

Codd E.F., "A Relational Model of Data for Large Shared Data Banks". CACM, Vol. 13, No. 6, June 1970.

The paper that started everything! Unlike scores of others, it ages very well. It presents a set-theoretic definition of relations, mentions the table view (called array representation), and describes the algebraic operations.

Codd E.F., "A Database Sublanguage Founded on the Relational Calculus". Proc. ACM SIGFIDET workshop, San Diego (November 1971).

This is the definition of ALPHA, derived directly from the tuple relational calculus, which has directly or indirectly influenced the design of many relational query languages.

Codd E.F., "Relational Completeness of Database Sublanguages". In: Data Base Systems, Courant Computer Science Symposium 6, Prentice-Hall (1972).

A standard reference for a definition of the relational algebra, the tuple relational calculus, and relational completeness. The paper also discusses the translation of calculus formulas into algebraic expressions.

Codd E.F., "Extending the Database Relational Model to Capture More Meaning". TODS, Vol. 4, No. 4 1979.

This paper contains a definition of basic concepts of the RDM, using set theory and rules for a tabular view of relations. The definition includes a version of the algebraic operations. The paper suggests two rules which control the representation of entities by relations and constrain the alteration operations of the RDM. The terms "fully relational" and "semirelational" are given a definition which was referenced in Section 1.5 of the Feature Catalogue [BROD80c]. The paper also proposes a number of extensions to the relational model (representations of entities, associations, treatment of null values, etc.).

Date, C.J., "An Introduction to Database Systems", Third Edition, Addison-Wesley, 1981.

The part of this book that deals with the RDM has been revised and expanded in the third edition. It includes among other things a set-oriented definition of relations, illustrated by table representations, a definition of relational algebra, the tuple, and domain relational calculi. Codd's definitions of "fully relational" and "semirelational" are given, and the observation is made that no currently existing systems are fully relational, or even semirelational.

Delobel C., "An Overview of the Relational Data Theory". IFIP Congress 1980.

This paper summarizes a number of results from the theory of relational databases and their application to database design. Among other topics, the following are described: decomposition of relations; the universal relation assumption; functional, multivalued, hierarchical and join dependencies, and their axiomatization; tableaux and their use to formulate problems of dependencies. The paper also contains a definition of relations which mentions both the set and the predicate aspects of a relation.

Gallaire H., Minker J. (Eds.), Logic and Data Bases, Plenum Press 1978.

This book contains a number of papers which explicitly deal with the relational model. The typical logician's view of a relation is that it represents a predicate on domains, and the typical calculus language is some version of domain relational calculus. The notion of a tuple (or record) is a natural one in the database field, but much less so in logic, artificial intelligence, and natural language studies.

Lacroix M., Pirotte A., "Domain Oriented Relational Languages". VLDB, Tokyo 1977.

This paper defines the domain relational calculus, where variables range over domain values rather than over relation tuples as in the tuple relational calculus. The paper also discusses the design of domain-oriented languages.

Sandberg G., "A Primer on Relational Database Concepts". IBM Systems Journal, Vol. 20, No. 1, 1981.

This paper presents a tabular view of the RDM. One advantage of the relational model is that its data structures can be informally described as tables. The following description is an informal definition of the table view of relations (other characterizations of similar descriptions are the "record view" or the "flat file view" of relations).

"The data are structured in the form of tables consisting of columns and rows, with the rows corresponding to records or segments, and the columns representing fields within the records. The following rules must be followed:

- Each table contains only one record type.
- Each record (row) has a fixed number of fields, all of which are explicitly named.
- Fields are distinct (atomic) so that repeating groups are not allowed.
- Each record is unique -- duplicates are not allowed.
- Records may come in any order; there is no predetermined sequence.
- Fields take their value from a domain of possible field values.
- The same domain may be used for many different field types, thus becoming the source of field values in different columns in the same or different tables."

This definition can be derived from the definition given in Section 2.3 if the following correspondence of terms is introduced:

- relation value = table
- relation scheme = record type (or table type)
- attribute = column name
- tuple = row = record
- N-ary relation = table with N columns

Ullman J.D., Principles of Database Systems, Computer Science Press 1980.

An important part of this book is devoted to relational concepts, with a set-theoretic definition of relations and a presentation of the basis of several query languages based on a thorough definition of tuple and domain relational calculus, relational algebra, and their correspondences.

2.3 Concepts, Terminology, and Examples

Disclaimer

Due to the pressure of time in completing the work of the RTG, it was not possible for this section of the Final Report to be reviewed by the full task group. This section was developed in substance by Joachim Schmidt, with contributions from Michael Brodie and Alain Pirotte, and with additional comments from other members of the group. Although this material does not necessarily represent the consensus of the group, it is included as an example of a relational data model description.

Essentially, a data model provides three kinds of mechanisms to support data management: structures, operations, and constraints. This section discusses the basic mechanisms with emphasis on the Relational Data Model (RDM). Terminology associated with relations and relational operations (such as relational algebra and calculus) are described.

2.3.1 Basic Concepts of the RDM

This section presents one view of the fundamental RDM concepts. Other views are possible as is indicated by general issues given in Section 2.1.2, by different definitions listed in Section 2.2, and by the following specific issues:

Can a relation schema have associated with it exactly one relation value or more than one relation value?

Are attribute names part of a relation value or not?

Should application-independent value sets, such as integers or strings, be considered to be domains?

Is it more appropriate to model a relation value as a time varying set of tuple values or as a time varying relation predicate? (A relation predicate defines set membership.)

The structuring mechanisms of a data model provide means for grouping elementary data items into compound data items. A fixed number of atomic data items can be composed to form structured data objects (such as tuples or records) and a variable number of data objects can be grouped into partitions (such as relations or DBTG sets). In the RDM, the partitions are defined so that:

- members of the same partition have the same structure, i.e., each member has the same number of components and corresponding components of different members have comparable values, i.e., values of the same kind;
- members of the same partition are distinct, i.e., a data object can exist only once in one partition; and
- members of different partitions may have comparable component values (this provides the basis for relating partitions and for constructing new partitions through operational means).

The operational mechanisms of a data model serve two purposes: first, to derive data from one or more partitions and, second, to alter data in a partition. In the RDM, the result of a derivation operation can be viewed as a partition. Derived data objects depend on the derivation operation in two ways:

- their values fulfill a selection criterion provided by the derivation operation; and
- their structure consists of components denoted explicitly or implicitly by the derivation operation.

As a result of an altering operation, the altered partition contains either more data objects, fewer data objects, or data objects with different component values.

Constraint mechanisms of a data model can be used to define and maintain time invariant properties of partitions that cannot be expressed directly by the structural and operational means. Constraints control the altering operations to maintain data integrity. Typically, RDM constraints are based on the value of data objects. The presence of some data objects can be made dependent on the presence or absence of other data objects. Constraints can be associated with individual relations, with collections of relations, and with transitions between database states.

2.3.2 Relations and Tuples

In the RDM, partitions are called relations. A relation has various distinct properties.

A relation value is the set of values of the data objects that are currently members of the relation. Derivation operations refer to the relation value. It is the relation value that is changed by an altering operation, thus, a relation value is a time varying property of a relation.

A relation schema defines the time invariant properties of a relation. Relation structure is one such property; a relation has the structure of a set of identically structured data objects.

Additional time invariant properties can be defined by relation constraints. One class of relation constraints enforces uniqueness of values of distinguished components (keys) throughout the data objects of a relation.

Example 1: Definition of a Relation and a Relation Schema; Display of a Relation Value

definition of a relation and a relation schema:

```
parts = relation pnr : partnumbers ;
                 pname : partnames ;
                 color : colortype ;
                 weight : weightunits ;
                 city : citynames
                 key <pnr>
        end ;
```

display of a value of relation "parts" :

pnr	pname	color	weight	city
p1	nut	red	12	Vienna
p2	bolt	green	17	Paris
p3	screw	blue	17	Rome
p4	screw	red	14	Vienna
p5	cam	blue	12	Paris
p6	cog	red	19	Vienna

Remark:

The key constraint defined for the relation "parts" requires uniqueness of pnr values.

Data objects in a relation are called tuples. Each tuple in a relation consists of the same number of components called attributes.

The time invariant properties of a tuple are defined by its tuple structure that associates with each attribute an attribute name and a domain. Domains are sets of atomic values that are used to define, for each attribute, the legal attribute values. Domains are also used to define attribute compatibility and thus tuple and relation

compatibility. Compatibility is required in derivation operations where attributes of one tuple are compared with attributes of another tuple. Compatibility is also required in altering operations where attribute values are altered or tuple values are inserted or removed.

Example 2: Definitions of User Defined Domains

```
partnumbers = string of [5] char ;
partnames = string of [20] char ;
colortype = (red, blue, yellow, green, black) ;
weightunits = 0..9999 ;
citynames = string of [25] char .
```

The time varying properties of a tuple are given by its value. A tuple value is defined by its attribute values just as a relation value is given by its tuple values. While the number of tuple values that define a relation value may vary in time, the number of attribute values that define a tuple value is fixed. Tuple values can be decomposed into attribute values; for that purpose attribute values are defined as pairs of attribute names and values. The values have to be elements of the domain associated with the attribute identifier by the tuple structure definition.

A relational database schema is an extension of a relation schema. It defines the time invariant properties of a number of relations that constitute a relational database.

A relational database schema defines three properties:

1. Database structure: the relations that constitute the database; each relation specified by its schema and relation name.
2. Database domains: the domains used for relation schema definition.
3. Database constraints: additional constraints can be defined. While relation constraints refer to one relation only, database constraints may affect more than one relation.

Example 3: Definition of a Relational Database Schema

```
partnumbers = string of [5] char ;
partnames = string of [20] char ;
colortype = (red, blue, yellow, green, black) ;
weightunits = 0..9999 ;
citynames = string of [25] char ;
suppliernumbers = string of [5] char ;
suppliernames = string of [20] char ;
quantity = integer ;
creditstatus = 0..999 ;

businessdatabase =
database parts = relation
                     pnr : partnumbers ;
                     pname : partnames ;
                     color : colortype ;
                     weight : weightunits ;
                     city : citynames
                     key <pnr>
                 end ;
     shipments = relation
                     snr : suppliernumbers ;
                     pnr : partnumbers ;
                     qty : quantity
                     key <snr,pnr>
                 end ;
     suppliers = relation
                     snr : suppliernumbers ;
                     sname : suppliernames ;
                     status : creditstatus ;
                     city : citynames
                     key <snr> .
                 end
end ;
```

2.3.3 Queries and Altering Operations

Database operations that derive relations are often called queries. The result of a query is a relation. This is the basis for a "closure" property of relational operations. A query definition has to provide the following information:

1. the "source relations", i.e., relations from which tuple values are selected to be transformed into the values of the result tuples;
2. the "selection criterion", i.e., a condition which source tuples have to fulfill to be selected; and
3. the "target transformation", i.e., rules that define how result tuple values are derived from selected tuple values.

Conceptually, the process of query evaluation consists of three steps:

1. provide access to tuples of the source relations;
2. evaluate the selection criterion for any tuple combination that can be formed according to 1); and
3. apply the target transformation to any tuple combination that fulfills the selection criterion, and store the result in a tuple of the result relation.

Example 4: The Three Steps of Query Evaluation

Query: Name the parts that are manufactured in Vienna and weigh less than 17 units.

Step 1: provide access to the tuples of the source relation, i.e.,

"parts" relation :

pnr	pname	color	weight	city
p1	nut	red	12	Vienna
p2	bolt	green	17	Paris
p3	screw	blue	17	Rome
p4	screw	red	14	Vienna
p5	cam	blue	12	Paris
p6	cog	red	19	Vienna

Step 2: evaluate the selection criterion, i.e., (city = "Vienna") and (weight < 17);

Criterion is :

	pnr	pname	color	weight	city
true	p1	nut	red	12	Vienna
false	p2	bolt	green	17	Paris
false	p3	screw	blue	17	Rome
true	p4	screw	red	14	Vienna
false	p5	cam	blue	12	Paris
false	p6	cog	red	19	Vienna

Step 3: apply target transformation to qualified tuples, i.e., choose pname values, and store the result in the result relation;

V

Result relation:

	pnr	pname	color	weight	city
-->	p1	nut	red	12	Vienna
	p2	bolt	green	17	Paris
	p3	screw	blue	17	Rome
-->	p4	screw	red	14	Vienna
	p5	cam	blue	12	Paris
	p6	cog	red	19	Vienna

derive result relation

V

pname
nut
screw

In the RDM, the result of a query evaluation is always a relation. This fact is the basis for what is called the "closure property" of the relational operators.

Dependent on the specific approach to relational query languages, the functionality required for querying a relational database is provided by different sets of operators.

The basic operations of the relational algebra approach can be characterized as follows:

1. Algebraic operators refer to a fixed number of relations; selection and projection are unary operators, i.e., they operate on one relation. Binary operators are union, intersection, difference, join, and division and operate on two relations.

2. Algebraic operators apply simple selection criteria: the selection operator applies a criterion that refers to constants and attribute values of individual tuples in one relation only.+ The join operator applies a selection criterion that refers to two attribute values of individual tuples in two relations. The division operator tests sets of attribute values in two relations for set inclusion.

3. Algebraic operators provide target transformations for tuple restructuring. Projection and division operators choose distinguished attribute values to form derived relations.

+This class of selection criteria can be defined formally by the expressions of the propositional logic.

Example 5: A Select Operation in the Relational Algebra Notation

Query: Derive data on suppliers that are either located in Paris or have a status different from 20.

Query definition (in PRTV Relational Algebra [TODD76]):
suppliers : (city = "Paris" or status ≠ 20)

"suppliers" relation :

snr	sname	status	city
s1	Smith	20	Vienna
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	Vienna
s5	Adams	30	Athens

Query evaluation: Select tuples of the "suppliers" relation that have either the value "Paris" in the attribute "city" or a value different from 20 in the attribute "status". Copy the entire value of each qualified tuple, i.e., all its attribute values, into the result relation.

Result relation :

snr	sname	status	city
s2	Jones	10	Paris
s3	Blake	30	Paris
s5	Adams	30	Athens

Remarks:

The select operator refers to only one relation, e.g., "suppliers". The selection criterion refers only to tuple attributes and constants; it contains comparison operators, such as =, <, >, ≤, ≥, ≠, and logical operators, such as and, or, not. The relation "suppliers" is only read, not altered.

Example 6: A Join Operation in the Relational Algebra Notation

Query: Derive data on the suppliers and on the shipments from these suppliers.

Query definition (in PRTV Relational Algebra [TODD76]):
suppliers * shipments

"suppliers" relation :

snr	sname	status	city
s1	Smith	20	Vienna
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	Vienna
s5	Adams	30	Athens

"shipments" relation :

snr	pnr	qty
s1	p1	300
s1	p2	200
s1	p3	400
s1	p4	200
s1	p5	100
s1	p6	100
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200
s4	p4	300
s4	p5	400

Query evaluation: Select tuple pairs from relations "suppliers" and "shipments" that have the same value in the snr attribute, i.e., the same supplier number. Copy the value of the qualified tuple pairs into the result relation (copy only one of the two identical snr values).

Result relation :

snr	sname	status	city	pnr	qty
s1	Smith	20	Vienna	p1	300
s1	Smith	20	Vienna	p2	200
s1	Smith	20	Vienna	p3	400
s1	Smith	20	Vienna	p4	200
s1	Smith	20	Vienna	p5	100
s1	Smith	20	Vienna	p6	100
s2	Jones	10	Paris	p1	300
s2	Jones	10	Paris	p2	400
s3	Blake	30	Paris	p2	200
s4	Clark	20	Vienna	p2	200
s4	Clark	20	Vienna	p2	200
s4	Clark	20	Vienna	p4	300
s4	Clark	20	Vienna	p5	400

Remarks:

In PRTV, attributes over which the join operation is formed are implicitly defined by the attribute names the joined relations have in common. In other languages, attributes to be compared must be named explicitly;

The above example shows an "equi-" join operation, i.e., compares attribute value for equality. In some languages, comparison operations other than equality can be applied.

Example 7: A Project Operation in the Relational Algebra Notation

Query: Derive the supplier names

Query definition (in PRTV Relational Algebra [TODD76]):

```
suppliers % sname
```

"suppliers" relation :

snr	sname	status	city
s1	Smith	20	Vienna
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	Vienna
s5	Adams	30	Athens

Query evaluation: For each tuple in the "suppliers" relation, select the attribute "sname", i.e., the supplier's name. Copy the value of the attribute "sname" into the result relation.

Result relation :

sname
Smith
Jones
Blake
Clark
Adams

Remarks:

A relation may have more than one tuple with the same value for a given set of (non-key) attributes. However, the result of a project operation on those attributes contains a certain value only once.

In the relational algebra approach, complex queries are expressed by nesting simple queries. Thus, algebraic query languages rely heavily on the closure property of its operators.

Example 8: A Join/Select/Project Operation in the Relational Algebra Notation

Query: Derive the names of those suppliers that shipped some part and either are located in Paris or have a status different from 20.

Query definition (in PRTV Relational Algebra [TODD76]):

```
suppliers * shipments : (city = "Paris"
                        or status ≠ 20) % sname
```

"suppliers" relation :

snr	sname	status	city
s1	Smith	20	Vienna
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	Vienna
s5	Adams	30	Athens

"shipments" relation :

snr	pnr	qty
s1	p1	300
s1	p2	200
s1	p3	400
s1	p4	200
s1	p5	100
s1	p6	100
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200
s4	p4	300
s4	p5	400

Query evaluation: Select tuple pairs from relations "suppliers" and "shipments" that have the same value in the attribute "snr" and have either the value "Paris" in the attribute "city" or a value different from 20 in the attribute "status". Copy the value of the attribute "sname" of each qualified tuple into the result relation.

Result relation :

sname
Jones
Blake

Query languages based on the relational calculus support the definition of complex queries by expressions. The basic expression of the tuple relational calculus approach can be characterized as follows:

1. Calculus expressions may refer to any number of source relations.

2. Calculus expressions may apply complex selection criteria that refer to constants, to tuples in source or any other relations. References to tuples are established through so-called tuple variables, free variables for the tuple to be selected, bound variables for the tuples that form the selection criterion.+

3. Calculus expressions may provide transformations to choose distinguished attribute values from the selected tuples or to apply standard computations such as average, maximum, or minimum.

Example 9: A Join/Select/Project Operation in the Tuple Relational Calculus Notation

Query: Derive the names of those suppliers that shipped some part and are either located in Paris or have a status different from 20.

Query definition (in PASCAL/R Relational Calculus [SCHM77]):

[<s.sname> of each s in suppliers :
((s.city = "Paris") or (s.status ≠ 20))
and some sp in shipments (s.snr = sp.snr)]

"suppliers" and "shipments" relation: see Ex. 8

Query evaluation: Select tuples of the "suppliers" relations that have the same value in the attribute "snr" as some tuple in the "shipments" relation and have either the value "Paris" in the attribute "city" or a value different from 20 in the attribute "status". Copy the value of the attribute "sname" of each qualified tuple into the result relation.

Result relation :

sname
Jones
Blake

Remarks:

The meaning and result of example query 9 is supposed to be identical to that of example query 8.

+This class of selection criteria can be defined formally by the expressions of the first-order predicate logic.

In the domain relational calculus, a query requests a set of tuples of values from "target" database domains. Only those tuples of values which satisfy a "selection criterion" are selected. This criterion is a property expressed in terms of values and additional simpler properties. In the domain calculus, relations express properties of values.

Conceptually, the process of query evaluation in the domain relational calculus consists of three steps:

1. access the elements of the source or target domains;

2. evaluate the selection criterion for any tuple of values that can be formed by taking one value from each target domain. Relations (more precisely, relational predicates) are accessed during that process. References are expressed through so-called domain variables, which range on database domains or on relation columns (i.e., single-attribute projections);

3. add each tuple of values that fulfills the selection criterion to the result relation.

Example 10: A Join/Select/Project Operation in the Domain Relational Calculus

Query: Derive the names of those suppliers that shipped some part and are either located in Paris or have a status different from 20.

Query definition (in DRC [LACK77b]):

{sn: ∃ s : suppliers (sname : sn, snr : s)
and (suppliers (snr : s, city : "Paris")
or suppliers (snr : s, status : ≠ 20))
and ∃ part : shipments (pnr : part, snr : s)}

Note that "sn", "s", and "part" are domain variables.

Query evaluation: Consider in turn the values of the "sname" attribute of relation "suppliers". Keep as an element of the result each value which occurs in relation "suppliers" with a supplier number "s", that occurs in "suppliers" either with "Paris" as a value of the "city" attribute or with a value different from 20 as a value of the "status" attribute, and that occurs in relation "shipments" with some part number "part" as the value of the "pnr" attribute.

Result relation :

sname
Jones
Blake

Remarks:

The meaning and result of examples 8, 9, and 10 are the same.

Since the evaluation of a relational calculus expression results in a relation, calculus expressions may be nested. The nesting structure of calculus expressions is that of the predicate-calculus; it is the nesting of ranges of bound variables in the selection criterion. Nesting of queries is necessary in the algebraic approach to express more complex queries. Nesting in the algebra is more obvious than in the calculus since it is operations that are nested not variable ranges as in the calculus.

Relational algebra, tuple relational calculus and domain relational calculus mark three positions in the spectrum of relational query languages. Many query languages designed to improve language usability have characteristics of all three approaches.

Example 11: A Join/Select/Project Operation in a notation that combines algebra and calculus features

Query: Derive the names of those suppliers that shipped some part and either are located in Paris or have a status different from 20.

Query definition (in SQL [CHAM76b]):

```
select sname
   from suppliers
      where (city = "Paris" or status ≠ 20)
      and snr in (select snr
                  from shipments)
```

"suppliers" and "shipments" relation: see Ex. 8

Query evaluation: Select tuples of the "suppliers" relation whose values in the attribute "snr" are in the set of "snr" values of the "shipments" relation and have either the value "Paris" in the attribute "city" or a value different from 20 in the attribute "status". Copy the value of the attribute "sname" of each qualified tuple into the result relation.

Result relation :

sname
Jones
Blake

Remarks:

The meaning and result of example query 11 is identical to that of example queries 8, 9, and 10.

In order to alter the value of a relation, its name and its new value have to be provided. In principle, a single altering operator that assigns the new value to the named relation would suffice. There are, however, notational and implementational advantages in distinguishing three altering operators.

1. The insert operator alters a relation value by inserting new tuples into the relation. An insert operation has to provide the name of the relation and the values for the new tuple.

   If none of the relational constraints ar violated, the new relation value is equal t the value defined by the union of the ol tuples and the new tuple values.

2. The delete operator alters a relation value b deleting tuples. A delete operator has t identify the relation to be altered and th tuples to be deleted.

   If none of the relational constraints ar violated, the new relation value is equal t the set difference of the old tuple values an the new tuple values.

   Since the tuples to be deleted are neces sarily elements of the relation to be altered the query mechanisms can be used to selec them.

3. The replace operator alters a relation value b replacing the value of distinguished tuples. replacement operation must name the relation t be altered, identify the tuples that will ge new values, and provide new values for thes tuples. The effect of a replacement operatio can also be achieved by a combination of delet and insert operations.

Example 12: An Insert Operation

Operation: Insert tuples with the values
<s7, Green, 7.5, Zurich>,
<s3, Smith, 15, Paris>,
<s6, Maier, 1, Vienna>,
<s8, Brown, 13, 1982, Toronto>
into "suppliers" relation.

Insert statement (in PASCAL/R [SCHM77]):

```
suppliers :+ [<s7, Green, 7.5, Zurich>,
              <s3, Smith, 15, Paris>,
              <s6, Maier, 1, Vienna>,
              <s8, Brown, 13, 1982, Toronto>]
```

"suppliers" relation (before operation):

snr	sname	status	city
s1	Smith	20	Vienna
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	Vienna
s5	Adams	30	Athens

Insert operation: new tuples with the above values are supposed to be inserted into the relation "suppliers." The RDBMS has to make sure that none of the structural or integrity rules defined for the relation "suppliers" are violated by the insert operation.

"suppliers" relation (after operation):

snr	sname	status	city
s1	Smith	20	Vienna
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	Vienna
s5	Adams	30	Athens
s6	Maier	1	Vienna

Remarks:

Three out of the four tuple values to be inserted

violate structural, domain, or integrity constraints defined for the relation "suppliers": tuple value <s7, Green, 7.5, Zurich> has status value 7.5 which is not an integer (domain violation). Tuple value <s3, Smith, 15, Paris> has key value s3 which is already present in the relation "suppliers" (key violation). Tuple value <s8, Brown, 13, 1982, Toronto> has five attributes instead of four as defined for relation "suppliers" (structure violation).

There are different ways of handling operations that would violate a structural or integrity rule. Depending on its nature, a violation can be detected at parsing time (e.g., the violations caused by the tuples with key values s7 and s8) or at execution time (e.g., the violation caused by the tuple with key value s3). Execution time violations may be "resolved" by, e.g., program abortion, trigger invocation, or by eliminating the tuple values that cause the violation (as in the example above).

3. ANALYSIS OF RELATIONAL SYSTEMS

In order to "identify or establish aspects of the RDM and RDBMSs that might be appropriate for standards development", the RTG developed a "Feature Catalogue" of Relational Concepts, Languages, and Systems". The catalogue was intended as an abstract (implementation independent) characterization of RDM and RDBMS concepts. It was used as a common basis for a detailed feature analysis of a number of existing DBMSs that were claimed to support aspects of the RDM. Due to limited resources, only a few DBMSs could be examined.

The objective of this chapter is to identify potential features of a relational standard. The feature analyses of 12 DBMSs were used to prepare this chapter.(1) Different interpretations of the base documents(2) made it necessary in some cases to go back to the originators of the feature analysis. Their answers were not always reflected in the original feature analysis. Furthermore, this also led to some inevitable overlap between the different sections of this chapter.

The structure of this chapter follows closely that of the feature catalogue. Features from each section of the catalogue are summarized in tabular form.

3.1 Development of the Feature Catalogue

A feature catalogue(2) was developed to guide the analyses of selected relational database management systems. The purpose of using a feature catalogue was to ensure that all systems were described in the same format and that all features of interest to the RTG were considered in the analyses.

The object of the analyses was to get insight into:

1. RDM:
   a. essential or core concepts, and
   b. additional concepts or extensions.
2. Systems issues concerning the support and use of the RMD:
   a. aspects closely tied to the model, such as the design of interfaces, and
   b. general system features, e.g., locking, concurrency, security, access control.

Nonrelational features were included in the feature catalogue for two reasons:

1. The analyses were required to decide whether a particular common DBMS feature was related to the RDM or not. One objective of performing the analyses was to determine what DBMS features were dependent on the RDM.
2. One reason for supporting the development of a relational standard is the task group's finding that there are a significant number of complete, useable DBMSs supporting aspects of the RDM. If the only implementations of the RDMs were research vehicles then a standard might be premature. Therefore it was important to determine the full capabilities of the systems being analyzed.

The intent of the task group was to categorize the features on the basis of the analyses according to the following taxonomy [ARTH80]:

1. Features that appear in every system which are claimed to be relational.
2. Features that do not appear in every system but are rapidly being added to many systems.
3. Features that are felt to be important but do not yet appear except in the literature or in research systems.
4. Features that have been in some systems for some time but do not appear to be spreading to other systems.
5. Features that distinguish the relational approach from other approaches.

Features in categories 1 and 2 would form the basis for a standard, while items from categories 3 and 4 could be used as a basis for future developments. Features in Section 5 would be used to exclude nonrelational systems.

As well as supporting a standard, the analyses based on the feature catalogue are expected to be useful in the following ways:

1. To relate specific knowledge of the state of the art to research and development results.
2. To demonstrate the amount and nature of work in implementations, applications, and literature.
3. To determine to what extent RDM concepts are constrained by implementation concerns.

The feature catalogue on which the analyses are based is the third version developed by the RTG. The initial version was based on the Generalized DBMS Survey completed by the CODASYL System Committee in 1969. This version was tested by asking members to analyze selected systems. The catalogue was found to be incomplete because some hierarchical DBMSs could be described as "relational" following the catalogue. A second version of the catalogue placed more weight on purely relational features, but was too detailed for our purposes and somewhat repetitive. In the third version of the feature catalogue this problem is not completely solved. However, the overall organization of information is clear and easy to follow.

The relative importance of relational concepts, languages, and systems is reflected in the structure of the catalogue. First, RDM concepts are considered, followed by associated functional capabilities and ending with DBMS features. Within the functional capabilities, query and manipulation

---

(1) These analyses will be published in Brodie, M.L., and Schmidt, J.W. (Eds.), "Relational Database Management Systems: Analysis and Comparisons," Springer-Verlag, Heidelberg/New York, in press.

) "RTG 80-81 Feature Catalogue of Relational Concepts, Languages, and Systems" and "RTG 80-90 Notes on Completing an RTG Feature Analysis."

are considered before definition, generation, and administration. This ordering indicates the distinction between RDM concepts and RDBMS features in response to the RTG charter.

The following systems were analyzed using the final feature catalogue:

IDM (Britton-Lee)
INGRES (University of California, Berkeley)
IPIP (Boeing Computer Services) -- incomplete feature analysis
MRDS (Honeywell)
MRS (University of Toronto)
NOMAD (National CSS, Inc.)
ORACLE (Relational Software Incorporated)
PASCAL-R (Hamburg University)
PRTV (IBM, United Kingdom)
QBE (IBM, Thomas J. Watson)
RAPID (Statistics Canada)
RAPPORT (LOGICA, United Kingdom)
SYSTEM R (IBM San Jose)

3.2 Database Constituents

3.2.1 Definitions

Analysis of the Feature Catalogue indicated that the following seven constituents appeared in a majority of the systems evaluated:

1. Database. The relational database is a collection of relations. Depending on the system, relations can be base relations, views, and/or snapshots.

2. Relation. A relation is a set of relation elements of the same type. A relation may be presented to the user as a table of rows and columns.

3. Tuple. A relation element is called a tuple. A tuple is a row in the tabular presentation of a relation.

4. Attribute. The attributes of a tuple are defined by the tuple type. The legal values of an attribute are chosen from a specific set of values, called domains.

5. User defined domain. The set of values over which an attribute can range is called a domain. A user defined domain allows the user to describe specific domains, such as the domain COLOR where COLOR can be red, yellow, blue, or green.

6. View. A view is a relation derived from existing relations. After its declaration, modifications to its defining relations are reflected in the view, according to its definition.

7. Snapshot. A snapshot is a relation derived from existing relations. After its declaration, it is independent of its defining relations and does not reflect updates made to its defining relations.

The following concepts were listed as constituents for systems that were analyzed.

8. Transaction/Lock. A transaction/lock is a set of queries to be executed without interference from any other query. A lock is normally an interactive command, while a transaction is usually batch. In some systems, however, a transaction can be issued in either batch or interactive mode.

9. Trigger. A trigger is a set of user specified commands that are executed automatically after a specific situation has occurred in the database. For example, modification of a specific relation may cause a trigger to be activated.

10. Authority/Assertion. Authority and asserti allow control privileges over a relation.

11. Relational Dictionary. A data dictionary which is presented to the user in relational form and can be queried using relational operators.

12. Relation Logging. A user can request that a log be maintained of all atomic changes to a specified relation.

13. Workspace. A workspace is an area where a relation can be manipulated and changed without affecting the database.

14. Image. An image defines the output picture for the data.

15. Index. An index is an access mechanism that can be defined for faster access.

16. Macro. A macro is a set of queries defined by a name. When the name is issued, the macro is automatically executed.

3.2.2 Results

The results are displayed in Table 3.1.

3.2.3 Observations on Database Constituents

The twelve feature analyses indicate unifo mity in the following database constituents: database, relation, tuple, and attribute. Snapshots and views, while not supported by all of the systems, are well represented with 9 systems supporting snapshots and 8 systems supporting views. PASCAL/R supports snapshots as a temporary relation only. The user defined domain was supported by only half of the systems. Under the heading of other database constituents it was found that the transaction, relational dictionary, and macro were the most prevalent.

3.3 Functional Capabilities

Thirty two features of the functional capabilities of an RDBMS were considered. They involved qualification, retrieval and presentation, alteration, and additional operations. Table 3.2 characterizes these features for the 12 DBMSs and presents a summary.

3.4 Schema Definition

3.4.1 Descriptions

Schema definition features of a system can put into 11 categories. A description of each fo lows:

TABLE 3.1 DATABASE CONSTITUENTS

	IDM	INGRES	MRDS	MRS	NOMAD	ORACLE
1. Database	Yes	Yes	Yes	Yes	Yes	Yes
2. Relation	Yes	Yes	Yes	Yes	Yes	Yes
3. View	Yes	Yes	Yes	No	Yes	Yes
4. Snapshot	Yes	Yes	Yes	No	Yes	Yes
5. Tuple	Yes	Yes	Yes	Yes	Yes	Yes
6. Attribute	Yes	Yes	Yes	Yes	Yes	Yes
7. Domain (user defined)	No	No	Yes	No	No	No
8. Other	Transaction, Macro, Relation Logging, Index	Index Macro, Authority	Transaction, Macro	Index Transaction		Macro, Transaction

TABLE 3.1 DATABASE CONSTITUENTS (continued)

	PASCAL/R	PRTV	RAPPORT	SYSTEM R	QBE	RAPID
1. Database	Yes	Yes	Yes	Yes	Yes	Yes
2. Relation	Yes	Yes	Yes	Yes	Yes	Yes
3. View	No	Yes	No	Yes	No	Yes
4. Snapshot	Yes	Yes	No	Yes	Yes	No
5. Tuple	Yes	Yes	Yes	Yes	Yes	Yes
6. Attribute	Yes	Yes	Yes	Yes	Yes	Yes
7. Domain (user defined)	Yes	Yes	No	No	Yes	Yes
8. Other	Relational Dictionary	Workspace Macro	Trigger, Transaction, Index	Index Transaction, Dictionary	Macro Authority, Image, Index Transaction, Dictionary	Index

TABLE 3.1 DATABASE CONSTITUENTS SUMMARY

Database	Supported by 12 Systems
Relation	Supported by 12 Systems
View	Supported by 8 Systems
Snapshot	Supported by 9 Systems
Tuple	Supported by 12 Systems
Attribute	Supported by 12 Systems
User Defined Domain	Supported by 5 Systems

1. Define Database. An explicit command allows a user to define an entire database. Defining one or more relations is not considered to be an explicit command for defining an entire database.

2. Generate Database. An explicit command exists for populating a database. In addition to this, the generation facility can dump data from the database and into the database, i.e., a fast load/unload facility.

3. Destroy Database. An explicit command exists that will destroy an entire database. Destroying one relation at a time is not considered to be an explicit command to destroy a database.

4. Define Relation. An explicit command defines a relation.

5. Destroy Relation. An explicit command destroys a relation.

6. Reorganize Relation. An explicit command reorganizes a relation.

7. Define Snapshot. An explicit command defines a snapshot.

8. Define View. An explicit command defines a view.

9. Drop Attribute. An explicit command causes an attribute to be deleted from a relation if the original relation still exists. A projection command that chooses all but the unwanted attribute (and forms this into a relation) is not considered to be an explicit command to delete an attribute.

10. Add Attribute. An explicit command will add an attribute to a relation. A join command that forms another relation (which includes all attributes of the original relation and an additional attribute) is not considered to be an explicit command to add an attribute.

TABLE 3.2 OPERATIONS

	IDM	INGRES	MRDS	MRS	NOMAD	ORACLE
1. Retrieval tuple handling	Duplicates removed under user request	Duplicates removed under user request	Duplicates removed under user request	Duplicates removed under user request	Duplicates never removed	Duplicates removed under user request
2. Nesting	Fully	Fully	Fully	Restricted	Restricted	Fully
3. Closure	Fully closed	Fully closed	Restricted		Fully closed	Fully closed
4. Set membership operator	No	No	No		Yes	Yes
5. Set operators (union, intersect, difference, equality)	No	No	Yes	No	No	No
6. Group by operator	Yes	Yes	No	Restricted	Yes	Yes
7. Arithmetic operators	Yes	Yes	Yes	No	Yes	Yes
8. String operators	Yes	No	Yes	No	Yes	No
9. Transaction operators	Yes	No	No	No	No	Yes
10. Existential quantifier	Explicit operation	Implied	Implied	Implied	Implied	Implied
11. Universal quantifier	Implied	Implied	Implied		Implied	Implied
12. Boolean expressions	Yes	Yes	Yes	Yes	Yes	Yes
13. Query language environment	Stand alone	Stand alone and host			Stand alone	Stand alone
14. Self-join capability	Full	Full	Full	None	Full	Full
15. Equi-join capability	Full	Full	Full	Full	Full	Restricted
16. Natural join capability	Full	Full	Full	Full	Full	Restricted
17. Projection capability	Implicit	Implicit	Implicit	Implicit	Implicit	Implicit
18. Sort capability	Explicit	No	No	No	Explicit	Explicit
19. Insert	Explicit	Explicit	Explicit	Explicit	Explicit	Explicit
20. Delete	Explicit	Explicit	Restricted	Explicit	Explicit	Explicit
21. Modify	Explicit	Explicit	Explicit and restricted	Explicit	Explicit	Explicit
22. Update over more than one relation	No	Yes			No	No
23. Commit/undo	Explicit	No	No		Implicit	No
24. Triggers	No	No	No	No	Yes	No
25. Aggregate functions	Yes	Yes	Yes	Yes	Yes	Yes
26. User-defined functions	No	No	Yes	No	Yes	No
27. Library functions	No	Yes			Yes	Yes
28. User-defined error conditions (exit capabilities)	Yes	Yes	No	No	Yes	No
29. Format of schema information	Relational	Relational	Other	Other	Other	Relational
30. System performance data available	Yes	No	No	No	Yes	No
31. Report generator	No	Yes	No	No	Yes	Yes
32. Relationally complete	Yes (self-contained)	Yes (self-contained)			Yes (self-contained)	No

11. Rename Attribute. An explicit command will rename an attribute. A command which allows a synonym for an attribute name is not considered to be a command which renames an attribute. The original name of the attribute must not exist after this command is issued.

12. Define Tuple. An explicit command defines a tuple.

Other schema definitions listed for the systems include displaying and modifying a directory, specifying backup, defining a secondary index, checking a domain, grouping a set of relations, encoding and decoding data, and opening and closing a relation.

3.4.2 Results

The results are displayed in Table 3.3.

TABLE 3.2 OPERATIONS (continued)

	PASCAL/R	PRTV	RAPPORT	SYSTEM R	QBE	RAPID
1. Retrieval tuple handling	Duplicates always removed	Duplicates always removed	Duplicates never removed	Duplicates removed under user request	Duplicates not allowed	Duplicates removed under user request
2. Nesting	Fully	Fully	None	Fully	Fully	Restricted
3. Closure	Fully closed	Fully closed	No	Fully closed	Fully closed	Fully closed
4. Set membership operator	Yes	No	Yes	Yes	No	
5. Set operators (union, intersect, difference, equality)	Yes	Yes	No	Union-supported	Yes	Yes
6. Group by operator	No	Yes	Restricted	Yes	Yes	No
7. Arithmetic operators	Yes	Yes	Yes	Yes	Yes	No
8. String operators	Yes	Yes	No	No	No	No
9. Transaction operators	No	No	Yes	Yes	No	Yes
10. Existential quantifier	Explicit operation	Implied	Implied	Implied	Implied	Implied
11. Universal quantifier	Explicit operation	Implied	No	Implied	Implied	Implied
12. Boolean expressions	Yes	Yes	Limited	Yes	Yes	Yes
13. Query language environment	Stand alone and host	Stand alone	Stand alone and host	Stand alone and host	Stand alone and host	Stand alone
14. Self-join capability	Full	Full	No	Full	Full	Full
15. Equi-join capability	Full	Full	No	Full	Full	Full
16. Natural join capability	Full	Full	No	Full	Full	Full
17. Projection capability	Explicit	Explicit	Restricted (only on output)	Implicit	Explicit	Restricted (only on output)
18. Sort capability	Restricted	No	Explicit	Explicit	Explicit	No
19. Insert	Explicit	Implicit	Explicit	Explicit	Explicit	Explicit
20. Delete	Explicit	Implicit	Explicit	Explicit	Explicit	Explicit
21. Modify	Explicit	Implicit	Restricted	Explicit	Explicit	Explicit
22. Update over more than one relation	No	No	No	Yes	Yes	No
23. Commit/undo	No	No	Explicit	Explicit	Explicit	No
24. Triggers	No	No	No	No	No	No
25. Aggregate functions	No	Yes	Yes	Yes	Yes	No
26. User-defined functions	Yes	Yes	Yes	No	Yes	No
27. Library functions	No	Yes	No	Yes	Yes	No
28. User-defined error conditions (exit capabilities)	No	No	Yes	Yes	No	Yes
29. Format of schema information	Relational	Other	Non-existent	Relational	Relational	Other
30. System performance data available	Yes	No	Yes	No	No	Yes
31. Report generator	Yes	No	No	No	No	No
32. Relationally complete	Yes (self-contained)	Yes (self-contained)	No contained)	Yes (self-contained)	Yes (self-	No

3.4.3 Observations

In general, the systems reviewed dealt with relations rather than with the database itself. systems had the ability to define a relation, a great majority allowed the user to destroy reorganize relations and define snapshots and views. Only eight systems had an explicit command to define a database, and of those only six had an explicit command to populate the database. This is because in several systems, the database is defined and populated implicitly when the relations are defined and generated. Most systems had explicit commands to drop, add, and rename attributes. Only two systems explicitly defined tuples. In the other systems, a tuple is defined implicitly when the relation of which it is an element is defined.

TABLE 3.2 OPERATIONS SUMMARY

1. Retrieval tuple handling	Duplicates removed under user request:	7
	Duplicates always removed:	2
	Duplicates never removed:	2
	Duplicates not allowed:	1
2. Nesting	Full nesting:	8
	Restricted nesting:	3
	Nesting not allowed:	1
3. Closure	Full closure:	7
	Restricted closure:	2
	Not closed:	1
4. Set membership operator	Supported by 5 systems	
	Not supported by 5 systems	
5. Set operators	Supported by 5 systems	
6. Group by operator	Full:	7
	Restricted:	2
	Not supported:	3
7. Arithmetic operators	Supported by 10 systems	
8. String operators	Supported by 5 systems	
9. Transaction operators	Supported by 5 systems	
10. Existential Quantifier	Explicit:	2
	Function provided by other operator combinations:	10
11. Universal quantifier	Explicit:	1
	Function provided by other operator combinations:	9
	Function not supported:	1
12. Boolean expressions	Full:	11
	Limited:	1
13. Query language environment	Stand alone:	4
	Stand alone and host:	5
14. Self-join capability	Supported by 10 systems	
15. Equi-join capability	Full:	10
	Restricted:	1
	Not supported:	1
16. Natural join capability	Supported by 10 systems	
17. Projection capability	Explicit:	3
	Implicit:	7
	Restricted to output:	2
18. Sort capability	Explicit:	6
	Restricted:	1
	Not supported:	5
19. Insert	Explicit:	11
	Implicit:	1
20. Delete	Explicit:	10
	Implicit:	1
	Restricted:	2
21. Modify	Explicit:	9
	Restricted:	2
	Implicit:	1
22. Update over more than 1 relation	Supported by 3 systems	
23. Commit/undo	Explicit:	4
	Implicit:	1
	Not supported:	6
24. Triggers	Supported by 1 system	
25. Aggregate functions	Supported by 10 systems	
26. User defined functions	Supported by 6 systems	
27. Library functions	Supported by 5 systems	
28. User defined error conditions	Supported by 6 systems	
29. Format of schema information	Relational:	6
	Other:	5
30. System performance data available	Yes:	5
	No:	7
31. Report generator	Yes:	4
	No:	8
32. Relationally complete	Yes:	7
	No:	2

3.5 Additional Definition, Generation, and Administration Facilities

3.5.1 Background

Additional definition, generation, and administration facilities are shown in Table 3.4. These facilities were chosen based on the importance of the relational model as determined by the RTG. Facilities are considered to be not applicable (N/A) if the constituent about which the facility is defined is not supported in the system.

1. Attribute Synonyms. An explicit command exists to define a synonym for an attribute name of a relation.

2. Relation Synonyms. An explicit command exists to define a synonym for a relation name.

TABLE 3.3 SCHEMA DEFINITIONS

	IDM	INGRES	MRDS	MRS	NOMAD	ORACLE
Define database	Yes	Yes	Yes	No	Yes	No
Generate Database	Yes	Yes	No		Yes	No
Fast load/unload	Yes	Yes	No		Yes	No
Destroy database	Yes	Yes	No	Yes	Yes	No
Define relation	Yes	Yes	Yes	Yes	Yes	Yes
Destroy relation	Yes	Yes	No	Yes	Yes	Yes
Reorganize relation	Yes	No	No	Yes	Yes	No
Define snapshot	Yes	Yes	Yes		Yes	Yes
Define view	Yes	Yes	Yes		Yes	Yes
Drop attribute	No	No	No	Yes	Yes	No
Add attribute	No	No	No	Yes	Yes	Yes
Rename attribute	Yes	No	Yes	Yes	Yes	No
Define tuple	No	No	No		No	No
Other	Open, close		Index, check domain	Display database, backup	Group a set of relations, encode/decode	

TABLE 3.3 SCHEMA DEFINITIONS (continued)

	PASCAL/R	PRTV	RAPPORT	SYSTEM R	QBE	RAPID
Define database	Yes	Yes	Yes	Yes	Yes	Yes
Generate Database	Yes	No	Yes	Yes	Yes	
Fast load/unload	Yes	No	Yes	Yes	Yes	
Destroy database	Yes	Yes		Yes	Yes	
Define relation	Yes	Yes	Yes	Yes	Yes	Yes
Destroy relation	No	Yes		Yes	Yes	Yes
Reorganize relation	Yes	Yes	Yes	No	Yes	No
Define snapshot	Yes	Yes		Yes	Yes	No
Define view	No	Yes	No	Yes	No	Yes
Drop attribute	No	Yes		Yes	Yes	Yes
Add attribute	No	Yes		Yes	Yes	Yes
Rename attribute	Yes	Yes	Yes	No	Yes	Yes
Define tuple	Yes	No	No	No	No	Yes
Other						Open, close, status

TABLE 3.3 SCHEMA DEFINITIONS SUMMARY	
Define database	Supported by 10 systems
Generate database	Supported by 7 systems
Fast load/unload	Supported by 7 systems
Destroy database	Supported by 8 systems
Define relation	Supported by 12 systems
Destroy relation	Supported by 9 systems
Reorganize relation	Supported by 7 systems
Define snapshot	Supported by 9 systems
Define view	Supported by 8 systems
Drop attribute	Supported by 6 systems
Add attribute	Supported by 7 systems
Rename attribute	Supported by 9 systems
Define tuple	Supported by 2 systems

TABLE 3.4 ADDITIONAL FACILITIES

	IDM	INGRES	MRDS	MRS	NOMAD	ORACLE
Attribute synonyms	Yes	No	Yes	No	Yes	Yes
Relation synonyms	Yes	No	Yes	No	No	Yes
Duplicate tuples	Yes	No	Yes (view) No (snpsht)	Yes	Yes	Yes
Can view be defined over more than one relation	Yes	Yes	Yes	N/A*	Yes	Yes
Can view be defined from other views	Yes	No	Yes	N/A	No	Yes
User/DBA can define mapping to explicit update view	No	No	No	N/A	No	No
Selection criterion	Yes	Yes	Yes	Yes	Yes	Yes
Aggregate function	Yes	Yes	Yes	Yes	Yes	Yes
Primary keys required	No	No	Yes	No	No	No
Key can be modified	Yes	Yes	No	N/A	Yes	Yes
Key can be declared unique	Yes	No	Yes	N/A	Yes	Yes
Concatenated primary key	Yes	No	Yes	N/A	Yes	No
Views inherit keys	No	Yes	Yes	No	No	N/A
Null Valued keys allowed	No	No	Yes	No	Yes	No

*N/A = not applicable

TABLE 3.4 ADDITIONAL FACILITIES (continued)

	PASCAL/R	PRTV	RAPPORT	SYSTEM R	QBE	RAPID
Attribute synonyms	Yes	No	Yes	No		No
Relation synonyms	Yes	Yes	No	Yes		No
Duplicate tuples	No	No	No	Yes	No	Yes
Can view be defined over more than one relation	N/A	Yes	N/A	Yes	N/A	No
Can view be defined from other views	N/A	Yes	N/A	Yes	N/A	
User/DBA can define mapping to explicit update view	N/A	No	N/A	Yes	N/A	
Selection criterion	Yes	Yes	Yes	Yes	Yes	Yes
Aggregate function	No	Yes	Yes	Yes	Yes	No
Primary keys required	Yes	No	Yes	No	No	Yes
Key can be modified	No	N/A		Yes		Yes
Key can be declared unique	Yes	N/A	Yes	Yes	Yes	No
Concatenated primary key	Yes	N/A	Yes	Yes	Yes	Yes
Views inherit keys	N/A	N/A	N/A	Yes	N/A	No
Null Valued keys allowed	No	N/A	No	Yes	No	Yes

3. Duplicate Tuples. The system supports storage of duplicate tuples. This may be due to the way the system has been implemented or it may be a user option.

4. Can a view be defined over more than one relation, e.g., in the case of join, union, intersection, and relative complement operations?

5. Can a view be defined from other views, i.e., can a dynamic derived relation be used in the definition of another dynamic derived relation?

6. User/DBA can define a mapping to update a view, i.e., a view can be explicitly updated, causing updates to be mapped down to its defining relations?

7. Selection Criterion. Tuples of a relation can be selected based on meeting a user specified criterion.

8. Aggregate Function. The system supports aggregate functions such as MIN, MAX, and COUNT.

9. Primary Keys. The system requires primary keys as part of the logical data definition.

10. Keys can be modified. A user can modify a key value.

11. Keys can be declared unique. Once a key is declared, its value must be unique for each tuple. Note that this feature allows, at the

TABLE 3.4 ADDITIONAL DEFINITION, GENERATION, AND ADMINISTRATION FACILITIES SUMMARY

Attribute synonyms	Supported by 6 systems
Relation synonyms	Supported by 6 systems
Duplicate tuples	Supported by 8 systems
Can view be defined over more than one relation	Supported by 7 systems
Can view be defined from other views	Supported by 5 systems
User/DBA can define mapping to update view	Supported by 1 systems
Selection criterion	Supported by 12 systems
Aggregate function	Supported by 10 systems
Primary keys required	Supported by 4 systems
Key can be modified	Supported by 6 systems
Key can be declared unique	Supported by 9 systems
Concatenated primary key	Supported by 8 systems
Views inherit keys	Supported by 3 systems
Null valued keys allowed	Supported by 4 systems

user option, definitions of "primary" and "candidate" keys for certain relations.

12. Concatenated Key. A key can consist of more than one attribute of a relation.

13. Views inherit keys. When a view is declared, the key is inherited from its underlying relations.

14. Null valued keys are supported. A primary key may have a null value.

3.5.2 Results

The results are displayed in Table 3.4.

3.5.3 Observations

The selection criterion facility is supported by all of the systems. Often used with selection is the aggregate function, which is found in ten of the systems. Eight of the systems support duplicate tuples; however, in most of these systems, the user can specify if he/she wishes to see unique tuples only. Synonyms are allowed for both attributes and the relations half of the systems. The domain of an attribute can be operated on in only five of the systems. An example of such an operation might be limiting the domain of an attribute SALARY such that the fixed decimal value is greater than zero and less than $100,000. In most systems the domain is an implicit entity used in the definition of attributes.

Primary keys are supported in 10 of the systems. In four systems — SYSTEM R, ORACLE, IDM, and NOMAD -- the key is unique only if the user specifies. A majority of the systems that supported keys allowed modification of key values and allowed a key to be a set of concatenated attribute values.

Views can be defined in terms of more than one ...tion in seven of the nine systems which support views. Less than half of the systems allow views to be defined in terms of other views, to inherit keys of their defining relations, and to be defined prior to run time. Only SYSTEM R allows the explicit update of views.

3.6 Functional Classes

3.6.1 Definitions

In analyzing the feature catalogue, 12 different functional classes were defined. These classes were used to group logically relational functions and are described below.

1. Database Schema Definition. These functions are included in interfaces 1, 4 of ANSI/SPARC DBMS framework and include the conceptual and external schema source definitions.

2. Database Retrieval. These functions include the set oriented and record-at-a-time oriented retrievals from the database. They include functions found in interfaces 7, 8, 9, 10, 11 of the ANSI/SPARC DBMS framework.

3. Database Altering. These functions include the set oriented and record-at-a-time oriented inserts, modifications, and deletes in the database. They include functions found in interfaces 7, 8, 9, 10, 11 of the ANSI/SPARC DBMS framework.

4. Integrity Constraints. These functions include the definitions of constraints on data values and their relationships. They include functions that would be in interfaces 1, 4 of the ANSI/SPARC DBMS framework.

5. Database Generation and Regeneration. These functions include facilities, separate from the database altering functions, for logical bulk loading, unloading, and reloading of relations. These functions can be found in interface 39 of the ANSI/SPARC DBMS framework.

	IDM	INGRES	MRDS	MRS	NOMAD	ORACLE
1. Database schema definition	QL commands dynamic	QL commands dynamic	Utility static	QL commands dynamic	QL commands dynamic	QL dynamic
2. Database retrieval Set-oriented	Yes	Yes	Yes	Yes	Yes	Yes
Record-at-a-time		Yes	Yes		Yes	Yes
3a. Database insert Set-oriented	Yes	Yes				Yes
Record-at-a-time		Yes	Yes	Yes	Yes	Yes
3b. Database modify Set-oriented	Yes	Yes	Yes	Yes	Yes	Yes
Record-at-a-time			Yes	Yes	Yes	Yes
3c. Database delete Set-oriented	Yes	Yes	Yes	Yes	Yes	Yes
Record-at-a-time			Yes	Yes	Yes	Yes
4. Integrity constraints	QL commands	QL commands	DD utility		DD command	
5. Database generation, regeneration	Logical bulk load	Logical bulk load	Logical bulk load	Logical bulk load	Logical bulk load	Logical bulk load
6. Database schema redef. & renaming	Indirect	Indirect	Indirect	Indirect	Indirect	Indirect
7. Report generation		Print command	Print command		Print command	Utility
8. Special data entry			Prompting			Prompting
9. Security, monitoring	QL commands	QL commands	DD utility	QL commands	QL commands	DD utility
10. LOAD/DUMP/recovery	Yes	Yes	Yes	Yes	Yes	
11. Definition of access paths	QL commands	QL commands	DD utility	QL commands	QL commands	QL commands
12. Database dictionary	QL access	QL access	Special commands	QL access	Special commands	QL access

TABLE 3.5 FUNCTIONAL CLASSES (continued)

	PASCAL/R	PRTV	RAPPORT	SYSTEM R	QBE	RAPID
1. Database schema definition	Utility static	QL command dynamic	Utility static	QL command dynamic	QL command dynamic	User program dynamic
2. Database retrieval Set-oriented	Yes	Yes	Yes	Yes	Yes	
Record-at-a-time	Yes	Yes	Yes	Yes	Yes	Yes
3a) Database insert Set-oriented	Yes	Yes		Yes	Yes	
Record-at-a-time	Yes	Yes	Yes	Yes	Yes	Yes
3b) Database modify Set-oriented	Yes			Yes	Yes	
Record-at-a-time	Yes	Yes	Yes	Yes	Yes	Yes
3c) Database delete Set-oriented	Yes	Yes	Yes	Yes	Yes	Yes
Record-at-a-time	Yes	Yes		Yes	Yes	Yes
4. Integrity constraints	DD utility	none	DD utility	QL command	QL command	DD utility
5. Database generation, regeneration	Logical bulk load	Logical bulk load			Logical bulk load/unload	Logical Bulk load/unload
6. Database schema redef. & renaming	Indirect	QL commands	Indirect	QL command	QL command	Utility
7. Report generation	PRINT command	Utility		PRINT command	PRINT command	Utility
8. Special data entry		Utility				Forms
9. Security, monitoring				QL command	QL command	
10. LOAD/DUMP/recovery	Yes	Yes	Yes	Yes	Yes	Yes
11. Definition of access path	DD utility		DD utility	QL command	QL command	Utility
12. Database dictionary	QL access	QL access	QL access		QL access	Utility

TABLE 3.5 FUNCTIONAL CLASSES SUMMARY

1. Database schema definition	Separate utility/static : 3 systems QL command/dynamic : 8 systems User program/dynamic : 1 system
2. Database retrieval	
Set-oriented	Supported by 11 systems
Record-at-a-time	Supported by 10 systems
3a) Database insert	
Set-oriented	Supported by 7 systems
Record-at-a-time	Supported by 11 systems
3b) Database modify	
Set-oriented	Supported by 9 systems
Record-at-a-time	Supported by 10 systems
3c) Database delete	
Set-oriented	Supported by 12 systems
Record-at-a-time	Supported by 9 systems
4. Integrity constraints	QL commands : 4 systems DD utility : 5 systems None : 2 systems
5. Database generation, regeneration	Bulk load supported by 10 systems
6. Database schema redef. & renaming	indirect (unload, destroy, create, load) : 8 utility : 1 separate command : 3
7. Report generation	PRINT command : 7 systems utility : 3 systems
8. Special data entry	Prompting of att. names : 3 systems Forms approach : 3 systems
9. Security, monitoring	QL commands : 6 systems Part of data def. : 2 systems
10. LOAD/DUMP/recovery	Supported in 11 systems These 6 others depend on O.S.
11. Definition of Acess Paths	QL commands : 7 systems Part of data def. : 4 systems
12. Database dictionary	QL access : 8 systems Special commands : 3 systems

6. Database Schema Redefinition and Renaming. These functions include facilities for renaming attributes and relations and adding or deleting attributes to already existing relations and adding or deleting entire relations. These functions would be supported through functions 1 and 4 of the ANSI/SPARC framework.

7. Report Generation. These functions include facilities that would be found in interface 8 of the ANSI/SPARC framework and are used for formatting and producing printed reports.

8. Special Data Entry. These functions would include screen entry or form entry processors and correspond possibly to function 11 of the ANSI/SPARC framework.

9. Security Definition and Monitoring. These functions are used to define protection and access rights and to indicate whether and how security violations are to be housed. They would be included in functions 1 and 4 of the ANSI/SPARC framework.

10. Database Utilities. These functions would support physical dumping and loading of databases and rollback and rollforward from database logs. They could correspond to function 24 of the ANSI/SPARC framework.

11. Definition of Storage Structure and Access Paths. These functions are used to define the static internal schema of the database and as such are similar to function 13 of the ANSI/SPARC framework.

12. Database Dictionary. These functions are used for retrieval of schema data and auxiliary user supplied textual descriptions, and can include functions found in functions 34, 35, 36, 37, 38, 2, 3, 5, 6, 14 and 15 of the ANSI SPARC architecture.

3.6.2 Results

The results are displayed in Table 3.5.

3.6.3 Observations

The completed feature catalogs of 12 relational systems were analyzed in order to locate the existence of these functional classes. Note that this analysis was concerned with the existence of these classes and the functional similarity rather than with their power or completeness.

systems on some of the classes. These classes, in decreasing order of uniformity, include set oriented retrieval operations (ignoring duplicate removal), set oriented tuple deletion and modification operations, record-at-a-time insertion operations, query language style commands to define the database, and query language style commands to retrieve those definitions. These classes of functions appeared somewhat similar in 8 or more than 7 of the systems.

Other function classes were either not present in more than 7 of the systems or exhibited considerable variability in this style.

3.7 Interface Flavors

3.7.1 Definitions

The functions to the various systems can be characterized by expected uses of the function. Three such categories include:

1. Interactive End User Interface. In this function the user issues "query" language commands through a keyboard (CRT or hard copy) terminal to access the database.

2. Host Programming Language Interface. In this function, a programming language such as COBOL, PL/1, or FORTRAN is used to access the database. These function can be implemented with the use of a preprocessor which maps database commands to host language subroutine calls, with direct (user controlled) subroutine calls, or with actual extensions to an existing (or new) language and compiler.

3. Special Interfaces. These functions generally require special hardware, such as graphics terminals, for accessing the database.

3.7. Results

The results are displayed in Table 3.6.

3.7.3 Observations

The availability of these interfaces in the systems analyzed are shown in Table 3.6. Most systems supported an interactive end user interface and a host programming language interface. The realization of the programming language functions varied considerably, as did the special functions.

3.8 System Architectures

3.8.1 Results

The systems analyzed can be loosely grouped into two types of architectures. Those architectures are shown in Figure 3.1. Systems of the Type A architecture have separate utilities for defining the database and for manipulating the data items. In those systems the data definitions are stored in an internal form where they are accessed (retrieval only) in order to process data manipulation commands. Some systems in this architecture also store the data definitions in relations for convenient and powerful access by database users. Systems of the Type A architecture include MRDS, NOMAD, PASCAL/R and RAPPORT.

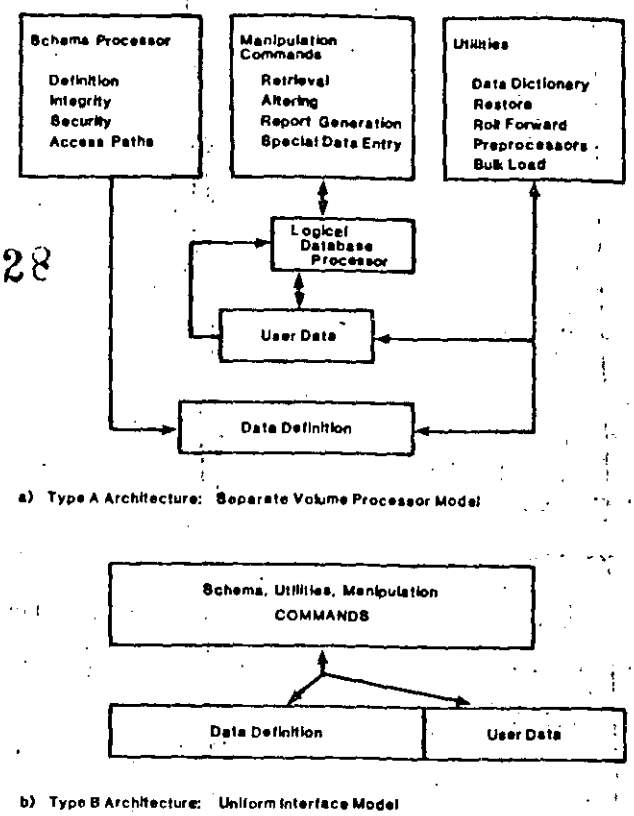


Figure 3.1 Relational Architectural Styles

Systems of the Type B architecture, on other hand, generally have one query level type interface (for both retrieval and update) through which schema commands and data manipulation commands can be freely intermixed. Systems with this type of architecture include IDM, INGRES, MRS, ORACLE, PRTV, SYSTEM R and QBE.

3.8.2 Observations

A few observations can be made about the architectures and functions of the systems. First, many of the systems store system data in relations. Those relations, protection permitting, can be accessed by the same retrieval commands that access user relations. However, special create or define commands are required to populate the system data. In a few systems it may be possible to apply "user relation" update commands to the system relations.

Secondly, in most of the systems, the record-at-a-time operators appear to go through the same level of function as do the set-at-a-time operators. In no system is it made explicit to the user that the "user set oriented operators" are mapped to the "user individual record oriented operators." In another system, IPAD (for which the architecture part of the feature analysis was not completed), that type of "set operator" to "record operator" mapping is suggested.

TABLE 3.6 INTERFACE FLAVORS

	Interactive End User	Host Programming Language Pre-processor	Calls	New Language	Special Interface
1. IDM					Yes - Back End Machine
2. INGRES	Yes	Yes			CUPID
3. MRDS	Yes		Yes		
4. MRS	Yes				Interactive Subsystem (tuple editor)
5. NOMAD	Yes		Yes	Yes	SCREEN Editor for Data
6. ORACLE	Yes		Yes		Forms for Data, Editor for Commands
7. PASCAL/R	Yes			Yes	
8. PRTV	Yes		Yes		
9. RAPPORT	Yes	Yes (?)			
10. SYSTEM R	Yes	Yes			
11. QBE	Yes		Yes		QBE Tabular Form
12. RAPID			Yes		Yes - various utilities
	10 systems	3 systems	6 systems	2 systems	7 Special Interfaces have been defined
	10 systems				

3.9 Operational Aspects

3.9.1 Definitions

In section 6 of the feature analysis, the urity, concurrency control, and crash recovery chanisms of the 12 relational database systems were reviewed. Information was provided in the following areas:

1. Security -- User Related. This section, labeled access control, was concerned with how (and if) users were identified to the DBMS, Three types of identification were identified:

   a. User Id from Operating System. The DBMS, through a system call, identifies the current user in order to determine the users capabilities.

   b. Passwords. The DBMS requires a user to supply a password in order to determine the user's capabilities.

   c. File Access Control. The Operating System controlled access to the database files. For some DBMSs, this method is the only form of access control. For others, the O.S. control is used in conjunction with user identification of type a or b. For the latter type of system, the DBMS itself is the owner of the database files.

2. Security -- Data Related. This section, labeled capabilities, was concerned with the granularity and/or level of protection.

   a. Item protected for most systems was a relation, view, or both, but had to be explicitly named. Some systems allowed protection on specific attributes of relations.

   b. Value based protection indicated that the data value can be used to define the items to be protected. Note this feature is distinct from the value dependent view mechanism in that the value based granules need not be 'named'.

3. Concurrency Control. This section was concerned with the consistency of the database during multiple user update. The particular items noted were:

   a. Multiple User Updates. Could users simultaneously (at least from their perspective) update the database?

   b. Multiple Update Commands. If so, could the user group multiple update commands into a transaction?

   c. Degree of Consistency. If multiple users could simultaneously update the database, what degree of consistency was permitted? Degree 3 consistency implies that no user sees the update of another user until the other user has completed a transaction (possibly a one command transaction). Some DBMSs achieve Degree 3 consistency by locking everything that a transaction reads or writes and holding those locks until the end of the transaction. With Degree 1 consistency, one user can see updates of other user transactions that have not completed. Some of the DBMSs achieve Degree 1 consistency by locking records (or pages) while they are in main memory but not when the records are returned to the disk.

   d. Explicit Locks. If multiple users could simultaneously update the database, can they explicitly preclaim resources by requesting locks for those resources.

4. Crash Recovery. Does the system make any provisions for protecting the consistency of the database in the presence of system and disk crashes?

a. Database Back-up and Restore: Does the Database System have special functions for backing up the disk copy of the database and restoring the database from that back-up? If the database system does not have those functions, some feature analyses suggested that operating system file dump and restore utilities could be used on the database files.

b. Logging. Is logging of record changes or page changes supported? By supported, we mean that the log can be 'rolled forward' together with the DB dump/restore to return the database to a consistent state in the case of a disk crash.

c. Recovery from System Crash. Does the system have any provisions for protecting the database from inconsistencies due to partially complete transactions and soft (non-disk damaging) crashes?

3.9.2 Results

The results are displayed in Table 3.7.

3.9.3 Observations

The results shown in Table 3.7 indicate that there are a variety of approaches to these operational aspects of DBMSs and that approaches do not seem related to the relational model.

The results are important, however, establishing the fact that some of the relational systems are in some sense 'complete' DBMSs; they are suitable for shared databases used by multiple users simultaneously updating the database, and they allow for environmental realities such as soft and hard crashes.

TABLE 3.7 OPERATIONAL ASPECTS

	IDM	INGRES	MRDS	MRS	NOMAD	ORACLE
1. Security (user related)						
a. User ID from OS	Yes	Yes			Yes	Yes
b Passwords						Yes
c. File access by OS		Yes	Yes	Yes	Yes	
2. Security (data related)						
a. Item of protection	Views/ relations/ attributes	Relations/ attributes	View/ relations	Relation	Relation/ Attributes	View/ Relations
b. Based on data value		Yes			Yes	
3. Concurrency control						
a. Multiple user update	Yes	Yes	Yes		Yes	Yes
b. Multiple commands per transaction	Yes	No	Yes		Yes	Yes
c. Degrees of consistency	3	3	3		1	3 if explicit preclaim 1 otherwise
d. Explicit locks	No	No	Yes			Yes
4. Crash recovery						
a. DB back-up/restore	Yes	OS	OS	Yes	OS	OS
b. Logging	Record					
c. Recovery from system crashes.	Yes	Yes	Yes		Yes	

TABLE 3.7 OPERATIONAL ASPECTS (continued)

	PASCAL/R	PRTV	RAPPORT	SYSTEM R	QBE	RAPID
1. Security (user related)						
a. User ID from OS				Yes	Yes	
b. Passwords	Yes					
c. File access by OS	Yes	Yes		Yes	Yes	Yes
2. Security (data related)						
a. Item of protection				Views/ relations/ attributes	Relation/ Attribute	View
b. Based on data value					Yes	
3. Concurrency control						
a. Multiple user update	No	No	Yes	Yes	No	Yes
b. Multiple commands per transaction				Yes		Yes
c. Degrees of consistency				3,2 or 1 user selects		
d. Explicit locks			Yes	No		Yes
4. Crash recovery						
a. DB back-up/restore	OS	OS	Yes	Yes	Yes	Yes
b. Logging				Page logging	Page logging	Page logging
c. Recovery from system crashes			Record level	Yes	Yes	

TABLE 3.7 OPERATIONAL ASPECTS SUMMARY

1. Security (user related)	
a. User ID from OS	6 systems
b. Passwords	2 systems
c. File access by OS	9 systems
2. Security (data related)	
a. Item of protection	Views: 5 systems Relations: 8 systems Attribute(R,W): 5 systems
b. Based on data value	3 systems
3. Concurrency control	
a. Multiple user update	Yes: 8 systems
b. Multiple commands per transaction	Yes: 6 systems
c. Degrees of consistency	3 : 5 systems
d. Explicit locks	Yes: 4 systems
4. Crash recovery	
a. DB back-up/restore	6 systems (OS for others)
b. Logging	Record : 3 systems Page : 2 systems
c. Recovery from system crashes	6 systems

4.1 The Importance of Architecture

Database management systems are complex hardware and software systems used for a wide variety of applications. This complexity and variety has resulted in DBMSs with different functions and architectures. The differences in DBMSs are further complicated by the fact that DBMSs interface to various programming languages and to end user query languages. The functions and architectures of database management systems also differ in terms of the data models they support.

An architectural framework for DBMSs is frequently proposed in order to unify the diversity in systems and approaches. Such a framework can serve many purposes.

First, the framework can be used to classify objects in a database and identify and define the operations on those objects independent of any particular implementation. Within such a framework, related functions can be grouped, the semantics of those functions defined, and uniform terminology established.

In the presence of such a 'semi-formal' database system definition, the difference and commonalities of different models can be determined. This is the second purpose of an architectural framework. Some of the objects and functions will be the same, regardless of the database model. Other objects and functions will only be defined for specific data models or definitions will vary depending on the data model. In addition to identifying the model dependent components (objects and functions), the framework may suggest mappings between the different models.

The third purpose of an architectural framework is to guide the development of DBMS standards. An architecture will allow for the standardization of both internal and external interfaces. The standardization of an internal interface such as 'logical database processor', could result in one DBMS product that is used to support external user interfaces provided by different DBMS vendors. Similarly, a competitor could develop a different logical database processor that conformed to the standards, and market that product without developing any end user interfaces. The designers of such internal and external standards use the framework definitions to determine which interfaces can be the same for all data models and which interfaces must be model dependent.

Additional purposes for an architectural framework have been proposed. The architecture could be used to guide the first stages of designing and developing a database management system. The framework could also serve as a basis for DBMS research efforts for identifying and formally specifying interfaces, defining data models, etc.

In this section, the impact of the relational model on architectural frameworks is analyzed. In Section 4.2, five approaches to architectural frameworks are reviewed. In Section 4.3, several open issues concerning relational standards are discussed. In Section 4.4, the architectures of two of the best known relational systems, INGRES and System R, are reviewed. Finally, in Section 4.5, a potential RDBMS architecture is presented and compared to several of the proposed architecture frameworks. The central goal of this section is to examine the relationship between potential RDBMS standards and more general DBMS standards.

4.2 Alternative DBMS Architecture Frameworks

Several different approaches to DBMS architecture frameworks are currently being developed. Some of the more prominent of these architectures are briefly described below. Note that it is not the jurisdiction or intention of the RTG to select or evaluate these different approaches. Rather, the RTG considered whether or not some or all of the approaches are compatible with a relational data model standard.

A. "ANSI/X3/SPARC DBMS FRAMEWORK" by An ANSI/X3/SPARC Study Group of Database Management Systems. [TSIC78].

The study group presents a generalized framework for the description of database management systems. In particular, the group recommends 42 interfaces for standardization. In addition, a schematic of processors for interrelating those interfaces is proposed. Those processors can be thought of as internal processing functions, program preparation and executive subsystems, and people acting in specific roles.

The study group's report emphasizes three levels of a schema corresponding to the external, conceptual, and internal view of the database. The external view of the database is used to describe the data as seen by the programmer. The conceptual view of the database describes how the information of an enterprise is modelled in the database. Finally, the internal view of the database is used to describe the data to the system. One of the main goals and contributions of the ANSI/SPARC framework was to allow data independence in the presence of evolving changes in an organization at the external levels without requiring extensive changes at the internal level.

The ANSI/SPARC framework is the basis upon which the DBS-SG must develop and maintain a Reference Model for standardization. That Reference Model is designed to serve as a single global map of a generic DBS onto which specific standardization efforts can be projected. The DBS-SG Reference Model can be classified into two categories dealing with the direct and indirect properties of DBMSs. The direct properties include functional components, the connections between the components, and the interfaces. A function supported by an interface is directly processed by its immediate component. However, the processing of that function may require indirect interactions with schema definitions, system logs, user authentication, etc. These activities are referred to as indirect properties of the database architecture and include the placement of the data declarations, and the propagation of those declarations through the Reference Model.

B. "On a Functional Framework for Database Management Systems" by Michael L. Brodie [BROD80a].

A DBMS framework is defined as "An implementation-free characterization of a DBMS in terms of its functional components". The programming language concept of data abstractions is used

base together with the DBMS operations on those components. In particular, an abstract, functional DBMS component is defined. It consists of a language or syntax for initiating functions, the functions, and the objects that the functions act on.

As an example of the framework abstractions, thirteen types of objects and ten basic functions are introduced. The potential functional components can then be defined as an entry in the matrix (13 by 10) which represents the application of each basic function to each object.

C. "NBS Strawman Architecture for a Family of DBMS Standards" by the Computer Corporation of America for an NBS sponsored project [CCA80].

The Computer Corporation of America project is to develop a framework architecture which can be used to guide the evolutionary development of database standards. There are four key concepts in the CCA architecture:

1. a comprehensive list of DBMS functions are grouped into components that could be marketed as independent database products;
2. a family of components are defined for allowing different standards for different data models;
3. a standard schema of schemas and its associated functions are defined; and
4. both internal and external interfaces are defined.

D. "Union of Existing Data Models" as proposed by Johnson (IPAD) [JOHN80], Date (UDL) [DATE 76; 79a; 79b; 79c; 79d; 80], and others.

This approach emphasizes the commonality between the hierarchical, network, and relational data models. Rather than develop a separate standard for the relational model, it could be considered as just a subset of a unified model. Under this approach, the database architecture could closely follow the existing CODASYL architectures.

4.3 Model Related Architectural Issues

4.3.1 The Schema Architecture

A database model can be described by its data structure, the operations on that structure, and the integrity constraints on these operations and structures. The relationship between the different models and how that relationship is encapsulated can affect DBMS standards in at least two important ways. First, the components of an architecture can be arranged in such a way that the different models could share some of the same internal components. For example, the following three approaches could be used to integrate the models.

1. The ANSI/SPARC DBMS framework introduced the concept of external, conceptual, and internal schemas. It has been suggested that a "semantically rich" model for the conceptual schema, such as an entity relationship model, generalized be defined. In this approach, the conceptual schema would be considered "rich", i.e., define many problem oriented properties. The external schemas could consist of relational, hierarchical, or CODASYL objects and use their respective operations and integrity constraints to map to a common, rich conceptual schema.
2. The relational model itself is considered a conceptual schema independent of CODASYL, or hierarchical conceptual schemas. Those independent conceptual schemas could be mapped to common access methods (internal schema).
3. The different models are mapped to a common data model such as a set-theoretic model. Such models can be called sparse because they consist of a small number of concepts (i.e., a set and an element of a set). These concepts are augmented by extremely powerful and general operators. Such models are not usually considered as "conceptual models" with which to represent the information in an enterprise.

The first approach would allow different external schemas (i.e., based on relational, hierarchical, or network models) to be defined over the same database. However, users of a relational external schema may not be aware of multiple entity interdependencies. Either the user will have to understand the underlying conceptual schema, or considerable research will be needed to adequately specify and implement the mappings between conceptual and relational objects. Furthermore, new data models may be difficult to map onto the rich model used for the conceptual schema.

An advantage of the second approach is the 'simpler' view: that what a user sees is realized in the conceptual model. Any multiple entity constraints are value dependent and expressed in a manner similar to the user's query language. Within the second approach an individual database would have to be declared as a relational or a network model, but not both.

The third approach has the advantage of being more extendible to new data models. However, there may be performance penalties related to both mapping overhead and lost optimization opportunities between the semantically weak schema and the access methods.

The above discussion concerns the effects of data models on the component architecture for DBMSs. The second effect of data models on DBMS standards concerns the "exact" definitions of the data models. DBMS standards need to specify both the syntax and semantics of the model operations on the model objects in the presence or absence of model integrity constraints. In order to specify the model semantics and make comparisons between the relative power of the models, a formal and consistent definition of the models may be appropriate. Note that approaches 1) or 3) mentioned above could be used for model definitions independently of any component architecture.

4.3.2 Multiple Models in User Interfaces

There are two approaches to supporting multiple models in query languages and host language interfaces in one DBMS system:

1. One approach to the different models takes the union of the functions in the different models. This approach, as suggested in UDL and IPAD, would then allow a relational or hierarchical system to be restricted to a subset of the language constraints.

2. An alternate approach to the different models takes the intersection of the functions in the different models. This approach, as suggested by the CCA architecture, would then allow a network, hierarchical, or relational system to be an independent extension of the base set of functions.

An advantage of the first approach would be that relational and hierarchical systems could be defined as just a subset of the CODASYL like systems. Unfortunately, if the models do not suitably interact, the definitions of the data models may have to be changed. An advantage of the second approach would be that features that are easier to implement in one of the models (i.e., query languages and their optimizers are easier to implement in a relational system) could be standardized without waiting for the feature to be designed and implemented for all of the models. A disadvantage of the second approach is that some interfaces for different models could needlessly evolve in different directions.

4.3.3 Separate or Unified Components of an Architecture

Approaches to incorporating different models into database framework architecture can be grouped into three classes:

1. One architecture with multiple components and interfaces can be defined. Each component and interface in the framework would have to be defined to accommodate all of the models. This architecture approach is similar to the first approach for supporting multiple data models in one user interface.

2. The architecture can be divided into those components and interfaces which are model dependent and those which are not. The components that are model independent could be standardized on their own. Separate standards for each model dependent component could be developed for each model. This approach is similar to the alternate approach for supporting multiple data models.

3. Separate architectures could be defined for each model. There would be no architectural relationship between standards developed for different data models. This approach is not compatible with the current ANSI/SPARC DBS-SG charter.

Approach three might be the 'easiest' to implement because it requires no coordination between committees working on standards for different data models. Note, however, that there would be no uniformity between the resultant standards. Approach one, on the other hand, would be the most difficult to define but would result in the most uniformity between standards for different models. Approach one is perhaps the theoretical ultimate goal of the DBS-SG reference model. Approach two allows for more flexibility and would serve as a more practical reference model to support today's technology.

4.3.4 Granularity of Building Blocks in the Architecture

Software components and their interfaces are the key constituents of an architecture framework. The issue addressed in this section is the granularity or size of the components. For example, is a DBMS considered to be one large software component, or should some grouping of the functions supported by the DBMS be identified? Identifying the components and their interfaces of a DBMS produces an architecture. The interfaces between components will be called internal. DBMS users deal only with external interfaces.

Three approaches to partitioning the functions supported by a DBMS into components are:

1. Components could be defined along the lines of external interfaces. For example, one component for a DBA interface could support all functions (e.g., creating and maintaining a database). Alternately, DBA functions could be supported by more than one component (e.g., two: one for logical database definition and one for physical database definition). These components and their external interfaces constitute the architecture.

2. One component could be defined for each function supported by a DBMS. In this approach each function that is identified would result in a separate component in the architectural framework. Separate internal components that perform a function for other components (and not directly for end users) could also be identified. For example, file read and file write components could be identified in the architecture.

3. One component could be defined for a class of external or internal interfaces. This approach differs from the first approach because internal components and their interface are also identified in the architecture. An internal interface could be used by multiple components that provide different external interfaces. For example, an internal file access component could be used by components interfacing to DBAs and by components interfacing to end users. This approach differs from the second approach because logically related functions are grouped into one component.

The first partitioning, based on external interfaces only, is normally considered in DBMS standardization activities. The second approach is useful for identifying DBMS functions and understanding their interrelationships in a DBMS architecture. The third approach modularizes those functions into potentially standardizable components that include internal interfaces. The standardization of internal interfaces may become increasingly important with the development of database machines and distributed databases and would allow for the integration of separate DBMS products.

4.4 Internal Architecture of Relational Systems

The internal architecture of two of the better known relational DBMSs, INGRES and the System R are reviewed and the commonality and differences in the architectures are discussed. INGRES and System R were considered due to the availability of published documentation of their architectures.

4.4.1 The INGRES Architecture

The INGRES architecture can be broken into six logical components as shown in Figure 4.1 [STON76, STON80]. The user interface is responsible for interacting directly with the end user and could be

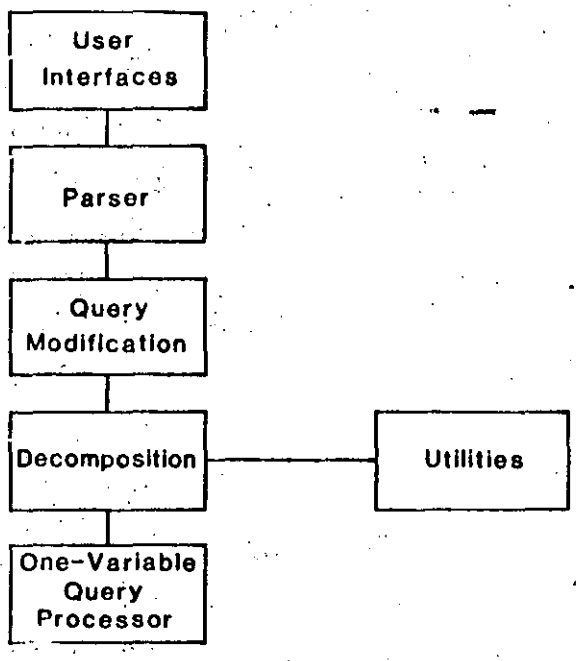


Figure 4.1 The INGRES Architecture

the INGRES terminal monitor, an Equal program, or a CUPID interface. Each of those interfaces sends commands to the parser.

The parser is responsible for syntax checking, the binding of relation and attribute names, and the construction of the parse tree. Note that since relation and attribute names are stored as part of the user database, and the parser must access this information; the parser includes the database access modules.

The Query Modification component modifies the parse tree as required in order to incorporate a view mechanism, integrity constraints, and protection constraints. The query modification component also establishes user identification, date, and time of execution and physical terminal identification. Note that access to the view, integrity, and protection definitions also requires access to the database.

The Decomposition component of the INGRES architecture is responsible for the concurrency control, query optimization, and executes the joining operation. This component invokes the needed utilities to create, sort, or destroy temporary or permanent relations as required for the execution of the query. Note that the decomposition component must read the schema data to determine size characteristics, and must read the database to obtain values for performing a join operation.

The one variable query processor is responsible for access path selection on one relation, retrieval of data from that relation, application of a predicate, updating a relation, and returning data values to the user interface.

The utilities modify the schema, reorganize the access paths on relations, provide logical bulk load and unloading, and provide some data dictionary facilities. The utilities thus read and write both the schema and user data.

4.4.2 The System R Architecture

The System R architecture is generally described as consisting of three components as shown in Figure 4.2 [ASTR76], [BLAS79], [CHAM81]. The user interface could be a general purpose query language, SQL embedded in a PL/1 program or other user interfaces.

The Relational Data System (RDS) is responsible for parsing, authorization, integrity constraints, catalogue management, and optimization. Note that the parsing, authorization and integrity constraints require access to the schema. Optimization requires access to the physical schema. Catalog management requires the retrieval and update of the system data which is also stored in relations. The Relational Storage System (RSS) provides a "record-at-a-time" interface for the relational data system. RSS is responsible for searching indexes, retrieving and storing relation occurrences in the database, concurrency control and recovery. RSS supports both clustered and non-clustered indexes and links between entity types. However, RSS makes no decisions about which access paths to select, but instead is explicitly directed by higher level directives. Note that RSS is thus different than the INGRES one-variable query processor.

A major difference between the System R and INGRES architectures is that System R compiles SQL commands that are embedded in a PL/1 program. These commands are compiled into specific access modules which are stored in the database. As shown

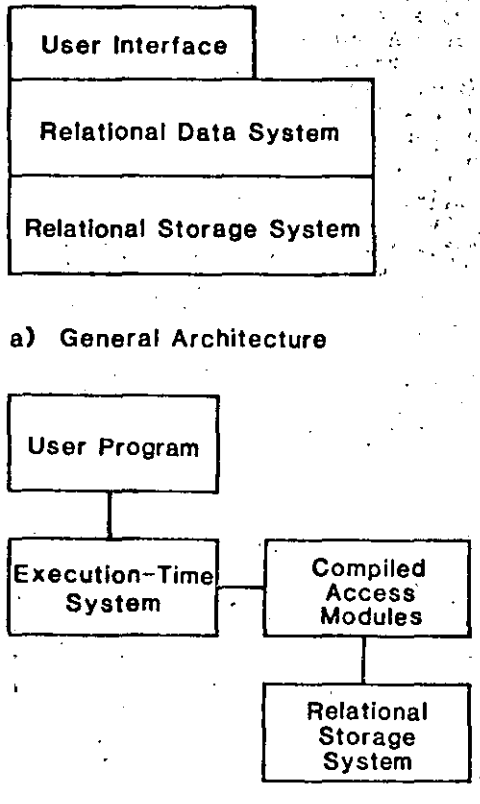


Figure 4.2 The System R Architecture

Figure 4.2, a user program then makes calls on a special execution time interface which retrieves the access modules from the database. Those access modules make direct calls on RSS.

4.4.3 Observations

Several observations are in order regarding these architectures and the uses of these systems.

1. Nonprocedural set oriented relational interfaces make a wide variety of end-user interfaces feasible. System R can support ad hoc queries and PL/1 programs through the Relational Data Interface to the Relational Data System. INGRES provides similar support through its parser. That INGRES interface has been used to implement a diverse set of End User interfaces such as CUPID and Geoquel at the University of California at Berkeley, a menu selection, retrieval and reporting system at Lawrence Livermore Laboratory and a graphics oriented database browsing system (SDMS) at the Computer Corporation of America.

2. Existing low level DBMSs do not provide sufficient physical schema and actual database size information to support the optimization of a high level non-procedural interface. RDS of System R and the Parser, Decomposition, etc., of INGRES requires extensive knowledge of the physical and logical schemas and various instance specific cardinalities of the user data in order to optimize user queries. In addition, both systems require the ability to modify the schema internally. Most Codasyl systems, IBM's IMS, or Cincom's Total, in general do not provide the needed physical schema knowledge at runtime and do not allow the dynamic updating of such schemas. The record at a time interface that INGRES and System R use are tailored to support their specific implementations.

3. Existing DBMS interfaces to programming languages may not provide sufficient logical schema information to support the general purpose interpretive interface typically associated with "relational" systems. For example, consider the problem of supporting a user interface which simply accepts a record type and item(s) name(s) and performs a selection on the record type or simply prints all of the record occurrences. The needed logical schema information on the existence, type, and size of each of the named objects cannot be readily obtained from most CODASYL schemas. Furthermore, the CODASYL COBOL DML used to retrieve the record occurrences could not be readily compiled (or precompiled) at runtime. Some CODASYL and other low level implementations may provide the schema information through subroutine calls and may also allow database navigation through subroutine calls. Such calls, however, are not part of the CODASYL standard and are, in fact, implementation specific. Thus while a vendor might extend its system to support the relational model, a third party vendor or user group could not.

4. Beneath the Parser interface of INGRES and the Relational Data Interface of System R, the architecture of the systems vary widely. Both systems depend on direct knowledge of their schemas, i.e., they do not use a "purely" relational interface to the schema data. Perhaps one of the most dramatic differences between the system architectures is that System R compiles its programming language embedded SQL commands while INGRES is fully interpretive. The significance of this discussion to architecture is that no internal components of the system could be easily interchanged.

4.5 Candidate RDBMS Architecture

The architecture shown in Figure 4.3 illustrates the components and interfaces of a Relational DBMS which may be suitable for standardization. Those components and interfaces are discussed in Section 4.5.1. That architecture is compared to the CCA architecture in Section 4.5.2 and to the ANSI/SPARC DBMS framework in Section 4.5.3.

4.5.1 A Proposed Architecture

In Figure 4.3, an architecture or grouping of interfaces are presented. Each of the components is discussed below. Note that the lists of interfaces for the various components are for illustrative purposes only. It is not claimed that the lists are complete or that all of the elements of the list are necessary. The Class 1 DBA interfaces are those that deal with a database in its entirety. These could include:

1. The creation of a database
2. The destruction of a database
3. Physical Database Dump, Restore

There is nothing particularly relational about these functions. They probably should not be part of a relational standard.

The Class 2 DBA functions are those that define, destroy, or modify parts of a schema. These functions include:

1. Relation definition, destruction or modification
2. View definition, destruction or modification
3. Access paths for or between relations

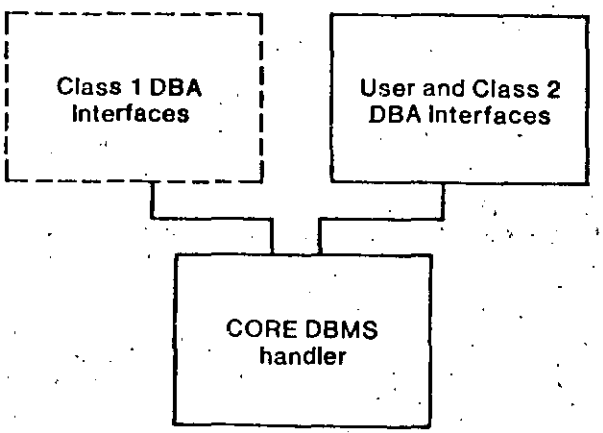


Figure 4.3 A RDBMS Architecture

5. Security constraints

Since these functions are model dependent, they should be addressed in a Relational DBMS standard.

The User interfaces that could be addressed in a Relational DBMS Standard include:

1. User Query Languages (Retrieval)
2. Set-oriented Update Languages
3. Logical Bulk Load and Unload of Relations
4. Report Generator
5. User Query Language applied to schema data

Note that these interfaces are model dependent because the operations they specify are on the objects in the model. Also note that these interfaces are prevalent in existing relational systems. Other interfaces, such as specific programming language interfaces, perhaps should not be considered for standardization since the individual implementations differ widely.

Class 2 DBA functions are combined with the user interfaces because in existing relational systems the query language is used in the definition of views, integrity constraints, and view constraints. It would seem appropriate, therefore, that if an algebraic (calculus, etc.) query language were standardized, that same standard should be used in the DBA functions.

The core database handler could also be standardized. The core database handler would perform the functions of mapping logical access to physical access for queries and updates, view mapping, integrity control, security control, optimization, and schema management. Note that this interface is an "internal" interface and as such its standardization should be in the form of subroutine or procedure calls. Note also that the retrieval and update interfaces to the database schema are also standardizable. In comparison to the INGRES architecture, this interface would be at a level similar to the query modification component. In comparison to the System R architecture, this interface would be a level inside the RDS, immediately following the parser.

There are two major purposes in standardizing

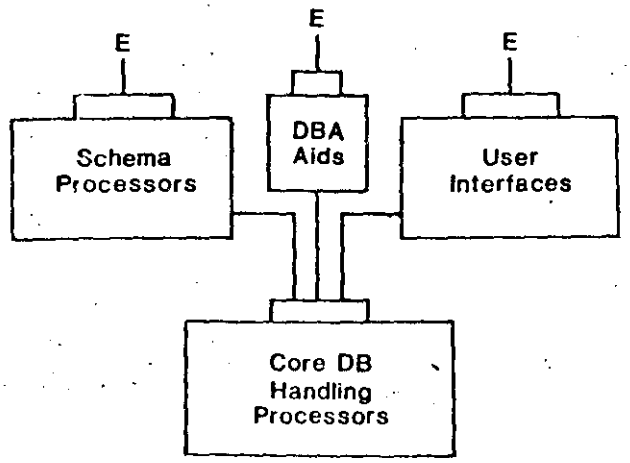


Figure 4.4 The CCA Strawman Architecture

First, a standard interface at this level could be used to uniformly define the functionality of a Relational DBMS without dependence on a particular query language interface. Multiple user interfaces based on different query languages could be separately standardized (and implemented) on top of the CORE database handler interface.

The second purpose in standardizing the CORE database handler interface is to provide extensibility for innovative and nonstandard interfaces. The availability of such a high level interface in INGRES has resulted in a wide variety of graphics interfaces, programming language interfaces, and menu selection forms interfaces being built "on top of" INGRES. By standardizing the internal interface, these special purpose user interfaces could be designed to be marketed for standard relational DBMSs from a variety of vendors. As previously mentioned, it is the lack of a standard internal interface and inaccessibility of the schema data through such an interface that has hindered users from developing higher level interfaces on top of the existing CODASYL systems.

4.5.2 The CCA Strawman Architecture

The CCA Strawman Architecture consists of 38 components which can be grouped into four major blocks as shown in Figure 4.4. This architecture has influenced the design of the proposed RDBMS architecture and in this section we review the differences.

The Strawman Architecture separates schema processors, DBA aids, and user interfaces into separate logical groupings. In the RDBMS architecture, the schema processors are combined in the same grouping as the user interfaces in order to reflect the fact that the user query languages in many relational systems are used in certain types of schema definitions.

The RDBMS architecture includes a subset of the user interfaces described in the strawman architecture. The subset represents those interfaces common to many of the relational systems analyzed. Most notably lacking are the programming languages' interfaces.

The Relational DBMS architecture does not include DBA aids such as physical and logical database design tools as defined in the strawman architecture mainly because no such relational DBA aids were found in the feature analysis. Such DBA aids are not prevented in DBMSs for relational or other models.

The strawman architecture and the RDBMS architecture both include a Core DB handler for standardizing both the functionality of a given data model and the schema management of the model. The interfaces differ in two important ways. Firstly, the strawman architecture contains a rich set of record-at-a-time operators at the CORE DB handler level. Relational systems that were analyzed supported a much weaker set of record-at-a-time operations and the use of those operations varied widely from system to system. Thus, record-at-a-time operations are not suggested for relational standardization at this time.

The second difference is that the strawman architecture interface does not support special commands for database management. Rather, the schema is to be stored in a database of its own and the usual data manipulation operators are to be used for both retrieval and updating of the schema

The ANSI/SPARC external data manipulation language interfaces (8, 9, 10) correspond to the report generator, query language, and update specifics of the RDBMS architecture respectively. Note that the ANSI/SPARC framework does not suggest a logical bulk load and unload utility for end users but instead includes it as part of an internal database utility (17).

Some of the ANSI/SPARC internal interfaces (2, 5, 14, and 12) specify object format interfaces and correspond to the internal RDBMS interface to the CORE Database handler.

The remaining ANSI/SPARC interfaces either are not prevalent or uniform in existing relational systems or are not particularly model dependent.

The two major concepts in the ANSI/SPARC architecture that are not emphasized in the proposed RDBMS architecture are the conceptual model and role definitions for individuals. The conceptual model is not explicitly emphasized in the RDBMS architecture due to the still open issues regarding the conceptual model and that research into conceptual models has not stabilized. The role definitions for individuals is also not explicitly emphasized in the RDBMS architecture. In INGRES, database users can create their own relations, views, and access paths. In System R the users can create those objects and share them with other users. In general, with dynamic schemas, the roles can become blurred and were not considered for standardization.

data. While the RDBMS architecture does not preclude this implementation, it does not suggest it for standardization at this time. Many of the relational systems analyzed supported retrieval from relations containing schema information. However, very few systems would allow updating of these system relations through the normal DM update commands.

The strawman architecture describes additional internal components including a physical database processor and a file access processor. These interfaces were not suggested for a relational DBMS standard since these interfaces vary widely in the analyzed systems and may not be particularly relational in nature.

4.5.3 The ANSI/SPARC Architecture

The ANSI/SPARC architecture specifies 42 interfaces. The proposed RDBMS architecture suggests standardization of some of those interfaces and introduces other interfaces not in the ANSI/SPARC architecture. The source schema interfaces (numbers 1, 4, 13 in the ANSI/SPARC architecture) are identical to the RDBMS interfaces in the DBA Class 2 grouping. The conceptual interface (1) could correspond to the relation schema and integrity constraints interfaces. The external interface (2) could correspond to the view schema interfaces. The internal schema interface (13) could correspond to the physical schema interfaces.

The RTG has concluded that the development of a relational standard is justifiable on a number of rounds. Section 1 of this report discusses three motives for a relational standard, namely: 1) the appropriate time for introducing a standard; 2) the desire for an alternative approach to data definition, manipulation, querying, and integrity; and 3) the inherent benefits of the relational approach. Section 2 illustates the adequacy of relational theory, i.e., the soundness of the RDM. Section 3 gives a detailed demonstration of the practicality of relational technology and describes the features of important research and commercially available RDBMSs. Finally, Appendix C indicates the strength of interest in the DBMS marketplace for RDBMSs by listing over 75 systems that are said to support significant aspects of the RDM. Although the 75 DBMSs have similarities they do not all provide the same "relational" functionality.

The RTG recommends that a standard be developed for interfaces to RDBMSs. This section presents guidelines, issues, and recommendations for developing such a standard.

5.1 Guidelines and Issues

The RTG's primary objective was to provide clear guidance to a subsequent technical committee while not unnecessarily restricting the possible alternatives. To do this, the RTG attempted to identify issues that arise in developing a relational standard, to determine the possible alternatives, and to outline their consequences when possible. The RTG did not attempt to resolve issues.

The RTG applied the following two simple guidelines in its investigation and strongly recommends their application in the development of the relational standard.

1. Concentrate on RDM concepts and on the RDM per se.

2. Concentrate on the semantics or functionality of RDM concepts and of interfaces to RDBMSs.

These guidelines are elaborated and supported in the remainder of this section.

Following the first guideline, the RTG emphasized RDM concepts and the RDM per se over more general topics such as:

- The role of the RDM and RDBMSs in a DBMS architectural framework such as the ANSI/X3/SPARC prototypical architecture.
- The role of a relational standard in a coherent family of DBMS standards.
- The relationship of RDM and RDMBS concepts and terms with general data model and DBMS concepts and terms.

However, the RTG identified issues concerning these topics whenever it was possible.

The general topics received less emphasis for several reasons. First, the general topics are all pen research problems, hence currently inappropriate for standardization. Second, architectural issues are, for the most part, independent of data models. The RDM can be implemented following many DBMS architectural frameworks and one such framework can be used to implement different data models (see Section 4). Third, dealing with more general topics may result in an overlap and possible incompatibilities with the work of other standards projects (e.g., ANSI/X3H2, ANSI/X3H4, DBSSG task groups addressing DBMS architectural, concepts and terminology, etc.). Finally, the RDM and its concepts were emphasized since a DBMS standard should not be based solely on the current state of DBMS technology in this rapidly changing field. There should be some emphasis on the conceptual basis for the standard, the concepts that shape the common understanding of the RDM. A standard with such a basis will accommodate the evolution of both the technology and the standard itself.

Following the second guideline, the semantics or functionality of RDM concepts and of interfaces to RDBMSs were emphasized over topics such as:

- The grouping of functions (e.g., for data definition, manipulation, query, and integrity control) into one or more languages.
- One or more of the many possible syntactic variations of languages that support RDM concepts.
- The embedding of RDM data definition, manipulation, and integrity functions in host languages.

Again, whenever it was reasonable to do so, issues concerning these topics were identified and discussed.

RDM semantics were emphasized over languages and syntactic issues for a number of reasons. First, the RDM provides a basis for a wide range of high level languages [CODD70] [PIRO79], a number of which are equally important to different classes of users. These languages may be self-contained or embedded in host languages. They may provide different representations of relations (e.g., as sets, arrays, predicates, or functions) and different forms of operators (e.g., based on relational algebra, tuple-oriented or domain-oriented relational calculus). Second, based on a RDM semantics specification, languages can be designed (and possibly standardized) for particular user or application needs. Since the languages are based on the same RDM semantics they can be readily, perhaps automatically, translated from one to the other. Third, the choice of languages and their relationship is an architectural issue which, following the first guideline, is of less consequence to the investigation.

Standardizing syntax without an adequate semantic definition would not produce an adequate standard. Currently, the SQL syntax [ASTR76] is provided by four systems: SYSTEM/R, ORACLE, MRS, and MRDS. Each system implements different semantics. For example, each imposes different restrictions on the join operation.

The nature and development proposed for a relational standard should be considered in light of the CODASYL proposal, the only comparable DBMS standards development project. The CODASYL development began over 15 years ago. At that time database management system concepts were not widely known or developed. Few DBMSs existed; hence there was only limited user or application experience. The CODASYL approach was based on one such system, Bachman's IDS. Currently, the relational approach is widely understood. Over 70 RDBMSs have been developed but are not yet in widespread use.

Three important aspects of the CODASYL development are in marked contrast to the two guidelines proposed for the development of the relational standard. First, the DBTG committee of

CODASYL defined a specific syntax using a metalanguage and gave the semantics in English prose. Second, from 1971 to 1981 the data definition and data manipulation languages (DDL and DML) were developed independently by separate committees. There is no high level query language. Third, although several DMLs were considered (both self-contained and host-language embedded), only the COBOL embedded version has been developed. The recent development of a FORTRAN embedded DML has posed consistency problems with respect to the COBOL DML. The two guidelines are proposed for the relational standard to avoid these and other problems in developing a relational standard and subsequent interfaces to RDBMSs.

Issues concerning the development of a relational standard can be classified into those concerning RDM concepts and the RDM itself, and those concerning RDBMSs. RDM issues are discussed in Sections 2.1 and 2.3. This section concludes by outlining several RDBMS issues.

First there is the issue of the possible spectrum of support of the RDM by a DBMS. The spectrum is based on DBMS features, on the supported RDM concepts, and on the restrictions placed on them. Two points on the spectrum are of interest for a relational standard: one indicating fully relational, the other indicating nonrelational. The spectrum between the two indicates semirelational systems or perhaps those languages or DBMSs that conform to the standard. How are these points to be defined? Fully relational could be defined as support of all core RDM concepts (see Chapter 2) without restriction. Nonrelational may include concepts that conflict with or preclude some core RDM concepts. If so, what restricted support would still be considered in conformance with the standard? Could a DBMS be considered relational if it supported relational manipulation or query functions but not data description functions? These issues remain unresolved.

A DBMS that places restrictions on the join operation may be considered semirelational or even nonrelational. In a fully relational system a join is permissible between any two relations based on union compatibility of the attributes on which the join is based. As a consequence, a relation can be joined with itself and relations can be joined in several ways if there is more than one pair of compatible attributes.

Second, there are issues concerning the embedding of sublanguages in a host language: Are RDM concepts supported directly in the host language or are they mediated with host language concepts through workspaces? How are general computation operations (e.g., arithmetic) integrated with relational operations? Other issues include: Are the operators set-oriented or tuple-oriented? What are the restrictions placed on defining, accessing, and manipulating derived relations? Are static or dynamic derived relations supported? What are the mechanisms for handling constraint violations? Are nonrelational DBMS features such as concurrency or security visible in the language? Finally, is the language relationally complete without relying on iteration and recursion constructs in the language? Some of these issues and others were considered in Chapters 2 and 3.

Third, there are architectural issues that should also be considered but excluded from a relational standard. Architectural issues were discussed with the first guideline. There are also basic questions such as: What is an architectural framework? What are the alternatives? There is the issue of mappings between relational schemas and between a relational schema and a nonrelational schema. There is the question of a conceptual data model. These and other issues are addressed in Chapter 4.

The RTG did not consider in detail capabilities that are not directly related to the RDM, e.g., general computational operations, privacy control, concurrent update, rollback, and recovery. The RTG does not regard standards activities in such areas as necessarily tied to a data model.

5.2 Recommendations for a Relational Standard

1. A Relational Standard is justifiable because of the:
   - Stability of the relational approach
   - Marketplace and educational interest: 70 systems, considerable user interest

2. An initial Relational Standard should concentrate on:
   - RDM concepts and the RDM *per se*
   - The functionality or semantics of RDM concepts and of interfaces to RDBMSs

3. An initial Relational Standard should exclude:
   - Architectural aspects
   - Specific language design issues, e.g., syntax
   - Aspects likely to be treated by other standards, e.g., database dictionary
   - Non-data model aspects of DBMSs, e.g., concurrency, recovery, distribution, security, physical storage aspects

Examples of functions that might define a Relational Standard Interface are given in Appendix A.

6. REFERENCES

We distinguish between two kinds of documents:

1. Relational Bibliography, i.e., partial list of publications of importance to the relational approach.

2. Reference Papers, i.e., additional documents referenced in this report.

6.1 Relational Bibliography

[AMBLE79]
Amble, T. "ASTRAL: A Structured and Unified Approach to Data Base Design and Manipulation". IFIP TC-2 Working Conf. on Data Base Architecture, June 1979.

[ASTR80a]
Astrahan, M.M., W. Kim and M. Schkolnick. "Evaluation of System R Access Path Selection Mechanism". Research Report RJ2797, IBM Research Laboratory, San Jose, California, April 1980.

[ASTR80b]
Astrahan, M.M. et al. "A History and Evaluation of System R". Research Report RJ2843, June 12, 1980.

[ASTR76]
Astrahan, M.M. et al. "System R: Relational Approach to Database Management". ACM TODS, Vol. 1, No. 20, June 1976.

[BJOR73]
Bjørner, D. et al. "The Gamma-0 n-ary Relational Data Base Interface Specifications of Objects and Operations". Research Report RJ1300, IBM Research Laboratory, San Jose, California, April 1973.

[BLAS79]
Blasgen M.W. et al. "System R: An Architectural Update". Research Report RJ2581, July 1979.

[BLAS77]
Blasgen, M.W. and K.P. Eswaran. "Storage and Access in Relational Data Bases". IBM Systems Jour., Vol. 16, No. 4, 1977.

[BOYCE73]
Boyce, R.F. and D.D. Chamberlin. "Using a Structured English Query Language as a Data Definition Facility". Research Report RJ1318, December 1973.

[BRAT80]
Bratbergsengen, K. et al. "A Neighbor Connected Processor Network for Performing Relational Algebra Operations". 5th Workshop on Computer Architecture for Non-Numeric Processing, March 1980.

[BRAT79]
Bratbergsengen, K. and R. Larsen. "Design of a Relational Data Base System". Computer Manufacturing International Inc. Data Base Seminar, March 1979.

[BROD80b]
Brodie, M.L. Data Abstraction, Database and Conceptual Modeling: An Annotated Bibliography. NBS Special Pub. 500-59, May 1980.

[BROD80q]
Brodie M.L. and J.W. Schmidt. "Issues in Investigating a Standard for the Relational Approach to Databases". Proc. NBS Conf. of DBMS Standards, Gaithersburg, MD, September 1980.

[BUNE79]
Buneman, P. and R.E. Frankel. "FQL -- A Functional Query Language". Proc. 1979 SIGMOD, Boston, May 1979.

[CHAM81]
Chamberlin et al. "Support for Repetitive Transactions and Ad Hoc Queries in System R". ACM TODS, Vol. 6., No. 1, March 1981.

[CHAM79]
Chamberlin, D.D. et al. "Support for Repetitive Transactions and Ad-Hoc Query in System R". Research Report RJ2551, May 1979.

[CHAM77]
Chamberlin, D.D. et al. "Data Base System Authorization". Research Report RJ2041, July 1977.

[CHAM76a]
Chamberlin, D.D. "Relational Data-Base Management Systems". Computing Surveys, Vol. 8, No. 1, March 1976.

[CHAM76b]
Chamberlin, D.D. et al. "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control". IBM Jour. of Research And Development, November 1976, pp. 560-575.

[CHAM75]
Chamberlin, D.D., J.N. Gray and I.L. Traiger. "Views, Authorization and Locking in a Relational Data Base System". Proc. 1975 AFIPS National Computer Conf., pp. 425-430.

[CHANG79]
Chang, C.L. "On Evaluation of Queries Containing Derived Relations in a Relational Data Base". Research Report RJ2667, IBM Research Laboratory, San Jose, California, October 1979.

[CHILDS77]
Childs, D.L. "Extended Set Theory". Proc. Third Int'l. Conf. on Very Large Data Bases, Tokoyo, October 1977.

[CODD80]
Codd, E.F. "Data Models in Database Management". in Brodie, M.L. and S.N. Zilles (Eds.) Proc. Workshop on Data Abstraction, Databases and Conceptual Modeling, to appear, 1980.

[CODD79a]
Codd, E.F. "Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks". Research Report RJ2599, IBM Research Laboratory, San Jose, California, August 1979.

[CODD74]
Codd, E.F. "Recent Investigations in Relational Data Base Systems". Information Processing '74, Amsterdam, Holland.

[DATE80]
Date, C.J. An Introduction to the Unified Database Language (UDL). Proc. 6th Conf. on Very Large Databases, October, 1980, Montreal, Canada.

[DATE79a]
Date, C.J. An Architecture for HIGH LEVEL Database Extensions: (UDL) COBOL Version. IBM Santa Teresa Laboratory, San Jose, California, April 1979.

[DATE79b]
Date, C.J. An Architecture for HIGH LEVEL Database Extensions: (UDL) COBOL Version, Relational Subset. IBM Santa Teresa Laboratory, San Jose, California, April 1979.

[DATE79c]
Date, C.J. An Architecture for HIGH LEVEL Database Extensions: (UDL) PL/I Version. IBM Santa Teresa Laboratory, San Jose, California, April 1979.

[DATE79d]
Date, C.J. An Architecture for HIGH LEVEL Database Extensions: (UDL) PL/I Version, Relational Subset. IBM Santa Teresa Laboratory, San Jose, California, April 1979.

[DAYAL78]
Dayal, U. and P.A. Bernstein. "On the Updatability of Relational Views". IEEE, June 1978.

[DRIS78]
Driscoll, J.R., B.A. Dutton and K.C. Kinsley. "A Relational Storage Scheme Suitable for Derived Views". Proc. of the 1978 ACM Annual Conf. (ACM '78).

[EPST80a]
Epstein, R. and P. Hawthorn. "Aids in the '80s". DATAMATION, February 1980.

[EPST80b]
Epstein, R. and P. Hawthorn. "Design Decisions for the Intelligent Database Machine". Proc. of the 1980 NCC.

[FAGIN79]
Fagin, R. "Normal Forms and Relational Database Operators". Research Report RJ2471, IBM Research Laboratory, San Jose, California, February 1979.

[FURT78]
Furtado, A.L. "Formal Aspects of the Relational Model". Information Systems 3, 2 (1978), pp. 131-140.

[FRY76]
Fry, J.P. and E.H. Sibley. "Evolution of Database Management Systems". Computing Surveys, Vol. 8, No. 1, March 1976.

[HALL75a]
Hall, P.A.V. "Optimization of a Single Relational Expression in a Relational Data Base System". Research Report UKSC 76, June 1975.

[HALL75b]
Hall, P.A.V., P. Hitchcock and S.P.J. Todd. "An Algebra of Relations for Machine Computation". Conf. Record of 2nd ACM Symposium on Principles of Programming Languages, Palo Alto, California, January 1975.

[HANS76]
Hansel, A. "A Formal Definition of a Relational Database System". Research Report UKSC 80, 1976.

[HARD78]
Hardgrave, W.T. "The Relational Model: A Reformulation of some Mathematical Aspects Using Positional Set Notation". IFSM T.R. NO. 25, Dept. of Information Systems Management, University of Maryland, March 1978.

[HEATH71]
Heath, I.J. "Unacceptable File Operations in a Relational Data Base". ACM SIGFIDET Workshop, 1971, pp. 19-33.

[HELD75]
Held, C.D., M. Stonebraker and E. Wong. "INGRES: A Relational Database System". Proc. ACM Pacific 75 Regional Conf., pp. 409-416.

[HITC76]
Hitchcock, P. "User Extensions to the Peterlee Test Vehicle". Systems for Large Data Bases. Amsterdam, Holland, 1976, pp. 169-180.

[HOUS79]
Housel, B.C. "QUEST: A HIGH LEVEL Data Manipulation Language for Network, Hierarchical, and Relational Databases". Research Report RJ2588, July 1979.

[JARD79]
Jardine, C. and J. Owlett. "Applying a Relational Database System to Historical Data". IBM UK Technical Note 74, 1979.

[JOHN79a]
Johnson, H.R., J.A. Larson and J.D. Lawrence. Data Description Language for Network and Relational Modelling. Sperry Univac, Roseville, Minnesota, 1979.

[JOHN79b]
Johnson, H.R., J.A. Larson and J.D. Lawrence. "Network and Relational Modelling in a Common Data Base Architecture Environment". Research Report TMA00720, Sperry Univac, Roseville, Minnesota, March 1979.

[JONES78]
Jones, S.E. and D.R. Ries. A Relational Data Base Management System for Scientific Data. Lawrence Livermore Laboratory, February 1978.

[KIM79]
Kim, W. "Relational Database Systems". ACM Computing Surveys, Vol. 11, No. 3, September 1979.

[LACR76]
Lacroix, M. and A. Pirotte. "Generalized Joins". ACM SIGMOD Record, Vol. 8, No. 3, 1976, pp. 14-15.

[LACR77a]
Lacroix, M. and A. Pirotte. "ILL: An English Structured Query Language for Relational Databases". Proc. IFIP TC-2 Working Conference on Modeling in Database Management Systems, Nice, France, January 1977, Nijssen (Ed.), North-Holland (1977), pp. 23-260.

[LACR78]
Lacroix, M. and A. Pirotte. Example Queries in Relational Languages. Tech. Note 107, MBLE Research Laboratory, Brussels, Belgium, revised April 1978 (see also Research Reports 351 and 367).

[LACR80]
Lacroix, M. and A. Pirotte. "User interfaces for Database Application Programming". Proc. 74th Infotech State of the Art Conference on Databases, London, October 1980.

[LACR81]
Lacroix, M. and A. Pirotte. "Associating Types with Domains of Relational Databases". Proc. NBS-ACM Workshop on Data Abstraction, Databases, and Conceptual Modeling, Pingree Park, Colorado, June 1980, in: SIGART Newsletter 74 (January 1981), SIGMOD Record, Vol. 11, No. 2 (February 1981), SIGPLAN Notices, Vol. 16, No. 1 (January 1981), pp. 144-146.

[LAME80]
Lamersdorf, W. and J.W. Schmidt. "Semantic Definition of PASCAL/R". Berichte Nr. 73 and 74, University of Hamburg, Federal Republic of West Germany, July 1980.

[LORIE79]
Lorie, R.A., R. Casajuana and J.L. Becerril. "GSYSR: A Relational Database Interface for Graphics". Research Report RJ2511, April 1979.

[McDO75]
McDonald, N. and M. Stonebraker. "CUPID: The Friendly Query Language". Proc. ACM Pacific 75 Regional Conf., April 1975.

[MYLO75]
Mylopoulos, J., S. Schuster and D. Tsichritzis. "A Multi-Level Relational System". Proc. 1975 AFIPS National Computer Conf.

[OSBO79]
Osborn, Sylvia L. "Towards a Universal Relational Interface". IEEE, August 1979, pp. 52-60.

[OWENS71]
Owens, R.C. "Evaluation of Access Authorization Characteristics of Derived Data Sets". Proc. ACM SIGFIDET Workshop on Data Description, Access and Control, 1971, pp. 263-278.

[OWLE77]
Owlett, J. "Deferring and Defining in Databases". A section of: "Architecture and Models in Data Base Management Systems". Proc. of the IFIP Working Conf. on Modelling in Data Base Management Systems, November 1977.

[PIRO79]
Pirotte, A. "Fundamental and Secondary Issues in the Design of Non-Procedural Languages". Proc. 5th VLDB, Rio de Janiero, Brazil, August 1979, pp. 239-250.

[PIRO78]
Pirotte, A. "Linguistic Aspects of High Level Relational Languages". Infotech State of the Art Report on Data Bases Technology, Vol. 2, pp. 271-300, 1978.

[PIRO77a]
Pirotte, A. "The Entity-Property-Association Model: An Information-Oriented Data Base Model". Proc. ACM International Computing Symposium (ICS77), North-Holland (1977), pp. 581-597.

[PIRO80]
Pirotte, A. "High Level Database Query Languages". in "Logic and Databases", Gallaire and Minker (Eds.), Plenum (1978), pp. 409-435.

[SCHM80a]
Schmidt, J.W. and M. Mall. "PASCAL/R Report". Bericht Nr. 66, University of Hamburg, Federal Republic of Germany, January 1980.

[SCHM79]
Schmidt, J.W. "Parallel Processing of Relations: A Single-Assignment Approach". VLDB5, Rio de Janeiro, 1975.

[SCHM77]
Schmidt, J.W. "Some High Level Language Constructs for Data of Type Relation". ACM TODS, Vol. 2, No. 3, September 1977.

[SELI79]
Selinger. P.G. et al. "Access Path Selection in a Relational Database System". Research Report RJ 2429, January 1979.

[SHIP81]
Shipman, D. "The Functional Data Model and the Data Language DAPLEX". ACM TODS, Vol. 6, No. 1, March 1981.

[SMITH77]
Smith, J.M. and D.C.P. Smith. "Database Abstraction: Aggregation and Generalization". ACM TODS 2, 2 (June 1977).

[STON80b]
Stonebraker, M. "Retrospection on a Database System". ACM TODS, Vol. 5, No. 2, June 1980.

[STON78]
Stonebraker, M. "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES". ERL Technical Memorandum Reprint 1702, University of California, August 1978.

[STON76]
Stonebraker, M. et al. "The Design and Implementation of INGRES". ERL Technical Memorandum Reprint 1468, University of California, September 1976.

[STON75]
Stonebraker, M. "Implementation of Integrity Constraints and Views by Query Modification". Proc. Int'l. Conf. on Management of Data, 1975, pp. 65-78.

[STRN71]
Strand, A.J. "The Relational Approach to the Management of Data Bases". Proc. of IFIP 1971 Congress, 1971, pp. 901-904.

[TIBU76]
Tibuya, S. "Practice of Noun Phrase Model". Proc. of IBM Int'l. Conf. in Relational Database, Bari, 1976.

[TODD77a]
Todd, S.P.J. "Automatic Constraint Maintenance and Updating Defined Relations". Information Processing '77, IFIP Congress Series, pp. 145-148.

[TODD77b]
Todd, S.P.J. "Database Research at the IBM UK Scientific Centre, Peterlee: A Survey". Research Report UKSC 93, 1977.

[TODD76]
Todd, S.P.J. "The Peterlee Relational Test Vehicle - A System Overview". IBM Systems Jour., No. 4, 1976.

[TODD74]
Todd, S.P.J. "Implementation of the Join Operator in Relational Data Bases". IEEE/IERE Colloquium on Information Structure and Storage Organisation. London, April 1974.

[TSIC78]
Tsichritzis, D. and Klug, A. "The ANSI/X3/SPARC DBMS Framework". Report of the Study Group on a Database Management System". Information Systems, Vol. 3, No. 4, 1978.

[VERH79]
Verhofstad, J.S.M. "An Evaluation of the PRTV Optimiser". Research Report UKSC 91, preparation.

[VERH76]
Verhofstad, J.S.M. "The PRTV Optimiser: The Current State". Research Report UKSC 83, May 1976.

[WONG77]
Wong, H.K.T. and J. Mylopoulos. "Two Views of Data Semantics: A Survey of Data Models in Artificial Intelligence and Database Management". INFOR, Vol. 15, No. 3, October 1977.

[ZLOOF78a]
Zloof, M.M. "Query-By-Example: Operations on the Transitive Closure". IBM Research Report RC5526, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, February 7, 1978.

[ZLOOF78b]
Zloof, M.M. "Security and Integrity Within the Query-By-Example Data Base Management Language". IBM Research Report RC6982, February 7, 1978.

[ZLOOF77]
Zloof, M.M. "Query-By-Example: A Data Base Language". IBM Systems Jour., No. 4, 1977.

[ZLOOF76]
Zloof, M.M. "Query-By-Example: Operation on Hierarchical Data Bases". AFIPS Conf. Proc., N.C. 45, 1976, pp. 845-853.

[ZLOOF75a]
Zloof, M.M. "Query-By-Example". AFIPS Conf. Proc., Vol. 44, AFIPS Press, Montvale, N.J., 1975.

[ZLOOF75b]
Zloof, M.M. "Query-By-Example: The Invocation and Definition of Tables and Forms". Proc. of the Int'l. Conf. on Very Large Data Bases. Boston, Massachusetts, September 22-24, 1975, pp. 1-24.

6.2 Reference Papers

[BEER78]
Beeri, C., P. Bernstein, and N. Goodman. "A Sophisticate's Introduction to Database Normalization Theory". VLDB, Berlin, 1978.

[BEKI79]
Bekic, H., D. Bjørner, W. Henhapl and C.B. Jones. "A Formal Definition of PL/1 Subset, Parts I and II". Tech. Report TR25.139, IBM Laboratory, Vienna, December 1979.

[BJOR78]
Bjørner, D. and C.B. Jones. The Vienna Development Method: The Meta Language. Springer-Verlag, New York, 1978.

[BJOR80]
Bjørner, D. "Formalization of Database Models". in "Abstract Software Specifications", Springer Lecture Notes in Computer Science 86, 1980.

[BROD80a]
Brodie, M.L. "A Functional Framework for Database Management Systems". TR-78, Dept. of Computer Science, University of Maryland, February 1980.

[BROD80c]
Brodie, M.L. and J.W. Schmidt. "Feature Catalogue of Relational Concepts, Languages and Systems." RTG Working Doc. 80-81, May 1980.

[BROD80e]
Brodie, M.L. "Standardization and the Relational Approach to Databases". Proc. 6th Int'l Conf. on Very Large Databases, October 1980, Montreal, Canada.

[CCA80]
Computer Corporation of America. "Overview of NBS Strawman Architecture for Family of DBMS Standards". Presented at NBS Conference, September, 1980.

[CODA73,78]
CODASYL Data Description Language Committee, Journal of Development, 1973, 1978.

[CODA69,71]
CODASYL Database Task Group Reports 1969, 1971.

[CODD70]
Codd, E.F. "A Relational Model of Data for Large Shared Data Banks". Comm. of ACM, Vol. 13, No. 6, June 6, 1970.

[CODD71a]
Codd, E.F. "Relational Completeness of Data Base Sublanguages". Data Base Systems, Courant Computer Science Symposia, Vol. 6, May 1971.

[CODD71b]
Codd, E.F. "A Database Sublanguage Founded on the Relational Calculus". Proc. ACM SIGFIDET Workshop, San Diego, Calif., November 1971.

[CODD72]
Codd, E.F. "Relational Completeness of Database Sublanguages". in Data Base Systems, Courant Computer Science Symposium 6, Prentice-Hall (1972).

[CODD79b]
Codd, E.F. "Extending the Relational Data Base Model to Capture More Meaning". ACM TODS, Vol. 4, No. 4, 1979.

[CODD81a]
Codd, E.F. "The Significance of the SQL/Data System Announcement". Computerworld, February 16, 1981.

[CODD81b]
Codd, E.F. "The Capabilities of Relational Database Management Systems". IBM Research Laboratory Report, RJ3132, May 11, 1981.

[DATE81]
Date, C.J. "An Introduction to Database Systems", Third Edition, Addison-Wesley, 1981.

[DELO80]
Delobel, C. "An Overview of the Relational Data Theory". IFIP Congress, 1980.

[DONA76]
Donahue. "Complementary Definitions of Programming Language Semantics". Lecture Note 42, Springer Verlag, New York, 1976.

[EARL71]
Earley, J. "Towards an Understanding of Data Structures". Comm. of ACM 14, 10 (October 1971), pp. 617-627.

[EGLI74]
Egli, H. "Programming Language Semantics Using Extensional Lambda-Calculus Models". TR-206, Cornell University, April 1974.

[GALL78]
Gallaire, H. and J. Minker (Eds.) Logic and Databases. Plenum Press, New York, 1978.

[GOGU77]
Goguen, J.A., J.W. Thatcher and E.G. Wagner. "An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types". RC 6487, IBM Yorktown Heights, New York, April 1977.

[GORD79]
Gordon, M.J.C. The Denotational Description of Programming Languages. Springer Verlag, New York, 1979.

[GUTT78]
Guttag, J.V. and J.J. Horning. "The Algebraic Specification of Abstract Data Types". Acta Informatica 10, pp. 27-52 (1978).

[HARD80]
Hardgrave, W.T. "Positional Set Notation". Draft Paper, National Bureau of Standards, February 1980.

[HARD76]
Hardgrave, W.T. "Set Processing: A Tool for Data Management". IFSM T.R. NO. 6, Dept. of Information Systems Management, University of Maryland, April 1976.

[HOARE74]
Hoare, C.A.P. and P.E. Lauer. "Consistent and Complementary Formal Theories of the Semantics of Programming Languages". Acta Informatica 3, pp. 135-153 (1974).

[HOARE73]
Hoare, C.A.P. and N. Wirth. "An Axiomatic Definition of the Programming Language PASCAL". Acta Informatica 2, pp. 335-355 (1973).

[KINS80]
Kinsley, K. and J. Driscoll. "Preliminary Feature Analysis". Working Paper of RTG, 1980.

[LACR77]
Lacroix, M. and A. Pirotte. "Domain Oriented Relational Languages". Proc. of Third VLDB, October 1977, pp. 370-378.

[LARS79]
Larson, J.A. and T.B. Wilson. Bridging the Gap between Network and Relational Data Bases. Sperry Univac, Roseville, Minnesota.

[LUCAS71]
Lucas, P. "Formal Definition of Programming Languages and Systems". IFIP Congress 71, August 1971.

[MILLS79]
Mills, H.D., R.C. Linger and B.I. Witt. "Structured Programming Theory and Practice". Addison-Wesley, 1979.

[SAND81]
Sandberg, G. "A Primer on Relational Data Base Concepts". IBM Systems Jour., Vol. 21, No. 1, 1981.

[SCOTT72]
Scott, D. "Lattice Theory, Data Types, and Formal Semantics". NYU Symposium on Formal Semantics, Prentice Hall, 1972.

[STON80a]
Stonebraker, M. "A Tale of Three Standards". Proc. 1980 NBS Workshop on a Family of DBMS Standards, NBS Publication, October 1980.

[ULLM80]
Ullman, J.D. "Principles of Database Systems". Computer Science Press, 1980.

APPENDIX A. NATURE OF A RELATIONAL STANDARD INTERFACE

The SD-3 proposes the standardization of RDBMS interfaces. The functions that will constitute the interfaces will support:

1. the definition of structures, domains, and integrity constraints;
2. the execution of query and altering operations over relational databases in accordance with 1); and
3. access to the definition of structures, domains, and constraints.

This appendix describes, briefly, examples of functions that might constitute the interface. Other sets of functions may serve equally well. According to their purpose the functions are divided into three groups:

1. database schema definition -- structure, domain, and constraint definition;
2. database operations -- deriving and altering relation values; and
3. schema access -- deriving information that defines a relational database schema.

The syntax used in the examples is "abstract." Many different forms could be used equally well.

A.1 Interface Functions for Database Definition

This section presents examples of functions that have to be provided in one form or another, by an RDBMS in order to accomplish a relational database schema definition.

Example A1:

The interface function

```
function dfn_domain (domain_id : string ;
                     value_set : {val})
```

defines a domain by a set of values, {val}, and identifies it by a string. The value set may be given extensionally, e.g., by {red, blue, yellow}, or intentionally, e.g., by the key word "integer" or by an interval {1..999}.

Example of function invocation:

```
dfn_domain ("creditstatus", {1..999}).
```

Example A2:

The interface function

```
function dfn_relation_structure (
     rel_structure_id : string ;
     attribute_set : {< attribute_id : string,
                        domain_id : string>} )
```

defines a relation structure by a set of attributes, i.e., by a set of ordered pairs of attribute and domain identifiers. Furthermore, it identifies the structure definition by a string.

Example of function invocation:

```
dfn_relation_structure("supplier_structure",
                       {<"snr", "suppliernumbers">,
                        <"sname", "suppliernames">,
                        <"status", "creditstatus">,
                        <"city", "citynames">}).
```

Example A3:

The interface function

```
function dfn_relation_schema (
                    rel_schema_id : string ;
                    rel_structure_id : string ;
                    key_attribute_set :
                        {attribute_id : string} )
```

defines a relation schema by a structure definition and by a set of attribute identifiers that are to constitute the relation key. Furthermore, it names the schema definition by a string. A more complete specification of this function would require that the key attribute set is a subset of the attribute identifiers specified in the corresponding relation structure definition.

Example of function invocation:

```
dfn_relation_schema ("supplier_schema",
                     "supplier_structure",
                     {"snr"}.
```

Additional functions are required for database schema definition and for the deletion and alteration of definitions.

If the data structures that store a schema definition are defined as a relational database (subsequently called a meta or schema database) the above functions represent specific insert procedures for the schema database.

A.2 Interface Functions for Database Operations

The interface functions for database operations are somewhat more complicated. We discuss some of the problems using a select operation as an example (an extension of Example 5, Section 2.3 of the report): select the suppliers that either are located in Paris but not named Smith, or have a status different from 20.

The selection criterion involved is

((city = "Paris") and (sname ≠ "Smith")) or
(status ≠ 20)

Here we face the problem of how to parameterize such criteria. One solution is to pass the selection criterion simply as a string. Another solution is to adopt, as a convention, a normal form in which any selection criterion is transformed into a sequence of or-connected predicates that are sequences of and-connected comparison terms* (our example is already in that normal form). With this convention a selection criterion can be represented as a set of elements where the elements are sets of ordered triples, e.g.,

{ {<"city", =, "Paris">, <"sname" ≠ "Smith">},
{<"status", ≠, 20>} }

The interpretation of this set is obvious:

- Each triple represents a comparison term.
- The inner sets represent predicates that are and-connections of these comparison terms.
- The outer set represents a selection criterion that is an or-connection of these predicates.

*In formal logic this normal form is called a disjunctive normal form.

The following interface function uses this type of parameter to pass selection criteria.

47

Example A4:

```
function select (
        source_relation_id : string ;
        selection_criterion :
            {{<lhs_id, comparison_op, rhs_id>}};
        result_relation_id : string    )
```

(lhs = left hand side, rhs = right hand side)

This function requires the name of a source relation, a selection criterion, and an identifier for a (temporary) result relation.

Example of function invocation:

```
select ("suppliers",
       {{<"city",=,"Paris">,<"sname",≠,"Smith">},
        {<"status",≠, 20>}},"temp_relation").
```

The definition of the above function can be generalized by extending

- the first parameter to be a set of source relation identifiers, and
- the second parameter to accept selection criteria that include quantifiers*.

Whereas the first definition of the select function directly supports the select operator of the relational algebra, the extended definition provides the basic support for a query language in the relational calculus approach.

Additional functions are required to support projection (for both the algebra and the calculus approach), join, union, difference, and division (these functions are required for the algebra approach only).

The interface functions for the altering operations are straightforward.

Example A5:

```
function insert (lhs_relation_id : string ;
                 rhs_relation_id : string)
```

The second parameter identifies a set of tuple values to be inserted into the relation that is identified by the first parameter. The tuple values to be inserted may be given by a temporary relation resulting from a database query or by a set of new tuple values provided by the user (see Example 11 in Section 2.3 of the report).

Delete and replace functions can be defined in a similar way.

A.3 Interface Functions for Database Schema Access

A third class of interface functions provides access to a database schema definition. Access to schema information is required for a variety of reasons. For example, schema information is required to check if an altering operation is legal (i.e., does not violate any of the structure, domain, and integrity constraints).

If the database schema information is stored in a relational meta database the interface functions defined in the previous section can be used to access the meta database. The subsequent example follows this approach.

Example A6:

By Example A2 we sketched the structure definition of a "suppliers" relation. If the meta database had a relation, "relation_structures", that holds all structure definitions this (meta) relation may have the following value:

(meta) relation "relation_structures":

attribute names and relation value:

relation_structure_id	attribute _id	domain_id
supplier_structure	snr	suppliernumbers
supplier_structure	sname	suppliernames
supplier_structure	status	creditstatus
supplier_structure	city	citynames
part_structure	pnr	...
...	...	...
shipment_structure	qty	...
...	...	...

Access to the structure definition of relation "suppliers" (as, e.g., required by the legality test for an insert operation, see Example 11, Section 2.3 of the report) can be achieved by the select function defined above.

```
select ("relation_structures",
       {<"relation_structure_id", =,
                          "supplier_structure">},
       "actual_structure")
```

The resulting value of the relation "actual_structure" will be

result relation value:

supplier_structure	snr	suppliernumbers
supplier_structure	sname	suppliernames
supplier_structure	status	creditstatus
supplier_structure	city	citynames

This result can be used to determine, for example, the number of attributes or the domains of the "suppliers" relation.

In this appendix we sketched the meaning (semantics) of some interface functions using the English language. The level of precision that can be achieved by informal or semi-formal definitions is considerable. There are, however, important arguments (compare Section 2.1 of the report) to supplement a nonformal definition of a standard by a formal and precise specification. The definition of the RDM provided in Appendix B can serve as an example of a formal specification.

*In formal logic a corresponding normal form is called prenex disjunctive normal form.

MODEL

The 'Vienna Development Method' (VDM) [BJOR78, ME80] is used to specify formally the semantics of the Relational Data Model (RDM). A VDM-Specification consists of three parts:

1. The Semantic Domains define a structural model for all 'objects of interest' and their components.
2. The Syntactic Domains define the names of all operations and a structural specification for each of their operands.
3. The Elaboration Functions define the semantics of all operations introduced by the syntactic domains.

Additional constraints may be introduced either by Static Consistency Constraints, defining whether a semantic object is "well-formed", or by Dynamic Consistency Constraints, stating whether a syntactic object is "well-applied".

Auxiliary Functions may be used to improve the conciseness and readability of a definition.

B.1 Semantic Domains for Relational Databases

The main object specified is a Relational Database, RELDATABASE. It consists of three parts: the domain definition, DOMAINS; the database schema, RDBSCHEMA; and the database value, RDBVALUE.

A Domain definition is modeled as a mapping from the set of domain identifiers, DOMID, to the associated domains, DOM. A domain is a set of atomic domain elements, DEM.

A Database Schema defines the 'structure' of the database, RDBSTR, and the constraints, RDBCNSTR, to be maintained on the database. The database structure defines for each relation of a database a relation schema, RELSCHEMA, and a relation identifier, RELID. A relation schema is given by the relation's structure, RELSTR, and by relation constraints, i.e., the key attribute identifiers.

A Database Value, RDBVALUE, is defined by the relation values. A relation value, RELVALUE, is represented in two ways: as a set of tuple values, TUPLEVAL-Set, and as a predicate defined over all possible tuple values, TUPLEVAL-Pred.

Identifiers are not further specified and therefore are considered as 'TOKENS'.

B.1.1 Abstract Syntax

```
RELDATABASE     :: RDBSCHEMA RDBVALUE

RDBSCHEMA       :: RDBSTR DOMAINS RDBCNSTR
RDBSTR          =  (RELID -m-> RELSCHEMA)
RELSCHEMA       :: RELSTR RELCNSTR
RELSTR          :: TUPLESTR
TUPLESTR        =  (ATTRID -m-> DOMID)
RELCNSTR        =  KEYATTRID-Set

DOMAINS         =  (DOMID -m-> DOM)
DOM             =  DEM-Set
DEM             =  ...

RDBVALUE        =  (RELID -m-> RELVALUE)
RELVALUE        :: SETREP PREDREP
SETREP          =  TUPLEVAL-Set
PREDREP         =  TUPLEVAL-Pred
TUPLEVAL        =  (ATTRID -m-> DEM)

DOMID           =  TOKEN
RELID           =  TOKEN
ATTRID          =  TOKEN
KEYATTRID       =  TOKEN
```

B.1.2 Static Consistency Constraints

```
is-wf RELDATABASE(reldatabase) =

let mk-RELDATABASE(rdbschema,rdbvalue ) be reldatabase
    in let mk-RDBSCHEMA(vdbstr,domains) = rdbschema
       in dom rdbstr = dom rdbvalue
       and (∀ relid ∈ dom rdbvalue)
           (is-wf-Relation(rdbvalue(relid) ,
              rdbstr(relid) , domains ))
       and test-rdb-constraints(reldatabase)

type : RELDATABASE ---> BOOL


test-rdb-constraints(reldatabase) =

    ...

type : RELDATABASE ---> BOOL


is-wf-RELATION(relvalue,relschema,domains) =

    let mk-RELVALUE(setrep,predrep) be relvalue ,
        mk-RELSCHEMA(relstr,relcnstr) be relschema
    in let mk-TUPLESTR(tuplestr) be relstr
       in rng tuplestr ⊆ dom domains

          and relcnstr ⊆ dom tuplestr

          and dom predrep =
              {[attrid ---> dem ] |
               attrid ∈ dom tuplestr and
               dem ∈ domains(tuplestr(attrid))}

          and setrep =
              {tupleval | tupleval ∈ dom predrep
                          and predrep(tupleval) }

          and test-key-constraints(setrep,relcnstr)

type : RELVALUE RELSCHEMA DOMAINS ---> BOOL


test-key-constraints(setrep,relcnstr) =

    let relkey be relcnstr
    in (∀ tupleval,tupleval' ∈ setrep)
       ((tupleval | relkey = tupleval' | relkey) ===>
                ( tupleval = tupleval' ))

type : SETREP RELCNSTR ---> BOOL
```

B.2 Syntactic Domains for Relational Databases

The basic database operations are queries, Query, and relation altering operations, Altop.

Queries can be database relations (given by relation identifiers), 'selections', 'joins', 'projections', 'unions', ... (in the algebraic approach), or 'relational calculus expressions' (in the calculus approach). Operands for these operations are either relation expressions, Rel, which are built up (recursively!) by other queries, or 'external' relations, Extrel, from outside the database. A 'rename' operation, Rename, is specified to achieve unique attribute identifiers for relational operands.

An altering operation, Altop, takes a relation, Rel, and alters the value of a database relation identified by its name, RELID.

B.2.1 Abstract Syntax

```
Op              =   Query | Altop

Query           =   RELID | Selection | Join |
                    Projection | Union
                    Division | ...   |
                    Rename | Calc-Expression

Selection       ::  Rel Selpred
Selpred         =   TUPLEVAL-Pred
Join            ::  Rel ATTRID Rel ATTRID
Projection      ::  Rel Attrids
Attrids         =   ATTRID-Set
Union           ::  Rel Rel
 ...
Rename          ::  Rel Bijection
Bijection       =   (ATTRID <-m->ATTRID)

Calc-Expression ::  Targetattrids Sourcerels Selpred
Targetattrids   =   ATTRID-Set
Sourcerels      =   Rel-Set

Altop           =   Insert | Delete | Replace
Insert          ::  RELID  Rel
Delete          ::  RELID  Rel
Replace         ::  RELID  Rel

Rel             =   Query | Extrel
Extrel          ::  RELVALUE RELSTR
```

B.2.2 Semantic Consistency Constraints

```
is-wf Op(op) =

    (is-Query(op) ---> is-wf-Query(op) ,

     is-Altop(op) ---> is-wf-Altop(op))

type : Op ---> BOOL


is-wf-Altop(altop) =

    let relid be s-RELID(altop) ,
        rel be s-Rel(altop)
    in is-wf-Rel(rel) and
       compatible(relid,rel)

type : Altop ---> BOOL


is-wf Query(query) =

    cases query :
    (mk-RELID(relid) ---> relid ϵ dom RDBVALUE,

     mk-Selection(rel,selpred) ---> is-wf-Rel(rel) and
            dom selpred = dom extract-predrep(rel) ,

     mk-Join(rel,attrid,rel',attrid') --->
       let tplstr be tpl-structure(rel)
           tplstr' be tpl-structure(rel')
       in is-wf-Rel(rel) and is-wf-Rel(rel') and
          attrid ϵ dom tplstr and attrid'
                                  ϵ dom tplstr' and
          tplstr(attrid) = tplstr'(attrid') and
          dom tplstr ∩ dom tplstr' = { } ,

     mk-Projection(rel,attrids) ---> is-wf-Rel(rel)
                and attrids⊆ dom tpl-structure(rel) ,

     mk-Union(rel,rel') ---> is-wf-Rel(rel)
                             and is-wf-Rel(rel')
                             and compatible(rel,rel'),

     mk-Division ...

     ...

     mk-Rename(rel,bijection) --->
       is-wf-Rel(rel) and
       dom bijection = dom tpl-structure(rel) ,

mk-Calc-Expression(targetattrids,sourcerels,selpred)
--->
   (∀ rel ϵ sourcerels)(is-wf-Rel(rel)) and
   (∀ rel,rel' ϵ sourcerels)((rel ≠ rel') ===>
      (dom tpl-structure(rel)∩
       dom tpl-structure(rel') = { } ))
   and targetattrids ⊆ {attrid |
                        attrid ϵ dom tpl-structure(rel)
                        and rel ϵ sourcerels }
   and dom selpred =
       {tupleval1 u...u tupleval_n |
        tupleval_i ϵ dom extract-predrep(rel_i)
        and rel_i ϵ sourcerels
        and 1 ≤ i ≤ n = card sourcerels }

type : Query---> BOOL


is-wf-Rel(rel) =

(is-Query(rel) ---> is-wf-Query(rel),
 is-Extrel(rel) --->
   let mk-Extrel(relvalue,relstr) be rel
   in let relcnstr be dom mk-TUPLESTR(relstr)
      in is-wf-Relation(relvalue,mk-RELSCHEMA(relstr,
                        relcnstr),DOMAINS))

type : Rel ---> BOOL


compatible (rel,rel') =

let tplstr be tpl-structure(rel) ,
    tplstr' be tpl-structure(rel')
in tplstr = tplstr'

type : Rel Rel --->BOOL


tpl-structure(rel) =

(is-Query(rel) --->  cases rel :
mk-RELID(relid) ---> mk-TUPLESTR(s-RELSTR(s-RDBSTR
                     (c RDBSCHEMA)(relid))) ,

mk-Selection(rel', ) ---> tpl-structure(rel') ,

mk-Join(rel', ,rel", ) ---> tpl-structure(rel') u
                            tpl-structure(rel") ,

mk-Projection(rel',attrids ) ---> tpl-structure(rel')
                                  | attrids ,

mk-Union(rel', ) ---> tpl-structure(rel') ,
```

```
mk-Division  ...

...

mk-Rename(rel',bijection) --->
  let tplstr be tpl-structure(rel')
  in [attrid' ---> domid |
                  attrid' = bijection(attrid) and
      attrid ∈ dom tplstr and domid = tplstr(attrid)] ,

mk-Calc-Expression(targetattrids,sourcerels, ) --->
  let tplstr be tpl-structure(rel_1) u ...
                              u tpl-structure(rel_n)
      s.t. rel_i ∈ sourcerels
           and 1 ≤ i ≤ n = card sourcerels
  in tplstr | targetattrids                ) ,

is-Extrel(rel) ---> mk-TUPLESTR(s-RELSTR(rel)))

type : Rel ---> TUPLESTR
```

B.3 Elaboration Functions for Relational Databases

The meaning of an operation is specified by:

- An evaluation function, eval-Query, if the operation is a query. This function states which relation value, RELVALUE, is computed as the result of a query.
- An interpretation function, int-Altop, if the operation is an altering operation. This function specifies the changes of the actual database value, RDBVALUE, caused by the interpretation of that relation altering operation.

```
elab-Op(op) =

  cases op:
  (is-Query(op) ---> eval-Query(op) ,
   is-Altop(op) ---> int-Altop(op))

type : Op ---> RELVALUE | (RDBVALUE --->RDBVALUE)


eval-Query(query)

    let setrep be extract-setrep(query) ,
        predrep be extract-predrep(query)
    in mk-REVALUE(setrep,predrep)

type : Query ---> RELVALUE


int-Altop(altop) =

    cases altop :
    (mk-Insert(relid,rel) --->
    let setrep be
        extract-setrep(relid u extract-setrep(rel) ,
       predrep be extract-predrep(relid)
                  or extract-predrep(rel)
    in let relvalue be mk-REVALUE(setrep,predrep)
       in RDBVALUE := c RDBVALUE
                      + [relid ---> relvalue],

    mk-Delete(relid,rel) --->
    let setrep be extract-setrep(relid)
                  \ extract-setrep(rel) ,
        predrep be extract-predrep(relid) and
                   not extract-predrep(rel)
    in let relvalue be mk-REVALUE(setrep,predrep)
       in RDBVALUE := c RDBVALUE
                      +[relid ---> relvalue],
```

```
    mk-Replace(relid,rel) ---> ...        )

type : Altop ---> (RDBVALUE ---> RDBVALUE )


extract-setrep(rel) =

    (is-Query(rel) ---> cases rel :
    (mk-RELID(relid) ---> s-SETREP(c RDBVALUE(relid)),

     mk-Selection(rel',selpred) --->
        {tupleval | tupleval ∈ extract-setrep(rel')
                and selpred(tupleval)} ,
     mk-Join(rel',attrid',rel",attrid") --->
        let setrep' be extract-setrep(rel') ,
            setrep" be extract-setrep(rel")
        in {tupleval' u tupleval" |
            tupleval' ∈ setrep' and
            tupleval" ∈ setrep" and
            tupleval'(attrid') = tupleval"(attrid")},

     mk-Projection(rel',attrids) --->
        {tupleval' | tupleval' = tupleval
                   | attrids and
                     tupleval ∈ extract-setrep(rel')}

     mk-Union(rel,rel") ---> extract-setrep(rel')
                             u extract-setrep(rel"),

     mk-Division ...

     ...

     mk-Rename(rel',bijection) --->
       {[attrid' ---> dem
                   | attrid' = bijection(attrid) and
                     attrid ∈ dom tupleval and
                     dem = tupleval(attrid)]
                   | tupleval ∈ extract-setrep(rel')}



mk-Calc-Expression(targetattrids,sourcerels,selpred)
--->
    let prodset be {tupleval_1 u ... u tupleval_n |
                    tupleval_i ∈ dom extract-setrep(rel
                    and rel_i ∈  sourcerels
                    and 1 ≤ i ≤ n = card sourcerels }
    in {tupleval' | tupleval' = tupleval
                  | targetattrids and
                    tupleval ∈ prodset and
                    selpred(tupleval) } ) ,

is-Extrel(rel) ---> s-SETREP(s-RELVALUE(rel)))

type : REL ---> SETREP


extract-predrep(rel) =

(is-Query(rel) ---> cases rel :

(mk-RELID(relid) ---> s-PREDREP (c RDBVALUE(relid)),

 mk-Select(rel',selpred) ---> extract-predrep(rel')
                              and selpred,

mk-Join(rel',attrid',rel",attrid") --->
   let predrep' be extract-predrep(rel') ,
       predrep" be extract-predrep(rel")
   in [(tupleval' u tupleval") --->
      predrep'(tupleval') and predrep"(tupleval")
      and tupleval'(attrid') = tupleval"(attrid")|
      tupleval' ∈ dom predrep'
      and tupleval" ∈ dom predrep"],
```

```
mk-Projection(rel´,attrids) --->
   let predrep be extract-predrep(rel´)
   in [tupleval´ ---> predrep(tupleval) |
       tupleval´ = tupleval | attrids and
       tupleval ϵ dom predrep]      ,

mk-Union(rel´,rel") ---> extract-predrep(rel´) or
                         extract-predrep(rel"),

mk-Division ...

...

mk-Rename(rel´,bijection) --->
 let predrep be extract-predrep(rel´)
 in [tupleval´ ---> predrep(tupleval) |
     tupleval´ = [attrid´ ---> dem |
                  attrid´ = bijection(attrid) and
                  attrid ϵ dom tupleval and
                  dem = tupleval(attrid)]
     and tupleval ϵ dom predrep ] ,
```

```
mk-Calc-Expression(targetattrids,sourcerels,selpred)
--->
   let prodpred be [(tupleval_1 ⌣ ... ⌣ tupleval_n)
                    --->
                    extract-predrep(rel_1)(tupleval_1)
                    and ... and
                    extract-predrep(rel_n)(tupleval_n)|
                    tupleval_i ϵ dom extract-predrep(r
                    and rel_i ϵ sourcerels
                    and 1 ≤ i ≤ n = card sourcerels]
   in [tupleval´ ---> prodpred(tupleval) |
                    tupleval´ = tupleval
                                | targetattrids and
                    tupleval ϵ dom prodpred] ),

    is-Extrel(rel) ---> s-PREDREP(s-RELVALUE(rel)))

   type : Rel ---> PREDREP
```

An objective of the RTG was to evaluate the state-of-the-art concerning DBMSs that are claimed to support aspects of the RDM. The RTG found that over 60 DBMSs were claimed to support significant aspects of the RDM. Due to limited resources, a detailed analysis of all such systems was not possible. The Relational DBMS Survey was conducted to evaluate the state-of-the-art in broad detail. The survey attempted to determine the extent of design, development, and use of Relational DBMSs (RDBMS). It was intended to:

1. identify all major RDBMS development efforts in both research and commercial settings, and

2. characterize each effort briefly in terms of interfaces, availability and extent of use.

C.1 Performing the Survey

The RTG solicited this information through announcements and press releases to trade and academic periodicals, by letter, and at database conferences. Shortly, after the RTG's formation, an announcement of the RTG's charter and a request for information on RDBMSs was placed in a variety of publications (e.g., SIGPLAN Notices Volume 14, Number 12, December 1979, and Performance Development Corporation Database Newsletter Volume 7, Number 6, November 6, 1979).

In January 1980, the RTG announced the survey and feature analysis and solicited contributions by means of the following SPARC press release:

ANSI Task Group Solicits Input
Towards Relational Standard

The Relational Database Task Group of the ANSI/X3/SPARC Database Systems Study Group is investigating the justifiability of a standard for relational data base management systems (RDBMS's). It will identify or establish:

- a working definition for the relational data base model;
- the role of the relational data base model appropriate for standards development (e.g., relational languages);
- terminology and concepts for the relational data base model; and
- the relationship of a RDBMS standard with other software standards that exist or are in development.

The task group solicits comments, position papers, and reports on related research and development projects, and on existing and proposed RDBMS's and relational languages. Responses received by January 31, 1980 will be used in the development of a survey and feature analysis of RDBMS's to be published in 1980.

Submissions should be sent to either of the task group co-chairmen:

Michael L. Brodie	Joachim W. Schmidt
Department of Computer Science	Fachbereich Informatik
University of Maryland	Universitaet Hamburg
College Park, Maryland	Schlueterstr. 70
20742 U.S.A	D-2000 Hamburg 13
(301) 454-2002	West Germany
	(040) 4123-4164

The press release was widely distributed and was included in: Performance Development Corp. Database Newsletter Volume 8, Number 1, January 1980; SIGPLAN Notices Volume 15, Number 2, February 1980; Communications of the ACM Volume 23, Number 2, February 1980; and many of the trade publications such as Computer World.

Representatives of twenty two systems responded to the press release. Twelve respondents ultimately completed a detailed feature analysis of their system (see Section 4). The RTG felt that many other systems existed for which no response was received. Hence, the survey became independent of the feature analysis. A one page questionnaire was created and the following press release was issued by SPARC in March 1981.

Survey of Relational DBMSs
by X3's Relational Database Task Group

Implementors of database management systems (DBMSs) supporting aspects of the relational data model are being invited to complete a brief, one page questionnaire to describe their systems. This survey is part of an investigation by X3's Relational Database Task Group (RTG) into the potential standardization of the relational approach to databases. The survey will be presented to X3 soon after it is compiled in early June 1981 in support of a future DBMS standardization development effort of the American National Standards Institute (ANSI).

The questionnaire must be obtained before

MAY 1, 1981

by writing to:

Harrison R. Burris
Neotechnic Industries Inc.
P.O. Box 277
Redondo Beach, CA 90277

Only those completed questionnaires received by

JUNE 1, 1981

will be included in the survey results. A copy of the survey will be sent to all contributors.

For further information telephone the RTG chairman, Prof. Michael L. Brodie at the University of Maryland, 301-454-2002 or Harrison R. Burris at TRW Defense and Space Systems Group, Redondo Beach, CA, 213-535-1047.

The press release was widely published in trade and academic periodicals (e.g., SIGPLAN Notices Volume 16, Number 5, May 1981).

The following cover letter and questionnaire was mailed to all respondents of both press releases, RTG members, and individual and organizations known to the RTG to be candidates for the survey.

American National Standards Committees
X3 -- Computers and Information Processing
X4 -- Office Machines Supplies

operating under the procedures of the
American National Standards Institute

March 24, 1981

Dear Colleague:

Implementors of relational database management systems are being invited to complete the attached

questionnaire. This survey is part of an investigation by the American National Standards Institute (ANSI) X3's Relational Database Task Group (RTG) into the potential standardization of the relational approach to databases.

The questionnaire must be returned to:

Harrison R. Burris
TRW DSSG
One Space Park, 114/2740
Redondo Beach, CA 90278

no later than 31 July in order to be included in the survey results. A copy of the survey will be sent to all contributors after presentation to X3 sometime in October 1981.

Sincerely,

Harrison R. Burris

QUESTIONNAIRE

The Relational Task Group will use this questionnaire to construct a roster of relational database systems. Your assistance will ensure a more thorough job; please complete as much of the questionnaire as possible.

1. Name of System:_____

2. Name of vendor/implementor:_____

3. Contact person (name, address, and phone number):

4. General description (250 words maximum):

5. Does the system have the following interfaces?

	Yes	No
Conceptual database description (schema)		
Relational external schema		
Interactive query/update language		
Data dictionary		
Relational sub-language adapted to host languages		

Specify programming languages:_____

6. Date system available:_____

7. Current Applications:

8. List published documentation

The 13 responses received by July 31, 1981, were incorporated into the survey results given in the remainder of the section. Additional information about RDBMSs has been included from [BROD80d]; [KIM79].

53

C.2 RDBMS Survey Results

Disclaimer

Whereas the RTG has attempted to identify systems supporting significant aspects of the RDM, the RTG does not explicitly or implicitly declare the systems named in the survey to be relational in the sense discussed in other parts of this report.

INTERFACE TYPE:
- CS = conceptual database description (schema) language
- ES = relational external schema language
- QL = interactive query/update language
- DD = data dictionary
- RSL = relational sub-language adapted to host languages
- "x" = indicates that the system supports such an interface. Generally, no "x" indicates that no such interface exists.

STATUS
- C = available for production use
- R = research prototype
- D = design stage
- E = education

SYSTEM	DEVELOPER/VENDOR	INTERFACE TYPE					AVAILABILITY	
		CS	ES	QL	DD	RSL	STATUS	DATE
ADAPLEX	Computer Corporation of America 575 Technology Square Cambridge, MA 02139	x	x	x	x	x	D	
ARCH:MODEL	Architectural Research Laboratory University of Michigan Ann Arbor, MI 48109	x		x		x	C	
ARTEMIS	Meitier Management Systems Inc. Derek Hardy 10175 Harwin Dr., Suite 100 Houston, TX 77036	x	x	x	x		C	6/77
ASTRAL	Department of Computer Science University of Trondheim N-7034 Trondheim-NTH Norway	x	x	x	x	x	E,R	
CAFS	International Computers Ltd. UK						C	
CASSM	University of Florida	x		x			E	
CREATE/3000	CRI, Inc. 2570 El Camino Real Mountain View, CA 94040	x		x		x	C	
CS4	KTH Sweden						C,R	
DTSS	DTSS Inc. (David Goldberg) 10 Allen Street Hannover, NH 03755	x	x	x	x	x	C	1/79
DAPLEX	Computer Corporation of America 575 Technology Square Cambridge, MA 02139	x	x	x	x	x	D	
DATACOM/DB	DATACOM Division Applied Data Research 8515 Greenville Ave., Suite 101 Dallas, TX 75243	x	x	x	x	x	C	2/73
DIRECT	Bancohio Corporation 770 W. Broad St. Columbus, OH 43222	?	?	?	?	?	C	?
DMS-170	Control Data Corporation 8100 34th Ave. South Minneapolis, MN 55440	x				x	C	?
ECOMPASS (ENFORM)	Tandem Computers Inc. 19330 Vallco Parkway Cupertino, CA 95014	x		x		x	C	1981
FADABS	Kernforschungszentrum Karlsruhe GmbH Institut fuer Datenverarbeitung Postfach 3640 D-7500 Karlsruhe, West Germany	x		x		x	C	1978
FQL	University of Philadelphia			x			R,E	
FRAMIS	Lawrence Livermore Laboratory	x		x	x		C	9/78
GAMMA-0	IBM, Yorktown Heights	x				x	R	1972
GMIS	MIT/IBM Donovan (MIT)			x			R	1976
GRAM	IBM	?	?	?	?	?		
IDBP (Intel Database Processor)	Intel Corporation 3065 Bowers Ave. Santa Clara, CA 95051	x	x	x	x	x	C	late 1981
IDM	Britton-Lee, Inc.	x	x	x	x		C	12/80

SYSTEM	DEVELOPER/VENDOR	INTERFACE TYPE					AVAILABILITY	
		CS	ES	QL	DD	RSL	STATUS	DATE
IMPS	IBM	x	x	x	x	x	C	1980
INFOSYSTEM	Henco Inc. Wellesley, MA	x	x	x		x	C	1975
INGRES	Relational Technology, Inc. 2855 Telegraph Ave., Suite 515 Berkeley, CA 94405	x	x	x	x	x	C	9/80
IPIP	Boeing Computer Services (IPAD Project)	x	x	x		x	D	
JANUS	Massachusetts Institute of Technology	x				x	C,E	
MADAM	Massachusetts Institute of Technology	?	?	?	?	?	E	?
MANAGE	CSC Infonet Division 650 N. Sepulveda Blvd E640 El Segundo, CA 90245	x	x	x	?	x	C	?
MAGNUM	Tymeshare Inc. Magnum Development Department 20705 Valley Green Dr. Cupertino, CA 95014	x	x		x		C	1975
MINISIS	Intrl Development Research Center 60 Queen St. Ottawa, Ontario Canada	?	?	?	?	?	C	?
MRDS/LINUS	Honeywell Inc. Corporate Technical Center 10701 Lyndale Ave. South Bloomington, MN 55420	x	x	x	x	x	C	3/78
MRS	Dennis Tsichritzis University of Toronto Toronto, Ontario Canada	x			x		C,R,E	
ODEX	Dr. Gianola Maurizio Olivetti OWPOA/TS ICO 2nd piano v. Jervis 77 Ivca, Italty	x	x		x	x	C	1981
NOMAD 2	NOMAD Development National CSS Inc. 187 Danbury Road Wilton, CN 06897	x	x	x	x	x	C	1975
ORACLE/SQL	Relational Systems Inc. 3000 Sand Hill Rd. Menlo Park, CA 94025	x	x	x	x	x	C	1979
OMEGA	University of Toronto	x	x			x	E,D	
PASCAL/R	Universitaet Hamburg Fachbereich Informatik Schlueterstrasse 70 D-2000 Hamburg 13 West Germany	x	x	x	x	x	R (D) E	1977
PHLOX	INRIA B.P. 105 F-78150 Roquencourt France	?	?	?	?	?	R	
PLAIN	Dept. of Medical Information Science University of California San Francisco, CA	x	x	x	x		R	
PLANES	University of Illinois			x			E,R	

SYSTEM	DEVELOPER/VENDOR	INTERFACE TYPE					AVAILABILITY	
		CS	ES	QL	DD	RSL	STATUS	DATE
POREL	Universitaet Stuttgart Azenbergstrasse 12 D-7 Stuttgart 1 Postfach 560 West Germany	x	x	?	?	?	E,R	
PRTV(IS/1)	IBM UK Scientific Center Athelston House St. Clement St. Winchester Hampshire SO23 8UT UK	x	x	x	x	x	C,R	1973
Extended QBE	IBM Scientific Center Heidelberg West Germany	x	x	x	x	x	R	1979
QBE	IBM 1133 Westchester Ave. White Plains, NY 10604	x	x	x	x	x	C	1978
PARIOU	M. Schneider, J.C. Perraud CUST BP 63 170 Aubiere France	x	x	x	x		E	1981
PINDAR	H. Fergen GMD/IIG D-5205 St. Augustin 1 Postfach 1290 West Germany	x		x			C	11/81
QUESTOR	Comshare, Inc. 3001 South State St. Ann Arbor, MI 48104	x		x			C	?
RAP	University of Toronto					R		
RAPID	Database Management Systems 12/P R.H. Coats Bldg Tunney's Pasture Ottawa H1A 0T6	x		x		x	C	?
RAPPORT	Logica Ltd. 341 Madison Ave. New York, NY 10017	x		x		x	C	?
RARES	University of Utah						R	
RELIANCE	Perkin-Elmer Data Systems 277 Bath Road Slough, Berkshire SL1 4AX UK	x	x	x	x	x	D	6/82
RENDEZVOUS	IBM Research, San Jose	?	?	x	?	?	R	
REL*STOR (RELATE 80)	GTE Sylvania P.O. Box 188 Mountain View, CA 94042	x		x		x	C	3/81
RIGEL	University of California, Berkeley	x	x				R	
RIM	Boeing Computer Services (IPAD Project)	?	?	?	?	?	C	?
RISS	Forrest Hospital	x		x			C	
RPS1100	Sperry-Univac		x	x			C	?
RTFILE	Inter Project Inc. P.O. Box 13 Brentwood, MD 20722	x					C	?
RDBMS	International Computers, Ltd. UK	?	?	?	?	?	C	?

SYSTEM	DEVELOPER/VENDOR	INTERFACE TYPE					AVAILABILITY	
		CS	ES	QL	DD	RSL	STATUS	DATE
RDBMS	Open University Mr. M.A. Newton Academic Computing Service Milton Keynes MK7 6AA UK	x	x	x	x		R	
RDMS	Massachusetts Institute of Technology	?	?	?	?	?	R	
RDMS(REGIS)	IBM UK	x	x	?	?	x	C	?
SESAM	SIEMENS AG Holger Conradi D ST DB22 Otto-Hahn-Ring 6 8000 Munich 83 West Germany	x	x	x		C	10/80	(v12.1) 10/82 (v13)
SDD-1	Computer Corporation of America 575 Technology Square Cambridge, MA 02139	x		x	x	x	R	1979
SYNTEX-II	R. Demolombe, J.-M. Nicolas, M. Lemaitre ONEA-CERT 2 Avenue Edouard Belin B.P. 4025 31055 Toulouse France	x		x		x	R	1975
SYSTEM 38	IBM Highway 52 & NW 37th St. Rochester, MN 55901	x	x	x	x	x	C	9/80
SYSTEM 3000	MRI Systems Corporation						D	
SYSTEM-D	Hannu Kangassalo University of Tampere P.O. Box 607 SF-33101 Tampere 10 Finland	x	x	x	x	x	D	
SYSTEM-R/SQL	IBM Research, San Jose	x	x	x	x	x	R	1975
SQL/DS	IBM Armonk, NY 10504	x	x	x	x	x	C	1982
HIBE	STIS 3135 South State St. Suite 104 Ann Arbor, MI 43104	x	x	x	x	x	C	1982
VERSO	INRIA F. Bancilhon, M. Scholl B.P. 105 F-78153 Le Chesnay Cedex France	x		x	x		R	
ZETA/TORUS	Dennis Tsichritzis CSRG Toronto, Ontario Canada	x	x			x	D	

Textual Responses

The following information was supplied by those that responded to questions 4, 5, 7, and 8.

ARTEMIS

General Description:

ARTEMIS is a mobile, self-contained Project Management Information System based on a dedicated minicomputer. In addition to in-built procedures for planning, scheduling, cost reporting, and probabilistic analysis, ARTEMIS also offers:

- a relational database featuring total field independence with dynamic dataset structuring and linking
- a simple yet powerful command language which is identical for all modules
- a versatile report writer
- a wide range of system options in both hardware and software
- communication with large mainframe computers
- concurrent processing of user-developed applications written in other programming languages
- graphical output of "S" curves, barcharts, histograms, and pie charts, using an 8-color flatbed plotter onto paper or transparencies, or projecting from the terminal directly onto a screen
- automated network logic drawings, either linear, variable, or non-timescaled, onto a 36" 4-color sum plotter
- annotated barchart drawings on either drum or flatbed plotter
- expandable up to 32 terminals, local or remote, and online disc storage up to one billion bytes

Current Applications:

Industry types include offshore oil, civil construction, aerospace and weapons, petrochemicals, automobiles, pharmaceuticals, energy, and government agencies. Application examples include planning and scheduling, cost/schedule integration, performance measurement, management reporting and forecasting, cost management, materials management, records management, plant and equipment registers, and maintenance planning and control.

Documentation

1. Network Planning System User Guide
2. Information Processing and Reporting System User Guide
3. Systems Managers Manual (4 versions for different memory capacities)
4. Management Graphics User Guide
5. Network Graphics User Guide
6. 2780 Emulator User Guide
7. PAN User Guide

DTSS (DaTaSyS)

General Description:

All data is stored in flat files. All access to data is via "views" of the files. Each file has full access to a master view. Subviews with restricted or limited access to selected records and fields may be created. Such subviews may have operation restrictions. Views may be combined using three operators, JOIN, INCLUDE, and INSERT. INCLUDE is similar to JOIN, but only zero or one matches are permitted. If view EMP INCLUDES view DEPT, each EMP record will occur in the result even if no matching DEPT record is found. INSERT creates "hierarchical" views. If view DEPT INSERTs view EMP, each DEPT record will appear once in the result, followed by the matching EMP records.

Interface Details:

Query Language: TELL*ME. Accessible from: COBOL, PL/1, BASIC, FORTRAN, assembly (GMAP).

Current Applications:

Treasury Management, Law Case Management, Executive Report Retrieval, DTSS Customer Billing, DTSS Sales Prospects, DTSS System Bugs and Proposals.

Documentation:

1. DaTaSyS, The DTSS Database Management System 5 MAR 80 (revision 2A).
2. DaTaSyS Reference Manual, Pub. 1311.
3. User's Manual for DaTaSyS***:GUIDE, Pub. 1331.
4. Introduction to DTSS Software, Pub. 5322.
5. User's Guide to the TELL*ME Query System.

ENCOMPASS

Interface Details:

Accessible from: COBOL, TAL, FORTRAN, MUMPS.

FADABS

General Description:

FADABS is a DBS for the use in minicomputer based technical, industrial, or scientific application systems. Its architecture makes it suited to this class of application systems: (1) proper modularization helps keeping programs small (application programs as well as the DBS tasks themselves); (2) the usually powerful process synchronization and communication features provided by minicomputers are utilized to achieve realtime behavior of DBS-application programs; (3) a low level (i.e., tuple-at-a-time) relation interface to FORTRAN, with binding of data structures at compile-time, as a means for the implementation of efficient, tailored user interfaces; and (4) two general, high level (i.e., set-oriented, descriptive) interfaces: interactive query facility, report generator.

FADABS is implemented mainly in FORTRAN, t achieve a high degree of portability, and available on SIEMENS 300 and PDP11/40 minicomputers.

Current efforts are directed at making FADABS a generable DBS.

Interface Details:

Accessible from FORTRAN.

Other Details:

Report generator, data entry facility.

Current Applications:

Nuclear materials information systems.

Documentation

1. Polster, F.J.: FADABS: Ein Datenbanksystem fuer die SIEMENS-Prozessrechner S330. Tagungsbericht, 9. Jahrestagung des SIEMENS-Anwenderkreises I, Kernforschungszentrum, 1978, KfK-Bericht 2642.

2. Polster F.J.: Using a Preprocessor to Implement a Data Manipulation Language for a Minicomuter Data Base System. Proceedings First Symposium on Small Systems. ACM SIGSMALL, N.Y., 1978, p. 40-44.

3. Stoeckle, D.; Polster, F.J.: The Report-Definition Language RDL and its Implementation in the Report-Generator FAREG. Report, KfK 2910, Kernforschungszentrum Karlsruhe, 1980. (in German)

4. Polster, F.J.; Tritschler, P.: The Database Query Language FAQUEL. Report KfK 3089, Kernforschungszentrum Karlsruhe, 1981. (in German)

Multics Relational Data Store (MRDS)

General Description:

The Multics Data Base Manager (MDBM) provides Multics users with a general database management facility that is callable from all Multics programming languages supporting a call interface and from Multics command level. A full range of database definition, retrieval, and update capabilities is available, together with facilities that provide a large measure of data independence and control of concurrent accesses to the database.

The MDBM is unique in that it provides an interface for both relational and network (CODASYL) database users. The relational interface, known as the Multics Relational Data Store (MRDS), supports network databases using a subset of the CODASYL standard specification.

Though related to each other at the implementation level, the user should view each interface as a separate and independent database management system.

Interface Details:

Query, manipulation: LINUS (The Logical Inquiry and Update System). Accessible from: FORTRAN, COBOL, PL/1, APL, and any language with a call facility supported on MULTICS.

HIBE (Highly Intelligent Back End)

Interface Details:

Accessible through only high level programming language with a call mechanism. Also has an interactive central language (e.g., self-contained high level programming language).

ODEX (Olivetti Database Executive)

General Description:

ODEX provides services of database definition and manipulation to application programmers; the data are defined and structured at the Logical Schema level and at the Storage Schema level by using the data and Storage Description, are organized at the logical level using a relation model and at the storage level using an indexed file organization. The data manipulation facilities are offered as extensions of the PLZ language, by the Data Manipulation Language. These extensions are processed by a precompiler which generates an extended PLZ source module and an access module, which will then be compiled by the PLZ compiler and linked to the run time support routines. The DML is very high level language based on the values of the data and is completely independent from the physical structures. ODEX provides mechanisms for controlling multiuser operations and facilities to recover a consistent database state in case of a system failure. In particular, actions performed on the database can be grouped in units called transactions, which are guaranteed to be executed as indivisible units. A log facility is provided.

Interface Details:

Accessible through PLZ.

Current Applications:

DBMS for a network monitor system.

ORACLE/SQL

Interface Details:

Query, data definition, manipulation, and control language: SQL. Accessible from: PL/1, COBOL, FORTRAN, C, assembly.

PARIOU

General Description:

PARIOU is a relational database management system which has been designed for educational and research purposes. He is presently available on PDP 10 but is entirely written in standard COBOL. The physical structure is based on inverted files. The system includes algorithms to optimize the process of queries. The actual query language is like ALPHA language. It is possible to implement easily other query languages since all queries in high level language are transformed in standard form by relational algebra operators.

Current Applications:

School attendance database.

Documentation:

1. J.C. Perraud. "Conception d'un système de gestion relationnelles PARIOU", Thesis.

2. B. Collardey. "Realisation du noyau du système PARIOU", Thesis.

3. R. Mpassi et J.L. Robillon. "L'algorithme de recherche du système PARIOU".

PINDAR (Adaptable Interactive Database and Report System)

General Description:

PINDAR is a form oriented interactive system with its own user language and includes functions for the definition of databases (Schema), input, update, deleting of data, comprehensive evaluation of databases and other files, and the generation of reports.

A PINDAR database consists fundamentally of a linear data file and an inverted file. The data file contains records made up of fields and subfields and the inverted file contains references to the data fields. Linkages may be defined between otherwise unrelated databases.

The user language consists of descriptive and procedural elements, and applications of various degrees of complexity may be programmed. A real time menu technique is used, where many options have predefined defaults.

Other Details:

KLDS (compatible database interface adapted to the linear data structure), i.e., host language interface to call PINDAR data management services as defined for the German public services.

Documentation:

PINDAR User Reference Manual (German Edition).

RDBMS (Open University)

General Description:

RDBMS was designed with a primary objective of demonstrating relational concepts for students. It supports all relational algebra operations and is perceived as a relational "calculator", with all operations expressed as a single command (e.g., JOIN) which operate on a workspace relation (using data from a second named relation as appropriate). It has integrated data dictionary facilities and commands to examine all types of data. This includes a domain concept, referred to as a "category" because it is considered more powerful than a mathematical domain, supporting join operations and acting as an integrity constraint for inserting new values for foreign keys (constraints not yet fully implemented). Tuple identification is based on explicitly specified keys, and if a project operation removes any key than it is necessary to make the new key(s) explicit using a KEY function. Aggregate functions are also supported during project operations when the number of tuples is reduced. The conceptual schema is expressed in terms of relations and each external schema is expressed a user view, specifying the relations, data items of each relation and procedures available to that user.

The system runs on a DEC-20, written largely in COBOL and using B*-trees for storage. The user friendly environment (help, recognition) is system dependent.

Current Applications:

University equipment database.

RELIANCE

General Description:

RELIANCE is a high performance transaction processing system with a relational database management capability. There is a Data Dictionary for the definition of data items, groups, and data views. The dataviews represent Join and Projection operations and are available both to the forms driven query/report generator and to the COBOL and FORTRAN programmer interface. The underlying disc manager is optimized for transaction processing with advanced data integrity features.

Interface Details:

Accessible (tuple-at-a-time) from COBOL and FORTRAN.

Documentation:

RELIANCE Manuals (5 volume set).

SESAM V 12.1, SESAM V 13

General Description:

SESAM is a commercially oriented Database System whose Data Structures are compatible with the Relational Modul. Its principle features are selection, (field-) projection, Join, update of Relations and Password Protection at Relation, Field, and Activity-level. The system can be accessed by Assembler, COBOL, FORTRAN, ALGOL, RPG and PLI user programs. User programs and DBMSs may be distributed at the users convenience within a computer network.

Single user program can work with several DBMSs simultaneously which may be local, remote, or both. A DBMS may also revive requests from both remote and local user programs).

Interfaces for TP monitors UTM and COSMOS are available. Several Report generators may also be used with SESAM.

Transactions of consistence level 3 with records (Primary Key) as the lock unit will be possible when Version 13 is released.

Current Applications: Approximately 200.

Documentation:

1. Software Product SESAM (BS2000/BS1000) V 12.1, Database Creation and Maintenance, Reference Manual Part 1, No. U 275-J-Z55-1-7600.

2. Software Product SESAM (BS2000/BS1000) V 12.1, Data Retrieval and Direct Update, Reference Manual Part 2, No. U276-J-Z55-1-7600.

3. SESAM auf dem Weg zum relationalen Datenbanksystem
   Part 1 Systemberater published 1.5.80
   Part 2 Systemberater published 1.3.80
   Part 3 Systemberater published 1.9.80
   (W. Schmittel, B. Nauer).

SYNTEX-II

General Description:

SYNTEX-2 is a relational DBMS whose DML is predicate calculus oriented with a mix of indivi-

dual and tuple variables. The DML can be used either in stand alone or embedded into the programming language COBOL. The internal mode relies upon two types of data structures: indexed files and rings. This internal model together with an efficient query optimizer enable good performance.

SYNTEX-2 is programmed into a low level language, LP 70 (very similar to PL 360) on a CII-HB IRIS 80 machine (of a capacity similar to IBM 370-158). The minimum size of core memory needed is 45 K-words (1 word = 32 bits).

Interface Details:

Accessible from COBOL.

Current Applications:

SYNTEX-2 is a prototype which was used for 5 different applications of various sizes, but it is not now under use.

Documentation:

1. "The language of SYNTEX-2 an implemented relational-like DBMS in: Information Technology" Ed.: J. Moneta, North-Holland, JCIT3, 1978.

2. "SYNTEX-2 un prototype de SGBD relationnel" Conf. CIPS-DPMA, Quebec, 1979.

SYSTEM-D

General Description:

SYSTEM D is an experimental, integrated tool for information systems design, implementation, and database management. It is under development at the University of Tampere. It can be used as a relational database management system but it can also be used as a tool for database design and data processing systems design. It provides a user an external view both on the conceptual level and on the relational level. It provides also a spectrum of different query languages. The database administrator (DBA) can see all four schema levels: conceptual, relational, data structure, and storage structure levels. The DBA has at its disposal a special language, called Data Administration Language, which can be used to design, define, and maintain the database. SYSTEM D has an Adaptive query optimizer which can use different sets of optimization rules in different situations. If information requirements of a user are formulated as a query, then SYSTEM D can be used as an automated design tool which produces a set of programs and optimized intermediate files which corresponds to the original query.

Interface Details:

Accessible from PASCAL AND FORTRAN.

Documentation:

H. Kangassalo, H. Jaakkola, K. Jarvelin, T. Lehtonen, and T. Niemi: SYSTEM D — An integrated tool for systems design, implementation, and database management. Report, University of Tampere, Department of Mathematical Sciences, June 1981.

VERSO

General Description:

This is a database machine on which we are implementing a relational system. The hardware is based on the idea of "on the fly" filtering of data. The DBMS offers a standard low level interface (Physical Management, Algebraic queries and update processing, concurrency control and recovery) but no protection on end user interface.

Other Details:

Supports concurrency and functional dependency checking.

Documentation:

1. "Design of an I/O Processor for a Relational Database Machine", SIGMOD80, Bancilhon/Scholl.

2. "The database Machine VERSO: Binary Operation", 6th Workshop on Nonnumeric Architecture, 1981 (Hyers).

DISEÑO DE BASES DE DATOS

EVALUATION CRITERIA FOR LOGICAL DATABASE DESIGN METHODOLOGIES

MAYO, 1985.

Evaluation criteria for logical database design methodologies

A P Buchmann* and A G Dale†

The increasing acceptance of database systems has shown the need for logical database design methodologies and evaluation criteria to select the one most appropriate for a particular task. Application of currently existing methodologies to a portion of the data handled during chemical process plant design has highlighted strengths and shortcomings of the methodologies analysed. Using these results, evaluation criteria are established and synthesis of a methodology which is responsive to the identified criteria is proposed.

INTRODUCTION

Interest in database design methodology has developed in the context of applying database management technology to support the integration of engineering design of chemical process plants.

In all engineering desciplines evolution has led from intuitive to systematic design procedures. Database design is no exception. Many investigators in the area have recognized the need for a systematic approach. Their attempts have been reviewed by Fry and Novak[1] and Yao Navathe and Weldon[2]. While existing attempts at proposing a database design methodology have many points of interest, they rely to some extent on shortcuts and intuitive assumptions. A logical database design methodology which does not omit any steps, and which covers the whole range of logical database design activities is required.

In this paper the environment in which the present work evolved is discussed along with a view of the database design process and the reasons for use of database design methodologies. Some of the proposed methodologies and how they perform in the environment analysed are presented. From their observed strengths and shortcomings a set of criteria for evaluating a methodology is derived. The last section suggests the synthesis of a new methodology responsive to the stated criteria.

ENVIRONMENT

Databases can play several basically different roles in supporting the engineering design of a chemical plant. For example:

- as an updatable description of the current status of a project, from which information used as input to a particular task can be obtained, thus enhancing communication amongst users
- as a private working-space for a small group of users serving as an archival system for preliminary design alternatives.

If the database is viewed as a record of progress of a project, it will serve as a communication medium among groups of designers carrying out various tasks in the overall design and will contain already-approved data released for use by other groups. Well-defined channels of communication exist for this purpose within an organization and information is traditionally presented to the user in form of familiar documents, e.g. flow sheets, load sheets, schematic P and I diagrams, specification sheets, etc. Data have to be supplied also to design–software packages which should rely primarily on the information contained in the project-wide database. This type of database reflects the gradual refinement of the project and grows monotonically. Various degrees of detail have to be represented, depending on the stage of completion of the project. For example, bidding information can range from coarse estimates to data obtained from an almost completed design, but will be in most cases less detailed than information used for subcontracting or the shop information and the final project documentation. The dependence of many applications on the same data or dependence on data generated by intermediate applications cause any update to propagate throughout the database. An example is the process design stage whose output serves as problem definition for later tasks in the design chain. Updates, therefore, have to be well-documented in the form of revisions and alerts to users of the updated information are necessary.

In particular, the interfaces among the process engineering group, the systems group and one equipment design group, the heat exchanger design group are examined. Information is transferred from the process engineering group in the form of a process release, consisting of a flowsheet, a material and energy balance, a major equipment list with identification of accounts, and load sheets. Detailed specification sheets and drawings are considered as the documents produced by the heat exchanger group, and valve, pipeline calculations, connection information sheets, etc. as the information provided by the systems group. The characterization of information was provided by the Pullman–Kellogg Company in the form of document blanks and through discussions with one of the authors.

As a first step in using database management technology, modelling the existing communication system

*Department of Chemical Engineering, University of Texas, Austin, Texas, USA
†Department of Computer Science, University of Texas, Austin, Texas, USA

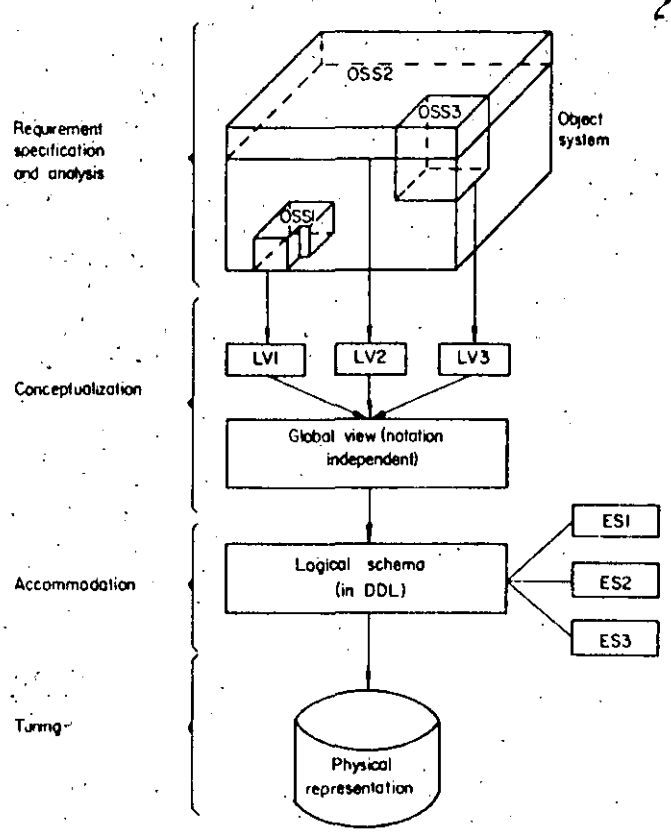


Figure 1. Database design process

which has proven to be adequate for engineering purposes is proposed. The following examination of the problems of database design methologies is made in the context of this application.

DATABASE DESIGN PROCESS

The term 'database' is an elusive one. In this context a database is considered to be a model of a portion of reality represented by an evolving collection of interrelated data accessible to several applications. These data have a controlled redundancy and the user has the capability of inserting, deleting, updating and retrieving data in a restrained form through the actions of a database management system (DBMS).

The extraction of all the characteristics of the object system which is to be represented in the database is the first step in any database design, and initially involves user requirements specification and analysis. The representation of the object system's properties and their integration into one conceptual model is the next portion of the logical database design. The model obtained by this process is an abstract model which represents the inherent properties of the object system and is independent of any model or notation supported by the DBMS. In order to be processable the conceptual model has to be expressed in terms compatible with a particular DBMS. This is the database accommodation step. Since local views were integrated at a high level of abstraction to yield a binding model common to all applications, it is necessary to represent now the user's views in terms of the data description language (DDL) supported by the particular DBMS. The whole process, from determination of user requirements through conceptual model and final accommodation in terms of the selected DBMS, is the logical database design process.

The internal organization of the data representing the object system and its models on physical devices constitutes the physical design of the database. Figure 1 shows schematically a view of the database design process.

The strong effect that any misconception of the object system or misrepresentation of its characteristics in the conceptual model has on the ability of the database to satisfy the users' information needs has centred attention on the logical database design stage. The systematic process by which one traverses the different stages of the logical database design and performs the mappings from one level of abstraction to the next is commonly called a logical database design methodology.

Two parameters have a strong influence on the complexity of database design:

- the number of objects to be represented,
- the number of interrelationships amongst objects

The area above the curve in Figure 2 represents the area in which intuitive design becomes most difficult. To model an object system in which many objects and activities are interrelated a systematic and well-structured approach is necessary. It is therefore instructive to observe how proposed methodologies perform in a large-scale application, what criteria should be applied for evaluation of existing data modelling techniques, and what consequences can be drawn for the development of future ones.

Since database design techniques have evolved largely for the purpose of supporting business applications it is useful to analyse the capabilities of various methodologies in the technical design environment described in the previous section. The remainder of this paper will therefore discuss observed strengths and weaknesses of the examined methodologies as a basis for defining a set of practical criteria for evaluating a methodology. Since no examined methodology fulfills all the criteria established, the final section states the need for the synthesis of a new methodology responsive to the postulated criteria. As the term 'synthesis' indicates, this should not be a complete redevelopment, but an assembly of the best parts of already proposed

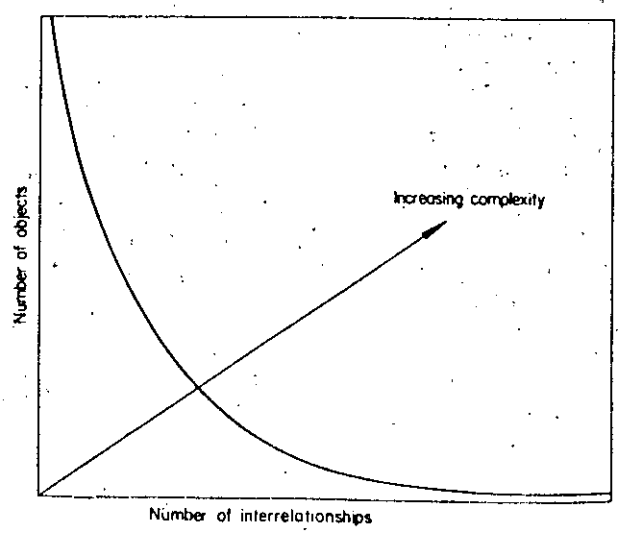


Figure 2: Boundary of feasibility for intuitive logical database design

design techniques integrated into a framework capable of bridging existing methodological shortcomings.

METHODOLOGIES ANALYSED

Database design methodologies with markedly different characteristics and origin are examined in order to cover a wide spectrum of features. The selection under consideration is limited but is a fair representation of the tendencies noticeable in research in this field. Two basically different trends can be observed. One approach first generates a global schema and then derives local views from it; this approach is characteristic of much current work in Europe. More popular in the US is an approach which first models local views of different users and then integrates them to form a global view. Modelling approaches that assume different data models, and which have the potential for future extension and refinement were also selected. The result was the selection of 3 methodologies, namely: Bubenko[3,4], Kahn[5,6] and Smith and Smith[7–9].

Space limitations do not allow an extensive review of these methodologies, and it is suggested that the original work be consulted for further details. Only the salient characteristics and specifically those which will influence the evaluation criteria will be cited.

Bubenko's methodology

Bubenko's work has been evolving and the comments made here apply basically to the version presented in the Inferential Abstract Modelling (IAM) approach[3,4]. The process can be divided into seven iteratively executable groups of activities:

- collection and specification of information requirements
- entity classification
- specification of functional dependencies
- abstract object specification, integration and analysis
- implied information analysis
- derivability (precedence) analysis
- transformation to an external name-based model

The first step consists mainly in a narrative gathering of information requirements and their formulation in terms of queries and transactions. The second step, the classification of entities into concept classes is not further formalized and appears to be an extraction of all the subjects and objects (grammatically speaking) of the query and transaction descriptions. The results of this step are highly dependent on the care and degree of standardization with which the requirement formulation was carried out. It is conceivable that information is lost or strange entities are added because of poor requirements description by the users. An example occurs in the engineering design application context where document names are well-understood concepts for a designer. In this case, it can happen that query or transaction requirement statements will be formulated in these terms, thereby introducing a report format into the conceptual model as an object. Any change in reporting procedure would imply a change in the conceptual model.

The grouping into concept classes proposed by Bubenko is highly intuitive and not likely to be formalized because of the application dependence of this step. However, guidelines would be helpful.

The third step, the determination of functional dependencies, is a monumental task in a semantically-rich application. The IAM methodology recognizes various kinds of functional dependencies. In the analysed context some examples could be:

F(1): tubesheet – component of – → heat exchanger
F(2): equipment function ← – id – → equipment identifier
F(3): flowrate, ΔP_{max} – – d – → pipe diameter
F(4): composition, temperature, pressure – – mat – → valve material

Several problems are encountered in an engineering environment, the most difficult being who should define the dependencies. A database designer cannot be expected to be familiar with every aspect of an application and hence cannot discern all possible dependencies. The user, on the other hand, has a limited perception of the whole problem space. In addition, some functional dependencies [e.g. F(3) and F(4) above] represent design decisions and some dependencies can be expected to change from project to project.

The abstract objects are determined as a result of the previous steps and it may be necessary to iterate. It should be noted that this approach uses two levels of abstraction. Only later are the abstract objects mapped into a name-based model. The implied information analysis comprises the identification of implied and derived associations and the corresponding rules. An association is implied if it is functionally dependent on a proper subset of the initial or identifying associations related to a particular abstract object. Identifying associations are those that cannot be removed without destroying the 1:1 correspondence property. Associations are considered derivable if they are fully dependent on the set of identifying associations. The precedence or derivability analysis serves as a test to ensure that the initial abstract objects and the defined rules are sufficient to satisfy all the anticipated information requests. This abstract model is mapped in the last step into a name-based model through the introduction of name sets.

One of the interesting features of Bubenko's approach is the inclusion of a time dimension, but this characteristic appears not to be operational. A strong point of the methodology is the separation of an abstract and a name-based level. The use of inferred and deduced information provides a possibility for checking for consistency at the logical level, since various paths are available to obtain certain information, but problems are the implementation level could rise.

Kahn's methodology

Kahn's methodology appears to be a good candidate to serve as a framework. A characteristic of this approach is the early separation of the problem into two parallel perspectives, the information structure perspective, which describes the interconnections within an organization, and the usage perspective, which is concerned with satisfying the processing requirements within an organization, and amalgamated these two perspectives should provide the best overall logical design. The separation into two perspectives raises one important issue: should two different extreme schemata be designed, one processing, the other information oriented, which are merged at the end of the logical design to yield the conceptual schema of the enter-

prise or should the design of one, very flexible, information structure first be undertaken, introducing through a gradual refinement those modifications that allow the processing requirements to be fulfilled? An analysis of the evolution of Bubenko's and Kahn's work in the last two years shows a tendency towards the second approach, but neither author makes a strong argument for either alternative. Kahn's methodology basically consists of 6 levels and 5 steps to go from one level to the next:

Levels	Steps
Real world requirements	
	Requirements step
Local information structures	
	Entity step Relationship step
Global information structure	
	Entity structure step
Entity structure	
	Refinement step
Revised entity structure	
	DBMS accommodation step
Logical database structure	

One of the strengths of this methodology is the detailed specification of the input/output for each step, the major shortcoming is the lack of detail in the available papers as to how one should proceed to obtain part of this output.

The requirement step is subdivided into 5 activities: a gross system analysis; a selection of design path between information structure perspective and processing requirements perspective with the former being tacitly preferred; a requirement technique selection; requirement collection and specification; a requirement analysis to produce individual information requirements documents. However, little information is given as to how these documents are produced.

Kahn's methodology aggregates the local views into a community view and insists that each user be responsible for his requirements, thereby avoiding users anonymously stating unreasonable requirements. The entity step serves the purpose of designing global entities which are representative of the organization's aggregated needs. An analysis of the activities performed indicates that this methodology arrives directly at a name-based model. The aggregation to form a nonredundant collection of entities is one of the more difficult tasks and is left largely to the designers' intuition. An equivalent step is carried out for relationships creating global relationships, identifying conditional and existent relationships and eliminating all redundant relationships. It is important to realize that a strictly nonredundant global information structure is obtained here which might be considered as the initial conceptual schema.

In the entity structure step, the global information structure is subsequently mapped into functional dependencies and from there into normalized relations. The published papers on this methodology do not describe what specific algorithm is used for the synthesis of normalized relations. However, according to Bernstein[10] all known algorithms depend on two assumptions: the availability of a set of all functional dependencies, and their uniqueness, i.e. that at most, one functional relationship connects one set of attributes to another. The treatment of functional dependencies in the published algorithms for construction of normalized relations is strictly syntactic and, therefore, the algorithms not only depend on a complete set of functional dependencies, but on a complete set of functional dependencies that cannot lead to invalid syntactic inferences. Kahn's methodology tries to circumvent this problem by resolving the semantic ambiguities in previous steps. By obtaining the functional dependencies from the global information structure rather than directly from the user requirements it is more likely that the strictly syntactic approach will be successful. However, the FDs still have to be checked for compliance with the uniqueness requirements and the necessary axioms for construction of relations. If Bernstein's method is used, these axioms would be reflexivity, augmentation, and pseudotransitivity.

Once in the form of normalized relations, the refinement introduces controlled redundancy, ensures adequate processing and accommodates security and volatility of the data. However, it is possible that some of the actions described in this step will be used for garbage collection, grouping unresolved issues and putting them out of sight.

The DBMS accommodation step comprises all those actions necessary for expressing the conceptual schema in the DDL of a chosen DBMS and requires that a mapping be possible from normalized relations to the model supported by the selected DBMS.

Smith and Smith methodology

The methodology introduced by Smith and Smith ignores the requirement gathering and analysis stage and assumes that the concepts to be integrated are available and are fully understood by the database designer. The basic premise of this work is that abstractions are the means by which humans understand and manipulate complex systems. For database design only abstract objects are considered of interest and they can be formed through generalization and aggregation. An object is formed through generalization from a class of other objects ignoring the differences in favour of the common properties. An abstract object is formed through aggregation by naming a relationship among other objects. Repeated application of generalization and aggregation to the set of names provided by the requirement step results in abstraction hierarchies, which can be visualized as lying in perpendicular intersecting planes. In this methodology the names carry a large portion of the semantics. The authors contend that it is inappropriate to assign a fixed interpretation to an object at the conceptual level. An object can be viewed as entity, component, relationship, category, attribute, or instance, depending solely on the viewpoint of the user. It is claimed that set theoretical models are less restrictive in this sense than graph theoretical models which tend to fix the interpretation. If this is the case, it will be necessary to analyse what restrictions have to be imposed in order to map the conceptual schema obtained through abstractions into a graph oriented DBMS – an important step since graph (network) oriented DBMSs are commercially available.

The methodology consists of few basic steps: generalization, aggregation, instance identification and expression in terms of an abstract syntax. The activities of the abstraction steps are highly intuitive and depend on the insight and abstraction capacity of the database designer.

IDENTIFICATION OF CRITERIA

In the preceding discussion, the salient characteristics of some existing methodologies were identified. In this section a framework for evaluating database design methodologies is outlined and the desirable features of a specific methodology proposed for the area of interest are identified.

Database design methodologies may be characterized by the following tentative list of properties:

- completeness
- design strategy
- requirement specification procedure
- number of abstraction levels
- aggregation process
- separation of information and processing requirements
- scope of the model
- use of functional dependencies
- use of roles
- treatment of security, update propagation, etc.
- modularity
- adaptability to automated design tools

The completeness of a methodology, incorporating smooth transitions from one step to the next and adequate mapping facilities to go from one level of abstraction to another is a necessary condition for acceptability. Gaps in proposed methodologies frustrate practising database designers and lead to *ad hoc* extensions.

Design strategy for software is often discussed in the form of the debate 'top down vs. bottom up'[12]. The labelling of a methodology as top down or requirements first is often confusing. For example, an early version of Bubenko's methodology is referred to once as top down[11] and elsewhere as requirements first[12]. In the case of database design methodologies the debate can be reduced to the question 'what is the initial image of the object system?'. While requirements first methodologies relate directly to the physical object system, top down methodologies use a preconceived mental model of reality which will, most likely, be incomplete and biased. For engineering purposes a requirements first approach is more adequate.

If a requirements first logical database design methodology is advocated, it is necessary to ensure that a good requirement gathering–analysing procedure is part of the selected methodology. The need for this is recognized by most authors but is carried through only by a few. Frequently emphasis is put into the development of requirement specification languages and automated analysis procedures, which are helpful only when the physical reality has been correctly perceived. The problems which are most difficult to eliminate arise from false perceptions of reality, resulting from a natural language barrier between the 'database naive' user and the 'application naive' database designer and the lack of appropriate tools for communication between them.

Most methodologies, having their origin in a business environment, do not consider the major source of information available in engineering environments, the standardized design documents. Each document represents the local view of a small portion of reality and is therefore an excellent starting point in gathering information about that portion of the object system. This approach has proven useful in the application considered here and has the added advantage of being very economic as far as users' time is concerned.

Some of the analysed methodologies include two abstraction levels, while other include one. When two levels are used, the naming procedure is usually delayed and first the physical reality is modelled in terms of abstract objects. Naming occurs at a lower level of abstraction, i.e. a later stage in the design methodology. If only one level of abstraction is provided, the database designer arrives immediately at a name-based model which is more restrictive and may be more difficult to manipulate because of the previously mentioned language barrier and the semantics conveyed by the names. The high semantic content of names makes the assumption that the database designer understands the unique meaning of each name and can resolve any naming conflict questionable. This suggests the use of two levels of abstraction as the preferred approach.

A major difference among existing methodologies is their approach to the aggregation process. Such methodologies aim at a single conceptual model, spanning all the applications in one step. Other methodologies model each application separately in the form of local information structures and later integrate them into a global view. For large engineering applications the second approach appears to be better, but whatever the number of aggregation steps may be, except for the most trivial applications, some support is necessary whenever overlapping of entities and attributes exist. It is this point where most methodologies fall short and leave the problem to the designer's intuition.

Methodologies vary in their consideration of processing requirements. There appears to be a consensus that the information perspective should be given priority. Once this is obtained processing requirements can be introduced either by defining an extreme, processing oriented schema and amalgamating it with the information oriented schema, or a gradual refinement of the information requirements-based schema can be attempted. The latter offers several advantages. Changes are gradual and need only be carried out until the required performance is met, thereby avoiding an overshooting of the performance requirements at the expense of flexibility. However, it is necessary to have a good monitoring mechanism to measure the effects of a given modification.

The scope of the model determines the degree of redundancy at the conceptual level, i.e. should information be stored explicitly or should as much as possible be derivable. If little information is stored explicitly and many inference rules are established there may be alternate ways of deriving desired information, thereby allowing the designer to verify correctness at the conceptual level. There is, however, a trade-off which has to be considered, especially in complex environments. In order to obtain correct inference rules the functional dependencies have to be known and this may be difficult. It can be anticipated than any methodology depending on specifying a complete set of functional dependencies will not be suitable for the design of a complex engineering application database. Emphasis on inference of information may also result in conflicts with the users in our area of interest since many engineering design decisions

are inferences based on the data stored in the database. Therefore extensive use of inference can be part of a database system supporting local workspaces, but not in a system used as the official record of a project.

The varying flexibility with which methodologies treat individual data items, either assigning a fixed role or allowing free interpretation depending on the particular view has to be considered when synthesizing a methodology. It is not feasible to integrate portions of methodologies which differ in their perception of the role of an item.

Before selecting a methodology it is necessary to ascertain if there are any unresolved steps that may play a crucial part. Issues like security, and update propagation are often found in 'garbage collection steps' towards the end of the methodology, but they may influence early stages in the logical database design and should be identified at the policy definition step to be resolved, if necessary, as the design progresses and avoid costly iterations and redesigns.

Modularity is a special concern in relation to the problem of synthesizing a methodology, but it is also necessary when trying to carry out some tasks in an automated or computer-aided form. Design tools are seldom comprehensive and clearly defined stages with well-specified input and output are necessary for a design aid to be useful.

CONCLUSIONS

Some existing methodologies have been outlined and some of the characteristics discussed. This resulted in a series of practical issues to consider in constructing a logical database design methodology for application in an engineering design environment. For the environment under consideration all the screened methodologies presented shortcomings. Consequently we are currently engaged in synthesizing a methodology with the following characteristics:

- a requirements first approach is utilized, even though this carries some risk of bias in favour of existing applications and problems
- the methodology will be comprehensive, i.e. it will include as integral parts (a) the mapping from reality in the form of defined requirement gathering procedures and (b) the mapping to a chosen DBMS in the form of a database accommodation step
- the requirement gathering procedure will make extensive use of engineering documents
- issues like security and propagation of updates will be defined at early stages and included in the requirement gathering/analysis procedure
- guidelines for identification of entities and attributes will be generated
- two levels of abstraction will be incorporated
- algorithms or heuristics for support of the aggregation process will be part of the methodology
- a conceptual schema that stresses the information requirements is favoured, with introduction of processing requirements through gradual refinement. Refinement will occur upon mapping into a DBMS dependent representation
- specification of obvious functional dependencies will be made, but it is recognized that a complete set is very unlikely to be obtained. Inference rules for deriving information will be avoided in favour of explicit storage whenever inference would interfere with the users' decision process
- the methodology will be modular in order to be able gradually to adapt automated design tools

ACKNOWLEDGEMENTS

The authors thank the Pullman–Kellogg Company for providing data used in the present research, and particularly Mr Alex Myles, Mr Carl Albers and Dr Burnelle Landry for their active cooperation. Thanks are also expressed to Professor Janis A Bubenko Jr for his interest and useful comments.

The present research was partly sponsored by Consejo Nacional de Ciencia y Tecnologia, Mexico.

REFERENCES

1 Novak, D O and Fry, J P 'The state of the art of logical database design', *University of Texas Conf. on Computing Systems*, Austin, Texas (Oct. 1976)

2 Yao, S B, Navathe, S B and Weldon, J L, 'An integrated approach to logical database design', *NYU Symp. on Database Design*, New York City (May 1978)

3 Bubenko Jr, J A 'IAM: Inferential Abstract Modeling – an approach to design of large shared databases', *IBM Res. Rep. RC6 343*, Yorktown Heights, NY(June 1977)

4 Bubenko Jr, J A 'IAM: An Inferential Abstract Modeling Approach to Design of Conceptual Schema', *ACM SIGMOD Int. Conf. on Management of Data*, Toronto (August 1977)

5 Kahn, B K 'A method for describing the information required by the database design process', *Proc. SIGMOD Conf.*, Washington DC (June 1976)

6 Kahn, B K 'A structured logical database design methodology', *NYU Symp. on Database Design*, New York City (May 1978)

7 Smith, J M and Smith, D C P 'Database abstractions: aggregation', *Commun. ACM* (June 1977)

8 Smith, J M and Smith, D C P 'Database abstractions: aggregation and generalization', *TODS* (June 1977)

9 Smith, J M and Smith, D C P 'Principles of database conceptual design', *NYU Symp. Database Design*, New York City (May 1978)

10 Bernstein, P 'Synthesizing third normal form relations from functional dependencies', *TODS* (December 1976)

11 Bubenko Jr, J A, Berild, S, Lindencrona-Ohlin, E and Nachmens, S 'From Information Requirements to DBTG – Data Structures', *Proc. ACM-SIGMOD-SIGPLAN Conf. on Data: Abstraction, Definition and Structure*, Salt Lake City (March 1976)

12 Bubenko Jr, J A 'Validity and verification of information modeling', *Conf. on Very Large Databases*, Tokyo (October 1977)

DISEÑO DE BASES DE DATOS

PROGRAMMING-IN-THE LARGE VERSUS PROGRAMMING-IN-THE-SMALL

MAYO, 1985.

PROGRAMMING-IN-THE LARGE VERSUS PROGRAMMING-IN-THE-SMALL

Frank DeRemer
Hans Kron

University of California, Santa Cruz

Key words and phrases
Module interconnection language, visibility, accessibility, scope of definition, external name, linking, system hierarchy, protection, information hiding, virtual machine, project management tool.

Abstract
We distinguish the activity of writing large programs from that of writing small ones. By large programs we mean systems consisting of many small programs (modules), possibly written by different people.

We need languages for programming-in-the-small, i.e. languages not unlike the common programming languages of today, for writing modules. We also need a "module interconnection language" for knitting those modules together into an integrated whole and for providing an overview that formally records the intent of the programmer(s) and that can be checked for consistency by a compiler.

We explore the software reliability aspects of such an interconnection language. Emphasis is placed on facilities for information hiding and for defining layers of virtual machines.

1. Introduction

Programming a large system in any typical programming language available today is an exercise in obscuration. We work hard at discovering the inherent structure in a problem and then structuring our solution in a compatible way. Research into "structured programming" (Dijkstra 1972) tells us that this approach will lead to readable, understandable, provable, and modifiable solutions. However, current languages discourage the accurate recording of the overall solution structure; they force us to write programs in which we are so preoccupied with the trees that we lose sight of the forest, as do the readers of our programs!

Let us refer to typical languages as "languages for programming-in-the-small" (LPSs). Let us use the term "module" to refer to a segment of LPS code defining one or more named resources. Each "resource" is a variable, constant, procedure, data structure, mode, or whatever is definable in the LPS. Preferably a module is one to a few pages long and is easily comprehensible by a single person who understands the intended environment and function of the module.

Work reported herein was supported in part by the National Science Foundation via grant number GJ 36339.

We argue that structuring a large collection of modules to form a "system" is an essentially distinct and different intellectual activity from that of constructing the individual modules. That is, we distinguish programming-in-the-large from programming-in-the-small. Correspondingly, we believe that essentially distinct and different languages should be used for the two activities. We refer to a language for describing system structure as a "module interconnection language" (MIL); it is one necessity for supporting programming-in-the-large.

An MIL should provide a means for the programmer(s) of a large system to express their intent regarding the overall program structure in a concise, precise, and checkable form. Where an MIL is not available, module interconnectivity information is usually buried partly in the modules, partly in an often amorphous collection of linkage-editor instructions, and partly in the informal documentation of the project. Aside from the issue that each of these three areas is ill-suited to express interconnectivity, the smearing of the relevant information over disjoint media is highly unreliable. Even more unsatisfactory are the facilities for specifying and enforcing module disconnectivity via informating hiding, limiting access to resources, establishing protection layers, closing off subsystems, etc. The lack of such facilities invites undisciplined or even unsocial programming, as shown by one of Weinberg's case studies (Weinberg 1971, pp. 71-75), since there is no automated means of enforcing the surface consensus of the programming team.

That current languages fail to support the global task of composing large systems was well argued by Wulf and Shaw in their paper entitled "Global variables considered harmful" (Wulf 1973). A responding paper (George 1973) proposed a scheme of declarations to augment block structure as a solution to the problems associated with global variables. The scheme provided mechanisms to protect variables from violations of various sorts by contained blocks, and to allow limited access to certain variables by selected internal blocks. Similar approaches have been suggested by others (Clark 1971, White 1972, and Ichbiah 1974). We believe that some of the mechanisms proposed were appropriate, but that they were inappropriately placed in the LPS.

That is, we distinguish between block structure and module interconnectivity. Block structure works well on a small scale, but humans simply cannot keep track of nesting levels after a few pages. Furthermore, and perhaps most important, module interconnectivity must in many cases take the shape of a graph or partial ordering. The more limited tree structure of nested blocks forces us to place some low-level modules at high-level places, extending their scope of definition to inappropriate places. It follows, then, that we need a separate language, or at least separate language constructs, for describing module interconnectivity, rather than complicating existing constructs that are well suited for programming-in-the-small.

Similarly, we reject Liskov's strong association of module interconnectivity with data abstraction (Liskov 1974). We believe that both are necessary, but that they should not be tied together inextricably. The programmer is the best judge of when they ought to be used in concert; he should be able to state his intents and to rely on a compiling system to see that they are carried out correctly. We regard data abstraction as a technique that should be supported by an LPS, rather than by a global mechanism such as an MIL.

2. Objectives of any module interconnection language (MIL)

In general, an MIL should serve as (1) a project management tool, (2) a means of communication between members of a programming team, (3) a design tool for, and actual means of establishing overall program structure, and (4) a means of documenting that structure in a clear, concise, formal and checkable way.

Top-down. As a communications device it should be especially effective from the top down. That is, it should facilitate the structuring of a system by a project manager (Weinberg 1971) or chief programmer (Mills 1970) or "modularizer" (Maynard 1972), and it should facilitate the communication of that structure to the relevant programmers.

One level at a time. Furthermore, it must encourage the structuring on one level at a time, since we humans do not usually deal effectively with several levels simultaneously. Given a job a programmer should be able to switch to the role of chief programmer for his subtask, assigning subproblems to his assistant programmers via an MIL program, just as he was given his job assignment by his superior. Such separate assignment of task and subtasks, via separate MIL programs, is perhaps most important when the chief programmer and his assistants are, in fact, one and the same person. The most likely person to attempt to violate system structure, whether intentionally or accidentally, is the one who is familiar with more than one level.

Bottom-up. Of course, an MIL should also support bottom-up programming. After all, we frequently gain understanding of a problem by first working on some of its details. Also, we often can make use of subsystems that have already been written to solve parts of other problems. In such cases we must be able to compose systems from existing subsystems.

Horizontal. Similarly, programmers at a single level may need to communicate. Each may need a resource supplied by the other. Or they may be creating co-subsystems, in the sense of coroutines. Or they may be writing mutually recursive subsystems. Thus, module linkage in the horizontal, as well as the vertical direction is needed.

Composition. Finally, when all the partial MIL programs for a system are put together, they should constitute a complete definition of the overall system structure. The description should be readable by humans to aid in understanding the system. Moreover, the compiling system should print graphical representations of the system structure, preferably in several layers of detail, for ease of human consumption.

Linkage. One may regard an MIL as being a higher-level language for specifying how a "linker" is to prepare for "loading" a program comprised of separately compiled segments (Presser 1972). Roughly, the linker must resolve static references to "external" names; i.e. names defined externally to each separately compiled module. A distinction, however, is that we do not expect the linkage to happen after compilation but rather as part of it.

Proving-in-the-small. Thus, an MIL might help to alleviate the disadvantages of independent compilation of modules, as addressed by Hoare (Hoare 1973). Modules can be small without sacrificing a proper description of the problem structure, since the latter is adequately and explicitly expressed in the MIL program. Working with small modules, we may find it less prohibitive to prove their correctness.

Proving-in-the-large. After establishing the correctness of the modules, we may be able to prove separately, on the MIL level, that correct modules work together correctly. Due to formalizing the description of system structure via a language, we may build a compiler that can check that a system of modules does indeed conform to the intended structure. That structure might be designed, for example: (1) to hide certain information (Parnas 1971) or (2) to build layers of virtual machines (Dijkstra 1972). Any MIL should inherently support these two concepts.

Independent of formal and mechanized proofs of correctness, the protection and documentation provided by the MIL program enhance the likelihood of correct construction in the first place and correct modifications later on.

Trade-off. One cost of this approach will be a more complex compiling system than has been necessary heretofore. However, the compiler can be more helpful to us by providing more feedback at early stages of system development. The cost should be more than offset by the increased speed and accuracy with which we will be able to construct and maintain large systems.

In summary, then, an MIL should:
(1) encourage the structuring of a system before starting to program the details;
(2) encourage the programming of modules assuming a correct environment, but without knowledge of the irrelevant details of that environment;
(3) encourage system hierarchy, while allowing flexible, if disciplined connections between modules;
(4) encourage information hiding and the construction of virtual machines, i.e. subsystems whose

internal structure is hidden, but which provide desired resources; and
(5) encourage descriptions of module interconnectivity that are separate from the descriptions of the modules themselves.

3. An example

To demonstrate the use of an MIL we graphically display a sample program structure in Figure 1. The particular system illustrated is a theorem proving program written in Algol/W (Wirth 1966) by Professor Sharon Sickel at the University of California, Santa Cruz. The program was written without the aid of an MIL; we tediously reviewed it after it was complete and factored out the overall structure. The exercise took several hours because the structure was not completely clear in its author's mind, nor was it apparent from the program listing, and because the exercise suggested some ways to improve the structure; in fact, what is presented here is the improved version.
The missing part of Figure 1 is developed in Section 5, culminating in Figure 5. In that section, we also define the concepts that are only sketched next.

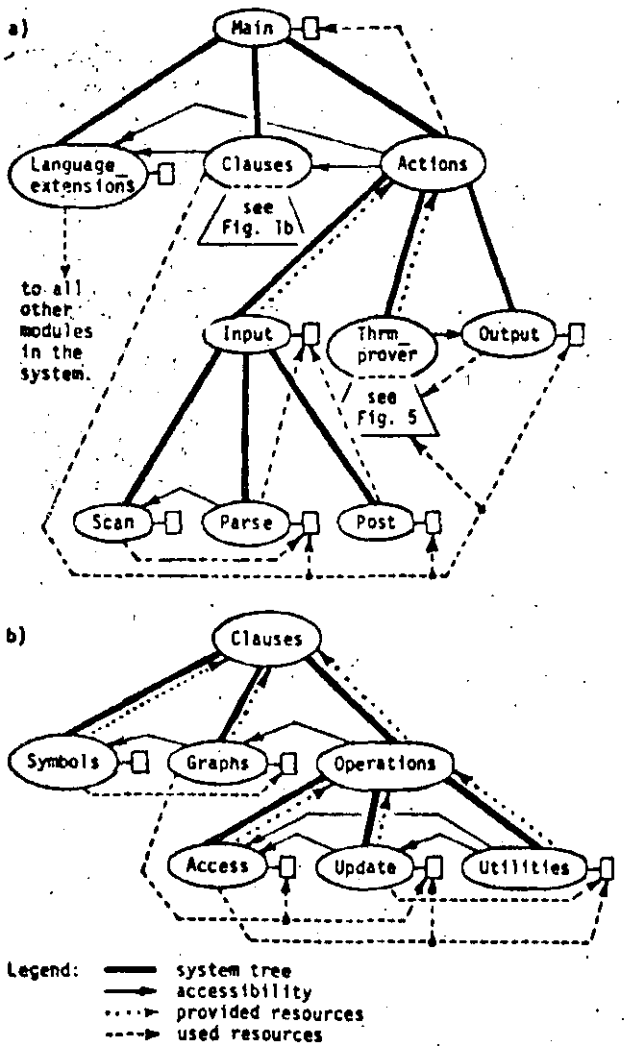


Figure 1. The system structure for a theorem proving program.

Graphical representation. Any system structure is represented as follows:
(1) Nodes and bold edges constitute the "system tree"; each edge connects a "parent" (system) to one of its "children" (subsystems).
(2) Dotted lines going upward along the tree edges from a child to its parent indicate that resources "provided" by the child are also "provided" by the parent. In other words, some of the resources provided by the child are passed via the parent to siblings and/or ancestors of the parent.
(3) Solid arrows, always between siblings, denote "sibling accessibility links". These are established by the parent, viz. chief programmer, so that the siblings may use each other's resources as necessary.
(4) Rectangular boxes attached to the nodes indicate LPS modules. Every leaf has a module. In our sample system, each module attached to a non-leaf serves the purpose of being the "driver" of its (sub-) system.
(5) Dashed arrows indicate that some resources provided by the node at the tail of the arrow are used by the module at the head of the arrow. The global aspects of the flow of resource names are addressed in detail in Section 5.

Refinements. Figure 1 represents the program at a high level of abstraction. As the diagram is further developed, specific resource names would start to appear and refinements would be made to accessibility. For examples, a parent might give a child accessibility to only a subset ("group") of the resources provided by siblings of the child, or the parent might hide some or all of its environment from the child.

Programming versus structuring. It is to be emphasized that "top-down programming" of a system would not necessarily proceed top-down relative to such a system diagram. Indeed, we are likely to start by programming the essential algorithms first, e.g. those inside Thrm_prover here, and deducing from their descriptions what would be the best form for the data to take and what innate operations should be supplied with the data, e.g. in the Clauses subsystem here. Adding the names of such data and operations to our diagram and appropriately refining related links would be working in a distinctly bottom-up fashion relative to the diagram. Nonetheless, we regard it as obvious that this overall approach is a top-down one relative to the problem being solved.

Misuse of structure. We now regard the modified Algol/W program as being reasonably well structured and the diagram of Figure 1 as a good display of its overall structure. One exception is that we believe the Clauses subsystem to be over-structured, in the sense that the Algol/W program structure is being used to try to achieve data abstraction. Unfortunately, this technique is both incomplete and uncheckable.
It is incomplete in that all of the irrelevant details of implementation of "clauses" cannot be hidden in Algol/W; one must rely upon programmer self-discipline as regards the correct use of data manipulation procedures provided with the virtual data type "clause". The technique is uncheckable by the compiler because it does not know that this

part of the program structure is being used for data abstraction, which in turn is due to Algol/W having no data abstraction facility to hide the details of implementation of new data types.

An LPS with general data abstraction facilities would obviate the need for such misuse of program structure. Likewise, an MIL would eliminate the necessity of using block structure to describe overall system structure.

4. The universe of discourse of MIL 75

We now present a particular language, MIL 75, for describing interconnections among modules. Being quite new, it has recieved little use thus far, so we expect it to evolve further toward an optimally useful language for describing system structure. Of course, the language is intended to satisfy the objectives stated in Section 2 above. It would, however, be a poor tool for enhancing reliability if it innately included "harmful" or unreliable features. Avoiding the latter must, of course, be a design objective of any language.

Names. The universe of discourse of MIL 75 consists of names: the names of resources originating in the separate modules, the names of the modules themselves, and the names of systems of modules.

External scope. An MIL 75 program addresses the question of who knows whom within a collection of modules. It defines the scopes of definitions of names across module and subsystem boundaries. It has nothing to say about the scopes of definitions within modules, these being defined by block structure and/or other constructs in the LPS.

Static, not dynamic. We emphasize that the interconnections addressed in MIL 75 are static ones, just as the names used to establish the connections are statically known, rather than being computed at run-time. In effect, the MIL program structures a global region through which the modules communicate. Thus, for example, MIL 75 could be used as a high-level language for programming the "global" declarations that are included in the compilation of each BCPL module (Richards 1969). However, MIL 75 admits of rather less restricted implementations, as may be deduced from Section 5.

No loading. Furthermore, it is to be stressed that MIL 75 does not address the problem of loading. That is, it has nothing to say about when modules or subsystems are to be loaded, nor does it say what, if any, overlay scheme is to be used. Perhaps we need, in addition to an MIL, a "subsystem loading language" to address exactly those issues. Presumably, the MIL program provides many of the right points of reference for describing loading and overlay strategies.

No functional specification. An MIL 75 program does not specify the nature of resources; it only specifies what those resources are to be named. Of course, the functional specification of modules and subsystems is important, but that is a separate issue not dealt with in this paper. A good solution may be to coalesce an MIL and a "function specification language". Perhaps the latter would be a language of axioms (Parnas 1972, Hoare 1972, Guttag 1974).

No types. Similarly, MIL 75 does not provide any ways of specifying the type of an object or defining language extensions. Rather it is used to specify paths for transmitting relevant information from one module to another. Such paths may be defined for any named entity that has its defined and applied occurences distributed over different LPS modules.

It is assumed, however, that the total LPS + MIL compiling system will do as much bookkeeping as necessary to do all static checking as soon as the necessary information is available. Clearly, this will require a non-trivial file system so that the compiler may keep summaries of each module and its external connections for the compiler's own use in subsequent compilations and recompilations. A modification of Liskov's "description units" (Liskov 1974) seem to be appropriate for such bookkeeping.

Accessibility. Finally, an MIL 75 program gives a particular module either unrestricted access to an object or none at all. We assume that any restrictions such as "read only", "write only", "read before write", "execute only", and other more general monitorings, are appropriate to the domain of, and programmable in, the LPS.

5. The semantics of MIL 75

The module interconnection language MIL 75 can be defined via attribute grammars (Knuth 1968); essentially, an MIL 75 program specifies a tree whose nodes are augmented by attributes. The latter are sets of resource names. In this section, however, we define the language by starting with a simple algebraic structure (a tree) and refining it stepwise. Simultaneously, the language concepts are presented and motivated by the stepwise development of the subsystem, Thrm_prover, which is to become part of the system in Figure 1.

5.1 System hierarchy

We concentrate on the overall system structure first. Since MIL 75 is intended to encourage structured programming, it imposes a tree structure on the system under construction. This "system tree" expresses nothing but the hierarchical relation between systems and subsystems. For now, we do not contemplate modules or resources at all. This will happen later, possibly forcing us to refine the system tree during the development of an actual system. Figure 2 shows the tree of the system Thrm_prover.

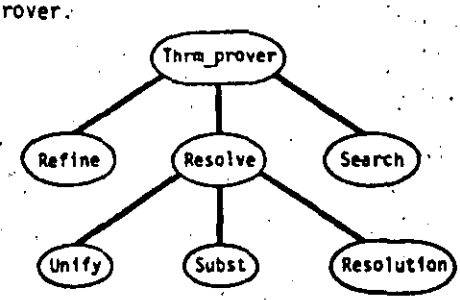


Figure 2. A sample system tree.

Our guideline for the rough decomposition of the project into a tree is that each node should finally encompass an intellectually manageable part of the whole problem, assuming that adequate support is provided by other nodes. Conceptually, there is

for each system tree node a designer who is responsible for the programming associated with "his" node and who supervises the designers of its children.

Definition: A "system tree" is a tuple $T = (N, S, Pa, Sn, t)$ where
(1) N is a finite set of "nodes";
(2) t is a distinguished member of N called the "root";
(3) Pa: $(N-\{t\}) \rightarrow N$ is a total function, called the "parent function", such that for any node n_1 in N there is a sequence of nodes $n_1, n_2, \ldots, n_k$ ($k \geq 1$) with $n_k = t$ and $Pa(n_i) = n_{i+1}$ ($1 \leq i < k$).
The terms "child", "sibling", etc. are defined in the obvious manner.
Finally,
(4) S is a finite set of "system names"; and
(5) Sn: $N \rightarrow S$ is a bijection; i.e. each node in N is associated with a unique system name.

When pre-existing subsystems are used in a new system, or when a very large system is constructed, the uniqueness of system names may prove difficult to achieve. Therefore, the syntax of MIL 75 allows "qualified names" (e.g. Resolve.Unify) for unambiguity and "aliases" for renaming.

5.2 Provided and derived resources

The next decisions to be made during the top-down development presumably concern the function of each subsystem. As the function of a subsystem can be completely described in terms of the resources it uses and provides, we now consider the association of resources with system tree nodes.

Ultimately, resources will originate in the LPS modules. Pursuing a top-down approach, however, the designer of any system tree node p states the set of resources provided by p. Then the question is where these resources come from. Some might originate in a module later to be attached to the node p, and thus are the direct responsibility of the designer of p. All other resources must come from any children of p. Therefore, the designer of p states the set of resources each child q must provide. This statement specifies the desired function of q, provided that all resources are adequately specified.

As seen from the node p, the resources it demands from its children are called "derived resources". The node p may derive resources from a child q and provide them to its own parent, in turn. In diagrams, such a case is indicated by a dotted arrow from q to p (cf. Figure 3 below).

Definition: A "resource-augmented system tree" is a tuple $T_R = (T, R, Pr, Mp)$ where
(1) $T = (N, S, Pa, Sn, t)$ is a system tree;
(2) R is a finite set of resources;
(3) Pr: $N \rightarrow 2^R$ is a total function (2^R denotes the powerset of R);
we say that "n provides r" iff $r \in Pr(n)$;
(4) Mp: $N \rightarrow 2^R$ is a total function;
we say that "n must provide r" iff $r \in Mp(n)$;
(5) $Mp(n) \subseteq Pr(n)$ for all n in N; and
(6) $Mp(p) \cap Mp(n) = \emptyset$ for all pairs of siblings p, n.

Naturally, it is a task of an MIL compiler to check that condition (5) is satisfied, i.e. that the bottom-up flow of derived resources is consistent. We allow set inclusion in (5) for facilitating a bottom-up approach, where pre-existing subsystems are used in a new parent system.

5.3 Accessibility

The next refinement is concerned with the interaction between siblings. The power and the responsibility to establish channels for transmitting names of resources between siblings rests solely with their parent. Here we follow Parnas' policy of a "designer controlled information distribution" (Parnas 1971). Consider Figure 3, where the "sibling accessibility links" are drawn as solid arrows between siblings.

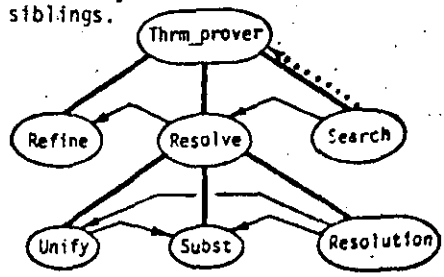


Figure 3. Sibling accessibility links.

These links do not represent individual connections between modules and resources. Rather, they allow the sibling at the tail of the arrow to access any resource provided by the sibling at the arrow head. In Figure 3, for example, Search has access to (any resource provided by) Resolve. For reliability reasons, access rights are non-transitive; for instance, Search has no access to Refine. Also, the children of Resolve are invisible to Search. Thus, Search can access a resource provided by Unify if and only if this resource is also provided by Resolve. In short, the substructure of one sibling is not apparent to another.

The accessibility links between a set of siblings may form any directed graph. Thus, the parent may allow mutual recursion between resources (e.g. procedures, coroutines, or data structures) of its children.

Inherited access. Typically, the access rights granted to a node are also useful for most of its children. In MIL 75, a child inherits by default all access rights that have been granted to its parent. In Figure 3, Resolution inherits access to its "uncle" Refine; in Figure 1a, Parse inherits access to its "granduncle" Clauses.

Alternatively, any parent may "will" a child nothing or an explicitly specified subset of its own access rights, thus making the child less "privileged" and formally asserting that the child and all its descendants cannot exploit or disturb certain resources. If a child is to be partially disinherited, the parent must list all access rights left to the child. Thus, if the parent later obtains additional access rights to vulnerable resources, they do not inadvertently shine through to the less privileged child.

Derived access. Naturally, a parent has access to the resources that it demands from any of its children. However, all descendants of its children are invisible to the parent. Thus, we can build layers of virtual machines as in Figure 4, where

the most privileged nodes are at the bottom.

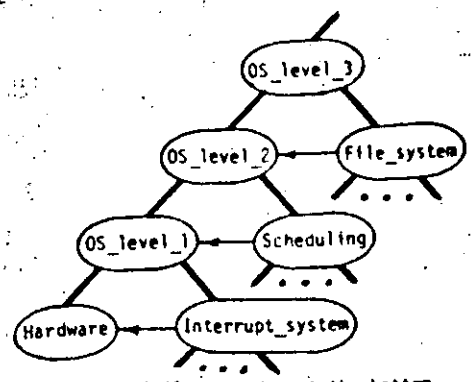


Figure 4. Privileged nodes at the bottom.

Definition: An "access-augmented system tree" is a tuple T_A = (T, Sac, Iac) where

(1) T = (N, S, Pa, Sn, t) is a system tree;
(2) Sac and Iac are relations on N-{t};
(3) p Sac n (pronounced "p has sibling-access to n") implies that p and n are siblings;
(4) p Iac n (pronounced "p inherits access to n") implies that either Pa(p) Sac n or Pa(p) Iac n.

Definition A node p "has access to" a node n iff either p Sac n, p Iac n, or p = Pa(n).

5.4 Module placement

We proceed to place modules into the system tree. With each node, we may associate at most one LPS module, as indicated by the following.

(1) With each leaf n, we must associate a module, the "leaf module" at n. A leaf without module is not allowed, since it cannot provide any resources.
(2) A module associated with a non-leaf n may act as a "driver" or "monitor" of the system named Sn(n). Such a "root module" at n must define all resources in Pr(n) that are not derived from the children of n.
(3) A non-leaf without root module serves as a structural entity only, making an integrated whole out of its subsystems and establishing a single interface to the outside.

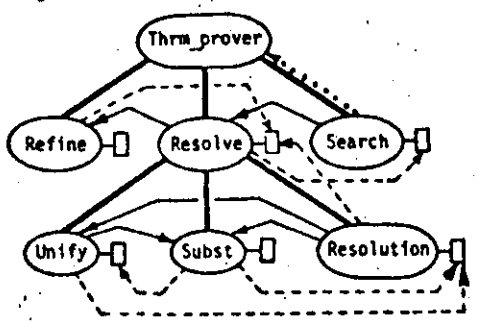


Figure 5. Module placement and usage links.

Different modules might be programmed in different LPSs. The site of the modules is primarily determined by their intellectual manageability, vis-a-vis programming-in-the-small, and secondarily by the quality of the MIL program that binds them together. For if the modules are too small, the MIL program is inconcise and introduces overhead; if the modules are too large, the MIL program does not give enough information about the system structure.

Origin and usage of resources. For each module m at a node n, there must be two statements in the MIL 75 program: (1) the "statement of origin" listing the resources defined in m, and (2) the "statement of usage", listing the resources that are used, but not defined, in m. For clarity, the latter statement is divided into a list of the "derived resources" provided by children of n, and a list of all others, i.e. those obtained through sibling or inherited access.

The compiling system must check that (1) the actual usage of resources by module m conforms to the access rights granted to node n, and that (2) any resource provided by n either comes from a child or originates in module m. No node may provide a resource that is obtained through sibling access or inherited access; such a flow of resources would probably have deleterious effects on reliability.

Usage links. The compiling system can now derive and graphically display the "usage links", drawn as dashed arrows in Figures 1 and 5. If a node n has access to a node p, and the module m at n uses a resource provided by p, then a usage link points from the node p to the module m. Figure 6 shows the three possible cases. Recall that the resource(s) provided by p might not originate in the module at p, if any. However, this is irrelevant to, and hidden from, the module m.

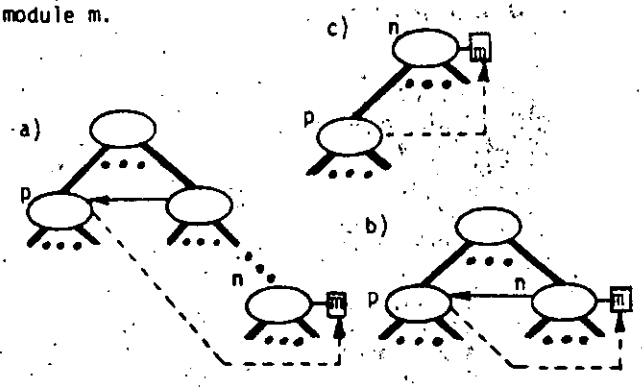


Figure 6. Usage links.

Figure 6 also suggests a graphical check on consistency: we can always form a cycle by traversing the usage link, zero or more tree edges upward, and finally one sibling accessibility link (parts a and b), or by traversing the usage link and one tree edge downward (part c).

Definition: A "module interconnection structure" is a tuple

T_M = (T, T_R, T_A, M, Mod, Or, Ud, Und) where

(1) T = (N, S, Pa, Sn, t) is a system tree, T_R = (T, R, Pr, Mp) is a resource-augmented system tree, and T_A = (T, Sac, Iac) is an acess-augmented system tree;
(2) M is a finite set of modules;
(3) Mod: N → M is a partial, injective function, such that Mod(q) is defined for every leaf q; we say that Mod(n) is the "module at n";
(4) Or, Ud, and Und are total functions N → 2^R, such that Or(n) = Ud(n) = Und(n) = ∅ if Mod(n) is undefined; we say that r ∈ Or(n) is

"resource originating in Mod(n)", $r \in Ud(n)$ is a "derived resource used in Mod(n)", and $r \in Und(n)$ is a "non-derived resource used in Mod(n)";
(5) for every p,n in N: $Or(p) \cap Or(n) = \emptyset$;
(6) we define for all $p \in N$ the set of "derived resources"
$D(p) = \{r \in R \mid \exists q \in N: Pa(q) = p \text{ and } r \in Mp(q)\}$;
then, for all $p \in N$,
(7) $Pr(p) \subseteq Or(p) \cup D(p)$;
(8) $Ud(p) \subseteq D(p)$; and
(9) $r \in Und(p)$ implies $(\exists n)[(p\ Sac\ n \text{ or } p\ Iac\ n) \text{ and } r \in Mp(n)]$.

5.5 Programming in MIL 75

A complete MIL 75 program consists of a sequence of one-level "system descriptions". Each is assumed to be (re-) compilable alone, or with any others. Put together, they can be translated into a module interconnection structure T_M.

A system description for a node p consists of statements specifying
(1) Sn(p), the designer's name, and a relevant date;
(2) Pr(p);
(3) Mod(p), Or(p), Ud(p), and Und(p);
and for each child q of p:
(4) Sn(q), Mp(q), and $(n \in N \mid q\ Sac\ n)$; and
(5) the set $W = (n \in N \mid q\ Iac\ n)$ either by enumeration (such that $W \subseteq I$ is satisfied) or by default (then $W = I$ is assumed), where
$I = (n \in N \mid p\ Iac\ n \text{ or } p\ Sac\ n)$.

Phrases specifying empty sets in (3) and (4) above may be omitted. Also, the name of Mod(p) may be left out if it is identical with the system name Sn(p). There follows a sample system description.

```
system Input
   author      'Sharon Sickel'
   date        'July, 1974'
   provides    Input_parser
   consists of
      root module
         originates  Input_parser
         uses derived     Parser, Post_processor
         uses nonderived  Language_extensions
      subsystem Scan
         must provide Scanner
      subsystem Parse
         must provide Parser
         has access to Scan
      subsystem Post
         must provide Post_processor
```

Groups. To increase the compactness of MIL 75 programs and to encourage an even more refined granting of access rights, MIL 75 also contains the concept of grouping (cf. George 1973). The designer of a parent can define named subsets or "groups" of the set of derived resources provided by its children. Then, the parent can grant its children access rights to these groups. A group acts as a virtual child of the parent, but has neither module, children, nor access rights. The compiler must check that no child q has access to a group that provides a resource also provided by q, lest the descendants of q inherit access to that resource.

6. Conclusions: The gain in reliability

A module interconnection language can be a significant asset. It embodies a design methodology for reliable software. It provides much needed forms of abstraction, expression, and verification, vis-a-vis programming-in-the-large. It also enhances the effects on reliability that are gained by other methods: management styles; top-down, modular, and structured programming; data abstraction; and information hiding. Reliability is enhanced during system design, the actual programming, system testing, and maintenance and modification.

System Design. Modularization, as a forethought rather than an afterthought, helps in finding reliable solutions to complex problems by applying the traditional method of "divide and conquer". The advantages of modularization, however, are often apparently offset by added complexity in the connections among modules (Liskov 1972).. This is not surprising; our problems are not solved by design methodologies that concentrate only on issues of programming-in-the-small. We will obtain more reliable software only if programming-in-the-large is recognized as a separate activity that relies, as heavily as does programming-in-the-small, on a language providing abstraction, structure, and style.

Programming. The results of the system design phase must be communicated to and among the members of the programming team or project. Such communication concerns, among other things, the modules that constitute the system, the position of each module in the hierarchy, the resources each module must provide, and the access rights with which each module is endowed. It is unreasonable to expect that this information can be reliably transmitted via anything but a formal language, or enforced by anything less than a rigorous compiling system.

Testing. A substantial amount of testing can be done by independently exercising each module. However, there is a wide gap between testing individual modules and testing the system as a whole. Performing only these two kinds of tests involves too big a jump in levels of abstraction and results in a gap in confidence. The hierarchical subsystem concept of an MIL suggests a more gradual bottom-up development of the testing phase. Also, the MIL program states clearly, for each subsystem, the connections to other subsystems and who is responsible for testing. In short, the MIL supports "structured testing".

Maintenance and modification. Each connection and dependency between modules is durably documented in the MIL program. No link between modules can be left out of the documentation and be forgotten--the compiling system will complain. Thus, system modifications are more likely to be successful, since their implications are more likely to be foreseen when using an MIL. Furthermore, the hierarchical structure imposed by the MIL makes it easy to replace modules and/or subsystems; and the compiler can support system modification by providing cross-references and graphs of system structure, accessibility links, and usage links. Finally, the modified system structure is automatically checked for consistency.

Outlook. It is obvious that we need languages for programming-in-the-large. An MIL is but a first approximation to such a language, since it does not include facilities for the specification of the function of modules. An MIL may even be regarded as only a "language feature" in the sense of Hoare (Hoare 1973). It seems, however, powerful enough to increase software reliability, even in the

absence of other needed extensions to current languages.

Acknowledgments

We are grateful to Frank Frazier, Nico Habermann, Jim Horning, Jean Ichbiah, Bernard Lorho, Bill McKeeman, Doug Michels, Dan Ross, Sharon Sickel, and Bill Wulf for many helpful comments and stimulating discussions.

References

Clark, B.L, and Horning, J.J. "The system language for project SUE." SIGPLAN Notices 6, 9 (October 1971).

Dijkstra, E.W. "Notes on structured programming." In: Dahl, O.J., Dijkstra, E.W., and Hoare, C.A.W. "Structured Programming." Academic Press, London, New York, 1972.

George, J.E., and Sager, G.R. "Variables--Bindings and protection." SIGPLAN Notices 8, 12 (December 1973).

Guttag, J. "The use of type for the definition of abstract data objects." Dept. of Computer Science, Univ. of Toronto, Ontario, Canada (March 1974).

Hoare, C.A.R. "Proof of correctness of data representations." Acta Informatica 1, 271-281 (1972).

Hoare, C.A.R. "Hints on programming language design." Memo AIM-224, Computer Science Dept., Stanford University (December 1973). Also in: Proc. Symposium on Principles of Programming Languages, Boston (October 1973).

Ichbiah, J.D. "Visibility and separate compilations." Proc. of IFIP WG 2.4, La Grande Motte, France (May 1974).

Knuth, D.E. "Semantics of context-free languages." Mathematical Systems Theory 2, 2 (1968).

Liskov, B.H. "A design methodology for reliable software systems." Proc. Fall Joint Computer Conference, 191-199 (1972).

Liskov, B.H., and Zilles, S. "Programming with abstract data types." Proc. Symposium on Very High Level Languages, SIGPLAN Notices 9, 4 (April 1974).

Maynard, J. "Modular Programming". Petrocelli Books, New York, 1972.

Mills, H.D. "Chief programmer teams: Techniques and procedures." IBM Internal Report (January 1970).

Parnas, D.L. "Information distribution aspects of design methodology." Technical Report, Dept. Computer Science, Carnegie-Mellon Univ. (February 1971).

Parnas, D.L. "A technique for software module specifications with examples." CACM 15, 5 (May 1972).

Presser, L., and White, J.R. "Linkers and loaders." ACM Computing Surveys 4, 3 (September 1972).

Richards, M. "The BCPL reference manual." Memo 69/1, The University Mathematical Laboratory, Cambridge, England (January 1969).

Weinberg, G.M. "The psychology of computer programming." Van Nostrand Reinhold Co., New York, 1971.

White, J.R., and Presser, L. "A tool for enforcing system structure." Report CS-11, Dept. of E.E., Univ. of California, Santa Barbara (April 19[illegible]

Wirth, N., and Hoare, C.A.R. "A contribution t development of ALGOL." CACM 9, 6 (June 1966).

Wulf, W., and Shaw, M. "Global variable considered harmful." SIGPLAN Notices 8, 2 (February 1973).

DISEÑO DE BASES DE DATOS

CONCEPTS FOR MODELLING INFORMATION

MAYO, 1985

(DIAM, of course, why hierarchies and networks are sometimes said to be "more natural".) When one goes to libraries, he must learn to specify the semantic information more explicitly. After some practice this process becomes easy, and is no longer a problem. In doing this the user gets the advantage of being more explicit about the meaning of his information structure and, if FORAL is any indication, he gets the advantage of using a language that looks more like "natural" English.

Other points of view on this topic are discussed in the papers of Hall, Falkenberg, and Bracchi at this conference.

Modelling in Data Base Management Systems, G.M. Nijssen, (ed.)
North Holland Publishing Company, 1976

CONCEPTS FOR MODELLING INFORMATION*

Eckhard Falkenberg
Institut für Informatik
University of Stuttgart
Stuttgart, F. R. of Germany

This paper is concerned with the problems of structuring and modelling information. The aim is to introduce concepts which are adequate for the design of conceptual schemas and which fulfill corresponding criteria. A method for modelling information is proposed which is based on only two basic concepts, object and role. The application of this system of concepts is illustrated. A comparison with some other approaches and a short evaluation are included in this paper.

1 THE AIM

In the present data base management research, there is a trend towards coexistence of different data models and languages within one system. To reach this coexistence, as a gross architecture the conceptual-schema approach is proposed (ANSI-SPARC [2], NIJSSEN [14]). Nevertheless, it is an open problem up to now what might be the most adequate concepts just for the conceptual schema. Adequacy can mean a lot of issues and criteria. Some of them are mentioned in the following.

We should be able to define the information contents of the application problem as precise as possible. For those precise definitions we should be able to choose the simplest, most elegant and straightforward solutions.

For the formal foundation of the concepts for modelling information we should choose as few as possible basic concepts.

We should take into account that the types in a data base evolve in course of time. This should not result in a permanent necessity of reprogramming the manipulation functions and application programs. This criterion we call evolvability.

Another important criterion we should consider is transformability. This means that the concepts on the conceptual-schema level should be chosen in a way, that the mappings between the conceptual schema and the various other schemas - user schemas and internal schema - become as simple as possible.

---

* The presentation, given by the author at the conference in Freudenstadt, is published elsewhere [11]. The present paper reflects to some extend several discussion points of this conference and has therefore the character of a postscript.

There are other criteria, like naturalness, comprehensibility, teachability and practicability. They are also important criteria, but difficult to grasp and to proof.

It is the aim of this paper to give an introduction into concepts which try to approximate these criteria of adequacy.

2 INTRODUCTION TO CONCEPTS FOR MODELLING INFORMATION

From a problem-oriented point of view, we can say that a data base is the description of an abstract model of a piece of reality. This description is performed by a language. It is an important and basic premise of the approach presented in this paper, to distinguish very carefully between (a) the abstract model of the application-specific piece of reality and (b) the linguistic reference to that model. In the literature, various terms are used for such an abstract model, e.g. "object system" (SUNDGREN [18]), "things of the real world" (SENKO et al. [16]), "information sphere" (FALKENBERG [10]). In this paper, we will use the term universe of discourse. A universe of discourse consists of discrete elements. During a time period, a universe of discourse may change, new elements may be added, existing elements may be deleted.

To describe a universe of discourse means to signify (specify, identify) all its elements uniquely, by aid of specific linguistic expressions. Such an expression, signifying exactly one element of a universe of discourse, we call a unique signification (more specific, see FALKENBERG [11]).

Considering elements of a universe of discourse, we may distinguish between (a) some kinds of fundamental elements and (b) some kinds of semantic rules concerning these fundamental elements. This distinction is of practical importance. While fundamental elements are present in each application, the necessity to use the one or the other kind of semantic rules depends on the application problem.

In this paper, we are only concerned with the question, what kinds of fundamental elements there should be in a universe of discourse. First, we give an introduction to the basic concepts and to some further concepts. Then we discuss some signification problems, in so far as this is necessary in our context. Furthermore, we consider changes of a universe of discourse.

2.1 BASIC CONCEPTS

A universe of discourse contains conceptual, mental units which are distinguishable from one another, atomic and discrete in nature. Those elements we call objects. When we ask for the information content of an object, the only knowledge we have a priori is that it exists in our universe of discourse. Except that, an object has no information content in itself. We get information content, if we say something about the object, if we consider facts concerning this object, if we associate this object with other objects. For single, atomic facts, for the smallest information-bearing* elements of a universe of discourse, we use the term association.

* term in colloquial sense

Informally speaking, an association contains and connects objects. We may now ask the question, what is the nature of this containment and connection. Is an association a set of objects or a tuple of objects, or what? Before answering that question, let us look at the two linguistic expressions "W. A. Mozart was born in Salzburg" and "W. A. Mozart lived in Salzburg". If we assume the normal historical knowledge as universe of discourse, we see that each expression is a unique signification of an association between two objects. In both cases the objects are the same, "W. A. Mozart" and "Salzburg". Nevertheless, the associations are different, which is indicated by different verbs, "born" and "lived". Therefore, we cannot say that the only components of associations are objects. There has to be a further kind of components which we call roles.

We have now introduced the two basic concepts of our approach, namely the concepts object and role. Therefore, we call our approach the object-role model. Other concepts, in particular the concepts association and type, are definable by aid of these two basic concepts. The most fundamental aspects of the object-role model are described formally in the appendix. In the following, this model and some peculiarities of it are illustrated by examples.

2.2 ASSOCIATIONS

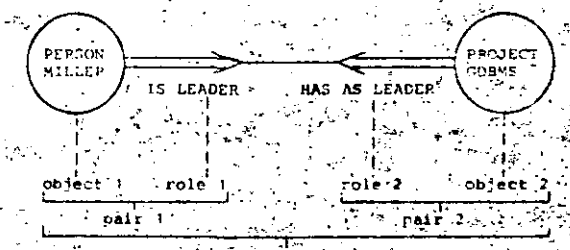
We have seen that the components of associations are objects and roles. Each object of an association plays a specific role within the association. It may be also the case that an association occurs as component of another association. In those cases we speak of nested associations.

Formally speaking, an association is defined as a set of object-role or association-role pairs. Within a universe of discourse, an object or an association may play several different roles, and also a role may be assigned to several objects or associations.

In the following, we give some examples of prominent cases. The unique significations we use to signify the associations are illustrated graphically.

In many application cases, associations are binary, that means they are sets of two object-role pairs.

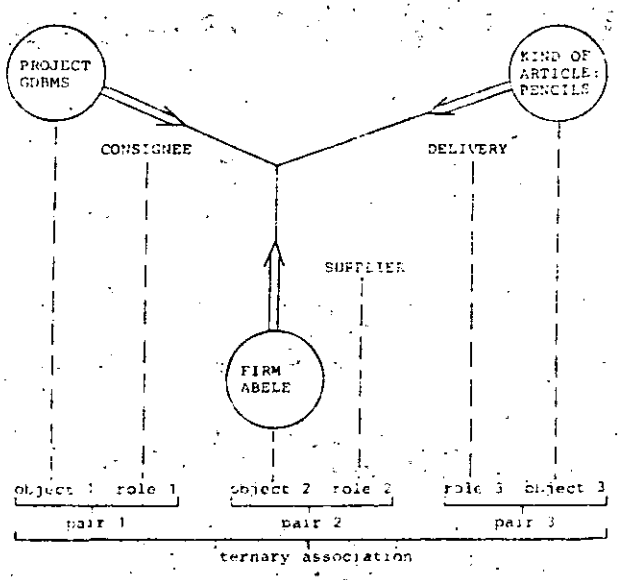
Example:



In this example two objects are associated, signified by "PERSON MILLER" and "PROJECT GDBMS". The person plays the role that he "IS LEADER" of a project, while the project plays the role that it "HAS AS LEADER" a person.

Not all application cases are as simple as the above example. In particular, sometimes more than two object-role pairs may be considered as one association, as one single fact.

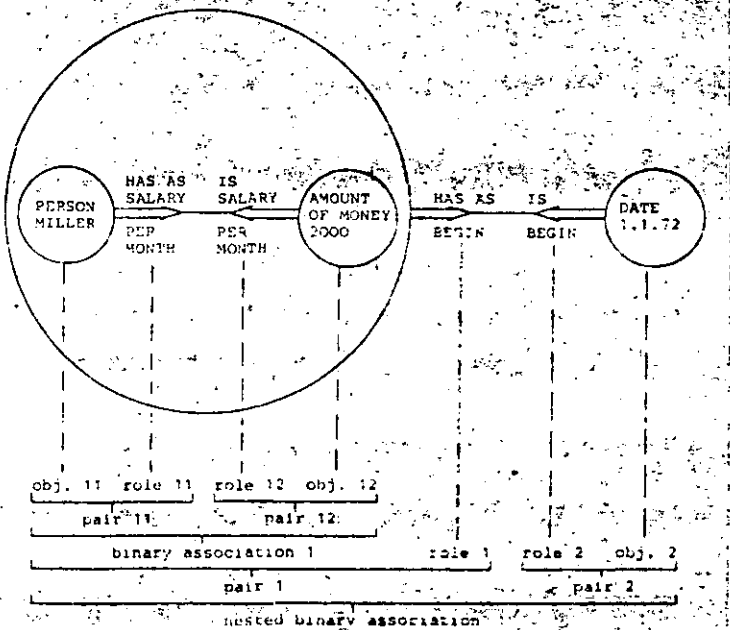
Example:



In this example we have three objects, signified by "PROJECT GDBMS", "FIRM ABELE" and "KIND OF ARTICLE: PENCILS". The role of the project is "CONSIGNEE", the role of the firm is "SUPPLIER", and the role of the kind of article is "DELIVERY". The three objects are, from a structural point of view, on the same level, because each object is associated to both other ones without preference.

There may be other cases, where such a preference exists. For example, we have the binary associations, that certain persons earn certain amounts of money per month. If we consider in addition the historical development of salaries, we have to add the dates, when those facts (persons earning amounts of money) begin or end.

Example:



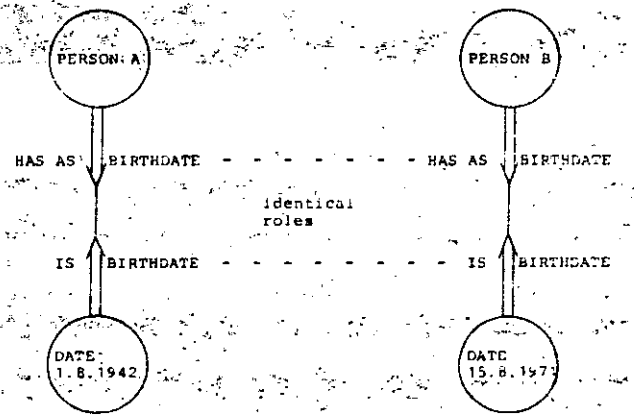
In terms of the object-role model, we can interpret this example as follows: There are two objects, signified by "PERSON MILLER" and "AMOUNT OF MONEY: 2000". The role of the person is that he "HAS AS SALARY PER MONTH" an amount of money, and the role of the amount of money is that it "IS SALARY PER MONTH" of a person. The whole salary association is associated to another object, signified by "DATE 1.1.72". The role of the salary association is that it "HAS AS BEGIN" a date, and the role of the date is that it "IS BEGIN" of a salary association.

2.1 TYPES

It is usual in communication and, in particular, also in data base technology to deal with types of elements. An actually at a certain instance of time in the universe of discourse existing set of elements, belonging to a specific type, is called a population of that type. A single element belonging to a specific type is called an instance of that type (DURCHHOLZ [9]).

In our approach we distinguish association types and object types. Association types may be defined by aid of the role concept in the following way: Several associations belong to the same type, if the sets of roles, occurring in each one of these associations, are identical.

Example:



These two associations belong to the same type, because in both associations the same roles occur, signified by "HAS AS BIRTHDATE" and "IS BIRTHDATE".

Also object types may be defined by aid of the role concept: Several objects belong to the same type, if these objects play at least one role in common. Following that definition in the above example, the two objects signified by "PERSON A" and "PERSON B" may belong to the same type, because they have a role in common, namely "HAS AS BIRTHDATE".

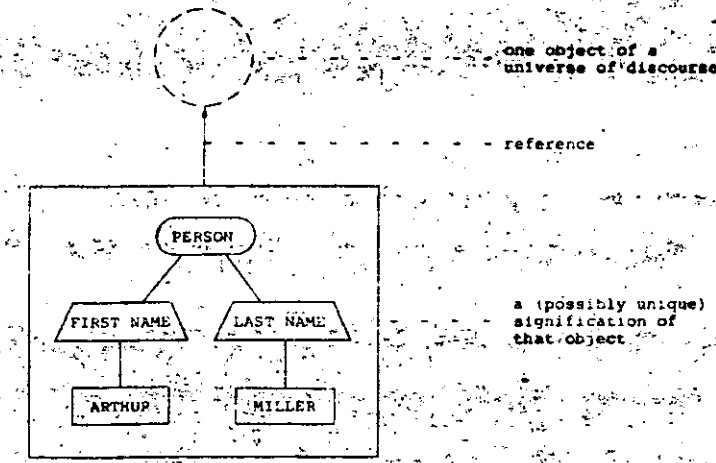
It is important to note that these definitions give in many cases a good approximation of that what may be intuitivly called types. Nevertheless, these definitions are not at all binding. Normally, in looking at things from several points of view, one element of a universe of discourse may be instance of several types. This means types may be overlapping.

2.4. SIGNIFICATION ASPECTS

In the above examples, we have chosen relatively simple significations to signify the objects and roles of the represented associations uniquely. For example, we have assumed that the string "PERSON MILLER" is a unique signification of an object. That assumption is only valid if the individual name "MILLER" is unique within the

actual population of the object type named "PERSON". If this condition is not true, we need another signification, probably a more complicated one. For example, we may use several types of names like "FIRST NAME" and "LAST NAME". The signification then probably becomes unique.

Example:



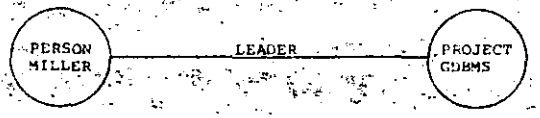
On the other hand, it is often possible to simplify the significations of roles. For example, if there are objects playing only one role within the universe of discourse, and the significations of these objects are chosen in an appropriate way, then this role may be obvious and needs not to be signified explicitly.

Example:

In this example the roles of a salary association are not signified explicitly, but implicitly by choosing the signification "SALARY 2000 PER MONTH" which is to some extend a mixture of object and role signification.

There may be other cases where it is sufficient to signify the various roles in an association by just one expression. Such an expression is to some extend similar to the linguistic "predicate".

Example:



2.5 CHANGES OF THE UNIVERSE OF DISCOURSE

To change a universe of discourse means to add or to delete information-bearing elements to respectively from the universe of discourse. This means for our approach that only associations may be added or deleted. It is allowed that the associations to be added belong to new types. It may also happen that the whole population of an association type will be deleted if this type is of no more interest. Changes of the universe of discourse mean therefore, that not only instances may change, but also the scope of interest, the types.

If only associations may be added or deleted, we have to answer the question, how can we add or delete objects and what means that. To introduce a new object into the universe of discourse means to add new associations from which this object is a component. To delete an existing object from the universe of discourse means to delete all the associations from which this object is a component. We see that objects are added or deleted indirectly, via associations. This is a conclusion of the point that we consider an object in itself not as information-bearing.

We might argue that we may be interested just only in the existence of an object in the application-suecific environment. But in this case, we have introduced unconsciously and implicitly an information-bearing element, namely the fact that our object exists in the environment. For conceptual clearness we should make this fact explicitly, that means we should introduce an existence association between our object and the object "environment".

3 COMPARISON WITH OTHER APPROACHES

In this section some points of comparison with other approaches and some arguments about our approach are collected.

3.1 UNIVERSE OF DISCOURSE AND SIGNIFICATIONS

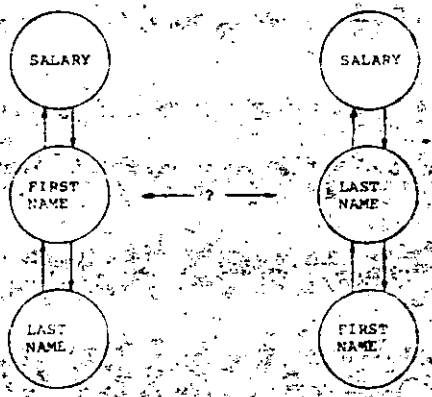
It is a matter of fact that in general one element of some universe of discourse may be signified in various different ways (FALKENBERG [11]). This is the main reason why we suggest to distinguish very carefully between (a) the problems arising in structuring a universe of discourse and (b) the problems arising in putting the linguistic

references to this universe of discourse. This distinction is an important premise for structuring and describing any relevant information content fairly precisely.

For example, let us consider a salary association between an amount of money and a person having a first name and a last name. Let us for a moment abstract from the signification problem, then we see that we have an association between two objects, namely between a certain amount of money and a certain person. The fact that we need a first name and a last name (or probably something else like a personnel number) to signify this person uniquely, belongs already to the signification problem. In this example we have the situation that the unique signification of the one object (the amount of money) is simple, while the unique signification of the other object (the person) is more complicated.

This attitude prevents us from having to answer the question whether the first name or the last name earns the money directly, as illustrated in the following.

Example:



For our feeling, the information content of this circle can be defined more precisely if one follows the approach presented in this paper (compare section 2.4; see on the other hand e.g. BRACCHI/FEDELI/PAOLINI [4] p. 214 and SENKO [17], see also the "scruples" of HALL/OWLETT/TODD [12]).

3.2 OBJECTS AND ROLES

We have seen that for the considerations about the universe of discourse two basic concepts are sufficient, namely object and role. In other approaches, often a larger number of basic concepts are used, like for example object, association, property and type (see e.g.

BENCI et al. [3]). It seems to us that for reasons of simplicity and comprehensibility as few as possible basic concepts should be introduced.

The concept of object is well known from many other approaches. The concept of role occurs in some less general form in the approaches of CODD [7] and SENKO et a... [17] (role names). It has also some similarity with ABRIAL's concept of access function [1].

3.3 PROPERTIES AND RELATIONSHIPS

A peculiarity of natural language is that one and the same fact can be signified in various different ways. Let us consider, for example, the two expressions "The house No. 11 is yellow" and "The house No. 11 possesses the colour yellow". Intuitivly, we could say that in the first expression "yellow" is a property of the "house No. 11", while in the second expression there is a relationship between the "house No. 11" and the "colour yellow". But if we abstract for a moment from the special linguistic paraphrase and ask what elements of our universe of discourse these expressions signify, we see that both expressions signify one and the same fact. The only point is that the one linguistic paraphrase may be used more often than the other. Thus, we may get the feeling that the one paraphrase is more "natural" than the other.

It seems to us, that those linguistic peculiarities motivate the distinction between properties and relationships in many approaches (see e.g. BENCI et al. [3], CHEN [5], HALL/OWLETT/TODD [12], MOULIN et al. [13], SCHMID/SWENSON [15], SUNDGREN [18]).

We have good reasons not to provide that distinction on the conceptual-schema level. Namely, this distinction is often of fuzzy nature, that means different users may come to different conclusions in classifying facts as properties or relationships. On the other hand, this classification is often considered as obvious which means that it can be dropped without loosing information.

There are even more serious problems with that distinction. Namely, there may be some evolution event so that a type of facts which has belonged, in the original universe of discourse, to the one kind belongs now, in the new universe of discourse, to the other kind. A property may become a relationship or vice versa. Data manipulation functions which deal with these types of facts and which refer to this distinction will no more work after the evolution event. This is the reason why the distinction between properties and relationships results in a lack of evolvability (FALKENBERG [10]).

These considerations refer to the conceptual-schema level. On the other hand, there may be user schemas where different linguistic paraphrases are possible and, in particular, where the users may distinguish between properties and relationships.

3.4 BINARY ASSOCIATIONS

It is often argued that one can manage all problems of structuring a universe of discourse by aid of binary functional association types (see e.g. the CODASYL-set concept [6]). This is true but disadvantageous in some cases. Namely, there may be an evolution event within the universe of discourse changing a functional association type into a relational one from the problem's point of view. If we want to save the functional approach in such a case, we have to introduce a new, additional object type (some people call it a dummy) which has now two new functional association types to the original two object types. The consequence of that is a lack of evolvability because those data-manipulation functions dealing with the original association types will no more work after the evolution event. Therefore, it is better to allow binary relational association types (ABRIAL [1], BRACCHI/PERDELI/PAOLINI [4], SENKO [17]).

But we should not restrict ourselves to that binary concept because this restriction has some disadvantages. Let us consider an evolution event, so that a new object type has to be introduced into an existing association type (and thus modifying it). It may be the case that the trial to split the modified, expanded association type into several binary association types results in a loss of information (see CODD [7] p. 385). To avoid this loss of information we have to introduce another, additional object type. Again, like in the case mentioned above, the consequence is a lack of evolvability. Therefore, we should allow n-ary relational association types.

3.5 N-ARY ASSOCIATIONS

As an association is considered as a single, atomic fact, we cannot take any number of objects into an association. Therefore, when we say "n-ary", we do not mean CODD's approach [7], where in general several single facts are combined within one n-tuple, even in third normal form.

A possible criterion to decide how many objects are involved in an association, so that we can say this is a single fact, is the criterion of semantical irreducibility. Informally speaking, a semantically irreducible n-ary association type cannot be split into several association types with a smaller number of object types, without having to introduce an additional object type or without loosing information (see e.g. HALL/OWLETT/TODD [12], FALKENBERG [10]).

There may be problems with that criterion when it is applied uncritically (DELOBEL [8]), but normally, it may result in a good approximation of what we intuitively call a single fact.

Another problem is to decide whether an association, containing more than two objects, may have an internal nested structure or not. Because we should be able to define information contents precisely and straight forward, and also for reasons of generality, we should allow nested associations.

4 SUMMARIZING EVALUATION

We have mentioned in section 1 some criteria of adequacy with regard to concepts for modelling information on the conceptual-schema level. Now we want to discuss the question, to what degree the concepts we introduced fulfill these criteria.

We have emphasized the importance of considering what are the smallest information-bearing elements of a universe of discourse, what are single facts. We allow n-ary associations as well as nested associations. This is an approach which enables us to solve the problem of structuring a universe of discourse in a precise and straightforeward manner. Another point we have emphasized is the clear distinction of the problems with regard to the universe of discourse from the signification problems. Also, this point can improve precision.

We have introduced a fairly simple system of concepts. There are only two basic concepts, namely object and role, by aid of which the others are definable.

To fulfillthe criterion of evolvability, we suggest (a) to avoid unnecessary distinctions like that between properties and relationships, and (b) to allow (semantically irreducible) n-ary association types.

Transformability has to do not only with the considerations about the universe of discourse, but also and, in particular, with the signification aspects. This point is demonstrated by the author elsewhere [11].

With regard to naturalness, comprehensibility, teachability and practicability we dispense with arguments. They may be left to the judgement and taste of the reader.

REFERENCES

[1] ABRIAL, J.R.: Data Semantics, in Klimbie/Koffeman (eds.): Data Base Management, North-Holland 1974

[2] ANSI-SPARC: Status Report 1975

[3] BENCI, E. et al.: Concepts for the Design of a Conceptual Schema, IFIP TC-2 Working Conference on "Modelling in Data Base Management Systems", Freudenstadt, January 1976

[4] BRACCHI, G.; FEDELI, A.; PAOLINI, P.: A Multilevel Relational Model for Data Base Management Systems, in Klimbie/Koffeman (eds.): Data Base Management, North-Holland 1974

[5] CHEN, P.P.: The Entity-Relationship Model - Toward a Unified View of Data, Proceedings of the Conference on "Very Large Data Bases", Framingham, Mass., September 1975

[6] CODASYL-DDLC: DDLC Journal of Development, 1973

[7] CODD, E.F.: A Relational Model of Data for Large Shared Data Banks, CACM, Vol. 13, No. 6, June 1970, p. 377-387

[8] DELOBEL, C.: Discussion point on the IFIP TC-2 Working Conference on "Modelling in Data Base Management Systems", Freudenstadt, January 1976

[9] DURCHHOLZ, R.: The Concept of Type, Bericht ADF 11, Gesellschaft für Mathematik und Datenverarbeitung, St. Augustin, June 1975

[10] FALKENBERG, E.: Structuring and Representation of Information at the Interface between Data Base User and Data Base Management System, doctoral thesis, University of Stuttgart, June 1975 (in German; in English forthcoming)

[11] FALKENBERG, E.: Significations: The Key to Unify Data Base Management; Information Systems, Vol. 2 (1), April 1976

[12] HALL, P.; OWLETT, J.; TODD, S.: Relations and Entities, IFIP TC-2 working Conference on "Modelling in Data Base Management Systems", Freudenstadt, January 1976

[13] MOULIN, P. et al.: Conceptual Model as a Data Base Design Tool, IFIP TC-2 Working Conference on "Modelling in Data Base Management Systems", Freudenstadt, January 1976

[14] NIJSSEN, G.M.: A Gross Architecture for the Next Generation Data Base Management Systems, IFIP TC-2 Working Conference on "Modelling in Data Base Management Systems", Freudenstadt, January 1976

[15] SCHMID, H.A.; SWENSON, J.R.: On the Semantics of the Relational Data Model, Proceedings of the ACM-SIGMOD Conference, San Jose, California, May 1975

[16] SENKO, M.E. et al.: Data Structuring and Accessing in Data-Base Systems, IBM Systems Journal, Vol. 12, No. 1, 1973, p. 30-93

[17] SENKO, M.E.: DIAM as a Detailed Example of the ANSI/SPARC Architecture, IFIP TC-2 Working Conference on "Modelling in Data Base Management Systems", Freudenstadt, January 1976.

[18] SUNDGREN, B.: Conceptual Foundation of the Infological Approach to Data Bases, in Klimbie/Koffeman (eds.): Data Base Management, North-Holland 1974

APPENDIX: THE OBJECT-ROLE MODEL

There are two basic concepts:

object o
role r

An association a is defined as a set of object-role or association-role pairs:

$$a = \{(x_i , r_i)\} ,$$

where $x_i = o_i \vee a_i$, $i \in I$, $|I| \geq 1$, $o_i \in O$, $r_i \in R$, $a_i \in A$,

I index set: places of the association,
O total set of objects,
R total set of roles,
A total set of associations;

An association type A may be defined as a set of associations where each association contains the same set of roles:

$$A = \{a_j\} ,$$

where $j \in J$, $\forall i,j \neq k : r_{ij} \equiv r_{ik}$ for fixed $i \in I$ and $j,k \in J$,

(r_{ij} or r_{ik} means the role r_i within the association a_j or a_k , respectively) , $a_j \in A$,

I index set: places of the association type;
J index set: population of the association type;

An object type O may be defined as a set of objects, each of which plays at least one role in common:

$$O = \{o_k\} ,$$

where $k \in K$, $\forall k : r_i \exists (o_k , r_i)$ for $i \in I$, $o_k \in O$,

K index set: population of the object type;

DISEÑO DE BASES DE DATOS

A SOFTWARE ENGINEERING VIEW OF DATA BASE SYSTEMS

MAYO, 1985

A SOFTWARE ENGINEERING VIEW OF DATA BASE SYSTEMS

HERBERT WEBER

Hahn-Meitner-Institut
Berlin

CONTENTS

ABSTRACT

This tutorial focuses on the application of software engineering techniques for the development of data base systems. It is aimed to explain a key concept in software engineering - the module or abstract data type concept - and to demonstrate its usefulness in data base system structuring and design.

1. Introduction

Data base systems, as any other kind of software systems, are expensive to produce and maintain, and usually of low quality. In a great number of empiric studies the nonexistence of a rational software technology has been identified as the main reason for this so-called software dilemma. Thus, the identification and formulation of fundamental principles for the development of software systems in general, and for the production of data base systems as well, are now of growing interest.

Due to the very well-known human limitations in dealing with complexity, both the complexity of the system development task, and the complexity of the system itself, are considered to be accountable for the difficulties in the development of large or even rather small, software systems. In order to manage these complexities, technologies are now available, or are under development, which impose a discipline on software production and structuring.

A programming concept central to the discussion about a software technology since a number of years - frequently termed module or abstract data type - is considered to be the base for a solution of the principle problem mentioned above (PAR 71, PAR 72, PAR 74, LIS 74, LIS 77, WIR 77, WUL 76). Consequently, this paper is aimed at explaining this concept and demonstrating the impact of its use in data base system development.

2. The Module Concept

The recent history in programming has proven that the languages used to formulate programs influence the style of programming and consequently the shape of the programs. It is therefore assumed that choosing the right language features may also encourage good programming. The module concept explained below is considered to support the production of well-structured, reliable, robust and verifiably correct programs. In order to ease its understanding, the concept will be introduced here step by step. For that purpose, we first give a definition of the concept in BNF notation (for those readers who find descriptions in a meta language more comprehendable) and explain then its characteristics and advantages with a simple but nontrivial example (for those readers who prefer a more informal explanation of the subject) in a number of iterations.

2.1 Basic Definitions

In a first iteration the concept may be defined as follows:

```
<module>    ::=<interface><body>
<interface>::=module <module identifier>
                     <operator list>
<body>      ::=begin <data definition>
                     {<procedure definition>;}^n
                end
```

According to this definition, a module consists of two parts: an interface and a body. The interface contains a set of identifiers which may be referenced to gain access to the body of the module. The body is a program in some programming language. As will be shown a little later, the concept has been defined this way to support the rather distinctive requirements of a module user and a module implementor.

A sample module definition using a high level language notation would then take the following format.

```
module FLIGHT SCHEDULE (op_1, op_2,..,op_n)
begin modulebody
  .
  .
  .
end modulebody
```

This definition introduces a module called FLIGHTSCHEDULE. It is worth noting here that the module is named after a certain sort of data, thus indicating that the module has been designed to be invoked for the creation and manipulation of data objects of this sort. The interface of the module identifies, therefore, this sort of data, and all the operations applicable on those data. Since the text of the module body is not of interest for the following discussion, we will ignore it for the moment.

This "data driven" module definition is rather different from the more intuitive use of the term module in today's programming practice. It will, however, be shown throughout the rest of the paper that this notion seems to be adequate to overcome a great number of today's programming problems.

A number of terms closely related to this notion of module, and equally important, for the understanding and further explanation of the module concept will be defined now in an informal way.

The desired relationship between all legal inputs and the possible outputs of all the operations of a module will be called the functions of that module.

The specification of a module is an implementation independent more-or-less formal description of the functions of that module.

The implementation of a module - its body - is the program text for the data definition part (denoting the implementation of the sort of data pertinent to that module), and for all the procedure definitions (denoting the implementation of the operations pertinent to that module).

A program consists of an arbitrary number of module definitions, as explained above, and an arbitrary number of statements each referencing a certain module and one operation defined in that module, e.g.,

```
begin
M_i : module definition a;
M_j : module definition b;
S_k : module identifier a.op_m;
S_e : module identifier a.op_n;
S_z : module identifier b.op_t;
end
```

A process denotes the execution of module definitions and statements for an appropriate set of input parameters as defined in a program.

2.2 The abstraction principle

We are now ready to explain how the data-driven module notion suits the requirements of the module user and the module implementor as well:

It is the user's interest to employ the module concept to construct programs. Provided he knows the function of a previously defined module, only the interface, which contains all the information necessary to make proper use of the

module, must be visible to him. The details of the module implementation contained in its body would be an unnecessary burden and will remain hidden.

The module implementor on the other hand is responsible for an implementation in accordance to a given specification of the functions of the module.

In both cases an implementation-independent specification of the functions of a module is necessary. The module user needs the specification to make sure the employed module has the intended functions and the module implementor uses the specification to implement this function in a complete and correct manner. Consequently, an implementation-independent specification of the functions of a module is an essential part of the module concept.
(It would, of course, be the ultimate goal to make the specification of the function of a module a part of the interface which can be checked automatically to ensure the proper use of the module. Since a concept for the computer representation and interpretation of function specifications does not exist at the moment, we will consider them as aside from the module.)

This partitioning of information according to a certain need to know, i.e., the retention of the essential information, and the suppression of the inessential details for a certain purpose, is considered to be the key concept to master the complexity in information handling and is usually called the abstraction principle.

The concept is not new in programming. All high level languages, for example, provide means to declare and initialize data without forcing the programmer to assign data to specific memory locations. This feature helps reducing the complexity of the programmers' task by hiding the memory allocation details in the language processing system.

Another very well known abstraction mechanism supported in many high level programming languages is the procedure concept. A procedure is designed to display its function to its users and to hide the implementation of this function. It is therefore considered as a suitable mechanism for a functional or procedural abstraction.

The data-oriented module concept as defined above - as will be shown in the next section- is a generalization of the known abstraction mechanisms. It is designed to display to its users the essential information on how to use a certain sort of data, and to hide the information on how those data are internally represented and manipulated. Hence, it provides a general data abstraction mechanism.

3. The Definition of a Module Interface

In a next refinement step we will first complete the definition of a module interface. Starting from the previous definition:

```
⟨module⟩    ::= ⟨interface⟩⟨body⟩
⟨interface⟩ ::= module ⟨module identifier⟩
                       ⟨operator list⟩
⟨body⟩      ::= begin ⟨data definition⟩
                      {⟨procedure definition⟩;}^n
                end
```

the more detailed description of the interface may then be given as follows:

```
⟨operator list⟩     ::= (⟨operator symbol⟩
                         [⟨parameter list⟩]
                        {,⟨operator symbol⟩
                         [⟨parameter list⟩]}^n)

⟨parameter list⟩    ::= ⟨parameter symbol⟩
                        {,⟨parameter symbol⟩}^k

⟨module identifier⟩::= CHARACTER STRING
⟨operator symbol⟩  ::= CHARACTER STRING
⟨parameter symbol⟩ ::= CHARACTER STRING
```

According to this definition each of the n legal operations on the sort of data identified in the interface gets a set of k parameters associated with it. For the execution of an operation the k parameters declared for this operation need to be passed to the module.

The example introduced in the previous section will be used to explain this feature.

```
module FLIGHTSCHEDULE
       (create-schedule [id],
        search-flight [id, f#],
        schedule-flight [id, f#, dest, st-t],
        cancel-flight [id, f#]);
```

This interface identifies the FLIGHTSCHEDULE sort of data and four operations on it. It is the function of the create-schedule operation to create a data object of the sort FLIGHTSCHEDULE. For its execution the parameter "id" denoting the data object's identifier must be passed to the module in an operation call.

The function of the search-flight operation is to access and display a flight identified by a certain flight number f# which is assumed to be recorded within a flightschedule data object with identifier id. With the schedule- flight operation one may record a new flight, i.e., the flight number, f#; the destination, dest; and the start time, st-t; within the flight schedule data object id. Finally, with the cancel-flight operation one may delete the entry identified by f# in the flightschedule data object id.

It is obvious now that a module may be used to create and manipulate an arbitrary number of data objects with different identifiers. Since the implementation and consequently the function of the operations will be the same for all objects, a module may be considered as a template for the creation and manipulation of data objects which exhibit exactly the same properties.

Because of this characteristic, the module concept closely resembles the data type concept in high level programming languages: A variable denoting a data object may be declared to be of a certain type thus determining the properties of the object, i.e., its possible manipulations. The type implementation is part of the compiler and hidden for its users. Because of these similarities the module concept is also frequently called abstract data type concept.

The term type will therefore be used in the sequel to denote the properties of a data object and also to refer to the module which implements the operations for the creation and possible manipulation of all objects of a given type.

4. The Implementation of a Module

An implementation is supposed to bring something into effect. The implementation of data and operations makes operations executable on data. Thus, for the implementation of a sort of data and the associated operations as defined in a module interface they need to be represented in terms of already machine-supported data objects and operations.

The implementation consisting of a data definition part and a procedure definition part will be expressed in BNF notation as follows:

```
<data definition> ::= rep as <constructor>
                      of <module identifier>

<constructor>       ::= CHARACTER STRING
<module identifier>::= CHARACTER STRING
```

The term "constructor" in the definition above denotes a structuring concept which is applied to compose objects of representing type from objects of component type. This composition of the representing types from component types (or from one component type) is defined in a module which implements the representing type. Thus, the term constructor refers to a certain type which is called the representing type. Data and operations of an abstract type as identified in a module interface are then implemented in terms of objects and operations of the (machine-supported) representing type which defines at the same time, a composition of component types. For the previously introduced example the data definition may have the following format:

```
module FLIGHTSCHEDULE (create-schedule [id],..)
rep as FILE of FS-RECORD
```

A data object of type FLIGHTSCHEDULE is represented by an object of the type FILE which in turn is composed - in a way defined in a FILE module - of a number of objects of type FS-RECORD. Thus the implementation of the types FILE and FS-RECORD is a prerequisite for the implementation of the type FLIGHTSCHEDULE.
The procedure definition part expressed in BNF will be given as follows:

```
<procedure definition>::=proc <procedure head>
                           <procedure body>
<procedure head>     ::=<operator symbol>
                        [<parameter list>]
                          --> <result>
<result>             ::=<data object>|BOOLEAN
<data object>        ::=<object identifier>:
                        <module identifier>
<procedure body>     ::={<statement>;}^n
<statement>          ::=<operation call>|
                        <conditional statement>|
                        <unconditional statement>|
                        <for statement>
<operation call>     ::=<module identifier>.
                        <operator symbol>
                        [<parameter list>]
```

(The remaining undefined nonterminal symbols in this grammar have either been defined before or should be understood as in the definition of a high level language like ALGOL 60.)

An abstract operation is implemented by a procedure which performs operations on objects of representing type. Thus, one may include calls of operations on objects of representing type - as defined in the module for the representing type - within this procedure.

On the basis of these definitions and based on the assumption that a data type FILE (create file, search record, insert record, delete record) has already been defined, one may now give a complete program text for the FLIGHT-SCHEDULE module in the following form:

```
module FLIGHTSCHEDULE
       (create-schedule [id],
        search-flight [id, f#],
        schedule-flight [id, f#, dest, st-t],
        cancel-flight [id, f#] );

begin

rep as FILE of FS-RECORD

proc create-schedule [id]--> id:FLIGHTSCHEDULE;
id:FILE:=FILE. create-file [o];
end create-schedule;

proc search-flight [id, f#]--> BOOLEAN;
FILE. search-record [ id, f#];
end search flight;

proc cancel-flight [id, f#]--> id:FLIGHTSCHEDULE;
FILE. delete-record [ id, f#];
end cancel-flight;

proc schedule-flight [id, f#, dest, st-t] -->
     id:FLIGHTSCHEDULE;
FILE.insert-record [id, f#, dest, st-t] ;
end cancel-flight;

end module
```

Each of the procedures pertinent to the FLIGHT-SCHEDULE module encloses a call of an operation of the FILE module. Assumed all the called FILE operations are already implemented the FLIGHT-SCHEDULE module may be executed and is then implemented as well.

4.1 The locality principle

After the detailed definition of a module implementation in the preceding section, we are now ready to identify another basic principle underlying the module concept. Obviously, operations and data are closely related to each other in the module concept. Each data object may be manipulated by only a certain predefined set of operations. Data not associated to a certain module but rather global to a number of modules do not exist. Thus, logical relationships between modules based on the shared use of global data cannot occur.

The module concept also prohibits

a module to refer to data declared in the body of another module,

one module to branch into the body of another module, and

one module to modify program statements within another module.

Hence, a module behaves like a self-contained entity which cannot cause nonlocal effects besides calls of other modules. To achieve this kind of locality is one of the goals in modular programming.

Modular programming is believed to have a number of advantages over more traditional program structuring concepts: (1) the avoidance of certain types of structural relationships will force programmers to design programs of drastically reduced structural complexity. (2) Since the complexity of the environment in which a module will be used may be ignored by the implementor of that module it may also reduce the complexity of the programming task. (3) Because changes which have to be made in the implementation of a module will not affect any other part of a system the concept will enhance system maintenance, adaptability and portability. (4) With the module concept the verification of the correctness of programs - the ultimate goal in program development - may be drastically simplified. Obviously, it is much simpler to show that the invariant properties of data will be preserved if the data is manipulated by its associated operations only, and not by other parts of a program. The verification may then be performed for each of the associated operations and not for all - usually unpredictable - uses of the data.

4.2 Protection of data

In a modular software system, each operation may only be applied to a certain type of data object. The operations are tailored to comply with the characteristics of the data they manipulate, e.g. it is common practice to manipulate integer data by a set of tailored arithmetic operations. Current high-level-languages compilers enforce this restricted application of operations defined in the language by means of a type checking capability for the built-in types of data. Since all the legal operations on a certain type of data are predefined in a module, the correct use of these operations may be enforced by a similar type checking mechanism.

This approach is in contrast to the current practice in systems programming. Universal operations like delete, insert or update may be applied to data of any kind. In order to preserve the data's characteristics it is usually necessary to implement access control and protection mechanisms.

The two aforementioned approaches to preserve data characteristics in high level programming languages and in systems programming are based on two fundamentally different philosophies: (1) Because of the awareness of the human limitations, the first approach follows the rule: anything not explicitly allowed is forbidden; (2) to guarantee the designers freedom and flexibility, the second approach follows the rule: anything not explicitly forbidden is allowed.

After a long period of freedom and flexibility in the design of systems it now seems to be clear that a discipline is essential for the enhancement of software. The module concept and an associated type checking mechanism seem to be the natural means to avoid not intended manipulations of data.

4.3 Extensibility

A module is characterized by a minilanguage: the abstract data and abstract operations of that module. A minilanguage is implemented in terms of another minilanguage provided by the module of representing type, e.g., the minilanguage

```
L1: FLIGHTSCHEDULE (create-schedule,
                    search-flight,
                    schedule-flight,
                    cancel-flight)
```

is implemented in terms of the minilanguage:

```
L2: FILE (create-file,
          search-record,
          insert-record,
          delete-record).
```

More generally, an abstract operation may be implemented by a number of representing type operations. Each abstract operation may then be considered as an identifier for a macro operation on data objects of representing types.

This is in fact an extension capability similar to the one found in extensible languages. With the repeated definition of new modules implemented in terms of macro operations on objects of representing type, which in turn may be implemented by macro operations on objects of component type, one may define arbitrary high level (mini) languages to suit particular users.

5. Module Interconnections

A discipline for the design and implementation of small programs (programming in the small) has been defined with the module concept. A similar discipline for the design and implementation of large software systems (programming in the large) must then provide rules for the interconnection of modules.

Since one module may employ other modules, one module interconnection mechanism - the nesting of modules - is already built-in in the module definition. With this structuring mechanism the overall structure of a software system may be organized in a hierarchic fashion. The hierarchic relationship between the modules may be characterized as a "uses" hierarchy because the services of one module may be used in another module (Par 74). A module in this hierarchy will be considered correct only if all modules it calls function properly. The module interconnection however, will be kept simple because neither the used nor the using modules impose any restrictions on each other. They all remain self-contained system components which function the same way in all environments.

This hierarchic organization of large software systems is accepted to provide means to keep the complexity of the system manageable and the function of the system understandable.

In order to keep the overall structure a simple hierarchy, other interconnection mechanisms - especially those neglecting the locality principle as explained above - are prohibited. The grammar introduced in the previous sections of this paper is in fact the description of a module definition and module interconnection language. The language is therefore suitable for the programming-in-the-small and programming-in-the-large. It imposes a discipline on the programming task and supports the design of simply structured software systems. The language is hoped to be appropriate to serve as a general data base system design and programming language.

It may be important to note here, that the hierarchic organization of software systems does not predetermine the way they are designed: top-down, bottom-up, or in a more iterative fashion.

6. The Module Oriented View of Data Base Systems

6.1 The Object Model of a Data Base

In accordance with common terminology a data base may be defined as the collection of all data objects stored on some storage devices and administered by a data base management system (DAT 75). The data base will be brought into existence through the initialization and execution of transactions of the administering data base management system.

Similarily, using the module concept, a data object will be brought into existence through the execution of a create operation of one particular module. The data object is then considered to be pertinent to that module. Only those operations defined in the module will ever be performed on the object. The object is said to be of the particular type defined by that module.

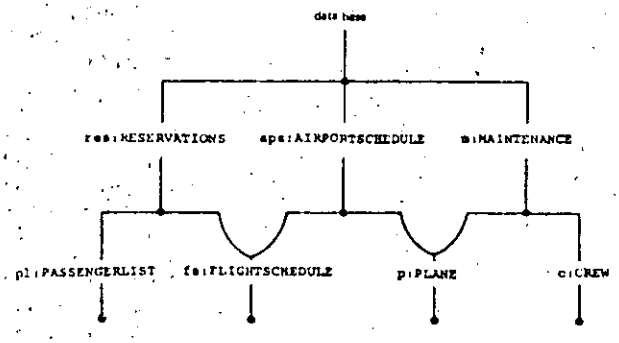
Because of the possible hierarchic compositions of modules, the create operation of one module may be designed to "use" the create operations of other modules to create and combine component objects. With this feature the module concept allows to create, store, and reference data on different levels of composition. At the same time, the concept automatically enforces the composition of objects out of components of the correct types as defined in the type module. (This may not seem to be very important for the standard compositions of data, like fixed format records in files, or files in blocks, but it is valuable for the composition of arbitrary user defined data types). Within the framework of the module concept one may then define the object model of a data base in the following way (WEB 76):

(1) A data base is a time varying set of data base objects.
(2) Data base objects may encompass other data base objects.
(3) Data base objects may belong to a number of different enclosing objects.
(4) Data base objects are identified by names. They are uniquely identified within one enclosing object. They may be differently named in different enclosing objects.
(5) Data base objects are characterized by a type and each object is characterized by exactly one type.
(6) The type of an object is defined in the associated type module which provides a definition of the composition of objects of other type and a definition of all permissible operations on composed objects.

The object model of a data base may be graphically represented as a D-graph (Data Object Graph). The graph may be defined as follows:

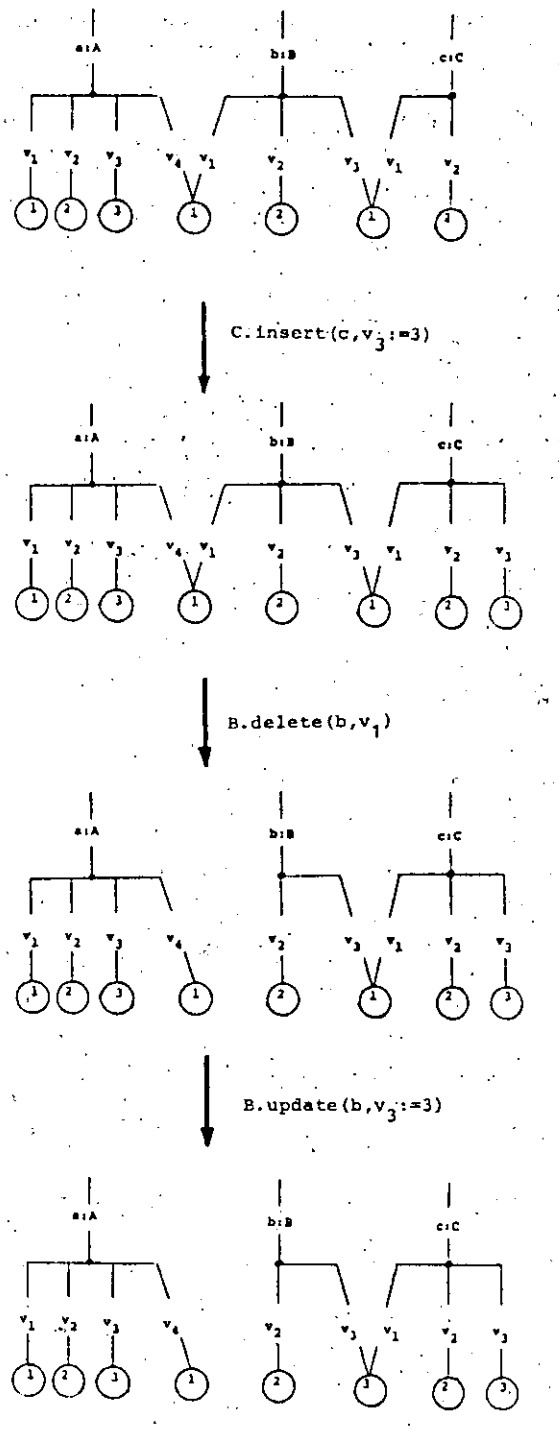
(1) Nodes of the D-graph represent data objects.
(2) The directed arcs represent an "is part" relationship between objects. If an object i is component of an object j, a directed arc is drawn from j to i.
(3) Labels on arcs identify component objects and the type of component objects.

Clearly, the "is part" relationship between data base objects defines an acyclic graph because the whole (e.g., a file) includes parts (e.g., records) but a part never includes the whole. All non-root nodes may be in an "is part" relationship with a number of nodes.(One may call them "shared" among a number of higher level nodes.) A sample graph may then be depicted as follows: (Arcs are supposed to point downward, they are represented by consecutive horizontal and vertical lines for representation convenience. Nonterminal nodes represent composed objects, terminal nodes represent primitive indivisible objects.)



The data base contains information on reservations which encompass information on passengers who have made reservations and on flights and their scheduling. The component of the data base termed airportschedule contains information on flights and their scheduling, and on planes which are allocated to those flights. The component termed maintenance contains the information on planes and on maintenance crews allocated to maintain those planes.

The possible changes of a data base by insertions, deletions and updates may be represented now in terms of modifications of the D-graph by insertions or removals of nodes or links and by replacements of nodes. They may be illustrated with the following depiction of three consecutive operations:



The object model conforms with the commonly accepted definition of a data base as quoted before although the object model definition does not refer to the data base management system for the determination of the membership of a data object to the data base. For the object model, each module may be considered as the administering (mini-) data base management system for a particular type of data object. The entire data base management system may be thought of as the set of all defined and implemented modules. (A more elaborate definition of such a modular data base management system will follow later.)

The D-graph model is different from existing data base models in the following sense. Existing data base concepts support the composition of data according to one particular structuring model usually called data model (e.g., the DBTG concept (DBTG 71) supports the owner-coupled-set data model, the relational concept supports the relational data model,etc.). The D-graph model of a data base supports hierarchic compositions of differently structured data instances as defined in the associated type modules.

6.2 The Conceptual Description of a Modular Data Base

Data bases are repositories of all the data of interest in an organization. They contain the information the organization needs for its operations. Thus, data in the data base have a perceivable information content.

Data in the data base are of course subject to change. A data base object may be changed by insertions, deletions and updates of component objects. It is therefore important to distinguish two different aspects of the information content: the extension and the intension of the data base. The term extension refers to the instantaneous and time dependent aspects of the information content (e.g., all the tuples in a relation at a certain time)*. The term intension refers to the time invariant aspects of the information content (e.g., the set of all permissible values an object can take). Data base objects may then be manipulated (i.e., the extension may be changed) according to their time invariant properties (i.e., according to their intension).

*The D-graph model introduced above is clearly a representation for the data base's extension. To represent the data base's extension at different times the D-graph was changed through the insertion or removal of nodes and node connections.

Consequently, in order to manipulate the data base correctly, both its extension and its intension must be represented within the data base. The data base's intension is represented in a so-called conceptual description.*

The time invariant aspects of the information of interest to an organization may be modelled in terms of the following concepts:

(1) Entity types or short entities are all concrete and abstract things or events an organization needs to know of for its operation. An entity is meant to denote a collection of instances with identical characteristics, e.g. the entity FLIGHTNUMBER denotes a variable set of instances F_1, F_2... which have the identical characteristic to identify flights. Instances have values, e.g., $F_i = \{100, 200, 300\}$.

Instances and values are subject to change. They are part of the extension aspects of information.

(2) Conceptual relationships are associations between the entities of interest in an organization, e.g. a conceptual relationship between the FLIGHTNUMBER and DESTINATION entities refers to the fact that each flight identified by a flight number has a destination.

For the representation of the time invariant aspects of information one may then describe entities and conceptual relationships in an arbitrary language.

The representation of information in a language is in fact an assignment of labels to entities and relationships. Representations in a data base are then encoded representations of labels in computer storage. In a simple representation each label is assigned to one entity or relationship and each entity or relationship gets only one label, thus, providing means for a unique representation and identification.

* Conceptual description of data bases are usually aimed to provide: (1) a high-level user-oriented description of the data base for its users; (2) support for a correct interpretation of the data base's information content; (3) means for the representation of restriction on the use of data in different applications, etc. All these aspects will not be considered here. It is the sole aim of this description to refer to the support it provides for the correct manipulation of the data base. Elaborate proposals for conceptual description may be found in the literature on data dictionaries or conceptual schemata (ANS 75, BCS 77, NIJ 76, VLDB 75, VLDB 76).

Labels, however, usually do not identify things uniquely. They rather denote collections of things which are identical with respect to some properties and distinct with respect to others. For example, the label FLIGHTSCHEDULE denotes things which are identical because they all identify flights, destinations and start times. At the same time the denoted things may be distinct because some of them identify regular flights, and others, night coach flights. Due to the different properties of interest in different situations the very same label refers to different sets of values the things can assume in the different contexts, e.g., if the FLIGHTSCHEDULE label is intended to denote flightschedules for regular flights all start times must be within the time span 5 AM to 10 PM; if it is intended to denote night coach flights, all start times must be within the time span 10 PM to 5 AM.

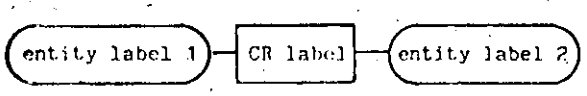
For a proper use of labels for the representation of entities and relationships, the information about the intended set of legal values of their instances must be represented as well. It is common practice to represent this information in terms of constraints associated with entities and relationships, e.g., the set of legal values of instances of the FLIGHTNUMBER entity may be defined by the constraint $0 \leq f\# \leq 100$. The computer representation of entities and relationships may then be based on the use of

(1) labels which denote entities and relationships, and

(2) constraints associated with entities and relationships.

As for the representation of the data base's extension a representational schema is usually defined for the representation of its intension (frequently called conceptual schema).

It is common practice to depict entities and relationships by the following kind of graphs:



This building block may be used to construct entity-relationship-nets of arbitrary shape and complexity.* Nets of this kind are then suitable to depict the information of interest in an organization.

So far this schema does not permit the representation of constraints. The constraints associated with entities and relationships will be represented as conditions which will be checked whenever modifications of the values of instances take place, e.g., for each execution of the operation add-number on an instance F# of entity FLIGHTNUMBER the following check will be performed

if $0 < f\# < 100$
then FLIGHTNUMBER add-number (F# ,f#)
else return FALSE

to make sure that new flightnumbers f# will only be added if they are taken from the set of natural numbers between 0 and 100.

The representation of entities, conceptual relationships, and their properties may therefore be given in terms of entity-relationship-nets and constraints.

(Another somewhat different - and probably even better known - terminology refers to some different concepts for the representation of a data base's intension in current data base technology. Contemporary concepts support only the representation of a very limited number of conceptual relationships (e.g., owner-coupled-set relationships among record types (DBTG 71), functional dependencies among attributes in a relation (Codd 70)). They provide therefore the representation of the data base's intension in terms of entities and some conceptual relationships among entities in a data base schema and of other relationships and of value constraints in integrity constraints.

It remains to be shown now that the module concept suffices for the representation of a data base's intension in terms of entity- relationship- nets and of constraints.

6.2.1 The representation of entities

As one may conclude from the previous discussion data types and entities (labels denoting entities) seem to be very similar tools for the representation of information: they both denote (real or abstract) things with common properties. Because of this similarity it seems to be natural to use modules as a means for the definition and implementation of entities in the same way they were used to define and implement types of data.

(1) Module interfaces denote entities and the set of permissible operations on instances of entities.

(2) The implementation of entities and of operations on their instances is defined in module bodies.

(3) Modules may be invoked to create instances and to modify the values associated to them.

Although there exist some obvious similarities between entities and abstract data types a

*For the sake of simplicity, the terms entity and relationship are used now to denote both real things and their representation by labels as well.

couple of important differences must be kept in mind:

(1) Entities have not been defined in conjunction with the set of permissible operations on their instances.

This feature of a module to define data and operations together, however, seems to be quite adequate for the representation of the invariant properties of entities. As one may recall, those properties have been represented in terms of constraints on the values of entity instances and have been enforced during value modifications. Because all possible value modifications are defined in a module, constraints on the possible values may be defined within the framework of a module and enforced in module executions in a straightforward way: Operation execution conditions may be associated with all value changing operations of a module. The operation will be executed only if the conditions set forth are satisfied. These conditions may be represented in a module within the procedures which are defined to implement those operations, e.g.,

```
module FLIGHTNUMBER (add-number, delete-number)

begin

rep as SET of NAT

proc add-number[ F#, f#]-->F#;
if 0<f#<100 TRUE
then SET.add-element[ S, f#]
else return FALSE;
end add-number;
.
.
.
end module
```

The add-element operation will be executed only if the operation execution condition (0<f#<100) is satisfied.

For the representation of entities and constraints on the values of their instances a module definition may then take the following general format:

```
module ENTITY IDENTIFIER (oper 1,
                          oper 2,
                          .
                          .
                          .
                          oper n)
begin

entity representation

proc oper 1
operation execution condition 1
operation execution condition 2
      .
      .
statement 1
statement 2
      .
      .
      .
end oper 1
      .
      .
      .
end module
```

Thus, the definition and implementation of an entity and its invariant properties by a module results in a rather implicit representation of constraints on the values of its instances in terms of constraint preserving operations. A module representation of an entity is then defined as

$$E = (V,O)$$

where
E is the entity
V is the set of permissible values of its instances
O is the set of operations applicable to its instances

The module representation of the entity is said to be constraint preserving if the following condition holds:

$$\forall v_E \in V_E : o_i(v_E) \in V_E$$

The application of an operation o_i on any permissible value v_E of an instance of E results in a permissible value v_E. This representation provides a means for hiding the information about the invariant properties of an entity within the implementation of the permissible operations. The feature seems then to support the structuring of the intensional part of a data base according to a certain need to know, e.g., information about the modifiability of the data base may remain hidden for its users.

(2) The second difference between entities and abstract data types results from another characteristic of modules. Modules have been designed to employ other modules for the implementation

of objects and operations of abstract type in terms of objects and operations of representing type. The mapping between objects of abstract type and of objects of representing type implies always a classification: All legal values of objects of abstract type will be mapped onto a subset of legal values of objects of the representing type. This classification of the set of legal values of representing type takes place because of the likeness of the selected values with respect to a certain property, e.g. an abstract object of type NIGHTCOACHFLIGHTSCHEDULE may be represented in terms of objects of type FLIGHTSCHEDULE; the set of legal values of objects of type NIGHTCOACHFLIGHTSCHEDULE is a subset of the legal values of type FLIGHTSCHEDULE; the legal values of objects NIGHTCOACHFLIGHTSCHEDULE are alike with respect to a certain starttime. This property is not common to all values of objects of the FLIGHTSCHEDULE type. Thus, objects of abstract types have properties which will not be inherited by objects of representing type.

If modules are used to define and implement entities, the above capability may be exploited to define classifications of instances according to certain properties. One may call this classification abstractive, since it has been made on the basis of some distinctive properties of interest, while all other properties have been ignored. Thus, those abstractive classifications may be introduced for the definition of new entities with new common characteristics. At the same time classifications delimit the scope of attention to some properties of the entities, and ignore all others. Based on this abstractive classification, information may be represented on arbitrary levels of detail or abstraction. One may refer to those classifications as schema abstraction which are equivalent to the data abstractions introduced in Section 2.2.

5.2.2 The Representation of Relationships

At a first glance the module concept does not seem to be very useful for the representation of relationships. Some considerations about the nature of entities and relationships, however, may offer some help. The distinction between entities and relationships is certainly not absolute: things may be considered as entities in one context and as relationships in another, e.g., a FLIGHTSCHEDULE may certainly be considered as an entity. It materializes, however at the same time a relationship between FLIGHTNUMBERS, DESTINATIONS, and STARTTIMES. If it is in general true, that the distinction between entities and relationships is just a matter of the context and not a matter of their representation, the question may be asked whether there is any real need to represent entities and relationships in a different manner. It seems to us that no principal difficulties exist to treat relationships in the same way as entities. A relationship and the entities connected through this relationship may be considered as a composed entity of its own right.

For the representation of relationships by modules, a capability is needed for the composition of entities and for the definition of the characteristics of the composed entity which reflect the relationship between the components entities.

The module concept has been defined to offer exactly this capability: arbitrary module interconnections may be defined in the module of representing type and the characteristics of the module reflect the nature of a relationship between the component modules.

The following example illustrates such a composition. Suppose the data base contains the following two entities

```
FLIGHTSCHEDULE: F#, DEST, ST-T,
      CREATE-SCHEDULE
      SEARCH-FLIGHT
      SCHEDULE-FLIGHT
      CANCEL-FLIGHT
PLANE:P#,TYPE,N°S
      CREATE-PLANE
      SEARCH-PLANE
      RESERVE-PLANE
      CANCEL-PLANERESERVATION
```

The relationship which must hold between these two entities is of the following nature:

The scheduling of a flight requires the corresponding allocation of a plane to this flight. The data base contains for each entry in the FLIGHTSCHEDULE instance a corresponding entry in the PLANE instance.

We therefore define a new entity whose specification reflects this relationship and which encloses the entities FLIGHTSCHEDULE and PLANE as components. This new entity will be called AIRPORTSCHEDULE:

```
AIRPORTSCHEDULE:FLIGHTSCHEDULE, PLANE
      CREATE-APSCHEDULE
      SEARCH-ENTRY
      ADD-ENTRY
      DELETE-ENTRY
```

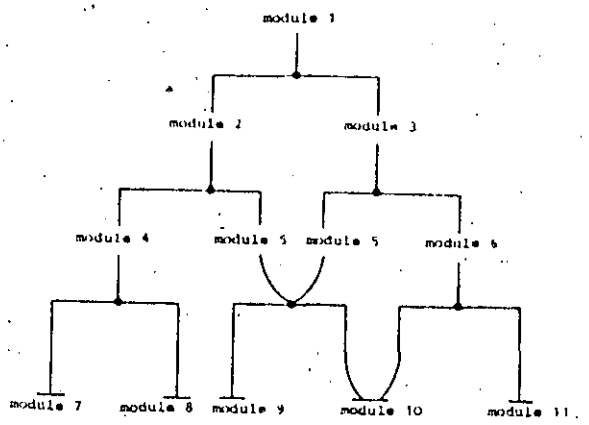
The relationship will be preserved if the operations on AIRPORTSCHEDULE instances are designed in such a way that the component instances of FLIGHTSCHEDULE and PLANE will be manipulated correspondingly, e.g.

```
module AIRPORTSCHEDULE (..., add-entry,...)

begin

rep as INTERCONNECTION
of (FLIGHTSCHEDULE, PLANE)
.
.
.
oper add-entry [aps, f#, dest, st-t, p#, type,
        n°s] --> aps:AIRPORTSCHEDULE;
FLIGHTSCHEDULE.add-flight [fs, f#, dest, st-t];
PLANE.reserve-plane [p, p#, type, n°s];
end add-entry;
.
.
.
end module
```

The add-entry operation will be performed by performing both the add-flight and the reserve-plane operations.

6.2.3 Conclusion

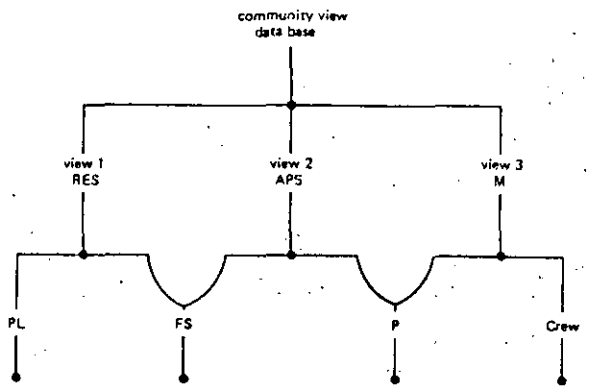
Both entities and relationship were represented as data types, their instances as objects of the respective type. The different values instances may assume were represented as different instantiations of changeable objects. Data types, and the definition of their implementation by modules, provide representation capabilities which are general enough for the conceptual description of a data base. Because of the built-in module interconnection mechanism in modules, the representational schema consists then of a collection of modules which are hierarchically structured in a graph similar to a D-graph. (Nodes represent now entities, links point to all modules which represent related component entities, the D-graph itself may be considered as an entity/relationship composition graph.)



A number of consequences follow from this rather uncommon concept, which incorporates the representation of declarative and procedural knowledge, and of schema abstractions. An elaborate comment on it must be left aside here. A lot of discussions about similar concepts may be found in the artificial intelligence literature. Although the concept is applied now to data bases as well, its impact is not fully explored yet.

6.3 The Representation of Data Base Views

The term view is used here to denote a conceptual description (frequently called external description) of the data base which suits one user or one group of users. A number of views of a data base will usually exist because a data base is shared among different users. Different users may be interested in different but overlapping parts of the data base, and may have a different understanding of the purpose of the data base. Nevertheless, views must be compatible with each other. One may define non-conflicting views within the framework of the module concept in the following way:
Views may be considered as different schema abstractions. Different views may then incorporate different subsets of data base objects, may have objects in common, and may see the objects they share involved in different relationships.



view 1: the flight reservation view sees FLIGHTSCHEDULE objects in a relationship with PASSENGERLIST objects.

view 2: the flight scheduling view sees FLIGHTSCHEDULE objects in a relationship with the PLANE objects.

view 3: the plan maintenance view sees the PLANE objects in a relationship with maintenance CREW objects.

If we allow different views to have objects in common and allow views to see shared objects in different relationships, we have to make sure that different views agree - at least to some extent - on the intension of the shared objects. (Otherwise the integrity of the data base may suffer from non-intended manipulations of shared objects.) It is shown in (WEB 76) that modules are suitable to define these sharing properties of data base objects. A brief description of the concept follows in the next paragraphs.

6.3.1 The State Space of Data Base Objects

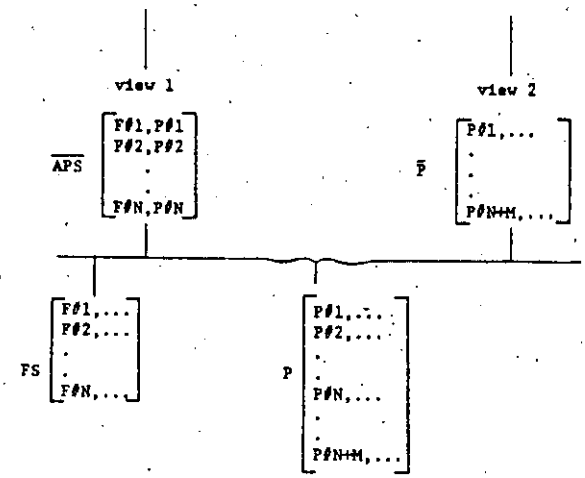
In order to explain the notion of agreement about the intension of data base objects, it is convenient to introduce the concept of a statespace and of substate spaces for changeable data base objects. An object may be changed by changes of the set of its components, they have however a particular set of components at a certain time. Consequently, they may be considered to be in a certain state at a certain time and their state will change in object manipulations. The set of legal states which corresponds to the objects's intension may be called the state space of an object. Each subset of the set of legal states may be called a substate space. A statespace of a data object may be considered as the unifying concept to describe the set of legal values of instances of both entities and relationships.

As explained before, the description of the representation of an object and the description of the associated manipulations in a module embodies in fact the definition of the space of all legal states for this object. A user who inserts a new object defines, with the declaration of its type, its state space, and will be called the owner of the object. Views may share an object in a nonconflicting manner, if they are defined by subsets of the owner view state space (i.e., a substate space).

6.3.2 The Image of Data Base Objects

In order to represent different states of different substate spaces of the same data base object (i.e., of different views), we have to introduce the notion "image of an object".

If the substate spaces of an object are different in different views, components of this object may exist at a certain time which may legally belong to one view (i.e., to one substate space) but not to another. One may say the different views have different images (or different parts of the same object's state) of an object, i.e., they see different instantiations of an object.



View 2 sees an object P which embodies an instantiation of P which represents all available planes. View 1 contains instantiations which represent all allocated planes. The different images of the object are defined through the selector relations $\overline{APS}$ (F#, P#) and $\bar{P}$ (P#,..) They both identify different subsets of the set of components of P and make them visible in different views.

The two different views may be created and maintained because of the following definition of the two view modules:

Assumed the type of P is defined by a module
PLANE (P#,TYPE,N°S)
CREATE-PLANE
SEARCH-PLANE
INSERT-PLANE
DELETE-PLANE

Different instantiations of an object P may be created and maintained in the two different views if different subsets of the set of legal operations may be called from the two different views.

VIEW 1 (The Schedulers' View)

SEARCH PLANE
Semantics: None of the changing operations on P can be initiated from view 1. This is to express the fact that view 1 is only authorized to initiate the allocation of already recorded planes. The set of components of P in view 1 will then be at any time a subset of the set of components seen in the subsequently defined view 2. Consequently, the set of possible states of P in view 1 is a subset of the set of possible states of P in view 2.

VIEW 2 (The Inventory View)

SEARCH PLANE;
INSERT PLANE;
Semantics: Insertions of components into P may be initiated from view 2 without any restriction to indicate its authority to record the existence of planes for further use. The dele-

tion and update of components, however, must be reserved to another view which has control over both views and is able to prevent the deletion or update of already allocated planes. (One may call this a superview.)

6.3.4 Communication Among Views

One may distinguish two basically different interrelationships between views in a data base. They may either coexist (i.e, there is no interference between views although they have components in common) or they may cooperate (i.e., a strictly controlled communication may occur by making changes of a shared object visible to all views which see the object). Both concepts will be briefly discussed in the following paragraphs.

Views may be defined to co-exist because the D-graph concept allows one to maintain and manipulate just images of data base objects. It therefore provides means to manipulate the data base via one view without affecting any other view. Views coexist in the data base, in general, if the following operations are performed:

(1) insert (component of object) Its effect would be a modification of the image of the object and of its state. The component inserted via one view, however, would not be visible in other views, since the object's image seen in the other views has not been changed.

(2) delete (component of object) Its effect may be just a change of the image of the object without affecting the state.

Views cooperate if update operations on shared objects may be performed. Update operations preserve all the images and change the state of an object. The caused changes are therefore visible in all views which reference the object. In order to keep the data base in a consistent state, a general policy for the performance of updates via different views must be established.

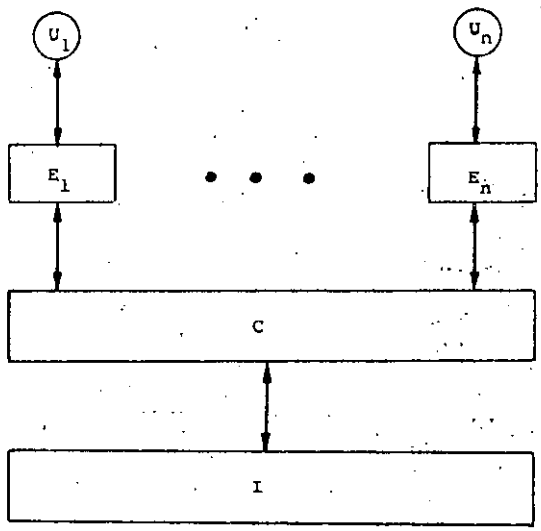
(1) If a non-owner view seeks the sharing of an object, it must agree to all possible changes of this object made by the owner of the object.

(2) An owner view may grant the opportunity to change an object it owns to all or a selected number of other views. A non-owner may then accept this opportunity.

Based on a more detailed explanation in (WEB 76) one can conclude that insertions and deletions may be performed not causing conflicts among user views. Updates may be performed to enable the communication between views if proper rules for this communication are set forth.

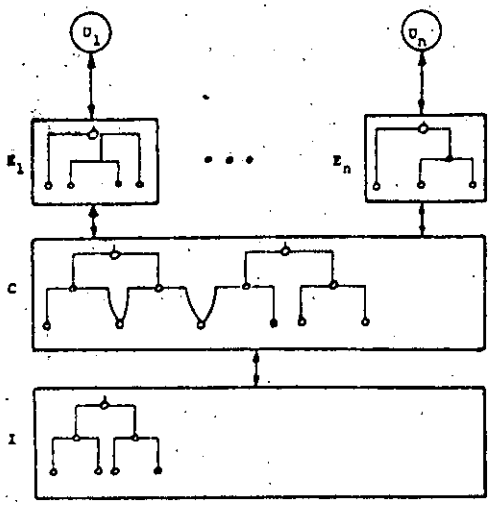
6.4 The Module Oriented View of Data Base Management Systems

Some existing and most proposed data base management systems are structured in a layered fashion. The layers correspond to the different modes for the representation of information in the data base (usually called logical and physical in a two-level architecture, or in a three-level architecture, external, conceptual and internal). The layers are employed successively for data base interrogations and manipulations.



A number of external "machines" will be designed to process different users' interaction languages. The external "machines" will be implemented in terms of a conceptual "machine" which in turn will be implemented by an internal "machine". Usually different types of data (i.e. differently structured or differently manipulatable data) will be processed by each of these "machines".

The module concept provides means for the implementation of those "machines" and conforms therefore with this principal data base management system architecture. It offers, however, some further system structuring capabilities which are appropriate to define arbitrary structural refinements for each machine. Refinements may be either functional (i.e. the gross function of a machine is decomposed into sub-functions of component machines) or data driven (i.e. a machine which processes a number of types of data will be decomposed in a number of component machines each processing a subset of the types of data). Such an architecture may then be depicted as follows:



This architecture may be developed in a coherent way in an overall design process. It may be also the result of an extension of already existing data management systems (which will then be employed by the modularily designed levels above).

7. Other Related Work

The module concept has been applied to a number of other problems in data base management which will not be described in detail here.

(1) A software design strategy based on the module notion has been applied to develop a methodology for the design of a family of data base systems. The strategy supports a top-down design of modular data base systems (Yeh 77, Yeh 78).

(2) A few attempts have been reported to develop a methodology for the formal specification and verification of data base systems. The methodology is based on concepts for the algebraic specification of modules (PAO 77, EKW 78, BR 78).

(3) The benefits which can possibly be gained from an application of the concept in designing distributed data base systems have been described in (HEB 78).

(4) Last, but not least, the concept has been applied to model security and privacy enforcement mechanisms (MIN 76).

8. Conclusion

The module concept has been shown to be suitable to model the main features of data base systems. It should consequently contribute to the simplification of the data base system development and to the enhancement of data base systems. The use of modules as a descriptive tool does not imply any redefinition of accepted basic concepts in data base management. It offers however, in some cases, a more precise definition of the concepts.

Although an increasing number of people are doing work on the subject, experimental projects along this line have not yet been reported so far. The presentation was aimed at stimulating some further work on the application of the concept to database systems.

Acknowledgement

The author gratefully acknowledges the careful reading of an earlier draft of the paper by M. Brodie and H.-J. Kreowski, and many helpful comments by N. Roussopoulos, H. Ehrig, and K. Kreplin.

Bibliography

(ANS 75) ANSI/X3/SPARC Study Group on Data Base Management Systems, Interim Report, ACM FDT Bulletin vol. 7, No. 2 (1975).

(BCS 77) The British Computer Society Data Dictionary, Systems Working Party Report, ACM SIGMOD RECORD, vol. 9, No. 4 (1977)

(BR 78) Brodie, M., Specification and Verification of Data Base Semantic Integrity, TR CSRG-91 1978, Univ. of Toronto

(COD 70) Codd, E.F., A relational model of data for large shared data banks, Comm.ACM, vol. 13, No. 6 (June 1970)

(DAT 75) Date, C. J. "An Introduction to Data Base Systems",Addison Wesley (1975)

(DBTG 71) CODASYL Systems Committee, Report of the CODASYL data base task group, ACM (April 1971)

(EKW 78) Ehrig, H., Kreowski, H.J., Weber, H. "Algebraic Specification Schemes for Data Base Systems" Technical Report HMI-B266, Hahn-Meitner- Institut Berlin (1978)

(HEB 78) Hebalkar, P. G., "Application Specification for Distributed Data Base Systems" submitted for publication, 1978

(LIS 74) Liskov, B., Zilles, S., "Programming with Abstract Data Types" ACM SIGPLAN Notices, vol. 9, No. 4 (April 1974)

(LIS 77) Liskov, B., "Abstraction Mechanisms in CLU," Comm. ACM, vol. 20, No. 8 (August 1977)

(MIN 76) Minsky, N., Intensional resolution of privacy protection in data base systems, Comm. ACM, vol. 23, No. 2 (April 1976)

(NIJ 76) Nijssen, G. M. (ed.) Modelling in Data Base Management Systems, North Holland (197 Data Base Management Systems, North Holland (1976)

(PAR 71) Parnas, D., "Information Distribution Aspects of Design Methodology", Information Processing 71, North Holland (1971)

(PAR 72) Parnas, D., "On the Criteria to be Used in Decomposing Systems into Modules", Comm. ACM, vol. 15, No. 12 (Dec. 1972)

(PAR 74) Parnas, D., "On a' Buzzword': Hierarchical Structure", Information Processing 74, North Holland (1974)

(PAO 77) Paolini, P., Pelagatti, G., "Formal Definition of Mappings in a Data Base", ACM SIGMOD Conference Proceedings (1977)

(VLDB 75) Kerr, D., ed. Proceedings of the First International Conference on Very Large Data Bases, available through ACM (1975)

(VLDB 76) Lockemann, P., Neuhold, E. Proceedings of the Second International Conference on Very Large Data Bases, North Holland (1976)

(VLDB 77) Proceedings of the Third International Conference on Very Large Data Bases, available though ACM (1977)

(WEB 76) Weber, H., "The D-Graph Model of Large Shared Data Bases: A Representation of Integrity Constraints and Views as Abstract Data Types", IBM Research Report RJ 1875 (Nov 1976)

(WIR 77) Wirth, N., "Modula: A Language for Modular Multiprogramming, Software-Practice and Experience, vol. 7 (1977)

(WUL 76) Wulf, W. A., LONDON, R. L., and SHAW,M., "An Introduction to the Construction and Verification of Alphard Programs", IEEE Transactionson Software Engineering, vol. SE-2, No. 4 (Dec. 1976)

(YEH 77) Yeh, R.T., Baker, J. W., "Toward a Design Methodology for DBMS: A Software Engineering Approach" in (VLDB 77)

(YEH 78) Yeh, R. T., Roussopoulos, N., Chang,P., "Data Base Design - An Approach and Some Issues" Technical Report SDBEG-4, University Texas at Austin

DISEÑO DE BASES DE DATOS

DATABASE DESCRIPTION WITH SDM: A SEMANTIC DATABASE MODEL

MICHAEL HAMMER
DENNIS MCLEOD

MAYO, 1985

Database Description with SDM: A Semantic Database Model

MICHAEL HAMMER
Massachusetts Institute of Technology
and
DENNIS McLEOD
University of Southern California

SDM is a high-level semantics-based database description and structuring formalism (database model) for databases. This database model is designed to capture more of the meaning of an application environment than is possible with contemporary database models. An SDM specification describes a database in terms of the kinds of entities that exist in the application environment, the classifications and groupings of those entities, and the structural interconnections among them. SDM provides a collection of high-level modeling primitives to capture the semantics of an application environment. By accommodating derived information in a database structural specification, SDM allows the same information to be viewed in several ways; this makes it possible to directly accommodate the variety of needs and processing requirements typically present in database applications. The design of the present SDM is based on our experience in using a preliminary version of it.

SDM is designed to enhance the effectiveness and usability of database systems. An SDM database description can serve as a formal specification and documentation tool for a database; it can provide a basis for supporting a variety of powerful user interface facilities, it can serve as a conceptual database model in the database design process; and, it can be used as the database model for a new kind of database management system.

Key Words and Phrases: database management, database models, database semantics, database definition, database modeling, logical database design
CR Categories: 3.73, 3.74, 4.33

1. INTRODUCTION

Every database is a *model* of some real world system. At all times, the contents of a database are intended to represent a snapshot of the state of an *application environment*, and each change to the database should reflect an event (or sequence of events) occurring in that environment. Therefore, it is appropriate

This research was supported in part by the Joint Services Electronics Program through the Air Force Office of Scientific Research (AFSC) under Contract F44620-76-C-0061, and, in part by the Advanced Research Projects Agency of the Department of Defense through the Office of Naval Research under Contract N00014-76-C-0944. The alphabetical listing of the authors indicates indistinguishably equal contributions and associated funding support.

Authors' addresses: M. Hammer, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139; D. McLeod, Computer Science Department, University of Southern California, University Park, Los Angeles, CA 90007.

that the structure of a database mirror the structure of the system that it models. A database whose organization is based on naturally occurring structures will be easier for a database designer to construct and modify than one that forces him to translate the primitives of his problem domain into artificial specification constructs. Similarly, a database user should find it easier to understand and employ a database if it can be described to him using concepts with which he is already familiar.

The global user view of a database, as specified by the database designer, is known as its (*logical*) *schema*. A schema is specified in terms of a database description and structuring formalism and associated operations, called a *database model*. We believe that the data structures provided by contemporary database models do not adequately support the design, evolution, and use of complex databases. These database models have significantly limited capabilities for expressing the meaning of a database and to relate a database to its corresponding application environment. The *semantics* of a database defined in terms of these mechanisms are not readily apparent from the schema; instead, the semantics must be separately specified by the database designer and consciously applied by the user.

Our goal is the design of a higher-level database model that will enable the database designer to naturally and directly incorporate more of the semantics of a database into its schema. Such a semantics-based database description and structuring formalism is intended to serve as a natural application modeling mechanism to capture and express the structure of the application environment in the structure of the database.

1.1 The Design of SDM

This paper describes *SDM*, a database description and structuring formalism that is intended to allow a database schema to capture much more of the meaning of a database than is possible with contemporary database models. SDM is designed to provide features for the natural modeling of database application environments. In designing SDM, we analyzed many database applications, in order to determine the structures that occur and recur in them, assessed the shortcomings of contemporary database models in capturing the semantics of these applications, and developed strategies to address the problems uncovered. This design process was iterative, in that features were removed, added, and modified during various stages of design. A preliminary version of SDM was discussed in [21]; however, this initial database model has been further revised and restructured based on experience with its use. This paper presents a detailed specification of SDM, examines its applications, and discusses its underlying principles.

SDM has been designed with a number of specific kinds of uses in mind. First, SDM is meant to serve as a formal specification mechanism for describing the meaning of a database; an SDM schema provides a precise documentation and communication medium for database users. In particular, a new user of a large and complex database should find its SDM schema of use in determining what information is contained in the database. Second, SDM provides the basis for a variety of high-level semantics-based user interfaces to a database; these interface facilities can be constructed as front-ends to existing database management systems, or as the query language of a new database management system. Such

interfaces improve the process of identifying and retrieving relevant information from the database. For example, SDM has been used to construct a user interface facility for nonprogrammers [28]. Finally, SDM provides a foundation for supporting the effective and structured design of databases and database-intensive application systems.

SDM has been designed to satisfy a number of criteria that are not met by contemporary database models, but which we believe to be essential in an effective database description and structuring formalism [22]. They are as follows.

(1) The constructs of the database model should provide for the explicit specification of a large portion of the *meaning* of a database. Many contemporary database models (such as the CODASYL DBTG network model [11, 47] and the hierarchical model [48]) exhibit compromises between the desire to provide a user-oriented database organization and the need to support efficient database storage and manipulation facilities. By contrast, the relational database model [12, 13] stresses the separation of user-level database specifications and underlying implementation detail (data independence). Moreover, the relational database model emphasizes the importance of understandable modeling constructs (specifically, the nonhierarchic relation), and user-oriented database system interfaces [7, 8].

However, the *semantic expressiveness* of the hierarchical, network, and relational models is limited; they do not provide sufficient mechanism to allow a database schema to describe the meaning of a database. Such models employ overly simple data structures to model an application environment. In so doing, they inevitably lose information about the database; they provide for the expression of only a limited range of a designer's knowledge of the application environment [4, 36, 49]. This is a consequence of the fact that their structures are essentially all record-oriented constructs; the appropriateness and adequacy of the record construct for expressing database semantics is highly limited [17, 22–24, 27]. We believe that it is necessary to break with the tradition of record-based modeling, and to base a database model on structural constructs that are highly user oriented and expressive of the application environment. To this end, it is essential that the database model provide a rich set of features to allow the direct modeling of application environment semantics.

(2) A database model must support a *relativist* view of the meaning of a database, and allow the structure of a database to support alternative ways of looking at the same information. In order to accommodate multiple views of the same data and to enable the evolution of new perspectives on the data, a database model must support schemata that are flexible, potentially logically redundant, and integrated. *Flexibility* is essential in order to allow for multiple and coequal views of the data. In a *logically redundant* database schema, the values of some database components can be algorithmically derived from others. Incorporating such derived information into a schema can simplify the user's manipulation of a database by statically embedding in the schema data values that would otherwise have to be dynamically and repeatedly computed. Furthermore, the use of derived data can ease the development of new applications of the database, since new data required by these applications can often be readily adjoined to the

existing schema. Finally, an *integrated* schema explicitly describes the relationships and similarities between multiple ways of viewing the same information. Without a degree of this critical integration, it is difficult to control the redundancy and to specify that the various alternative interpretations of the database are equivalent.

Contemporary, record-oriented database models do not adequately support relativism. In these models, it is generally necessary to impose a single structural organization of the data, one which inevitably carries along with it a particular interpretation of the data's meaning. This meaning may not be appropriate for all users of the database and may furthermore become entirely obsolete over time. For example, an association between two entities can legitimately be viewed as an attribute of the first entity, as an attribute of the second entity, or as an entity itself; thus, the fact that an officer is currently assigned as the captain of a ship could be expressed as an attribute of the ship (its current captain), as an attribute of the officer (his current ship), or as an independent (assignment) entity. A schema should make all three of these interpretations equally natural and direct. Therefore, the conceptual database model must provide a specification mechanism that simultaneously accommodates and integrates these three ways of looking at an assignment. Conventional database models fail to adequately achieve these goals.

Similarly, another consequence of the primacy of the principle of relativism is that, in general, the database model should not make rigid distinctions between such concepts as entity, association, and attribute. Higher-level database models that do require the database schema designer to sharply distinguish among these concepts (such as [9, 33]) are thus considered somewhat lacking in their support of relativism.

(3) A database model must support the definition of schemata that are *based on abstract entities*. Specifically, this means that a database model must facilitate the description of relevant *entities* in the application environment, *collections* of such entities, *relationships* (associations) among entities, and *structural interconnections* among the collections. Moreover, the entities themselves must be distinguished from their syntactic identifiers (*names*); the user-level view of a database should be based on actual entities rather than on artificial entity names.

Allowing entities to represent themselves makes it possible to directly reference an entity from a related one. In record-oriented database models, it is necessary to cross reference between related entities by means of their identifiers. While it is of course necessary to eventually represent "abstract" entities as symbols inside a computer, the point is that users (and application programs) should be able to reference and manipulate abstractions as well as symbols; internal representations to facilitate computer processing should be hidden from users.

Suppose, for example, that the schema should allow a user to obtain the entity that models a ship's current captain from the ship entity. To accomplish this, it would be desirable to define an attribute "Captain" that applies to every ship, and whose value is an officer. To model this information using a record-oriented database model, it is necessary to select some identifier of an officer record (e.g., last name or identification number) to stand as the value of the "Captain" attribute of a ship. For example, using the relational database model, we might have a relation SHIPS, one of whose attributes is Officer__name, and a relation

OFFICERS, which has Officer__name as a logical key. Then, in order to find the information about the captain of a given ship, it would be necessary to join relations SHIPS and OFFICERS on Officer__name; an explicit cross reference via identifiers is required. This forces the user to deal with an extra level of indirection and to consciously apply a join to retrieve a simple item of information.

In consequence of the fact that contemporary database models require such surrogates to be used in connections among entities, important types of semantic integrity constraints on a database are not directly captured in its schema. If these semantic constraints are to be expressed and enforced, additional mechanisms must be provided to supplement contemporary database models [6, 16, 19, 20, 45]. The problem with this approach is that these supplemental constraints are at best ad hoc, and do not integrate all available information into a simple structure. For example, it is desirable to require that only captains who are known in the database be assigned as officers of ships. To accomplish this in the relational database model, it is necessary to impose the supplemental constraint that each value of attribute Captain__name of SHIPS must be present in the Captain__name column of relation OFFICERS. If it were possible to simply state that each ship has a captain attribute whose value is an officer, this supplemental constraint would not be necessary.

The design of SDM has been based on the principles outlined above which are discussed at greater length in [22].

2. A SPECIFICATION OF SDM

The following general principles of database organization underlie the design of SDM.

(1) A database is to be viewed as a collection of *entities* that correspond to the actual objects in the application environment.
(2) The entities in a database are organized into *classes* that are meaningful collections of entities.
(3) The classes of a database are not in general independent, but rather are logically related by means of *interclass connections.*
(4) Database entities and classes have *attributes* that describe their characteristics and relate them to other database entities. An attribute value may be derived from other values in the database.
(5) There are several primitive ways of defining interclass connections and derived attributes, corresponding to the most common types of information redundancy appearing in database applications. These facilities integrate multiple ways of viewing the same basic information, and provide building blocks for describing complex attributes and interclass relationships.

2.1 Classes

An *SDM database* is a collection of entities that are organized into classes. The structure and organization of an SDM database is specified by an *SDM schema,* which identifies the classes in the database. Appendix A contains an example SDM schema for a portion of the "tanker monitoring application environment"; a specific syntax (detailed in Appendix B) is used for expressing this schema. Examples in this paper are based on this application domain, which is concerned

with monitoring and controlling ships with potentially hazardous cargoes (such as oil tankers), as they enter U.S. coastal waters and ports. A database supporting this application would contain information on ships and their positions, oil tankers and their inspections, oil spills, ships that are banned from U.S. waters, and so forth.

Each class in an SDM schema has the following features.

(1) A *class name* identifies the class. Multiple synonymous names are also permitted. Each class name must be unique with respect to all class names used in a schema. For notational convenience in this paper, class names are strings of uppercase letters and special characters (e.g., OIL__TANKERS), as shown in Appendix A.

(2) The class has a collection of *members*: the entities that constitute it. The phrases "the members of a class" and "the entities in a class" are thus synonymous. Each class in an SDM schema is a homogeneous collection of one type of entity, at an appropriate level of abstraction.

The entities in a class may correspond to various kinds of objects in the application environment. These include objects that may be viewed by users as:

(a) concrete objects, such as ships, oil tankers, and ports (in Appendix A, these are classes SHIPS, OIL__TANKERS, and PORTS, respectively);
(b) events, such as ship accidents (INCIDENTS) and assignments of captains to ships (ASSIGNMENTS);
(c) higher-level entities such as categorizations (e.g., SHIP__TYPES) and aggregations (e.g., CONVOYS) of entities;
(d) names, which are syntactic identifiers (strings), such as the class of all possible ship names (SHIP__NAMES) and the class of all possible calendar dates (DATES).

Although it is useful in certain circumstances to label a class as containing "concrete objects" or "events" [21], in general the principle of relativism requires that no such fixed specification be included in the schema; for example, inspections of ships (INSPECTIONS) could be considered to be either an event or an object, depending upon the user's point of view. In consequence, such distinctions are not directly supported in SDM. Only name classes (classes whose members are names) contain data items that can be transmitted into and out of a database, for example, names are the values that may be entered by, or displayed to, a user. Nonname classes represent abstract entities from the application environment.

(3) An (optional) textual *class description* describes the meaning and contents of the class. A class description should be used to describe the specific nature of the entities that constitute a class and to indicate their significance and role in the application environment. For example, in Appendix A, class SHIPS has a description indicating that the class contains ships with potentially hazardous cargoes that may enter U.S. coastal waters. Tying this documentation directly to schema entries makes it accessible and consequently more valuable.

(4) The class has a collection of attributes that describe the members of that class or the class as a whole. There are two types of attributes, classified according to *applicability*.

(a) A *member attribute* describes an aspect of each member of a class by logically connecting the member to one or more related entities in the same or another class. Thus a member attribute is used to describe each member of some class. For example, each member of class SHIPS has attributes Name, Captain, and Engines, which identify the ship's name, its current captain, and its engines (respectively).

(b) A *class attribute* describes a property of a class taken as a whole. For example, the class INSPECTIONS has the attribute Number, which identifies the number of inspections currently in the class; the class OIL__TANKERS has the attribute Absolute__legal__top__speed which indicates the absolute maximum speed any tanker is allowed to sail.

(5) The class is either a *base class* or a *nonbase class.* A base class is one that is defined independently of all other classes in the database; it can be thought of as modeling a primitive entity in the application environment, for example, SHIPS. Base classes are mutually disjoint in that every entity is a member of exactly one base class. Of course, at some level of abstraction all entities are members of class "THINGS"; SDM provides the notion of base class to explicitly support cutting off the abstraction below that most general level. (If it is desired that all entities in a database be members of some class, then a single base class would be defined in the schema.)

A nonbase class is one that does not have independent existence; rather, it is defined in terms of one or more other classes. In SDM, classes are structurally related by means of *interclass connections.* Each nonbase class has associated with it one interclass connection. In the schema definition syntax shown in Appendix A, the existence of an interclass connection for a class means that it is nonbase; if no interclass connection is present, the class is a base class. In Appendix A, OIL__TANKERS is an example of a nonbase class; it is defined to be a subclass of SHIPS which means that its membership is always a subset of the members of SHIPS.

(6) If the class is a base class, it has an associated list of groups of member attributes; each of these groups serves as a logical key to uniquely identify the members of a class (*identifiers*). That is, there is a one-to-one correspondence between the values of each identifying attribute or attribute group and the entities in a class. For example, class SHIPS has the unique identifier Name, as well as the (alternative) unique identifier Hull__number.

(7) If the class is a base class, it is specified as either *containing duplicates* or *not containing duplicates.* (The default is that duplicates are allowed; in the schema syntax used in Appendix A, "duplicates not allowed" is explicitly stated to indicate that a class may not contain duplicate members.) Stating that duplicates are not allowed amounts to requiring the members of the class to have some difference in their attribute values; "duplicates not allowed" is explicit shorthand for requiring all of the member attributes of a class taken together to constitute a unique identifier.

2.2 Interclass Connections

As specified above, a nonbase class has an associated interclass connection that defines it. There are two main types of interclass connections in SDM: the first

allows subclasses to be defined and the second supports grouping classes. These interclass connection types are detailed as follows.

2.2.1 *The Subclass Connection.* The first type of interclass connection specifies that the members of a nonbase class (S) are of the same basic entity type as those in the class to which S is related (via the interclass connection). This type of interclass connection is used to define a subclass of a given class. A *subclass* S of a class C (called the *parent class*) is a class that contains some, but not necessarily all, of the members of C. The very same entity can thus be a member of many classes, for example, a given entity may simultaneously be a member of the classes SHIPS, OIL_TANKERS, and MERCHANT_SHIPS. (However, only one of these may be a base class.) This is the concept of "subtype" [21, 25, 31, 32, 41] which is missing from most database models (in which a record belongs to exactly one file).

In SDM, a subclass S is defined by specifying a class C and a predicate P on the members of C; S consists of just those members of C that satisfy P. Several types of predicates are permissible.

(1) A predicate on the member attributes of C can be used to indicate which members of C are also members of S. A subclass defined by this technique is called an *attribute-defined subclass*. For example, the class MERCHANT_SHIPS is defined (in Appendix A) as a subclass of SHIPS by the member attribute predicate "where Type = 'merchant'"; that is, a member of SHIPS is a member of MERCHANT_SHIPS if the value of its attribute Type is "merchant." (A detailed discussion of member attribute predicates is provided in what follows. The usual comparison operators and Boolean connectives are allowed.)

(2) The predicate "where specified" can be used to define S as a *user-controllable subclass* of C. This means that S contains at all times only entities that are members of C. However, unlike an attribute-defined subclass, the definition of S does not identify which members of C are in S; rather, database users "manually" add to (and delete from) S, so long as the subclass limitation is observed. For example, BANNED_SHIPS is defined as a "where specified" subclass of "SHIPS"; this allows some authority to ban a ship from U.S. waters (and possibly later rescind that ban).

An essential difference between attribute-defined subclasses and user-controllable subclasses is that the membership of the former type of subclass is determined by other information in the database, while the membership of the latter type of subclass is directly and explicitly controlled by users. It would be possible to simulate the effect of a user-controllable subclass by an attribute-defined subclass, through the introduction of a dummy member attribute of the parent class whose sole purpose is to specify whether or not the entity is in the subclass. Subclass membership could then be predicated on the value of this attribute. However, this would be a confusing and indirect method of capturing the semantics of the application environment; in particular, there are cases in which the method of determining subclass membership is beyond the scope of the database schema (e.g., by virtue of being complex).

(3) A subclass definition predicate can specify that the members of subclass S are just those members of C that also belong to two other specified data-

base classes (C_1 and C_2); this provides a class *intersection* capability. To insure a type-compatible intersection, C_1 and C_2 must both be subclasses of C, either directly or through a series of subclass relationships. For example, the class BANNED_OIL_TANKERS is defined as the subclass of SHIPS that contains those members common to the classes OIL_TANKERS and BANNED_SHIPS.

In addition to an intersection capability, a subclass can be defined by class *union* and *difference*. A union subclass contains those members of C in either C_1 or C_2. For example, class SHIPS_TO_BE_MONITORED is defined as a subclass of SHIPS with the predicate "where is in BANNED_SHIPS or is in OIL_TANKERS_REQUIRING_INSPECTION." A difference subclass contains those members of C that are not in C_1. For example, class SAFE_SHIPS is defined as the subclass of SHIPS with the predicate "where is not in BANNED_SHIPS."

The intersection, union, and difference subclass definition primitives allow *set-operator-defined subclasses* to be specified; these primitives are provided because they often represent the most natural means of defining a subclass. Moreover, these operations are needed to effectively define subclasses of user-controllable subclasses. For example, class intersection (rather than a member attribute predicate) must be used to define class SHIPS_TO_BE_MONITORED; since BANNED_SHIPS and OIL_TANKERS_REQUIRING_INSPECTION are both user-controllable subclasses, no natural member attributes of either of these classes could be used to state an appropriate defining member attribute predicate for SHIPS_TO_BE_MONITORED.

(4) The final type of subclass definition allows a subclass S to be defined as consisting of all of the members of C that are currently values of some attribute A of another class C_1. That is, class S contains all of the members of C that are a value of A. This type of class is called an *existence subclass*. For example, class DANGEROUS_CAPTAINS is defined as the subclass of OFFICERS satisfying the predicate "where is a value of Involved_captain of INCIDENTS"; this specifies that DANGEROUS_CAPTAINS contains all officers who have been involved in an incident.

2.2.2 *The Grouping Connection.* The other type of interclass connection allows for the definition of a nonbase class, called a *grouping class* (G), whose members are of a higher-order entity type than those in the underlying class (U). A grouping class is *second order*, in the sense that its members can themselves be viewed as classes; in particular, they are classes whose members are taken from U.

The following options are available for defining a grouping class.

(1) The grouping class G can be defined as consisting of all classes formed by collecting the members of U into classes based on having a common value for one or more designated member attributes of U (an *expression-defined grouping class*). A *grouping expression* specifies how the members of U are to be placed into these groups. The groups formed in this way become the members of G, and the members of a member of G are called its *contents*. For example, class SHIP_TYPES in Appendix A is defined as a grouping class of SHIPS with the grouping expression "on common value of Type". The members of

SHIP_TYPES are not ships, but rather are groups of ships. In particular, the intended interpretation of SHIP_TYPES is as a collection of types of ships, whose instances are the contents (members) of the groups that constitute SHIP_TYPES. This kind of grouping class represents an abstraction of the underlying class. That is, the elements of the grouping class correspond in a sense to the shared property of the entities that are its contents, rather than to the collection of entities itself.

If the grouping expression used to define a grouping class involves only a single-valued attribute, then the groups partition the underlying class; this is the case for SHIP_TYPES. However, if a multivalued attribute is involved, then the groups may have overlapping contents. For example, the class CARGO_TYPE_GROUPS can be defined as a grouping class on SHIPS with the grouping expression "on common value of Cargo_types"; since Cargo_types is multivalued, a given ship may be in more than one cargo type category. Although the grouping mechanism is limited to single grouping expressions (namely, on common value of one or more member attributes), complex grouping criteria are possible via derived attributes (as discussed in what follows).

It should be clear that the contents of a group are a subclass of the class underlying the grouping. The grouping expression used to define a grouping class thus corresponds to a collection of attribute-defined subclass definitions. For example, for SHIP_TYPES, the grouping expression "on common value of Type" corresponds to the collection of subclass member attribute predicates (on SHIPS) "Type = 'merchant'," "Type = 'fishing'," and "Type = 'military'." Some or all of these subclasses may be independently and explicitly defined in the schema. In Appendix A, the class MERCHANT_SHIPS is defined as a subclass of SHIPS, and it is also listed in the definition of SHIP_TYPES as a class that is explicitly defined in the database ("groups defined as classes are MERCHANT_SHIPS"). In general, when a grouping class is defined, a list of the names of the groups that are explicitly defined in the schema is to be included in the specification of the interclass connection; the purpose of this list is to relate the groups to their corresponding subclasses in the schema.

(2) A second way to define a grouping class G is by providing a list of classes $(C_1, C_2, \ldots, C_n)$ that are defined in the schema; these classes are the members of the grouping class (an *enumerated grouping class*). Each of the classes $(C_1, C_2, \ldots, C_n)$ must be explicitly defined in the schema as an (eventual) subclass of the class U that is specified as the class underlying the grouping. This grouping class definition capability is useful when no appropriate attribute is available for defining the grouping and when all of the groups are themselves defined as classes in the schema. For example, a class TYPES_OF_HAZARDOUS_SHIPS can be defined as "grouping of SHIPS consisting of classes BANNED_SHIPS, BANNED_OIL_TANKERS, and SHIPS_TO_BE_MONITORED."

(3) A grouping class G can be defined to consist of user-controllable subclasses of some underlying class (a *user-controllable grouping class*). In effect, a user-controllable grouping class consists of a collection of user-controllable subclasses. For example, class CONVOYS is defined as a grouping of SHIPS "as specified." In this case, no attribute exists to allow the grouping of ships into convoys and individual convoys are not themselves defined as classes in the schema; rather, each member of CONVOYS is a user-controllable group of ships that users may

add to or delete from. This kind of grouping class models simple "aggregates" over a base class: arbitrary collections of entities manipulated by users.

2.2.3 *Multiple Interclass Connections.* As specifed above, each nonbase class in an SDM schema has a single interclass connection associated with it. While it is meaningful and reasonable in some cases to associate more than one interclass connection with a nonbase class, the uncontrolled use of such multiple interclass connections could introduce undesirable complexity into a schema. In consequence, only a single interclass connection (the most natural one) should be used to define a nonbase class.

To illustrate this point, consider for example the class RURITANIAN_OIL_TANKERS. Clearly, this class could be specified as an attribute-defined subclass of OIL_TANKERS (by the interclass connection "subclass of OIL_TANKERS where Country.Name = 'Ruritania'"), or as a subclass of RURITANIAN_SHIPS (by the interclass connection "subclass of RURITANIAN_SHIPS where Cargo_types contains 'oil'"); these definitions are, in a sense, semantically equivalent. The possibility of allowing multiple (semantically equivalent) interclass connections to be specified for a nonbase class was considered, but it was determined that such a feature could introduce considerable complexity: The mechanism could be used to force two class definitions that are not semantically equivalent to define classes with the same members. For example, one could associate interclass connections that define the class of all Ruritanian ships and the class of all dangerous ships with a single class, intending to force the sets of members of these two possibly independent collections to be the same. In sum, without a carefully formulated and powerful notion of semantic equivalence [30], it was determined that multiple interclass connections for a nonbase class should not be allowed in SDM. Of course, multiple class names and judiciously selected class descriptions can be used to convey additional definitions, for example, naming a class BANNED_SHIPS and RURITANIAN_OIL_TANKERS to indicate that the two sets of ships are intended to be one and the same.

2.3 Name Classes

Entities are application constructs that are directly modeled in an SDM schema. In the real world, entities can be denoted in a number of ways; for example, a particular ship can be identified by giving its name or its hull number, by exhibiting a picture of it, or by pointing one's finger at the ship itself. Operating entirely within SDM, the typical way of referencing an entity is by means of an entity-valued attribute that gives access to the entity itself. However, there must also be some mechanism that allows for the outside world (i.e., users) to communicate with an SDM database. This will typically be accomplished by data being entered or displayed on a computer terminal. However, one cannot enter or display a real entity on such a terminal; it is necessary to employ representations of them for that purpose. These representations are called SDM *names*. A name is any string of symbols that denotes an actual value encountered in the application environment; the strings "red," "128," "8/21/78," and "321-004" are all names. A name class in SDM is a collection of strings, namely, a subclass of the built-in class STRINGS (which consists of all strings over the basic set of alphanumeric characters).

Every SDM name class is defined by means of the interclass connection "subclass." The following methods of defining a class S of names are available.

(1) The class S can be defined as the intersection, union, or difference of two other name classes.
(2) The class S can be defined as a subclass of some other name class C with the predicate "where specified," which means that the members of S belong to C, but must be explicitly enumerated. In Appendix A class COUNTRY__NAMES is defined in this way.
(3) A predicate can be used to define S as a subclass of C. The predicate specifies the subset of C that constitutes S by indicating constraints on the format of the acceptable data values. In Appendix A, classes ENGINE__SERIAL__NUMBERS, DATES, and CARGO__TYPE__NAMES are defined in this way. CARGO__TYPE__NAMES has no format constraints, indicating that all strings are valid cargo type names. ENGINE__SERIAL__NUMBERS and DATES do have constraints that indicate the patterns defining legal members of these classes. Note that for convenience, the particular name classes NUMBERS, INTEGERS, REALS, and YES/NO (Booleans) are also built into SDM; these classes have obvious definitions. (Further details of the format specification language used here are presented in [26].)

2.4 Attributes

As stated above, each class has an associated collection of attributes. Each attribute has the following features.

(1) An *attribute name* identifies the attribute. An attribute name must be unique with respect to the set of all attribute names used in the class, the class's underlying base class, and all eventual subclasses of that base class. (As decribed in [30], this means that attribute names must be unique within a "family" of classes; this is necessary to support the attribute inheritance rules described in what follows.) As with class names, multiple synonymous attribute names are permitted. For notational convenience in this paper, attribute names are written as one uppercase letter followed by a sequence of lowercase letters and special characters (e.g., the attribute Cargo__types of class SHIPS), as shown in Appendix A.
(2) The attribute has a *value* which is either an entity in the database (a member of some class) or a collection of such entities. The value of an attribute is selected from its underlying *value class*, which contains the permissible values of the attribute. Any class in the schema may be specified to be the value class of an attribute. For example, the value class of member attribute Captain of SHIPS is the class OFFICERS. The value of an attribute may also be the special value *null* (i.e., no value).
(3) The *applicability* of the attribute is specified by indicating that the attribute is either:
    (a) a member attribute, which applies to each member of the class, and so has a value for each member (e.g., Name of SHIPS); or
    (b) a class attribute, which applies to a class as a whole, and has only one value for the class (e.g., Number of INSPECTIONS).

(4) An (optional) *attribute description* is text that describes the meaning and purpose of the attribute. For example, in Appendix A, the description of Captain of SHIPS indicates that the value of the attribute is the current captain of the ship. (This serves as an integrated form of database documentation.)

(5) The attribute is specified as either *single valued* or *multivalued*. The value of a single-valued attribute is a member of the value class of the attribute, while the value of a multivalued attribute is a subclass of the value class. Thus, a multivalued attribute itself defines a class, that is, a collection of entities. In Appendix A, the class OIL_TANKERS has the single-valued member attribute Hull_type and the multivalued member attribute Inspections. (In the schema definition syntax used in Appendix A, the default is single valued.) It is possible to place a constraint on the size of a multivalued attribute, by specifying "multivalued with size between X and Y," where X and Y are integers; this means that the attribute must have between X and Y values. For example, attribute Engines of SHIPS is specified as "multivalued with size between 0 and 10"; this means that a SHIP has between 0 and 10 engines.

(6) An attribute can be specified as *mandatory*, which means that a null value is not allowed for it. For example, attribute Hull_number of SHIPS is specified as "may not be null"; this models the fact that every SHIP has a Hull_number.

(7) An attribute can be specified as *not changeable*, which means that once set to a nonnull value, this value cannot be altered except to correct an error. For example, attribute Hull_number of SHIPS is specified as "not changeable."

(8) A member attribute can be required to be *exhaustive* of its value class. This means that every member of the value class of the attribute (call it A) must be the A value of some entity. For example, attribute Engines of SHIPS "exhausts value class," which means that every engine entity must be an engine of some ship.

(9) A multivalued member attribute can be specified as *nonoverlapping* which means that the values of the attribute for two different entities have no entities in common; that is, each member of the value class of the attribute is used at most once. For example, Engines of SHIPS is specified as having "no overlap in values," which means that any engine can be in only one ship.

(10) The attribute may be related to other attributes, and/or defined in terms of other information in the schema. The possible types of such relationships are different for member and class attributes, and are detailed in what follows.

2.4.1 *Member Attribute Interrelationships.* The first way in which a pair of member attributes can be related is by means of inversion. Member attribute A_1 of class C_1 can be specified as the *inverse* of member attribute A_2 of C_2 which means that the value of A_1 for a member M_1 of C_1 consists of those members of C_2 whose value of A_2 is M_1. The inversion interattribute relationship is specified symmetrically in that both an attribute and its inverse contain a description of the inversion relationship. A pair of inverse attributes in effect establish a binary association between the members of the classes that the attributes modify. (Although all attribute inverses could theoretically be specified, if only one of a

pair of such attributes is relevant, then it is the only one that is defined in the schema, that is to say, no inverse specification is provided.) For example, attribute Ships__registered__here of COUNTRIES is specified in Appendix A as the inverse of attribute Country__of__registry of SHIPS; this establishes the fact that both are ways of expressing in what country a ship is registered. This is accomplished by

(1) specifying that the value class of attribute Country__of__registry of SHIPS is COUNTRIES, and that its inverse is Ships__registered__here (of COUNTRIES);
(2) specifying that the value class of attribute Ships__registered__here of COUNTRIES is SHIPS, and that its inverse is Country__of__registry (of SHIPS).

The second way in which a member attribute can be related to other information in the database is by *matching* the value of the attribute with some member(s) of a specified class. In particular, the value of the match attribute A_1 for the member M_1 of class C_1 is determined as follows.

(1) A member M_2 of some (specified) class C_2 is found that has M_1 as its value of (specified) member attribute A_2.
(2) The value of (specified) member attribute A_3 for M_2 is used as the value of A_1 for M_1.

If A_1 is a multivalued attribute, then it is permissible for each member of C_1 to match to several members of C_2; in this case, the collection of A_3 values is the value of attribute A_1. For example, a matching specification indicates that the value of the attribute Captain for a member S of class SHIPS is equal to the value of attribute Officer of the member A of class ASSIGNMENTS whose Ship value is S.

Inversion and matching provide multiple ways of viewing n-ary associations among entities. Inversion permits the specification of binary associations, while matching is capable of supporting binary and higher degree associations. For example, suppose it is necessary to establish a ternary association among oil tankers, countries, and dates, to indicate that a given tanker was inspected in a specified country on a particular date. To accomplish this, a class could be defined (say, COUNTRY__INSPECTIONS) with three attributes: Tanker__inspected, Country, and Date__inspected. Matching would then be used to relate these to appropriate attributes of OIL__TANKERS, COUNTRIES, and DATES that also express this information. Inversions could also be specified to relate the relevant member attributes of OIL__TANKERS (e.g., Countries__in__which__inspected), COUNTRIES (e.g., Tankers__inspected__here), DATES, and COUNTRY__INSPECTIONS (see Figure 1).

The combined use of inversion and matching allows an SDM schema to accommodate relative viewpoints of an association. For instance, one may view the ternary relationship in the above example as an inspection entity (a member of class COUNTRY__INSPECTIONS), or as a collection of attributes of the entities that participate in the association. Similarly, a binary relationship defined as a pair of inverse attributes could also be viewed as an association entity, with matching used to relate that entity to the relevant attributes of the associated entities [30].

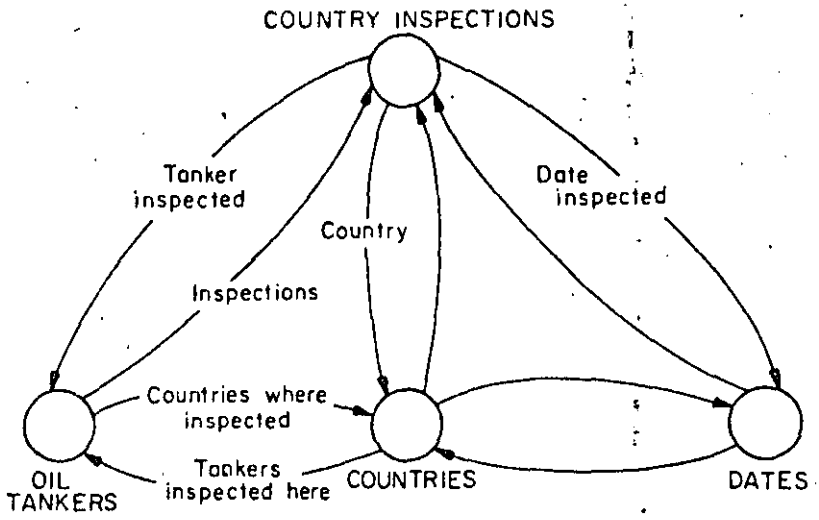


Fig. 1. Multiple perspectives on the "Country Inspections" association. Circles denote classes and are labeled with class names. Arrows denote member attributes, labeled by name, with the arrowhead pointing to the attribute's value class. For brevity, only some of the possible attributes are named (as would be the case in many real SDM schemata).

2.4.1.1 *Member Attribute Derivations.* As described above, inversion and matching are mechanisms for establishing the equivalence of different ways of viewing the same essential relationships among entities. SDM also provides the ability to define an attribute whose value is calculated from other information in the database. Such an attribute is called *derived*, and the specification of its computation is its associated *derivation*.

The approach we take to defining derived attributes is to provide a small vocabulary of high-level attribute derivation primitives that directly model the most common types of derived information. Each of these primitives provides a way of specifying one method of computing a derived attribute. More general facilities are available for describing attributes that do not match any of these cases: A complex derived attribute is defined by first describing other attributes that are used as building blocks in its definition and then applying one of the primitives to these building blocks. For example, attribute Superiors of OFFICERS is defined by a derivation primitive applied to attribute Commander, and in turn, attribute Contacts is defined by a derivation primitive applied to Superiors and Subordinates. This procedure can be repeated for the building block attributes themselves, so that arbitrarily complex attribute derivations can be developed.

2.4.1.2 *Mappings.* Before discussing the member attribute derivation primitives, it is important to present the concept of *mapping*. A mapping is a concatenation of attribute names that allows a user to directly reference the value of an attribute of an attribute. A mapping is written, in general, as a sequence of attribute names separated by quotation marks. For example, consider the mapping "Captain.Name" for class SHIPS. The value of this mapping, for each member S of SHIPS, is the value of attribute Name of that member O of

OFFICERS that is the value of Captain for S. In this case, the attributes Captain of SHIPS and Name of OFFICERS are single valued; in general, this need not be the case. For example, consider the mapping for SHIPS "Engines. Serial_number." Attribute Engines is multivalued which means that "Engines. Serial_number" may also be multivalued. This mapping evaluates to the serial numbers of the engines of a ship. Similarly, the mapping for SHIPS "Captain.Superiors.Name" evaluates to the names of all of the superiors of the captain of a ship. This mapping is multivalued since at least one of the steps in the mapping involves a multivalued attribute. The value of a mapping "$X.Y.Z$," where X, Y, and Z are multivalued attributes, is the class containing each value of Z that corresponds to a value of Y for some value of X.

2.4.1.3 *Member Derivation Primitives.* The following primitives are provided to express the derivation of the value of a member attribute; here, attribute A_1 of member M_1 of class C_1 is being defined in terms of the relationship of M_1 to other information in the database.

(1) A_1 can be defined as an *ordering* attribute. In this case, the value of A_1 denotes the sequential position of M_1 in C_1 when C_1 is ordered by one or more other specified (single-valued) member attributes (or mappings) of C_1. Ordering is by increasing or decreasing value (the default is increasing). For example, the attribute Seniority of OFFICERS has the derivation "order by Date_commissioned." The OFFICER with the earliest date commissioned will then have Seniority value of 1. Ordering within groups is also possible: "order by A_2 within A_3" specifies that the value of A_1 is the sequential position of M_1 within the group of entities that have the same value of A_3 as M_1, as ordered by the value of A_2. (A_2 and A_3 may be mappings as well as attributes.) For example, attribute Order_for_tanker of INSPECTIONS has the derivation "order by decreasing Date within Tanker," which orders the inspections for each tanker. The value class of an ordering attribute is INTEGERS.

(2) The value of attribute A_1 can be declared to be a Boolean value that is "yes" (true) if M_1 is a member of some other specified class C_2, and "no" (false) otherwise. Thus, the value class of this *existence* attribute is YES/NO. For example, attribute Is_tanker_banned? of class OIL_TANKERS has the derivation "if in BANNED_SHIPS."

(3) The value of attribute A_1 can be defined as the result of combining all the entities obtained by recursively tracing the values of some attribute A_2. For instance, attribute Superiors of OFFICERS has the derivation "all levels of values of Commander"; the value of the attribute includes the immediate commander of the officer, his commander's superiors, and so on. Note that the value class of Commander is OFFICERS; this must be true for this kind of recursive attribute derivation to be meaningful. It is also possible to specify a maximum number of levels over which to repeat the recursion, namely, "up to N levels" where N is an integer constant; this would be useful, for example, to relate an officer to his subordinates and their subordinates.

(4) When a grouping class is defined, the derived multivalued member attribute *Contents* is automatically established. The value of this attribute is the

collection of members (of the class underlying the grouping) that form the contents of that member. For example, each member of the grouping class SHIP__TYPES has as the value of its Contents attribute the class of all ships of the type in question.

(5) The value of a member attribute can be specified to be derived from and equal to the value of some other attribute or mapping. For instance, attribute Date__last__examined of OIL__TANKERS has the derivation "same as Last__inspection.Date." (Note that this, in effect, introduces a member attribute as shorthand for a mapping.)

(6) Attribute A_1 can be defined as a *subvalue* attribute of some other (multivalued) member attribute or mapping (A_2). The value of A_2 is specified as consisting of a subclass of the value of A_1 that satisfies some specified predicate. For example, attribute Last__two__inspections of class OIL__TANKERS is defined as "subvalue of Inspections where Order__for__tanker ≤ 2."

(7) The value of a member attribute can be specified as the intersection, union, or difference of two other (multivalued) member attributes or mappings. For example, attribute Contacts of OFFICERS has the definition "where is in Superiors or is in Subordinates," indicating that its value consists of an officer's superiors and subordinates.

(8) A member attribute derivation can specify that the value of the attribute is given by an arithmetic expression that involves the values of other member attributes or mappings. The involved attributes/mappings must have numeric values, that is, they must have value classes that are (eventual) subclasses of NUMBERS. The arithmetic operators allowed are addition ("+"), subtraction ("−"), multiplication ("*"), division ("/"), and exponentiation ("!"). For example, attribute Top__speed__in__miles__per__hour of OIL__TANKERS has the derivation "= Absolute__top__speed/1.1" (to convert from knots).

(9) The operators "maximum," "minimum," "average," and "sum" can be applied to a member attribute or mapping that is multivalued; the value class of the attributes involved must be an (eventual) subclass of NUMBERS. The maximum, minimum, average, or sum is taken over the collection of entities that comprise the current value of the attribute or mapping.

(10) A member attribute can be defined to have its value equal to the number of members in a multivalued attribute or mapping. For example, attribute Number__of__instances of SHIP__TYPES has the derivation "number of members in Contents." "Number of unique members" is used similarly. "Number of members" and "number of unique members" differ only when duplicates are present in the multivalued attribute involved.

2.4.1.4 *The Definition of Member Attributes.* We now specify how these derivation mechanisms for derived attributes may be applied. The following rules are formulated in order to allow the use of derivations while avoiding the danger of inconsistent attribute specifications.

(1) Every attribute may or may not have an inverse; if it does, the inverse must be defined consistently with the attribute.

(2) Every member attribute A_1 satisfies one of the following cases.

(a) A_1 has exactly one derivation. In this case, the value A_1 is completely specified by the derivation. The inverse of A_1 (call it A_2), if it exists, may not have a derivation or a matching specification.
(b) A_1 has exactly one matching specification. In this case, the value of A_1 is completely specified by its relationships with an entity (or entities) to which it is matched (namely, member(s) of some class C). The inverse of A_1 (call it A_2), if it exists, may not have a derivation. It can have a matching specification, but this must match A_2 to C in a manner consistent with the matching specification of A_1.
(c) A_1 has neither a matching specification nor a derivation. In this case, it may be the case that the inverse of A_1 (call it A_2) has a matching specification or a derivation; if so, then one of the above two cases ((a) or (b)) applies. Otherwise, A_1 and A_2 form a pair of primitive values that are defined in terms of one another, but which are independent of all other information in the database.

With regard to updating the database, we note that in case (c), a user can explicitly provide a value for A_1 or for A_2 (and thereby establish values for both of them). In cases (a) and (b), neither A_1 nor A_2 can be directly modified; their values are changed by modifying other parts of the database.

2.4.2 *Class Attribute Interrelationships.* Attribute derivation primitives analogous to primitives (5)-(10) for member attributes can be used to define derived class attributes, as these primitives derive attribute values from those of other attributes. Of course, instead of deriving the value of a member attribute from the value of other member attributes, the class attribute primitives will derive the value of a class attribute from the value of other class attributes. In addition, there are two other primitives that can be used in the definition of derived class attributes.

(1) An attribute can be defined so that its value equals the number of members in the class it modifies. For example, attribute Number of INSPECTIONS has the derivation "number of members in this class."
(2) An attribute can be defined whose value is a function of a numeric member attribute of a class; the functions supported are "maximum," "minimum," "average," and "sum" taken over a member attribute. The computation of the function is made over the members of the class. For example, the class attribute Total_spilled of OIL_SPILLS has the derivation "sum of Amount_spilled over members of this class."

2.4.3 *Attribute Predicates for Subclass Definition.* As stated earlier, a subclass can be defined by means of a predicate on the member attributes of its parent class. Having described the specifics of attributes, it is now possible to detail the permissible types of attribute predicates. In particular, an attribute predicate is a simple predicate or a Boolean combination of simple predicates; the operators used to form such a Boolean combination are "and," "or," and "not." A simple predicate has one of the following forms:

(1) MAPPING SCALAR_COMPARATOR CONSTANT;
(2) MAPPING SCALAR_COMPARATOR MAPPING;
(3) MAPPING SET_COMPARATOR CONSTANT;

(4) MAPPING SET__COMPARATOR CLASS__NAME;
(5) MAPPING SET__COMPARATOR MAPPING.

Here, MAPPING is any mapping (including an attribute name as a special case); SCALAR__COMPARATOR is one of "=," "≠," ">," "≥," "<," and "≤"; CONSTANT is a string or number constant; SET__COMPARATOR is one of: "is contained in," "is properly contained in," "contains," and "properly contains"; CLASS__NAME is the name of some class defined in the schema. For illustration, an example of each of these five forms is provided below along with an indication of its meaning; the first two predicates define subclasses of class OFFICERS, while the third, fourth, and fifth apply to class SHIPS:

(1) Country__of__license = 'Panama' (officers licensed in Panama);
(2) Commander.Date__commissioned > Date__commissioned (officers commissioned before their commander);
(3) Cargo__types contains 'oil' (ships that can carry oil);
(4) Captain is contained in DANGEROUS__CAPTAINS (ships whose captain in the class containing officers that are bad risks);
(5) Captain.Country__of__license is contained in Captain.Superior. Country__of__license (ships commanded by an officer who has a superior licensed in the same country as he).

2.4.4 *Attribute Inheritance.* As noted earlier, it may often be the case that an entity in an SDM database belongs to more than one class. SDM classes can and frequently do share members, for example, a member of OIL__TANKERS is also a member of SHIPS; a member of OIL__SPILLS is also in INCIDENTS. As a member of a class C, a given entity E has values for each member attribute associated with C. But in addition, when viewed as a member C, E may have additional attributes that are not directly associated with C, but rather are *inherited* from other classes. For example, since all oil tankers are ships, each member T of the class OIL__TANKERS inherits the member attributes of SHIPS. In addition to the attributes Hull__type, Is__tanker__banned, Inspections, Number__of__times__inspected, Last__inspection, Last__two__inspections, Date__last__examined, and Oil__spills__involved__in, which are explicitly associated with OIL__TANKERS, T also has the attributes Name, Hull__number, Type, etc.; these are not mentioned in the definition of OIL__TANKERS but are inherited from SHIPS (a superclass of OIL__TANKERS). The value of each inherited attribute of tanker T is simply the value of that attribute of T when it is viewed as a member of SHIPS; the very same ship entity that belongs to OIL__TANKERS belongs also to SHIPS, so that the value of each such inherited attribute is well defined.

The following specific rules of attribute inheritance are applied in SDM.

(1) A class S that is an attribute-defined subclass of a class U, or a user-controllable subclass of U, inherits all of the member attributes of U. For example, since RURITANIAN__OIL__TANKERS is an attribute-defined subclass of OIL__TANKERS, RURITANIAN__OIL__TANKERS inherits all of the member attributes of OIL__TANKERS; in turn, members of OIL__TANKERS inherit all of the member attributes of SHIPS.

Class attributes describe properties of a class taken as a whole and so are

not inherited by an attribute-defined or user-controllable subclass. In order for an attribute to be inherited from class U by class S, both its meaning and its value must be the same for U and S. This is not true in general for class attributes. Although a subclass may have a similar class attribute to one defined for its parent class, for example, Number_of_members, their values will in general not be equal.

(2) A class S defined as an intersection subclass of classes U_1 and U_2 inherits all of the member attributes of U_1 and all of the member attributes of U_2. For example, the class BANNED_OIL_TANKERS, defined as containing all members of SHIP that are in both BANNED_SHIPS and OIL_TANKERS, inherits all attributes of BANNED_SHIPS as well as all of the attributes of OIL_TANKERS. This follows since each member of BANNED_OIL_TANKERS is both an oil tanker and a banned ship and so must have the attributes of both. Note that since BANNED_SHIPS and OIL_TANKERS are themselves defined as subclasses, they may inherit attributes from their parent classes which are in turn inherited by BANNED_OIL_TANKERS.

(3) A class S defined as the union of classes U_1 and U_2 inherits all of the member attributes shared by U_1 and U_2. For example, the class SHIPS_TO_BE_MONITORED inherits the member attributes shared by BANNED_SHIPS and OIL_TANKERS_REQUIRING_INSPECTION (which turn out to be all of the member attributes of SHIPS).

(4) A subclass S defined as the difference of classes, namely, consisting of all of the members in a class U that are not in class U_1, inherits all of the member attributes of U. This case is similar to (1), since S is a subclass of U.

These inheritance rules determine the attributes associated with classes that are defined in terms of interclass connections. These rules need not be explicitly applied by the SDM user; they are an integral part of SDM and are automatically applied wherever appropriate.

2.4.4.1 *Further Constraining an Inherited Member Attribute.* An important constraint may be placed on inherited attributes in an SDM schema. This constraint requires that the value of an attribute A inherited from class C_1 by class C_2 be a member of a class C_3 (C_3 is a subclass of the value class of A). To specify such a constraint, the name of the inherited attribute is repeated in the definition of the member attributes of the subclass, and its constrained value class is specified. For example, attribute Cargo_types is inherited by MERCHANT_SHIPS from SHIPS; its repetition in the definition of MERCHANT_SHIPS indicates that the value class of Cargo_types for MERCHANT_SHIPS is restricted to MERCHANT_CARGO_TYPE_NAMES. Values of attribute Cargo_types of SHIPS must satisfy this constraint. If the value being inherited does not satisfy this constraint, then the attribute's value is null.

2.5 Duplicates and Null Values

As specified above, an SDM class is either a set or a multiset: It may or may not contain duplicates. If a class has unique identifiers, then it obviously cannot have duplicates. If unique identifiers are not present, then the default is that duplicates

are allowed. However, a class can be explicitly defined with "duplicates not allowed." Duplicates may also be present in attribute values, since attribute derivation specifications and mappings can yield duplicates.

In point of fact, the existence or nonexistence of duplicates is only of importance when considering the number of members in a class or the size of a multivalued attribute.On most occasions, the user need not be concerned with whether or not duplicates are present. Consequently, the only SDM primitives that are affected by duplicates are those that concern the number of members in a class and the size of an attribute. The SDM interclass connections and attribute derivation primitives are defined so as to propagate duplicates in an intuitive manner. For example, attribute-defined and user-controllable subclasses contain duplicates if and only if their parent class contains duplicates; and, if the class underlying a grouping has duplicates, the contents of the groups will similarly contain duplicates. Further details of this approach to handling duplicates are provided in [27].

As stated above, any attribute not defined as "mandatory" may have "null" as its value. While the treatment of null values is not a simple issue, we state that for the purposes here null is treated just like any other data value. A detailed discussion of null value handling is beyond the scope of this paper (see [14] for such a discussion).

2.6 SDM Data Definition Language

As noted above, this paper provides a specific *database definition language* (*DDL*) for SDM. The foregoing description of SDM did not rely on a specific DDL syntax although the discussion proceeded through numerous examples expressed in a particular sample DDL syntax. Many forms of DDL syntax could be used to describe SDM schemas, and we have selected one of them in order to make the specification of SDM precise.

The syntax of SDM DDL is presented in Appendix B, expressed in Backus-Naur Form style. The particular conventions used are described at the beginning of Appendix B. For the most part, the syntax description is self-explanatory; however, the following points are worthy of note.

(1) Syntactic categories are capitalized (with no interspersed spaces, but possibly including "_"s). All lowercase strings are in the language itself, except those enclosed in "*"s; the latter are descriptions of syntactic categories whose details are obvious.

(2) Indentation is an essential part of the SDM DDL syntax. In Appendix B, the first level of indentation is used for presentation, while all others indicate indentation in the syntax itself. For example, MEMBER_ATTRIBUTES is defined as consisting of "member attributes," followed by a group of one or more member attribute items (placed vertically below "member attributes").

(3) Many rules that constrain the set of legal SDM schemata are not included in the syntax shown in the figure. For example, in SDM, the rule that attributes of different applicability (member attributes and class attributes) must not be mixed is not included in the syntax, as its incorporation therein would be too cumbersome. A similar statement can be made for the rules that arithmetic expressions must be computed on attributes whose values are numbers, that a common underlying class must exist for classes defined by multiset operator interclass connections, and so forth.

2.7 Operations on an SDM Database

An important part of any database model is the set of operations that can be performed on it. The operations defined for SDM allow a user to derive information from a database, to update a database (adding new information to it or correcting information in it), and to include new structural information in it (change an SDM schema) [27]. Note that operations to derive information from an SDM schema are closely related to SDM primitives for describing derived information (e.g., nonbase classes and derived attributes). There is a vocabulary of basic SDM operations that are application environment independent and predefined. The set of permissible operations is designed to permit only semantically meaningful manipulations of an SDM database. User-defined operations can be constructed using the primitives. A detailed specification of the SDM operations is beyond the scope of this paper.

3. DISCUSSION

In this paper, we have presented the major features of SDM, a high-level data modeling mechanism. The goal of SDM is to provide the designer and user of a database with a formalism whereby a substantial portion of the semantic structure of the application environment can be clearly and precisely expressed. Contemporary database models do not support such direct conceptual modeling, for a number of reasons that are summarized above and explored in greater detail in [22]. In brief, these conventional database models are too oriented toward computer data structures to allow for the natural expression of application semantics. SDM, on the other hand, is based on the high-level concepts of entities, attributes, and classes.

In several ways, SDM is analogous to a number of recent proposals in database modeling, including [1, 3, 5, 9, 14, 31, 33, 34, 39–41, 43, 46]. Where SDM principally differs from these is in the extent of the structure of the application domain that it can capture and in its emphasis on relativism, flexibility, and redundancy. An SDM schema does more than just describe the kinds of objects that are captured in the database; it allows for substantial amounts of structural information that specifies how the entities and their classes are related to one another. Furthermore, it is a fundamental premise of SDM that a semantic schema for a database should directly support multiple ways of viewing the same information, since different users inevitably will have differing slants on the database and even a single user's perspective will evolve over time. Consequently, redundant information (in the form of nonbase classes and derived attributes) plays an important role in an SDM schema, and provides the principal mechanism for expressing multiple versions of the same information.

3.1 The Design of SDM

In the design of SDM, we have sought to provide a higher level and richer modeling language than that of conventional database models, without developing a large and complex facility containing a great many features (as exemplified by some of the knowledge representation and world modeling systems developed by the artificial intelligence community, e.g., [35, 51]). We have sought neither absolute minimality, with a small number of mutually orthogonal constructs, nor

a profusion of special case facilities to precisely model each slightly different type of application. There is a significant trade-off between the complexity of a modeling facility and its power, naturalness, and precision. If a database model contains a large number of features, then it will likely be difficult to learn and to apply; however, it will have the potential of realizing schemata that are very sharp and precise models of their application domains. On the other hand, a model with a fairly minimal set of features will be easier to learn and employ, but a schema constructed with it will capture less of the particular characteristics of its application.

We have sought a middle road between these two extremes, with a relatively small number of basic features, augmented by a set of special features that are particularly useful in a large number of instances. We adhere to the principle of the well-known "80–20" rule; in this context, this rule would suggest that 80 percent of the modeling cases can be handled with 20 percent of the total number of special features that would be required by a fully detailed modeling formalism. Thus, a user of SDM should find that the application constructs that he most frequently encounters are directly provided by SDM, while he will have to represent the less common ones by means of more generic features. To this end, we have included such special facilities as the inverse and matching mechanisms for attribute derivation, but have not, for example, sought to taxonomize entity types more fully (since to do so in a meaningful and useful way would greatly expand the size and complexity of SDM). We have also avoided the introduction of a huge number of attribute derivation primitives, limiting ourselves to the ones that should be of most critical importance. For example, there does not exist a derivation primitive for class attributes to determine what percentage the members of the class constitute of another class. Such special cases would be most usefully handled by means of a general-purpose computational mechanism.

SDM as presented in this paper is neither complete nor final. SDM as a whole is open to any number of extensions. The most significant omission in this paper is that of the operations that can be applied to an SDM database: the database manipulation facility associated with the database definition facility presented here. Such a presentation would be too lengthy for this paper and can be found in [27]. In brief, however, the design of SDM is strongly based on the duality principle between schema and procedure, as developed in [21]. From this perspective, any query against the database can be seen as a reference to a particular virtual data item; whether that item can easily be accessed in the database, or whether it can only be located by means of the application of a number of database manipulation operations, depends on what information has been included in the schema by the database designer. Frequently retrieved data items would most likely be present in the schema, often as derived data, while less commonly requested information would have to be dynamically computed. In both cases, however, the same sets of primitives should be employed to describe the data item(s) in question, since dynamic data retrieval and static definitions of derived data are fundamentally equivalent, differing only in the occasions of their binding. Thus the SDM database manipulation facility strongly resembles the facilities described above for computing nonbase classes and derived attributes. Among other beneficial consequences, this duality allows for a natural evolution of the semantic schema to reflect changing patterns of use and access: As certain

kinds of requests become more common, they can be incorporated as derived data into the schema and thereby greatly simplify their retrieval.

3.2 Extensions

Numerous extensions can be made to SDM as presented here. These include extending SDM by means of additional general facilities, as well as tailoring special versions of it (by adding application environment specific facilities). For example, as it currently is defined, derived data is continuously updated so as always to be consistent with the primitive data from which it is computed. Alternative, less dynamic modes of computation could be provided, so that in some cases derived data might represent a snapshot of some other aspect of the database at a certain time. Similarly, a richer set of attribute inheritance rules, possibly under user control, might be provided to enable more complex relationships between classes and their subclasses. In the other direction, a current investigation is being conducted with the goal of simplifying SDM and accommodating more relativism [30]. Further, an attempt is currently under way to construct a version of SDM that contains primitives especially relevant to the office environment (such as documents, events, and organization hierarchies), to facilitate the natural modeling and description of office structures and procedures.

3.3 Applications

We envision a variety of potential uses and applications for SDM. As described in this paper, SDM is simply an abstract database modeling mechanism and language that is not dependent on any supporting computer system. One set of applications uses SDM in precisely this mode to support the process of defining and designing a database as well as in facilitating its subsequent evolution. It is well known that the process of logical database design, wherein the database administrator (DBA) must construct a schema using the database model of the database management system (DBMS) to be employed, is a difficult and error-prone procedure [10, 30, 31, 37, 38, 42, 44, 50]. A primary reason for this difficulty is the distance between the semantic level of the application and the data structures of the database model; the DBA must bridge this gap in a single step, simultaneously conducting an information requirements analysis and expressing the results of his analysis in terms of the database model. What is lacking is a formalism in which to express the information content of the database in a way that is independent of the details of the database model associated with the underlying DBMS. SDM can be used as a higher-level database model in which the DBA describes the database prior to designing a logical schema for it. There are a number of advantages to using the SDM in this way.

(1) An SDM schema will serve as a specification of the information that the database will contain. All too often, only the most vague and amorphous English language descriptions of a database exist prior to the database design process. A formal specification can more accurately, completely, and consistently communicate to the actual designer the prescribed contents of the database. SDM provides some structure for the logical database design process. The DBA can first seek to describe the database in high-level semantic terms, and then reduce that schema to a more conventional logical

design. By decomposing the design problem in this way, its difficulty as a whole can be reduced.

(2) SDM supports a basic methodology that can guide the DBA in the design process by providing him with a set of natural design templates. That is, the DBA can approach the application in question with the intent of identifying its classes, subclasses, and so on. Having done so, he can select representations for these constructs in a routine, if not algorithmic, fashion.

(3) SDM provides an effective base for accommodating the evolution of the content structure, and use of a database. Relativism, logical redundancy, and derived information support this natural evolution of schemata.

A related use of SDM is as a medium for documenting a database. One of the more serious problems facing a novice user of a large database is determining the information content of the database and locating in the schema the information of use to him. An SDM schema for a database can serve as a readable description of its contents, organized in terms that a user is likely to be able to comprehend and identify. A cross-index of the schema would amount to a semantic data dictionary, identifying the principal features of the application environment and cataloging their relationships. Such specifications and documentation would also be independent of the DBMS being employed to actually manage the data, and so could be of particular use in the context of DBMS selection or of a conversion from one DBMS to another. An example of the use of SDM for specification and documentation is [15].

On another plane are a number of applications that require that SDM schema for a database be processed and utilized by a computer system. One such application would be to employ SDM as the conceptual schema database model for a DBMS within the three-schema architecture of the ANSI/SPARC proposal [2]. In such a system, the conceptual schema is a representation of the fundamental semantics of the database. The external views of the data (those employed by programmers and end-users) are defined in terms of it, while a mapping from it to physical file structures establishes the database's internal schema (storage and representation). Because of its high level and support for multiple views, SDM could be effectively employed in this role. Once occupying such a central position in the DBMS, the SDM schema could also be used to support any number of "intelligent" database applications that depend on a rich understanding of the semantics of the data in question. For example, an SDM schema could drive an automatic semantic integrity checker, which would examine incoming data and test its plausibility and likelihood of error in the context of a semantic model of the database. A number of such systems have been proposed [16, 19, 20, 45], but they are generally based on the use of expressions in the first-order predicate calculus that are added to a relational schema. This approach introduces a number of problems, ranging from the efficiency of the checking to the modularity and reliability of the resulting model. By directly capturing the semantics in the schema rather than in some external mechanism, SDM might more directly support such data checking. Another "semantics-based" application to which SDM has been applied is an interactive system that assists a naive user, unfamiliar with the information content of the database, in formulating a query against it [28].

It might even be desirable to employ SDM as the database model in terms of which all database users see the database. This would entail building an SDM DBMS. Of course, a high-level database model raises serious problems of efficiency of representation and processing. However, it can also result in easier and more effective use of the data which may in the aggregate dominate the performance issues. Furthermore, SDM can be additionally extended to be more than just a database model; it can serve as the foundation for a total integrated database programming language in which both the facilities for accessing a database and those for computing with the data so accessed are combined in a coherent and consistent fashion [18]. And, SDM can provide a basis for describing and structuring logically decentralized and physically distributed database systems [22, 29].

APPENDIX A. AN SDM SCHEMA FOR THE TANKER MONITORING APPLICATION ENVIRONMENT

SHIPS
  description: all ships with potentially hazardous cargoes that
    may enter U.S. coastal waters
  member attributes:
    Name
      value class: SHIP__NAMES
    Hull__number
      value class: HULL__NUMBERS
      may not be null
      not changeable
    Type
      description: the kind of ship, for example, merchant or fishing
      value class: SHIP__TYPE__NAMES
    Country__of__registry
      value class: COUNTRIES
      inverse: Ships__registered__here
    Name__of__home__port
      value class: PORT__NAMES
    Cargo__types
      description: the type(s) of cargo the ship can carry
      value class: CARGO__TYPE__NAMES
      multivalued
    Captain
      description: the current captain of the ship
      value class: OFFICERS
      match: Officer of ASSIGNMENTS on Ship
    Engines
      value class: ENGINES
      multivalued with size between 0 and 10
      exhausts value class
      no overlap in values
    Incidents__involved__in
      value class: INCIDENTS
      inverse: Involved__ship
      multivalued
  identifiers:
    Name
    Hull__number

INSPECTIONS
  description: inspections of oil tankers
  member attributes:
    Tanker
      description: the tanker inspected
      value class: OIL__TANKERS
      inverse: Inspections
    Date
      value class: DATES
    Order__for__tanker
      description: the ordering of the inspections for a tanker
        with the most recent inspection having value 1
      value class: INTEGERS
      derivation: order by decreasing Date within Tanker
  class attributes:
    Number
      description: the number of inspections in the database
      value class: INTEGERS
      derivation: number of members in this class
  identifiers:
    Tanker + Date
COUNTRIES
  description: countries of registry for ships
  member attributes:
    Name
      value class: COUNTRY__NAMES
    Ships__registered__here
      value class: SHIPS
      inverse: Country__of__registry
      multivalued
  identifiers:
    Name
OFFICERS
  description: all certified officers of ships
  member attributes:
    Name
      value class: PERSON__NAMES
    Country__of__license
      value class: COUNTRIES
    Date__commissioned
      value class: DATES
    Seniority
      value class: INTEGERS
      derivation: order by Date__commissioned
    Commander
      description: the officer in direct command of this officer
      value class: OFFICERS
    Superiors
      value class: OFFICERS
      derivation: all levels of values of Commander
      inverse: Subordinates
      multivalued
    Subordinates
      value class: OFFICERS
      inverse: Superiors
      multivalued

```
    Contacts
      value class: OFFICERS
      derivation: where is in Superiors or is in Subordinates
  identifiers:
    Name
ENGINES
  description: ship engines
  member attributes:
    Serial__number
      value class: ENGINE__SERIAL__NUMBERS
    Kind__of__engine
      value class: ENGINE__TYPE__NAMES
  identifiers:
    Serial__number
INCIDENTS
  description: accidents involving ships
  member attributes:
    Involved__ship
      value class: SHIPS
      inverse: Incidents__involved__in
    Date
      value class: DATES
    Description
      description: textual explanation of the accident
      value class: INCIDENT__DESCRIPTIONS
    Involved__captain
      value class: OFFICERS
  identifiers:
    Involved__ship + Date + Description
ASSIGNMENTS
  description: assignments of captains to ships
  member attributes:
    Officer
      value class: OFFICERS
    Ship
      value class: SHIPS
  identifiers:
    Officer + Ship
OIL__TANKERS
  description: oil-carrying ships
  interclass connection: subclass of SHIPS where Cargo__types
    contains 'oil'
  member attributes:
    Hull__type
      description: specification of single or double hull
      value class: HULL__TYPE__NAMES
    Is__tanker__banned?
      value class: YES/NO
      derivation: if in BANNED__SHIPS
    Inspections
      value class: INSPECTIONS
      inverse: Tanker
      multivalued
    Number__of__times__inspected
      value class: INTEGERS
```

```
      derivation: number of unique members in Inspections
    Last__inspection
      value class: MOST__RECENT__INSPECTIONS
      inverse: Tanker
    Last__two__inspections
      value class: INSPECTIONS
      derivation: subvalue of inspections where
        Order__for__tanker ≤ 2
      multivalued
  Date__last__examined
    value class: DATES
    derivation: same as Last__inspection.Date
  Oil__spills__involved__in
    value class: INCIDENTS
    derivation: subvalue of Incidents__involved__in
      where is in OIL__SPILLS
    multivalued
class attributes:
  Absolute__top__legal__speed
    value class: KNOTS
  Top__legal__speed__in__miles__per__hour
    value class: MILES__PER__HOUR
    derivation: = Absolute__top__legal__speed/1.1
RURITANIAN__SHIPS
  interclass connection: subclass of SHIPS where
    Country.Name = 'Ruritania'
RURITANIAN__OIL__TANKERS
  interclass connection: subclass of OIL__TANKERS where
    Country.Name = 'Ruritania'
MERCHANT__SHIPS
  interclass connection: subclass of SHIPS where Type = 'merchant'
  member attributes:
    Cargo__types
      value class: MERCHANT__CARGO__TYPE__NAMES
OIL__SPILLS
  interclass connection: subclass of INCIDENTS where
    Description = 'oil spill'
  member attributes:
    Amount__spilled
      value class: GALLONS
    Severity
      derivation: = Amount__spilled/100,000
  class attributes:
    Total__spilled
      value class: GALLONS
      derivation: sum of Amount__spilled over members of this class
MOST__RECENT__INSPECTIONS
  interclass connection: subclass of INSPECTIONS where
    Order__for__tanker = 1
DANGEROUS__CAPTAINS
  description: captains who have been involved in an accident
  interclass connection: subclass of OFFICERS where is a value of Involved__captain of
    INCIDENTS
BANNED__SHIPS
  description: ships banned from U.S. coastal waters
```

interclass connection: subclass of SHIPS where specified
member attributes:
Date_banned
value class: DATES

OIL_TANKERS_REQUIRING_INSPECTION
interclass connection: subclass of OIL_TANKERS where specified

BANNED_OIL_TANKERS
interclass connection: subclass of SHIPS where
is in BANNED_SHIPS and is in OIL_TANKERS

SAFE_SHIPS
description: ships that are considered good risks
interclass connection: subclass of SHIPS where is not in BANNED_SHIPS

SHIPS_TO_BE_MONITORED
description: ships that are considered bad risks
interclass connection: subclass of SHIPS where is in BANNED_SHIPS
or is in OIL_TANKERS_REQUIRING_INSPECTION

SHIP_TYPES
description: types of ships
interclass connection: grouping of SHIPS on common value of Type
groups defined as classes are MERCHANT_SHIPS
member attributes:
Instances
description: the instances of the type of ship
value class: SHIPS
derivation: same as Contents
multivalued
Number_of_ships_of_this_type
value class: INTEGERS
derivation: number of members in Contents

CARGO_TYPE_GROUPS
interclass connection: grouping of SHIPS on common value of
Cargo_types

TYPES_OF_HAZARDOUS_SHIPS
interclass connection: grouping of SHIPS consisting of classes
BANNED_SHIPS, BANNED_OIL_TANKERS,
SHIPS_TO_BE_MONITORED

CONVOYS
interclass connection: grouping of SHIPS as specified
member attributes:
Oil_tanker_constituents
description: the oil tankers that are in the convoy (if any)
value class: SHIPS
derivation: subvalue of Contents where is in OIL_TANKERS
multivalued

CARGO_TYPE_NAMES
description: the types of cargo
interclass connection: subclass of STRINGS

MERCHANT_CARGO_TYPE_NAMES
interclass connection: subclass of CARGO_TYPE_NAMES
where specified

COUNTRY_NAMES
interclass connection: subclass of STRINGS where specified

ENGINE_SERIAL_NUMBERS
interclass connection: subclass of STRINGS where format is

"H"
number where integer and ≥1 and ≤999
"_"
number where integer and ≥0 and ≤999999

DATES
description: calendar dates in the range "1/1/75" to "12/31/79"
interclass connection: subclass of STRINGS where format is
month: number where ≥1 and ≤12
"/"
day: number where integer and ≥1 and ≤31
"/"
year: number where integer and ≥1970 and ≤2000
where (if (month = 4 or = 5 or = 9 or = 11) then day ≤30)
and (if month = 2 then day ≤29)
ordering by year, month, day

ENGINE_TYPE_NAMES
interclass connection: subclass of STRINGS where specified

GALLONS
interclass connection: subclass of STRINGS where format is
number where integer

HULL_NUMBERS
interclass connection: subclass of STRINGS where format is
number where integer

HULL_TYPE_NAMES
description: single or double
interclass connection: subclass of STRINGS where specified

INCIDENT_DESCRIPTIONS
description: textual description of an accident
interclass connection: subclass of STRINGS

KNOTS
interclass connection: subclass of STRINGS where format is
number where integer

MILES_PER_HOUR
interclass connection: subclass of STRINGS where format is
number where integer

PORT_NAMES
interclass connection: subclass of STRINGS

PERSON_NAMES
interclass connection: subclass of STRINGS

SHIP_NAMES
interclass connection: subclass of STRINGS

SHIP_TYPE_NAMES
description: the names of the ship types, for example, merchant
interclass connection: subclass of STRINGS where specified

APPENDIX B. SYNTAX OF THE SDM DATA DEFINITION LANGUAGE

The following list is given to clarify and define some of the items and terms used in this appendix.

(1) The left side of a production is separated from the right by a "←."
(2) The first level of indentation in the syntax description is used to help separate the left and right sides of a production; all other indentation is in the SDM data definition language.

(3) Syntactic categories are capitalized while all literals are in lowercase.
(4) { } means optional.
(5) [ ] means one of the enclosed choices must appear; choices are separated by a ";" (when used with "{ }" one of the choices may optionally appear).
(6) ⟨ ⟩ means one or more of the enclosed can appear, separated by spaces with optional commas and an optional "and" at the end.
(7) ⟨⟨ ⟩⟩ means one or more of the enclosed can appear, vertically appended.
(8) * * encloses a "meta"-description of a syntactic category (to informally explain it).

```
SCHEMA ←
  ⟨⟨CLASS⟩⟩;
CLASS ←
  ⟨CLASS_NAME⟩
    {description: CLASS_DESCRIPTION}
    {[BASE_CLASS_FEATURES; INTERCLASS_CONNECTION]}
    {MEMBER_ATTRIBUTES}
    {CLASS_ATTRIBUTES}
CLASS_NAME ←
  *string of capitals possibly including special characters*
CLASS_DESCRIPTION ←
  *string*
BASE_CLASS_FEATURES←
  {[duplicates allowed; duplicates not allowed]}
  (⟨⟨IDENTIFIERS⟩⟩)
IDENTIFIERS ←
  [ATTRIBUTE_NAME; ATTRIBUTE_NAME + IDENTIFIERS]
MEMBER_ATTRIBUTES ←
  member attributes:
    ⟨⟨MEMBER_ATTRIBUTE⟩⟩
CLASS_ATTRIBUTES ←
  class attributes:
    ⟨⟨CLASS_ATTRIBUTE⟩⟩
INTERCLASS CONNECTION ←
  [SUBCLASS; GROUPING_CLASS]
SUBCLASS ←
  subclass of CLASS_NAME where SUBCLASS_PREDICATE
GROUPING ←
  [grouping of CLASS_NAME on common value of ⟨ATTRIBUTE_NAME⟩
    {groups defined as classes are ⟨CLASS_NAME⟩};
  grouping of CLASS_NAME consisting of classes ⟨CLASS_NAME⟩;
  grouping of CLASS_NAME as specified]
SUBCLASS_PREDICATE ←
  [ATTRIBUTE_PREDICATE;
  specified;
  is in CLASS_NAME and is in CLASS_NAME;
  is not in CLASS_NAME;
  is in CLASS_NAME or is in CLASS_NAME;
  is a value of ATTRIBUTE_NAME of CLASS_NAME;
  format is FORMAT]
ATTRIBUTE_PREDICATE ←
  [SIMPLE_PREDICATE; (ATTRIBUTE_PREDICATE);
  not ATTRIBUTE_PREDICATE;
```

```
  ATTRIBUTE_PREDICATE and ATTRIBUTE_PREDICATE;
  ATTRIBUTE_PREDICATE or ATTRIBUTE_PREDICATE]
SIMPLE_PREDICATE ←
  [MAPPING SCALAR_COMPARATOR [CONSTANT; MAPPING];
  MAPPING SET_COMPARATOR [CONSTANT; CLASS_NAME; MAPPING]]
MAPPING ←
  [ATTRIBUTE_NAME; MAPPING.ATTRIBUTE_NAME]
SCALAR_COMPARATOR ←
  [EQUAL_COMPARATOR; >; ≥; <; ≤]
EQUAL COMPARATOR ←
  [−; ≠]
SET_COMPARATOR ←
  [is (properly) contained in; (properly) contains]
CONSTANT ←
  •a string or number constant•
FORMAT ←
  •a name class definition pattern•
  (see [26])
MEMBER_ATTRIBUTE ←
  ⟨ATTRIBUTE_NAME⟩
    {ATTRIBUTE_DESCRIPTION}
    value class:CLASS_NAME
    {inverse:ATTRIBUTE_NAME}
    {[match:ATTRIBUTE_NAME of CLASS_NAME on ATTRIBUTE_NAME];
    derivation:MEMBER_ATTRIBUTE_DERIVATION]}
    {single valued; multivalued (with size between CONSTANT and CONSTANT)}
    {may not be null}
    {not changeable}
    {exhausts value class}
    {no overlap in values}
CLASS ATTRIBUTE ←
  ⟨ATTRIBUTE_NAME⟩
    {ATTRIBUTE_DESCRIPTION}
    value class:CLASS_NAME
    derivation:CLASS_ATTRIBUTE_DERIVATION]}
    {single valued; multivalued (with size between CONSTANT and CONSTANT)}
    {may not be null}
    {not changeable}
ATTRIBUTE_NAME ←
  •string of lowercase letters beginning with a capital and possibly
  including special characters•
ATTRIBUTE DESCRIPTION ←
  "•string•"
MEMBER_ATTRIBUTE_DERIVATION ←
  [INTERATTRIBUTE_DERIVATION;
  MEMBER-SPECIFIC_DERIVATION]
CLASS_ATTRIBUTE_DERIVATION ←
  [INTERATTRIBUTE_DERIVATION;
  CLASS-SPECIFIC_DERIVATION]
INTERATTRIBUTE_DERIVATION ←
  [same as MAPPING;
  subvalue of MAPPING where [is in CLASS_NAME; ATTRIBUTE_PREDICATE];
  where [is in MAPPING and is in MAPPING; is in MAPPING or is in MAPPING;
  is in MAPPING and is not in MAPPING];
```

= MAPPING_EXPRESSION;
[maximum; minimum; average; sum] of MAPPING;
number of (unique) members in MAPPING]
MEMBER-SPECIFIC_DERIVATION ←
[order by [increasing; decreasing] (MAPPING)
(within (MAPPING));
if in CLASS_NAME;
[up to CONSTANT; all] levels of values of ATTRIBUTE_NAME;
contents]
CLASS-SPECIFIC_DERIVATION ←
[number of (unique) members in this class;
[maximum; minimum; average; sum] of ATTRIBUTE_NAME over
members of this class]
MAPPING EXPRESSION ←
[MAPPING; (MAPPING); MAPPING NUMBER_OPERATOR MAPPING]
NUMBER OPERATOR ←
[+; −; •; /; !]

ACKNOWLEDGMENTS

The authors wish to thank the following persons for their comments on the current or earlier versions of this paper, SDM, and related work: Antonio Albano, Arvola Chan, Peter Chen, Ted Codd, Dennis Heimbigner, Roger King, Peter Kreps, Frank Manola, Paula Newman, Diane and John Smith. In particular, Diane and John Smith helped the authors realize some of the weaknesses of an earlier version of SDM vis-a-vis relativism. Ted Codd's RM/T model has also provided many ideas concerning the specifics of SDM. DAPLEX (of Dave Shipman) and FQL (of Peter Buneman and Robert Frankel) have aided us in formulating various SDM constructs (e.g., mapping). Work performed at the Computer Corporation of America (under Frank Manola) and at the Lockheed California Company (under Don Kawamoto) has provided valuable input regarding the practical use of SDM. Finally, the referees provided many helpful comments concerning both the substance and presentation of this paper; their observations and suggestions are gratefully acknowledged.

Note: Because of the lack of neuter personal pronouns in English, the terms "he," "his," etc., are used throughout this paper to refer to an individual who may be either male or female.

REFERENCES

1. ABRIAL, J.R. Data semantics. In *Database Management*, J. Klimbie and K. Koffeman, Eds. North-Holland, Amsterdam, 1974.
2. ANSI/X3/SPARC (STANDARDS PLANNING AND REQUIREMENTS COMMITTEE). Interim report from the study group on database management systems. *FDT (Bulletin of ACM SIGMOD)* 7, 2 (1975).
3. BACHMAN, C.W. The role concept in data models. In *Proc. Int. Conf. Very Large Databases*, Tokyo, Japan, Oct. 1977.
4. BILLER, H., AND NEUHOLD, E.J. Semantics of databases: The semantics of data models. *Inf. Syst. 3* (1978), 11–30.
5. BUNEMAN, P., AND FRANKEL, R.E. FQL—A functional query language. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, Boston, Mass., 1979.

6. BUNEMAN, P., AND MORGAN, H.L. Implementing alerting techniques in database systems. In *Proc. COMPSAC'77*, Chicago, Ill., Nov. 1977.
7. CHAMBERLIN, D.D. Relational database management systems. *Comput. Surv. 8*, 1 (March 1976), 43–66.
8. CHANG, C.L. A hyper-relational model of databases. IBM Res. Rep. RJ1634, IBM, San Jose, Calif., Aug. 1975.
9. CHEN, P.P.S. The entity-relationship model: Toward a unified view of data. *ACM Trans. Database Syst. 1*, 1 (March 1976), 9–36.
10. CHEN, P.P.S. The entity-relationship approach to logical database design. Mono. 6, QED Information Sciences, Wellesley, Mass., 1978.
11. CODASYL COMMITTEE ON DATA SYSTEM LANGUAGES. Codasyl database task group report. ACM, New York, 1971.
12. CODD, E.F. A relational model of data for large shared data banks. *Commun. ACM 13*, 6 (June 1970), 377–387.
13. CODD, E.F. Further normalization of the database relational model. In *Database Systems*, Courant Computer Science Symposia 6, R. Rustin, Ed. Prentice-Hall, Englewood Cliffs, N.J., 1971, pp. 65–98.
14. CODD, E.F. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 397–434.
15. COMPUTER CORPORATION OF AMERICA. DBMS—Independent CICIS specifications. Tech. Rep. CCA, Cambridge, Mass., 1979.
16. ESWARAN, K.P., AND CHAMBERLIN, D.D. Functional specifications of a subsystem for database integrity. In *Proc. Int. Conf. Very Large Databases*, Framingham, Mass., Sept. 1975.
17. HAMMER, M. Research directions in database management. In *Research Directions in Software Technology*, P. Wegner, Ed. The M.I.T. Press, Cambridge, Mass., 1979.
18. HAMMER, M., AND BERKOWITZ, B. DIAL: A programming language for data-intensive applications. Working Paper, M.I.T. Lab. Computer Science, Cambridge, Mass., 1980.
19. HAMMER, M., AND MCLEOD, D. Semantic integrity in a relational database system. In *Proc. Int. Conf. Very Large Databases*, Framingham, Mass., Sept. 1975.
20. HAMMER, M., AND MCLEOD, D. A framework for database semantic integrity. In *Proc. 2nd Int. Conf. Software Engineering*, San Francisco, Calif., Oct. 1976.
21. HAMMER, M., AND MCLEOD, D. The semantic data model: A modelling mechanism for database applications. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, Austin, Tex., 1978.
22. HAMMER, M., AND MCLEOD, D. On the architecture of database management systems. In *Infotech State-of-the-Art Report on Data Design*. Pergamon Infotech Ltd., Berkshire, England, 1980.
23. KENT, W. *Data and Reality*. North-Holland, Amsterdam, 1978.
24. KENT, W. Limitations of record-based information models. *ACM Trans. Database Syst. 4*, 1 (March 1979), 107–131.
25. LEE, R.M., AND GERRITSEN, R. Extended semantics for generalization hierarchies. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, Austin, Tex., 1978.
26. MCLEOD, D. High level definition of abstract domains in a relational database system. *J. Comput. Languages 2*, 3 (1977).
27. MCLEOD, D. A semantic database model and its associatetd structured user interface. Tech. Rep., M.I.T. Lab. Computer Science, Cambridge, Mass., 1978.
28. MCLEOD, D. A database transaction specification methodology for end-users. Tech. Rep., Computer Science Dep., Univ. Southern California, Los, Angeles, Calif., 1980.
29. MCLEOD, D., AND HEIMBIGNER, D. A federated architecture for database systems. In *Proc. Nat. Computer Conf.*, Anaheim, Calif., 1980.
30. MCLEOD, D., AND KING, R. Applying a semantic database model. In *Proc. Int. Conf. Entity-Relationship Approach to Systems Analysis and Design*, Los Angeles, Calif., Dec. 1979.
31. MYLOPOULOS, J., BERNSTEIN, P.A., AND WONG, H. K. T. A language facility for designing interactive database-intensive applications. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, Austin, Tex., 1978.
32. PALMER, I. Record subtype facilities in database systems. In *Proc. 4th Int. Conf. Very Large Databases*, Berlin, West Germany, Sept. 1978.

33. PIROTTE, A. The entity-property-association model: An information-oriented database model. Tech. Rep., M.B.L.E. Res. Lab., Brussels, Belgium, 1977.
34. ROUSSOPOULOS, N. Algebraic data definition. In *Proc. 6th Texas Conf. Computing Systems*, Austin, Tex., Nov. 1977.
35. SCHANK, R.C. Identification of conceptualizations underlying natural language. In *Computer Models of Thought and Language*, R.C. Schank and K.M. Colby, Eds. W.H. Freeman, San Francisco, Calif., 1973.
36. SCHMID, H.A., AND SWENSON, J.R. On the semantics of the relational data model. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, San Jose, Calif., 1975.
37. SENKO, M.E. Information systems: Records, relations, sets, entities, and things. *Inf. Syst. 1*, 1 (1975), 3–14.
38. SENKO, M.E. Conceptual schemas, abstract data structures, enterprise descriptions. In *Proc. ACM Int. Computing Symp.*, Belgium, April 1977.
39. SHIPMAN, D. W. The functional data model and the data language DAPLEX. *ACM Trans. Database Syst. 6*, 1 (March 1981), 140–173.
40. SMITH, J.M., AND SMITH, D.C.P. Database abstractions: Aggregation. *Commun. ACM 20*, 6 (June 1977), 405–413.
41. SMITH, J.M., AND SMITH, D.C.P. Database abstractions: Aggregation and generalization. *ACM Trans. Database Syst. 2*, 2 (June 1977), 105–133.
42. SMITH, J.M., AND SMITH, D.C.P. Principles of conceptual database design. In *Proc. NYU Symp. Database Design*, New York, May 1978.
43. SMITH, J.M., AND SMITH, D.C.P. A database approach to software specification. Tech. Rep. CCA-79-17, Computer Corporation of America, Cambridge, Mass., April 1979.
44. SOLVBERG, A. A contribution to the definition of concepts for expressing users' information system requirements. In *Proc. Int. Conf. Entity-Relationship Approach to Systems Analysis and Design*, Los Angeles, Calif., Dec. 1979.
45. STONEBRAKER, M.R. High level integrity assurance in relational database management systems. Electronics Res. Lab. Rep. ERL-M473, Univ. California, Berkeley, Calif., Aug. 1974.
46. SU, S.Y.W., AND LO, D.H. A semantic association model for conceptual database design. In *Proc. Int. Conf. Entity-Relationship Approach to Systems Analysis and Design*, Los Angeles, Calif., Dec. 1979.
47. TAYLOR, R.W., AND FRANK, R.L. CODASYL database management systems. *Comput. Surv. 8*, 1 (March 1976), 67–104.
48. TSICHRITZIS, D.C., AND LOCHOVSKY, F.H. Hierarchical database management: A survey. *Comput. Surv. 8*, 1 (March 1976), 105–124.
49. WIEDERHOLD, G. *Database Design*. McGraw-Hill, New York, 1977.
50. WIEDERHOLD, G., AND EL-MASRI, R. Structural model for database design. In *Proc. Int. Conf. Entity-Relationship Approach to Systems Analysis and Design*, Los Angeles, Calif., Dec. 1979.
51. WONG, H.K.T., AND MYLOPOULOS, J. Two views of data semantics: A survey of data models in artificial intelligence and database management. *INFOR 15*, 3 (Oct. 1977), 344–382.

Received February 1980; revised July 1980; accepted August 1980

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

DISEÑO DE BASES DE DATOS

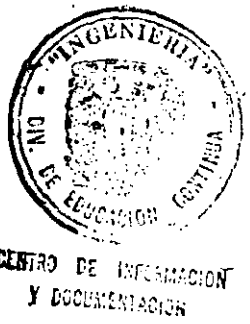
LIMITATIONS OF RECORD-BASED INFORMATION MODELS

WILLIAM KENT

MAYO, 1985

Limitations of Record-Based Information Models

WILLIAM KENT
IBM Corporation



Record structures are generally efficient, familiar, and easy to use for most current data processing applications. But they are not complete in their ability to represent information, nor are they fully self-describing.

Key Words and Phrases: records, information model, data model, semantic model, conceptual model, normalization, first normal form, entities, relationships
CR Categories: 3.70, 4.33, 4.34

INTRODUCTION

Records provide an excellent tool for processing information that fits a certain pattern. Other kinds of information do not fit as well into record structures. In all cases, the use of record structures depends on supplementary information, often reflected only in the special-purpose application programs written to process the data, and which may or may not still be remembered by the users of the data. Record structures do not provide the semantically self-describing base needed for conceptual schemas [2, 3, 21], or for generalized query processors or other end-user facilities.

In their capacity as data processing *tools*, records have a desirable versatility. That is, a given construct (e.g., field names, or compound fields) can be used for different purposes at different times. Unfortunately this virtue becomes a vice for semantic modeling: one has to know the special usage of each construct in each case, and there is no general rule for deducing the underlying semantic structure. While some information cannot be represented in records, other information can be represented in so many ways as to become ambiguous.

Models which provide additional file structure around the records (e.g., sequencing, hierarchies, CODASYL networks) overcome some of the functional limitations. None of them overcome all the limitations. Furthermore, by building on top of record structures, they retain all the underlying ambiguities. In some cases, they simply add more options for representing something which could already be represented in several ways in record structure.

This paper would be pointless if there were not any alternatives, but there are.

Author's address: General Products Division, IBM Corporation, 555 Bailey Avenue, P.O. Box 50020, San Jose, CA 95150.

There are various models which are essentially graph structured, based on such primitive concepts as binary relations, or entities and relationships. Such models tend to be more functionally complete in their information processing capability, and more precise in their semantic modeling. (We are not arguing their economic superiority to record structures for bulk data processing in today's applications.) A number of these are listed in the concluding section of this paper. We do not try to explain or defend these models. Our main purpose is to collect some problems concerning record structures (many of which have already been mentioned in the literature), as a resource to help defend alternative models. Nevertheless, much of the discussion is cast in terms of entities and relationships. The motivation for this paper originated in attempts to reconcile record structures with the characteristics of entities and relationships.

"RECORD" DEFINED

By *record* we mean here a fixed sequence of field values, conforming to a static description usually contained in catalogs and/or in programs. The description consists mainly of a name, length, and data type for each field. Each such description defines one *record type* (or, in relational terms, a "relation" or "table"). My remarks apply to any data model based on this kind of construct. This clearly includes the traditional hierarchical, relational, and (CODASYL) network models. (For introductions to these models, see [7, 9–13, 42, 43].) It also includes approaches mentioned in [2] and [18], to the extent that they speak of such things as "conceptual records." (It should be noted that such constructs are not mentioned in [3].) The comments also apply to the entity-relationship model of [8], which is really driven by record structures (relations) rather than by entities and relationships as the primitive concept (his Level 1 constructs appear to be constrained to match his Level 2 information structure).

Some record formats allow a certain variability by permitting a named field or group of fields to occur more than once within a record (i.e., as a list of values or sets of values). We will use the term *normalized system* to refer to systems which do not permit repeating groups or fields. This follows the relational model, which excludes such repetitions via its normalization requirements (specifically, first normal form [9, 20]).

BASIC ASSUMPTIONS BEHIND RECORD STRUCTURES

Record structure presumes a horizontal and vertical homogeneity in data: horizontally, each record of a given type contains the same fields; and vertically, a given field contains the same "kind" of information in each record.

Homogeneity of Relevant Facts

The records of a given type in a file describe a set of things in the real world (e.g., employees). Record structure fits best when the entire population has the same kinds of attributes (e.g., every employee has a name, address, department, salary, etc.). While exceptions are tolerated, the essential configuration is that of a homogeneous population of records, all having the same fields.

Although commercial data processing naturally focuses on areas which fit this

pattern, the pattern does not always hold. In many cases, although a certain group of individuals constitutes a single "kind" of thing, there is considerable variation in the facts relevant to each individual in that set.

Consider clothing. While we can agree that pants, socks, underwear, and hats are all items of clothing, it would be very hard to define a conventional "clothing" record type. There are many field names which are relevant; not many of them apply to any one kind of clothing. Consider: size, waist size, neck size, sleeve length, long or short sleeves, cup size, inseam length, button or zipper, sex, fabric type, heel size, width, color, pattern, pieces, season, number, collar style, cuffs, neckline, sleeve style, weight, flared, belt, waterproof, formal or casual, age, pockets, sport, washable, etc. How would you design the record format for clothing records?

Clothing is by no means the only such category. Tools, furniture, vehicles, and people are just a few other categories having inhomogeneous attributes over their populations.

The more that information deviates from the norm of homogeneity, the less appropriate is the record configuration. There are certain techniques for accommodating variability among instances in a record structure, but these need to be used sparingly. If there is considerable variation over a population, then the solutions become cumbersome and inefficient. Such solutions include:

(1) Define the record format to include the union of all relevant fields, where not all the fields are expected to have values in every record. (Often "maiden name" is defined as part of a record format for all employees, though it is only relevant to married females.) Thus many records might have null values in many fields. Furthermore, the limited relevance is not defined to the system; it is only the pattern of usage (and, sometimes, validation logic in programs) which reflects the limitation.

(2) Allow the same field to have different meanings in different records. Unfortunately, such a practice is never defined to the system. With respect to any processing done by the system, that field appears to have the same significance in every record occurrence. It certainly has only one field name, which in these cases usually turns out to be something totally uninformative, like CODE or FIELD 1. It is only the buried logic in application programs which knows the significance of these fields, and the different meanings they have in different records.

Such inhomogeneity is especially vexing if it affects an attribute we would like to use as an identifier. If it does not apply to all individuals in the set, then it cannot be used as a "key" for the record type. This situation is fairly common. The employees of a multinational corporation might not all have social security numbers, or employee numbers. Some books do not have "International Standard Book Numbers" (ISBN), others do not have Library of Congress numbers, and some have neither. Library of Congress numbers are also given to things which are not books (e.g., films and recordings); those would not have ISBN's. Oil companies have their individual conventions for naming their own oil wells, and the American Petroleum Institute has also assigned "standard" names to some wells—but not all.

The function needed here is equivalent to "self-naming" fields, i.e., redefining the concept of record to mean a chain of relevant field names and their values.

Homogeneity Within Fact Type

That was a kind of "horizontal" homogeneity, i.e., each record containing the same fields. There is also a "vertical" homogeneity assumed. Within a record type, a given field is expected to contain the same kind of value in every record, as though a certain kind of fact always involved the same kinds of objects.

Again, this is not always true. Suppose that company cars can be assigned either to employees or to departments. If employees are normally identified by six numeric digits and departments by four alphanumeric characters, how do we design a "vehicle assignment" record? Assignment is a simple fact, to which one might naively expect to be able to address a simple inquiry: "to whom is car 97 assigned?" In such a case, we might like a two-part answer: the type of the assignee (employee or department), plus the identification of the individual assignee (in a format which depended on the assignee type).

We could design a record format with four fields: vehicle number, assignee type, employee, and department. The second field would tell us the assignee type, and hence whether to look for the assignee name in the third or fourth field. We assume, of course, that only one of the last two fields is filled in—but there is not likely to be any system facility to enforce that. Thus the multifield format introduces a data integrity hazard. And, as far as the record definitions convey any meaning to the system, we have here three independent facts about vehicles, with no interdependence among the three. The data structure bears little resemblance to the semantic structure of the underlying relationships. (We have, incidentally, created a horizontal inhomogeneity. The employee field is relevant for some vehicles; and the department field for others. They are never both relevant for the same vehicle.)

If, later on, cars can be assigned to other kinds of things with different identifier formats (e.g., to divisions, or to branch offices) then the record formats have to be redesigned with additional fields for the new assignee types. The validation gets more complicated: only one of the last n fields may contain a value. And the file may have to be physically reloaded for each format change.

Another approach is to provide distinct record types, one for each type of assignee. The fields in one record type would be vehicle number and employee, in another they would be vehicle number and department, and so on. Each record type has its own name; instead of "vehicle assignment" being a single kind of fact, we have many kinds: "vehicle-employee assignments," "vehicle-department assignments," etc. (One is required to believe that employees and departments cannot have the same relationships with vehicles. A relationship between an employee and a vehicle is necessarily "different" from a relationship between a department and a vehicle.) Instead of going to one record type (or naming one relationship) to find the assignment of a vehicle, one now has to know how many such record types there are—and their names—and be prepared to interrogate each one of them. It is even worse if you are interested in some other information about the vehicle, not its assignment. That information might be in any one of the record types. Validation is still a problem: there is no system facility to keep

the same vehicle from appearing in more than one record type (i.e., having more than one assignee). Extensions are difficult, too: every new assignee type requires the introduction of another record type. And changing a vehicle's assignment is cumbersome, if the assignee type is also changing; a record of one type has to be deleted and a record of another type inserted.

Neither multifield records nor multiple record types offer a good solution to the vertical inhomogeneity problem. These approaches look even worse if there is inhomogeneity on both sides of the relationship. Suppose that instead of vehicle assignments, we were recording more general equipment assignments. Assignable equipment might include vehicles, furniture, tools, etc., each potentially having its own identifier formats. If there are m kinds of assignable equipment and n kinds of assignees, then the multifield approach requires $m + n + 2$ fields (two type fields), with only four fields containing values in any one record. The multiple record type approach would require $m \times n$ record types.

Still another solution to vertical inhomogeneity is to make it disappear. One way is to relax the field definition to a level that is general enough to handle all necessary identifier formats; a varying character string would do the job. Vehicle assignment records are then reduced to two fields again, where the second field might contain employee numbers, department codes, or the identifiers for any other kinds of assignees. Only users (and the code in their programs) know which is which, and what to do with them. The system is unable to furnish services such as syntax checking of the field values, or following a path to the corresponding employee or department record (by matching key values, as in the relational join), or verifying that a referenced employee or department does, in fact, exist.

This solution also fails if things in different categories might accidentally have the same identifiers (e.g., in the general equipment assignments, if a vehicle registration number might happen to be the same as a tool inventory number). Which points out that the vertical inhomogeneity problem is not simply a record format problem. Even if the record formats are compatible (e.g., employees and departments both having four-character codes), one has to guard against different entity types occurring in the same field if they might have the same identifiers.

But on the other hand, we do not have to have multiple entity types to encounter vertical inhomogeneity. Identifier formats can vary even within a single entity type. Employee numbers might differ in various subsidiaries of a corporation, or within a multinational corporation. Ship registry formats differ according to the country of registry. Oil companies have different formats for identifying their oil wells. The soldiers in a United Nations military group are likely to have different kinds of serial numbers. And so on.

There is still another way to force the disappearance of vertical inhomogeneity. One can provide a uniform reference to all the entities involved by aggregating them into one "supertype" and giving them a new arbitrary identifier, e.g., an "assignee number." (Analogous to such familiar constructs as "taxpayer identification number," or "capital equipment inventory number.") This permits assignees to be referenced uniquely and carefully in the assignment records, with a well-defined and checkable identifier format. Unfortunately, all of the entities involved have now acquired a new and additional identifier for which values have to be assigned, and by which they have to be recognized in various contexts. And, in

some cases a readable name (perhaps the department name) is replaced by an unintelligible code number, which has to be looked up somewhere else.

What could such an identifier represent? In a record structure, it should be the key of a record type, i.e., an "assignee" record type. But if we also wanted to have employees and departments represented in distinct record types, we have a conflict. The type of a record is either "department" or "assignee"; it cannot be both.

Still another drawback of this "supertype" approach is that it has to be reapplied for each different kind of multitype fact (i.e., potentially for each case of vertical inhomogeneity). Entities get aggregated one way for equipment assignment records, another way to keep track of who manufactures what, another way for who owns what, another way as "employers" (which might be people, companies, schools, government agencies, foreign organizations, etc.), perhaps still another way as "taxpayers," and so on. Each of these is potential grounds for another supertype, with its own identifier scheme. This touches on the problem of multiple types for an individual, which we will get to later; the immediate concern is that an individual might become attached to a great many serial numbers, potentially one for each aggregation to which it belonged.

Vertical inhomogeneity can introduce still another record formatting problem. The identifiers involved may differ in more than just length and character set (e.g., numeric vs. alphabetic). There may be differences in "structure," e.g., if qualified naming is involved. Consider a company in which the name of a department is unique within its division, but not necessarily within the company as a whole. Then corporate records would have to refer to a department using two fields: a department name plus a division name (serving as a qualifier for the department). At this point, it is not at all clear how many fields there are in a corporate vehicle assignment record. If assigned to an employee, there are two: vehicle number and employee number. When the assignee is a department, there are three fields: vehicle number, department name, and division name. (If we had to describe this relationship in terms of the relational model, we might have to call it a relation of degree two and a half, on the average.)

To net it all out, the record structure is not well suited to information exhibiting "vertical inhomogeneity." The function required here is equivalent to allowing a single fact (field) to include both a type and value, where the syntax and structure of the value depended on the type.

PRESUMPTIONS UNDERLYING TRADITIONAL IMPLEMENTATIONS

Although not intrinsic to the record structure, a number of features characterize most traditional implementations of record processing systems. These include such things as the separation of descriptions and data, minimal requirements for descriptions, and resistance to changing descriptions.

Descriptions Are Not Information

Information is obtained from a record structure by extracting the values of fields, and it is only field values which supply information. One can answer the question "who manages the Accounting department?" by finding a certain field which contains the manager's name. But it is not likely that the file can provide an

answer to "how is Henry Jones related to the Accounting department?" There are no fields in the file containing such entries as "is assigned to," "was assigned to," "on loan to," "manages," "audits," "handles personnel matters for," etc. Depending on how the records are organized, the answer generally consists of a field name or a record type name, which are not contained in the records. To a naive seeker of information from the database (e.g., via a high-level query interface), it is not at all obvious why one question may be asked and the other may not.

It is not just that he cannot get an answer; the interfaces do not provide any way to frame the question. The data management systems do not provide a way to ask such questions whose answers are field names or record type names.

Then consider the following questions:

(1) How many employees are there in the Accounting department?
(2) What is the average number of employees per department?
(3) What is the maximum number of employees currently in any department?
(4) What is the maximum number of employees permitted in any department?
(5) How many more employees can be hired into the Accounting department?

If the maximum number of employees permitted is fixed by corporate policy, then a system offering advanced validation capabilities is likely to place that number into a constraint in a database description, outside the database itself. Our naive seeker of facts will then again find himself unable to ask the last two questions. He might well observe that other things having the effect of rules or constraints are accessible from the database, such as sales quotas, departmental budgets, head counts, safety standards, etc. The only difference, which does not matter much to him, is that some such limits are intended to be enforced by the system, while others are not. It is not at all obvious to him why he can ask some questions and not others.

This suggests that we should represent such descriptions and constraints in the same format—and in the same database—as "ordinary" information, but with the added characteristic that they are intended to be executed and enforced by the data processing system.

There is, of course, an inherent difference between descriptions and other data, with respect to update characteristics. Changes to descriptions imply differences in the system's behavior, ranging from changes in validation procedures to physical file reorganizations implied by format changes. Thus the system has to be aware of, and control, changes to descriptions. But such descriptions need not be inherently different for retrieval purposes. And even with respect to update, the method need not be inherently different as perceived by users. It is only necessary that the authorization to do so be carefully controlled, and that the consequences be propagated into the system. There is already a precedent for such update controls: many implementations forbid the modification of key fields of records.

Some Descriptions Are Not Needed

While such things as field names and record types can be factored out of the data [32], they do not always wind up in the catalogs (record descriptions) either. Sometimes they do not appear anywhere in the data management system at all.

Catalogs are maintained primarily for the benefit of the system, not for users, and tend to contain only such information as is needed for the performance of system services. Quite often, only key fields are described, for which the system may provide such services as indexing, ordering, and uniqueness checking. Other fields might only be described in the declarations local to the various application programs, with no assurance that such descriptions are consistent with each other.

One of the major contributions of the relational model is to treat all fields in a record as constructs requiring description to the data management system.

Field Names Are Only Place Holders

When provided at all, field names are used by record management systems only to designate some space within a record. This suffices for the system to provide its services, such as matching keys or sequencing. And, certainly, one name is adequate for this purpose. But for information modeling, we may want to attach several labels to a field, indicating perhaps the kind of entity which may occur there (e.g., "date") and its relationship to the subject of the record— its reason for occurring in the record (e.g., "termination").

In practice, there has been no discipline in the usage of field names. Sometimes they name the entity type, sometimes the relationship, sometimes a hybrid of the two ("termination-date"), sometimes an identifier type ("social security number"), and sometimes nothing intelligible ("code 1," "field x"). And even when they do name entity types or relationships, field names are just mnemonic aids to human users, rather than anything which can be used by a system service to establish semantic connections. If the field name specifies the entity type, it is not likely to be the same as the corresponding record type name (while a record type might be named "dept-rec," the corresponding field in an employee record might be named "deptno"). The same entity type might be spelled differently in different field names ("dept," "deptnum," "deptno," etc.). And nothing prevents the same field name from meaning entirely different things in different records.

The relational model does improve on this situation by providing for both "selector" (column) names and "domain" names, but there is still relatively little discipline imposed. Domain names often specify identifier types rather than entity types [19]. Thus "social security number" and "employee number" are likely to be specified as the domains of two fields, giving no clue that the same entity might be named in both places. Even if the domain name identified an entity type, it might or might not be the same name as the record type representing those entities. And domain names give no clue when one entity type is a subset of another; unequal domains appear to be disjoint.

Some implementations of the relational model do not incorporate the domain construct at all.

Stability of Relevant Facts

Another implication of record formats, and of the file plus catalog configuration, is that the kinds of facts relevant to an entity are predefined and are expected to remain quite stable. It generally takes a major effort to add fields to records.

While this may be acceptable and desirable in many cases, there are situations

where all sorts of unanticipated information needs to be recorded, and a more flexible data structure is needed.

The need to record information of unanticipated meaning or format is crudely reflected in provisions for "comments" fields or records. These consist of unformatted text in which system facilities can do little more than search for occurrences of words. Thus ironically, we have the two extremes of rigidly structured and totally unstructured information—but very little in between.

CORRELATION WITH INFORMATION CONCEPTS

For information modeling purposes, one has to account for such concepts as entities and entity types, relationships and attributes, and naming. These are discussed in the following sections.

ENTITIES AND TYPES

There is a natural inclination to identify entities with records, since a record has the sense of being an integral object. It is an elementary unit of creation and destruction, as well as of data transmission, and records are classified into types just as entities are. Such a correspondence between entities and records would be enormously simplifying, giving us information modeling as a free by-product of data management technology.

Arguments against such a modeling approach are hampered by the lack of a good operational definition of the term "entity." But we can suggest some difficulties in reconciling record structures with certain "intuitively obvious" characteristics of entities. Thereafter we can either conclude that records have limited value for information modeling, or else adjust our intuitions about entities in order to get a better fit with record concepts.

The questions which might be asked to test the hypothesis that records represent entities are: How well do their characteristics match? Is there a 1:1 correspondence between them?

A Record Does Not Have All the Facts

Many facts have the form of a relationship between two entities (e.g., a department and an employee). Although it concerns both entities, such a fact is not likely to be replicated in the records representing both entities. At most, it will usually be included in the record of only one of the entities involved.

Quite often the fact will not occur in the record corresponding to either one. If the fact is a many-to-many relationship, such as employees and their skills, then normalized record systems do not permit the necessary repeating field to occur in either the employee record or the skill record. Normalized systems constrain a record to be a collection of single-valued facts. If a class has one instructor, then that can be mentioned in the class record, but not if there might be several instructors.

Thus a record cannot be characterized as containing "all the facts" about an entity.

Entities Are Not Always Single-Typed

If we intend to use a record to represent a real world entity, there is some difficulty in equating record types with entity types. It seems reasonable to view

a certain person as a single entity (for whom we might wish to have a single record in an integrated database). But such an entity might be an instance of several entity types, such as person, employee, dependent, customer, stockholder, taxpayer, parent, instructor, student, mammal, physical object, property (slaves?), etc. it is difficult, within the current record processing technologies, to define a record type corresponding to each of these, and then permit a single record to simultaneously be an occurrence of several of the record types.

Note that we are not dealing with a simple nesting of types and subtypes: all employees are people, but some customers and stockholders are not. Nor are subtypes mutually exclusive: some people are employees, some are stockholders, and some are both.

In order to fit comfortably into a record-based discipline, we need to perceive entity types as though they did not overlap. We should perhaps think of customers and employees as always distinct entities, sometimes related by an "is the same person" relationship. But we then have to make arbitrary decisions about the placement of common information such as addresses and birthdates. Furthermore, one has to be very careful about the number of entities being modeled. If an employee is a stockholder, there will be two records for him; is he two entities? If a committee has five employees and five stockholders, how big is the committee?

(Bachman and Daya [4] and Smith and Smith [40] propose models in which multiple records can represent the multiple types, or roles, of an entity.)

Type Is Not Always Homogeneous

Even within a single type, there may be facts and naming conventions which are relevant to some occurrences and not others. These points were covered in the earlier discussion of horizontal homogeneity.

Entities Without Records

Most of the things mentioned in a database do not have any distinct records to represent them. These are the things we treat as attributes of other things, such as salaries, colors, birthdates, birthplaces, employers, spouses, addresses, etc. (While such things may be mentioned in multiple records, I do not think we can say they are "represented" by any one record.) Unfortunately, apart from the listing of examples, it is difficult to identify precise criteria for deciding whether something is an entity, and whether it is to be represented by a record.

In a normalized system, an entity might also fail to be represented by a single record if there did not happen to be any single-valued information about the entity. Suppose one had in mind to treat projects as entities, but all the information to be maintained about them turned out to be multivalued (in relational terms, we find no functional dependences on projects). That is, our projects can have multiple managers, multiple objectives, multiple start and stop dates, multiple budgets, and so on. Each such fact needs to be maintained in a distinct intersection record, and there might be no motivation to define a single record type or relation to represent the projects themselves. One would have record types (relations) called "project-manager," "project-objective," "project-dates," and so on, but none called simply "project."

Entities With Many Records

We might have too many records. As mentioned earlier, a common solution to the problem of overlapping types (e.g., employees and stockholders) is to define them as disjoint types, and allowing an entity (person) to be represented by a record in each type.

The "generalization" approach of Smith and Smith [40] yields multiple records per entity; it is not clear that any one of them could be said to "represent" the entity. The approach of Bachman and Daya [4] is similar in this respect, but they do postulate one underlying record (never materialized) per entity.

More generally, there is no discipline preventing the definition of several record types corresponding to one entity type. That is, we could have several record types defined over the same key, with each record type containing different attributes of the subject entity. One might be tempted to do this for economic reasons, e.g., to group together attributes which tend to be accessed together, or to physically segregate rarely used data. Regardless of the motivation, such a configuration is permitted in all record-based systems. Thus none of these systems really has a well-defined semantic establishing a 1:1 correspondence between entities and records.

Records Without Entities

Normalized systems require many-to-many facts to be represented in distinct record types of their own (so-called intersection records). Employee-skill records are a good example. What entity does one of these records represent? Not the employee, nor the skill. If it represents anything at all, the record represents the *relationship* between the employee and the skill. This might suggest the principle that relationships are entities, and ought in general to be represented by records. But some relationships are not represented by records, e.g., the relationship between an employee and his department. (That relationship is recorded in an employee record, but not represented by a distinct record of its own.) Obviously, it is only the many-to-many relationships which must be represented by distinct records (in a normalized system); are they the only ones which are entities?

There are three ways to take a consistent view of this situation:

(1) All relationships are entities, and some records represent multiple entities (as the same record "represents" both an employee and his relationship to a department).
(2) Relationships are not entities, and intersection records do not represent entities.
(3) Some relationships are entities and some are not depending on whether or not they are represented by intersection records.

It is a matter of judgment as to whether any of these views is acceptable.

Depending on what definitions we like, some intersection records might not even represent relationships. You might wish to consider the color of a car to be an attribute, and not a relationship. But if cars are multicolored, then their colors must be split out into separate car/color intersection records. Does the attribute now become a relationship? Is it an entity? What entity does the record represent?

If we do not care to think of such multivalued attributes as being entities in themselves, then we again have records which do not represent entities.

Records With Many Entities

If there is a 1:1 correspondence between certain entities, then a single record might be perceived as "representing" all of them. Employees and spouses provide an example, in a monogamous society. Since each spouse occurs in exactly one employee record, one could view these records as representing spouses just as well as employees. The perception is even more plausible if the spouses are uniquely identified (as they might be, by social security number), and if they occurred in every record (if, perhaps, company policy required all employees to be married).

Summary

"Entity" is not very well defined, for our purposes. To be absolutely fair, we should only conclude that record structures do not correspond to everyone's intuitions about the characteristics of entities. But it is quite difficult to establish a definition of "entity" which puts it in 1:1 correspondence with a normalized record, unless one starts with that as the definition.

RELATIONSHIPS

One Concept, Many Representations

A binary relationship is a fairly simple concept: a named link between two entities. But there are about a half dozen ways to implement binary relationships in record structures. (Schmid and Swenson [31] make a similar analysis.)

Most of these ways to implement binary relationships involve pairing identifiers of the two entities in one record. It might be in the record representing one entity or the other. It might be in a separate record (intersection record) representing the relationship itself. And it might be embedded in a record representing some other entity altogether (an employee record may include a relationship between the employee's spouse and the spouse's employer).

These alternatives correspond to several combinations in which the two entity identifiers might occur as keys in a record. One or the other might be the key, or they might together constitute the key, or neither might be in the key.

(Actually, there are other possibilities as well. One identifier or the other might be a subset of the key, which probably violates third normal form. Or they might together constitute a subset of the key, in which case they might be part of a compound name—to be discussed later. Or they might each constitute a key for the record, if there was a 1:1 relationship between them.)

In addition, a relationship might be represented indirectly, being implied by other relationships. If projects are assigned to single departments, and if each employee works on all of his department's projects, then the way to discover if an employee works on a certain project is to match the department numbers in the employee and project records. That is, an employee's assignment to a department and a project's assignment to the department together imply the employee's working on the project.

In record management systems which provide file structure in addition to record structure, even more options become available. Relationships might be represented by file order, or by the linkage of records into hierarchies or (CODASYL) sets.

For information modeling, the problem is what to do with this plethora of options [12, 26]. Why is it necessary to make such choices? What are the criteria? Do the criteria have anything to do with the semantics of the information, as distinguished from the economics of storing or processing the data? Do all users have to know which options have been chosen, and to adapt their processing accordingly?

Normalized systems reduce the number of options for one-to-many and many-to-many relationships, and generally force them to be treated differently from each other. (But not necessarily; one-to-many relationships can be represented in separate intersection records, though they hardly ever are.) Some differences force the information modeler to prematurely impose processing techniques on end users: one-to-many relationships can often be altered by updating field values, while many-to-many relationships in intersection records can only be altered by deleting and inserting records.

Relationships Are Not Described

The various representations available for relationships are often used for other purposes as well. Thus record descriptions rarely provide clear evidence of the presence of relationships—neither explicitly nor implicitly. A record with multi-field keys might represent a relationship, or the keys might constitute a qualified name for an entity. If neither field is in the key (e.g., spouse and spouse's employer), there is no mention at all of the relationship; the two fields appear to be independent facts about the employee.

There is no regular way to reflect the name of the relationship in the file description. Sometimes it is a record type name (intersection records), sometimes it is a field name (or a part of one), and sometimes it does not occur at all (e.g., implied relationships, or joins). When file structure is available, the relationship name might also be a (CODASYL) set name or, again, it might not occur at all if represented by file order or hierarchical structure.

And, when the relationship names do occur as field names, there is no discipline. There is rarely any clue in the field name (or even in a relational domain name, if provided) as to the record type representing the related entity. Consider a generalized query processor asked to find the name of the manager of a certain department. The department record probably has a field named MANAGER (perhaps with a domain named EMPLOYEE NUMBER). What tells the processor to look into a record type named EMP-RECS to find the name of the employee whose number occurs in that field?

Sometimes the field name combines the relationship and the entity type ("assigned-dept").

The same field name might signify the same or different relationships in different records, and different names might be used for the same relationship in different records. Since field names cannot be duplicated within a record, a relationship occurring more than once in a given record necessarily has different

names (in a credit application, "employer" and "spouse's-employer" refer to occurrences of the same relationship).

In a good model for relationships, one might expect some direct way to declare relationships, specifying a name and some characteristics, without having to choose among a variety of (ambiguous) representational alternatives.

Attributes

There does not seem to be an effective way to characterize "attributes," or to distinguish them from relationships. Ironically, the most dominant correlate seems to be with record structures: if a field value is the key of some other record, then it represents a relationship; otherwise it is an attribute. This need to map things into recordlike terms seems to be the main force motivating a distinction between attributes and relationships. If we did not have a record-based implementation in mind, the distinction might go away [35].

NAMING, SYMBOLIC REFERENCE

In a pure record structure, most facts (relationships, attributes) are represented by including in one record symbolic identifiers of two or more things (e.g., employee number and department number, or employee number and salary) [32]. Such symbolic reference admits references to nonexistent entities (entities whose corresponding records are missing). Symbolic reference forces a strong interaction between the concepts of identifier and entity type, and encounters problems with synonyms and with changeable names.

When simple labels are not conveniently available, the record model permits arbitrary combinations of facts to be specified as identifiers capable of distinguishing among entity occurrences. This, in turn, leads to further problems, unless a number of constraints on the selected facts are carefully observed. Furthermore, in using such identifiers to refer to entities, multiple fields serve the function of a single field—generating ambiguities in the structure of the information represented by the record.

Simple Identifiers

To the extent that a record represents an entity (i.e., signifies its existence), symbolic reference permits references to nonexistent entities. A department number can occur in an employee record even if no corresponding department record exists. At best, there might be some check that the field contains a "plausible" department name, in terms of its syntax: the right number of the right kinds of characters. (In the proposals of Schmid and Swenson [31] and Smith and Smith [39] such existence dependences can be expressed and maintained.)

When arbitrary identifiers are assigned, such as employee numbers, then there is little question of uniqueness of identifier. But when some fact about the entity is given double duty as an identifying label for that entity, one has to be quite careful that the fact does, in fact, uniquely determine the entity. Presidential elections can be identified by the year in which they occur—provided we are absolutely certain there is no possible circumstance under which another presidential election might be held in the same year.

Symbolic identifiers rarely provide absolute identification of entities. At best,

the identifiers are unique within entity type. One cannot tell which entity is being referenced just by examining a field value; one has to have supplementary knowledge as to which entity type is involved. Nothing in the data tells us whether 123456789 identifies a person or a machine. Such information is almost never included with the data, nor with the record description. That is, a field description rarely specifies the entity type, or record type, whose keys will occur in that field. The domain construct of the relational model provides limited assistance in this area, as mentioned in the earlier discussion of field names.

In a pure record structure (one with no pointers or other file structure interconnecting the records), the only way to detect a reference to the same thing in several records is to find a match in some corresponding field values. (For example, that is the basis of the relational join.) Synonyms and aliases interfere with this process. At best, one could execute a chain of path-following operations, if one knew which record types provided which synonym linkages. That is the only way to detect that, e.g., a social security number in one record referred to the same person as an employee number in another record.

There are often several *kinds* of identifiers by which an entity can be identified uniquely, such as employee numbers and social security numbers. One now has to know not only which entity type is being referenced in the field, but also which kind of identifier is being used. And one has to know which other record type to go to in order to "translate" one name type into the other.

In an obvious way, the extent of an entity type affects the choice of identifier. Facts which are unique over a small set of entities may not be unique over a larger set.

Presidential elections in one country can be identified simply by the year of occurrence. But if the entity type were perceived as a larger set of elections, then the identification would have to include additional facts, such as the office (governor, mayor, etc.) and political unit involved (name of the country, state, city, etc.).

In a converse way, identifiers can affect the perception of entity types. In order to provide record keys, it is often necessary to arbitrarily choose one kind of identifier as a "primary" identifier for an entity. Two constraints then impose boundaries on entity types: a key value is meaningful within exactly one record type, and each record of the type must contain a key value. That is, key values must be in 1:1 correspondence with the set of entities represented, and the tendency is to think of a set as an entity type if it corresponds to the scope of a unique identifier. Unfortunately, different kinds of identifiers may have overlapping but unequal scopes, leading to conflicting choices of entity types. Employee numbers, social security numbers, military service numbers, etc., each identify different sets of people, each leading to a different concept of the "entity type" involved. When identified by ISBN's, books constitute a different entity type from films. When identified by Library of Congress numbers, they are all publications. Thus in general, the varying scopes of identifiers can have an excessive and potentially confusing influence on the establishment of entity types. If names were not tied so intimately to types, it might be possible to deal with types more naturally.

The tight coupling between types and names forces vertical homogeneity, i.e.,

the inability to represent facts which involve multiple entity types. The name formats are likely to be different between types; when the formats are the same, names might not be unique across types.

Sometimes an entity can have multiple identifiers of the *same* kind. A person can have several social security numbers, and a book might have several ISBN's. Names and addresses can be spelled and abbreviated in a variety of ways. Again, it becomes harder to detect references to the same entity when different identifiers are used. (Consider the difficulty of purging duplicates out of a mailing list.) Furthermore, these identifiers cannot all occur in the record which represents the entity (assuming normalized records), and they certainly cannot all be keys of the record. At most, one of them has to be arbitrarily selected as the primary identifier. For the others, a new intersection record type has to be defined, with each such record "translating" one secondary identifier into a primary identifier. Now, if some record contains a secondary identifier, it is necessary to know that the field is not a key into the entity record type, but must first be translated via the intersection records. If the field might contain either primary or secondary identifiers, then the retrieval algorithm is even more complex. (And, incidentally, one might ask what entities are being represented by the intersection records. They are a distinct record type from the one representing the entities. Their keys are the secondary identifiers, with a separate record for each one, hence several records per entity.)

Sometimes the identifier chosen for an entity is not, strictly speaking, the name of that entity, but of a related entity. This practice can confuse the underlying semantic structure of the information. Many facts are available, for example, about elections. One might reasonably expect to ask, in a symmetric way, who won Election No. 10 and when Election No. 10 was held. But if elections are named by their dates, e.g., the "1948 election," it suddenly becomes absurd to ask when it was held. Strictly speaking, we are dealing with an unnamed entity being identified by means of a related entity, but the record descriptions give no hint of this structure. (If it disturbs you to think of a year as being a related entity, then imagine that presidents could only be elected once, and we therefore named elections by their winners. Then it becomes equally absurd to ask who won the "Truman election," although it is meaningful to ask who lost it, and when it occurred.)

Another way to see the semantic problem is in terms of functional dependences. On an entity level, an election uniquely determines its date and its winner, and hence dates and winners ought to be functionally dependent on elections. But if the relation representing this information contains only a date column and a winner column, then there is no way to express these two functional dependences. In effect, the existence of three distinct entities is not acknowledged in any way. This is one respect in which relations and functional dependences do not mirror relationships among entities.

And, finally, it is not sufficient for a fact to uniquely identify an entity. In order to be useful in a record structure, it had better be an unchangeable fact. It is highly undesirable to have to change that fact in every place where the entity is referenced. Thus while a phone number might uniquely determine an office at any given moment, it is not a good way to identify offices if the phones tend to be moved around.

Compound Identifiers

All of these concerns—and more—apply when compound facts are used to identify entities in records.

We use for an example the identification of dependents of employees, perhaps for a benefits database. Names of dependents are certainly not unique, but we might assume that no employee has two dependents with the same name. Then a dependent could be identified by the combination of his name and the related employee's identifier (as in [8]). (We will refer to that related employee as the "sponsor.")

To begin with, the concerns about uniqueness, existence, synonyms, scope, and changeability must be examined more carefully, because there are more facts involved.

We are depending on the uniqueness of names of one employee's dependents, which might be a questionable assumption if the employee has adopted children, or has remarried, or if the employee's parents, grandchildren, siblings, etc. also qualified as dependents in addition to his children. More simply, his child and spouse might have the same name.

In references to the dependent from other records we have, as before, no assurance that the dependent record itself exists. But a second level of existence dependence is now introduced. In order for the dependent's identification to be meaningful, there must be some assurance that the sponsor's employee record also exists.

Synonyms can easily arise, if a dependent was related to more than one employee. The dependent has as many valid identifiers as he has sponsors. A compound identifier thus need not even be restricted to a many-to-one relationship. If all we are after is the ability to select dependents, then all we need is that no employee have two dependents with the same name. The relationship could be many-to-many.

This gives us a one-way uniqueness. A qualified name identifies exactly one dependent, but we cannot tell whether two qualified names might refer to the same dependent. If two employees had a dependent named "Joe," what indicates whether it is the same dependent? Special pains must be taken to detect that the dependents of several employees might be the same person, in order to properly coordinate benefits. A separate "translation" record type is now needed to establish the equivalences between identifiers. (And the synonym problem is compounded if the sponsors themselves have synonyms.)

It is possible to designate one identifier (involving one sponsor) as the "primary" identifier of the dependent. But it may be necessary in some situations to permit any identifier to be used, requiring a search of both the translation records and the dependent records. Furthermore, this leads to a potential violation of the constancy requirement: a dependent's primary sponsor will change if that sponsor leaves the company but the dependent is still related to other employees. Then all references to that dependent have to be modified. In fact, it is probably not a good idea to use employees to qualify dependent names in an implementation which forbids modification of record keys.

Scope of identifiers is, of course, still a crucial concern. The facts chosen for the compound name must be relevant and known for all occurrences of the entity type being identified. For example, if the company charitably expanded its

benefits program to provide aid to all needy people in the community, then some recipients would not have any related employee to use in their identifiers. Some other form of identification would have to be devised.

So far, all these problems of compound names have just been extensions of similar problems which existed for simple names. A whole new set of problems derives from the fact that compound names force an entity to be referenced by multiple fields in a record.

There is a three-way ambiguity. Multiple fields in a record are used to represent three distinct semantic constructs: the compound name of an entity, a relationship among entities, or multiple independent facts about an entity.

Spurious Joins. The resemblance between compound names and independent facts can lead to "phantom entities" and spurious joins. Whenever an employee number and a person's name occur in any two fields of a record, this could be construed as the compound identifier of a dependent. In particular, a relational join can be performed over such fields.

An employee's name and his manager's employee number, in one record, could be matched up with a customer's name and his salesman's employee number in another record, in the mistaken belief that such pairs of fields constituted qualified names for the same person. More simply, a dependent record might include another employee number field besides his sponsor (e.g., a benefits administrator); a join could be done on the wrong employee field, mistaking this person for a dependent of that other employee (who might not even have any dependents).

The domain concept in the relational model is supposed to control joinability. In relational theory, joins may only be performed on columns based on the same domains, presumably assuring that the same entity is occurring on both sides as a "link." The problem is that relations are fundamentally defined as aggregations of single columns, each column based on a domain. There is no concept of a domain encompassing multiple columns, although entities may be identified by multiple columns. The domain concept cannot be applied in such a way that two columns in a relation must always be treated as a unit, and may only be joined with other pairs of columns defined to constitute similar units. In the current example, the two fields might have separate domains specified as "employee number" and "person name"; there is no place to establish "dependents" as a domain. And even if the second domain was specified as "dependent name," it still could be paired with any other employee number in the same record.

Thus domains do not readily correspond to entity types, when the entities are identified by compound names.

Reducibility Dilemma. The resemblance between compound names and independent facts leads to a dilemma in the theory of reducible relations [15, 16, 19, 29].

Consider a person's birthday. On the face of it, this is an elementary (irreducible) fact—a simple binary relationship between a person and a certain day in the past. And, if we happen to represent dates in Julian notation (one field), then birthday actually has the structure of an elementary fact. But if we choose to change the *naming* of the date to the more conventional notation involving three fields, then we have a record containing four fields. This record can now be

reduced to three binary records: person and year, person and month; and person and day of month. The original birthday record can always be recovered by joining these three.

The same analysis, and ambiguity, applies whenever a composite naming convention is selected for an entity. City of birth, for example, is an irreducible fact if globally unique city codes are used; it is reducible if the city is identified by the composite of, e.g., city, state, and country names.

Composite names are in general *not* precisely equivalent in function to simple unique identifiers for the same entities. Composite names almost always convey additional information; when used in lieu of simple names they necessarily change the underlying structure of the information. A simple name simply designates an entity; a composite name does that, but it simultaneously informs us about other related entities. A city code simply designates a city; the conventional notation may additionally tell us the state and country in which it is located. A Julian date simply designates a certain day (if we do not bother to do certain computations); the conventional notation additionally tells us the year and month in which it occurred, as well as the day of the month. In the role of designating a single entity, a compound name could be part of an irreducible fact; in the role of providing auxiliary information about related entities, it leads to reducibility.

Confusion of Subject. The resemblance between compound names and relationships confuses the subject matter of a record, namely the question of what entity is being represented by a record. A record having employee number and dependent name as a compound key might be a dependent record (representing a dependent), or it might be an intersection record representing the relationship between an employee and a dependent. In the first case, it would be appropriate to include the dependent's age in the record, since that is a fact about the dependent:

EMPLOYEE-NUM	DEPENDENT-NAME	AGE

key

In the second case, the record might specify the kinship between the dependent and the employee:

EMPLOYEE-NUM	DEPENDENT-NAME	KINSHIP

key

The first fact is really about the dependent alone, while the second is about the relationship between the dependent and an employee. But the structure of the two, in "irreducible" form is indistinguishable. Thus if naming conventions are not separated from entity representation, "irreducible" records still do not model the structure of information unambiguously. (To see the significance, compare what happens to the preceding two structures if dependents were named simply, e.g., by social security numbers.)

In unreduced records, a composite key is likely to be serving both roles simultaneously. It would not be unusual to see the two records shown above combined into one (since they have the same key), containing both age (a fact about the dependent) and kinship (a fact about the relationship). It is thus ambiguous as to which entity is really represented by this record.

Ironically, although "intersection records" are commonly accepted as the construct which represents many-to-many relationships, it is, in fact, hard to know which records are serving as intersection records.

Surrogates

A precise model of information should distinguish carefully between the structure of entities being modeled and the various structures of names which might be associated with them. This implies a distinction in the model between entities and traditional data items.

Some alternative models suggest that some sort of an internal construct be used to represent an entity, acting as a "surrogate" for it [1, 19]. This surrogate would occur in data structures wherever the entity is referenced, and naming problems would at least be isolated by keeping structured or ambiguous identifiers off to one side, outside the structures representing attributes and relationships.

Since these surrogates must eventually be implemented inside the computer in some form of symbol string, it is sometimes held that such surrogates are themselves nothing but symbolic identifiers.

It is useful to be aware of some fundamental differences between surrogates and ordinary identifiers:

(1) A surrogate might not be exposed to users. Only ordinary identifiers need pass between user and system. In concept, models involving surrogates can behave as though a fact (e.g., the assignment of an employee to a department) was treated in two stages. First, the surrogates corresponding to the employee and department identifiers are located (i.e., name resolution). Then the two surrogates are placed in association with each other, to represent the fact.
(2) Users do not specify the format, syntax, structure, uniqueness rules, etc. for surrogates.
(3) Surrogates are globally unique, and have the same format for all entities. The system does not have to know the entity type (or the identifier type) before knowing which entity is being referenced, or before knowing what the surrogate format will be.
(4) Surrogates are purely information-free. They do not imply anything about any related entities, nor any kind of meaningful ordering.

SEMANTIC STRUCTURE AND THIRD NORMAL FORM

In the absence of any additional discipline, it is difficult to guess the semantic structure implied by a record format. A given field might include multiple facts by encoding, or with internal punctuation. A given fact might occur in any of several fields: in one of our example vehicle assignment records, the assignee would occur in the third field of some records and the fourth field of other records. A given field might give information about different things in different records: a

clever programmer might design a single "maiden name" field to refer either to an employee or to the employee's spouse, depending on which was female. In general, the possibility of tricks such as these obscures the underlying semantic structure.

The semantic structure suggested by a relation under the discipline of third normal form [11, 20] is one in which each record represents a single "subject" entity (identified by the key), with the other fields being "direct" facts about the subject entity. That is, the nonkey fields are functionally dependent on the key, and there are no other functional dependences.

There are some difficulties here also. In the first place, the role of functional dependences in relational theory is unclear, at least as reflected in implementations. The assumption tends to be that functional dependences (if specified at all) have been used during the design phase of the database to insure that relations are in third normal form, and then discarded. They do not seem to be present at run time to explain the semantic structure of the data.

Secondly, functional dependences deal only with 1:*n* relationships and not *m*:*n* relationships. However, some recent work [14] offers extensions into this area.

Thirdly, functional dependences merely specify dependences without naming the relationships involved. Thus, e.g., it cannot be determined whether several functional dependences (between different pairs of columns) are based on the same relationship. Consider, for example, a credit application record containing the applicant, employer, spouse, and spouse's employer. Functional dependences give no clue that the same relationship exists between the first and second columns as between the third and fourth.

Furthermore, third normal form allows several keys to remain in the same relation, even if they identify different entities. This can occur when entities are in a 1:1 relationship, as with employees and spouses. One of the keys is typically selected as "primary," which can suggest that only one entity (e.g., the employee) is the subject of the record. If functional dependences are discarded after the database design phase, there is nothing left to dispel this illusion. The remaining structure implies, for example, that "spouse's birthdate" is not an attribute of the spouse, but only of the employee. The structure is vulnerable to update anomalies, with the implication that the "spouse" and "spouse's birthdate" attributes of an employee may be updated independently of each other.

Still another concern is that it may not be possible to model the *attributes* of 1:*n* and 1:1 relationships, because only "full" functional dependences are considered in the determination of third normal form. A full functional dependence is based on the "smallest" possible subject which determines the related fact. Thus while the relationship between an employee and his department may have an assignment-date attribute, that date is not "fully" dependent on both employee and department. There is only one (current) assignment date for each employee, and hence that date has a full functional dependence on the employee alone. (That is, knowing the employee is sufficient to determine the date.) To the extent that only full dependences are considered, functional dependences distort the semantic structure of information. Attributes of 1:*n* and 1:1 relationships are always transformed into attributes of one of the related entities. Only the attributes of *m*:*n* relationships are preserved in functional dependences.

Finally, the most serious concern is that functional dependences are expressed

on the name level rather than the entity level. One consequence is the failure to model the existence of entities whose names are given to other entities (as in the earlier example of elections named via years or winners).

Another consequence is that dependences are only expressed when the subjects are uniquely named. In the example of employees and spouses, functional dependences on the spouses would only be expressed if the spouses were uniquely named (as by social security number). In the more common case, no functional dependences could be written, since the same spouse *name* might occur in different records with different birthdates, employers, etc. Spouses would not in any way be designated as the subjects of any information, or as candidate keys.

(We could uniquely identify spouses by the combination of spouse name and employee number, as a qualified name. But functional dependences based on this pair of fields would not be full dependences, since the attributes—such as "spouse's birthdate"—are uniquely determined by employee number alone. As mentioned earlier, such nonfull dependences are discarded in the determination of third normal form.)

Because of the dependence on naming, a large number of relationships in a typical database fail to be reflected in functional dependences, and hence fail to be subject to the discipline of third normal form. In a personnel file, attributes of such things as previous employers, banks, schools, relatives, etc. tend to be duplicated by embedding in multiple employee records. The functional dependence between a school and its address could not be specified, because several schools might have the same name. Hence, if several employees attended the same school, the address of that school might be included in each employee's record, without violating third normal form. All such cases are exposed to the update anomalies which third normal form is supposed to prevent.

A special case of this problem is not only tolerated but in fact encouraged by the relational model. Compound keys, in which the name of an entity includes the name of a related entity, plainly constitute a replication of information, thereby violating the spirit if not the letter of third normal form.

Consider an earlier example, where departments were uniquely named only within divisions, so that company records had to use a compound of department name qualified by division name. On the entity level, a department uniquely determines its division (a given department is, after all, in only one division). But on the name level, a department name does *not* uniquely determine the corresponding division—that is precisely why qualification is needed in the first place. Hence no functional dependence can be written here, even though there is a many-to-one relationship between the entities involved.

A department's division is mentioned in every record which references that department. If the division of a department changes, that single fact must be changed wherever that department is mentioned—precisely what third normal form should avoid.

There is a pragmatic solution. The update problem is best dealt with by legislating it away: forbid the update of keys. This is common in most implementations, though not a formal part of relational theory.

Third normal form is very sensitive to naming conventions. Giving departments globally unique identifiers would represent no real change in the semantic

structure of the information. But a functional dependence could now be written between departments and divisions, and the records in question would no longer be in third normal form.

To conclude: records in third normal form can include a great many relationships which are not suggested at all in the record descriptions, even in the form of functional dependences.

CONCLUSIONS

We have outlined a number of ways in which record structures fail to model the semantics of information accurately and unambiguously. Other models deal with these problems, with varying degrees of success. A discussion of such alternatives is generally beyond the scope of this paper.

Just briefly, we can observe that file structures (hierarchies, CODASYL sets) overcome some functional limitations, but by increasing rather than decreasing the variety of representational alternatives. Bachman and Daya [4] and Smith and Smith [40] address some of the problems of types, but still in a record-based framework. Binary relation (or "elementary fact") models are generally more successful in coping with the modeling problems, though none have done so completely. Such models are more directly based on semantic concepts, e.g., entities and the network of relationships among them, rather than on recordlike structures. The model described by Biller and Neuhold [5] is excellent in this respect. Other models along these lines include the works (referenced below) of Abrial, Bracchi, Falkenberg, Griffith, Hall, Roussopoulos, Schmid, Senko, and Sowa.

ACKNOWLEDGMENTS

Many people made valuable comments on earlier versions of this paper, including Chris Date, Bob Engles, Bob Griffith, Roger Holliday, Lucy Lee, and especially Paula Newman, who suggested some of the ideas. I am also deeply indebted to the referees, especially Mike Senko, who instigated major improvements in the paper.

REFERENCES

1. ABRIAL, J.R. Data semantics. In *Data Base Management*, J. W. Klimbie and K. L. Koffeman, Eds., North-Holland Pub. Co., Amsterdam, 1974.
2. ANSI/X3/SPARC. Study Group on Data Base Management Systems. Interim Rep., Feb. 1975; also FDT (Bulletin of ACM SIGMOD) 7, 2 (1975).
3. The ANSI/X3/SPARC DBMS Framework. Report of the Study Group on Data Base Management Systems, D. Tsichritzis and A. Klug, Eds., AFIPS Press, Montvale, N.J., 1977.
4. BACHMAN, C.W., AND DAYA, M. The role concept in data models. Proc. Third Int. Conf. Very Large Data Bases, Database (ACM) *9*, 2 (Fall 1977), SIGMOD Record (ACM) *9*, 4 (Oct. 1977), 464–476.
5. BILLER, H., AND NEUHOLD, E.J. Semantics of data bases: The semantics of data models. *Inform. Syst. 3*, 1 (1978).
6. BRACCHI, G., PAOLINI, P., AND PELAGATTI, G. Binary logical associations in data modeling. In *Modelling in Data Base Management Systems*, G. M. Nijssen, Ed., North-Holland Pub. Co., Amsterdam, 1976.
7. CHAMBERLIN, D.D. Relational data base management systems. *ACM Comptng. Surveys 8*, 1 (March 1976), 43–66.

8. CHEN, P.P.S. The entity-relationship model: Toward a unified view of data. *ACM Trans. Database Syst. 1,* 1 (March 1976), 9–36.
9. CODD, E.F. A relational model of data for large shared data banks. *Comm. ACM 13,* 6 (June 1970), 377–387.
10. CODD, E.F. Normalized data base structure: A brief tutorial. ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego, Nov. 1971, pp. 1–18.
11. CODD, E.F. Further normalization of the data base relational model. In *Data Base Systems* Courant Computer Science Symposia 6, R. Rustin, Ed., Prentice-Hall, Englewood Cliffs, N.J., 1972; also IBM Res. Rep. RJ909.
12. CODD, E.F., AND DATE, C.J. Interactive support for non-programmers: The relational and network approaches. ACM SIGMOD Workshop on Data Description, Access, and Control (Vol. 2), May 1974, pp. 11–41; also IBM Res. Rep. RJ1400.
13. DATE, C.J. *An Introduction to Database Systems,* Addison-Wesley, Reading, Mass., Second ed., 1977.
14. FAGIN, R. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst. 2,* 3 (Sept. 1977), 262–278; also IMB Res. Rep. RJ1812.
15. FALKENBERG, E. Concepts for modelling information. In *Modelling in Data Base Management Systems,* G. M. Nijssen, Ed., North-Holland Pub. Co., Amsterdam, 1976.
16. FALKENBERG, E. Significations: The key to unify data base management. *Inform. Syst. 2,* 1 (1976), 19–28.
17. GRIFFITH, R.L. Information structures. IBM Tech. Rep. TR03.013, IBM, San Jose, Calif., May 1976.
18. Data Base Management System Requirements. Joint Guide-Share Data Base Requirements Group, Nov. 1970.
19. HALL, P.A.V., OWLETT, J., AND TODD, S.J.P. Relations and entities. In *Modelling in Data Base Management Systems,* G.M. Nijssen, Ed., North-Holland Pub. Co., Amsterdam, 1976.
20. KENT, W. A primer of normal forms. Tech. Rep. TR02.600, IBM, San Jose, Calif., Dec. 1973.
21. KENT, W. New criteria for the conceptual model. In *Systems for Large Data Bases,* P. C. Lockemann and E. J. Neuhold, Eds., North-Holland Pub. Co., Amsterdam, 1977.
22. KENT, W. Entities and relationships in information. In *Architecture and Models in Data Base Management Systems,* G.M. Nijssen, Ed., North-Holland Pub. Co., Amsterdam, 1977.
23. KERSCHBERG, L., OZKARAHAN, E.A., AND PACHECO, J.E.S. A synthetic English query language for a relational associative processor. Proc. Second Int. Conf. Software Eng., San Francisco, 1976, pp. 505–519.
24. KLIMBIE, J.W., AND KOFFEMAN, K.L., Eds. *Data Base Management,* North-Holland Pub. Co., Amsterdam, 1974. (Proc. IFIP Working Conf. Data Base Management, Cargese, Corsica, France, April 1974.)
25. LOCKEMANN, P.C., AND NEUHOLD, E.J., Eds. *Systems for Large Data Bases,* North-Holland Pub. Co., Amsterdam, 1977. (Proc. Second Int. Conf. Very Large Data Bases, Sept. 1976, Brussels.)
26. NIJSSEN, G.M. Two major flaws in the CODASYL DDL 1973 and proposed corrections. *Inform. Syst. 1* (1975), 115–132.
27. NIJSSEN, G.M. *Modelling in Data Base Management Systems,* North-Holland Pub. Co., Amsterdam, 1976. (Proc. IFIP TC-2 Working Conf., Freudenstadt, W. Germany, Jan. 1976.)
28. NIJSSEN, G.M. *Architecture and Models in Data Base Management Systems,* North-Holland Pub. Co., Amsterdam, 1977. (Proc. IFIP TC-2 Working Conf., Nice, France, Jan. 1977.)
29. RISSANEN, J., AND DELOBEL, C. Decomposition of files, a basis for data storage and retrieval. IBM Res. Rep. RJ1220, IBM Res. Lab., San Jose, Calif., May 1973.
30. ROUSSOPOULOS, N., AND MYLOPOULUS, J. Using semantic networks for data base management. Proc. Int. Conf. Very Large Data Bases, 1975, pp. 144–172. (available from ACM, New York)
31. SCHMID, H.A., AND SWENSON, J.R. On the semantics of the relational model. Proc. ACM SIGMOD Int. Conf. Manage. of Data, 1975, pp. 211–223.
32. SENKO, M.E., ALTMAN, E.B., ASTRAHAN, M.M., AND FEHDER, P.L. Data structures and accessing in data base systems. *IBM Syst. J. 12* (1973), 30–93.
33. SENKO, M.E. The DDL in the context of a multilevel structured description: DIAM II with FORAL. In *Data Base Description,* B.C.M. Douque and G. M. Nijssen, Eds., North-Holland Pub. Co., Amsterdam, 1975, pp. 239–257; also IBM Res. Rep. RC5073.
34. SENKO, M.E. Information systems: Records, relations, sets, entities, and things. *Inform. Syst. 1,*

1 (1975), 1–13.
35. SENKO, M.E. DIAM as a detailed example of the ANSI SPARC architecture. In *Modelling in Data Base Management Systems*, G. M. Nijssen, Ed., North-Holland Pub. Co., Amsterdam, 1976.
36. SHARMAN, G.C.H. A new model of relational data base and high level languages. Tech. Rep. TR. 12.136, IBM United Kingdom, Feb. 1975.
37. ACM SIGFIDET Workshop on Data Description, Access, and Control, Nov. 1971, San Diego, Calif., E. F. Codd and A. L. Dean, Eds.
38. ACM SIGMOD International Conference on Management of Data, May 1975, San Jose, Calif., W. F. King, Ed.
39. SMITH, J.M., AND SMITH, D.C.P. Database abstractions: Aggregation. *Comm. ACM 20*, 6 (June 1977), 405–413.
40. SMITH, J.M., AND SMITH, D.C.P. Database abstractions: Aggregation and generalization. *ACM Trans. Database Syst. 2*, 2 (June 1977), 105–133.
41. SOWA, J.F. Conceptual graphs for a data base interface. *IBM J. Res. and Develop. 20*, 4 (July 1976), 336–357.
42. TAYLOR, R.W., AND FRANK, R.L. CODASYL data base management systems. *ACM Comptng. Surveys 8*, 1 (March 1976), 67–104.
43. TSICHRITZIS, D., AND LOCHOVSKY, F.H. Hierarchical data base management systems. *ACM Comptng. Surveys 8*, 1 (March 1976), 105–124.
44. Proceedings of the International Conference on Very Large Data Bases, Framingham, Mass., 1975. (available from ACM, New York)
45. Proc. of the Second Int. Conf. Very Large Data Bases, Brussels, 1976.
46. Proceedings of the Third International Conference on Very Large Data Bases, Oct. 1977, Tokyo, Japan, Joint Issue Data Base *9* (ACM) 2 (Fall 1977), SIGMOD Record (ACM) *9*, 4 (Oct. 1977).

Received June 1977; revised February 1978 and June 1978

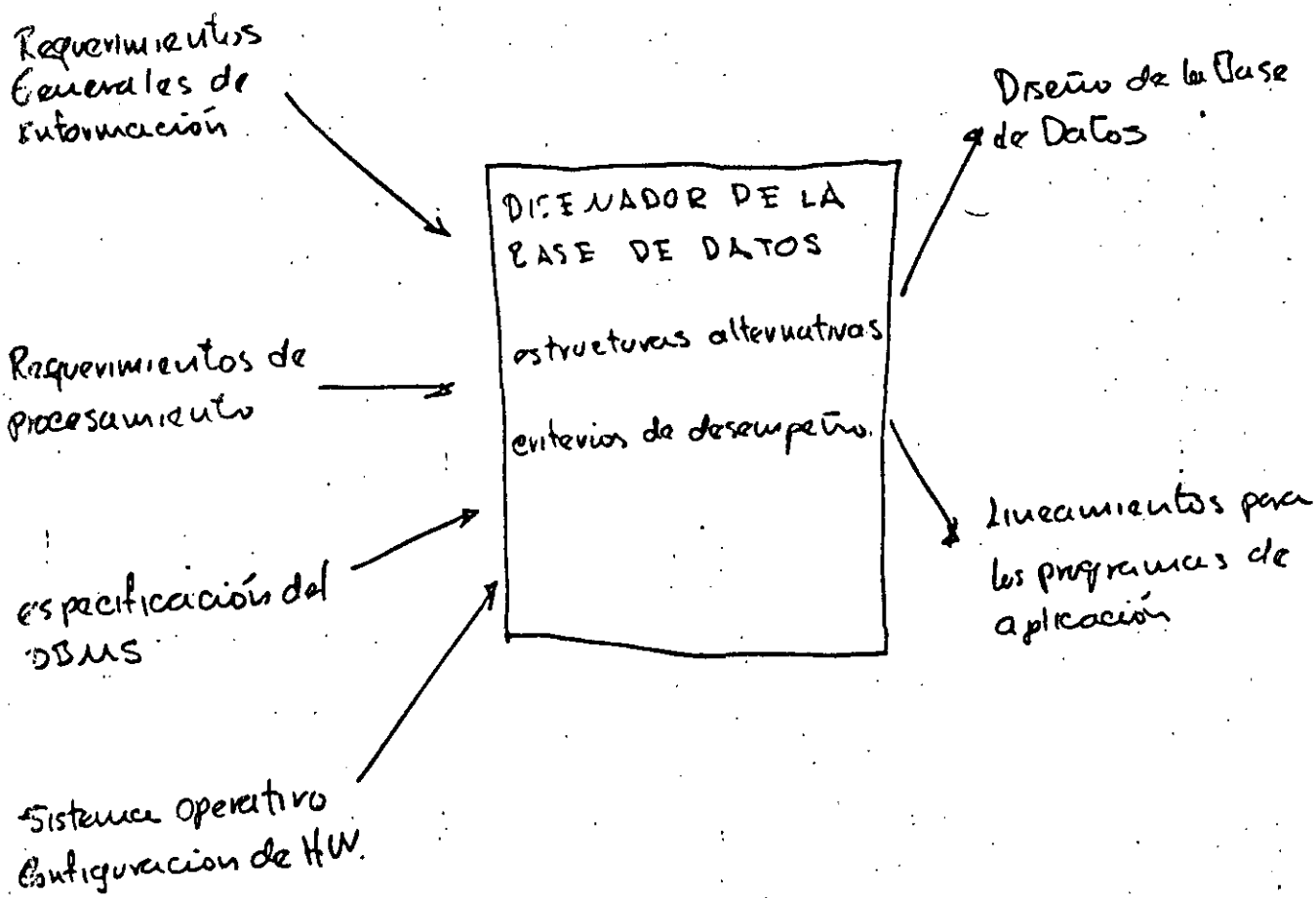
DISEÑO DE BASES DE DATOS

PROCESO DE DISEÑO DE LA BASE DE DATOS

Ing. Daniel Ríos Zertuche

MAYO, 1985

PROCESO DE DISEÑO DE LA BASE DE DATOS



CICLO DE VIDA DE UN SISTEMA DE BASE DE DATOS

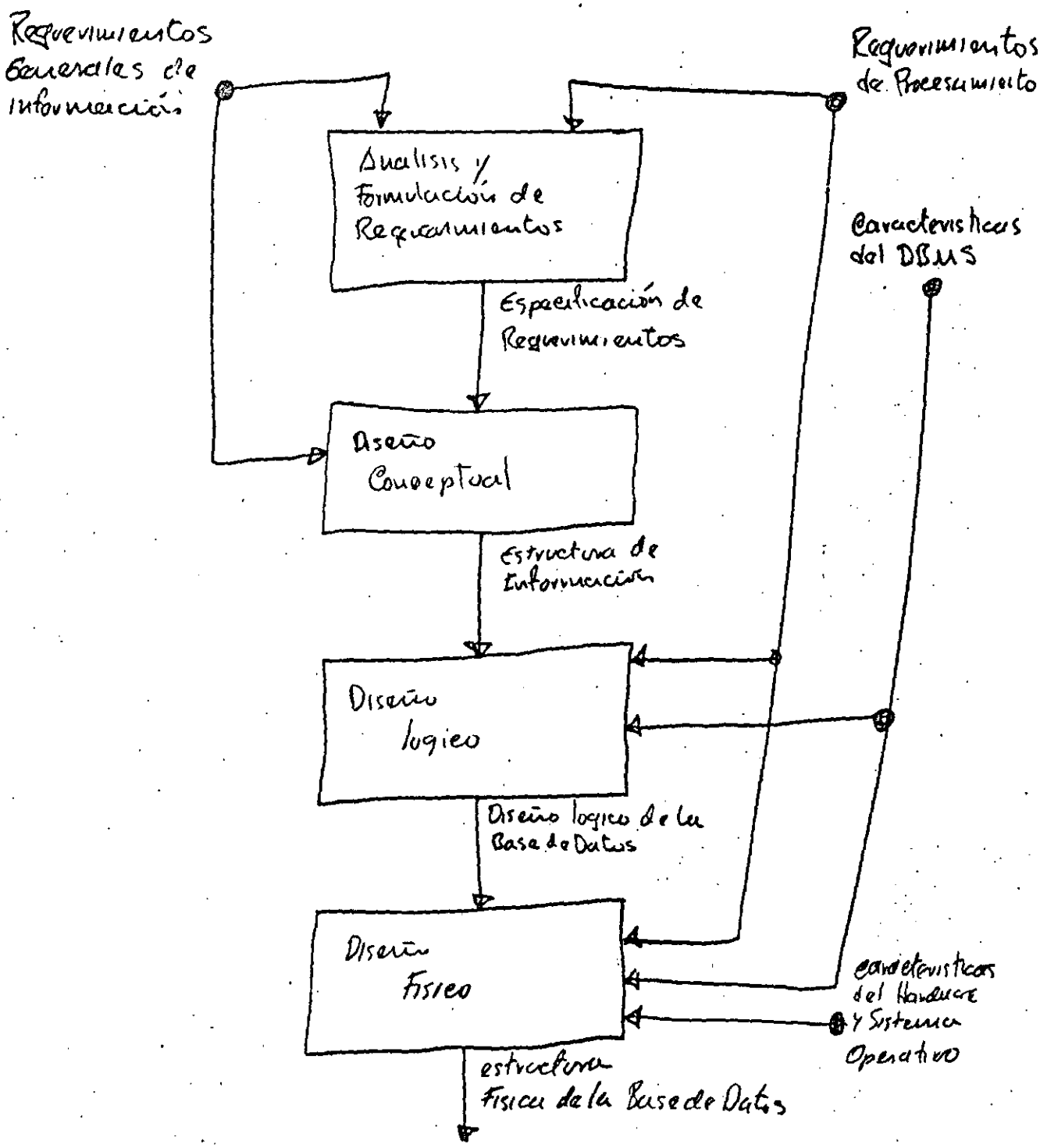
Fase de Analisis y Diseño

1.- Analisis y Formulación de Requerimientos
2.- Diseño Conceptual
3.- Diseño de la implantación
4.- Diseño Fisico

Fase de implantación y Operación de la Base de Datos

1. Implantación de la Base de Datos
2.- Operación y Monitoreo
3.- Modificación y Adaptación

PASOS BASICOS EN EL DISEÑO DE UNA BASE DE DATOS



ANÁLISIS Y FORMULACIÓN DE REQUERIMIENTOS F5

1. Identificación de las organizaciones que desempeñan las funciones operacionales

2. Solicitar a cada encargado la siguiente información
   Puestos de las personas en su área de responsabilidad
   funciones operativas desempeñadas por puesto
   Objetivo que cubre cada puesto

3. Clasifique los puestos en planeación
   control
   operación

4. entrevistar a dos personas en cada puesto operacional

5. entrevistas a los ~~funciones~~ gerencias respecto a las funciones de Planeación y Control.

Entrevistas Operativas

Objetivos:

+ Identificar cada función Operacional
+ Identificar sus datos
+ Identificar las reglas implícitas y explícitas de como y cuando cada función ocurre

Procedimiento para la entrevista

1.- para frecuencia = diaria, semanal, mensual, Trimestral y anual

pida al entrevistado describir

funciones o tareas

Documente en un diagrama de Flujo

acciones principales
decisiones
interfases.

Utilice el diagrama como un mecanismo para realimentación

2º Una vez corroborados los diagramas determine

documentos
archivos
referencias informales.

3º Determine la relación entre el diagrama de flujo y las referencias. De acuerdo con el entrevistado

4º Solicite una muestra de cada documento

5º Asocie en cada elemento dato en los documentos con el simbolo en el diagrama de flujo en el que sea utilizado.

6º anexe a cada Diagrama de Flujo una nota describiendo las reglas.

ENTREVISTA A GERENCIAS

F8

ENTREVISTA A NIVELES MEDIOS

Interfase entre las areas operativas

Reglas que gobiernas las operaciones diarias

Información requerida para medición y control del desempeño

Efectos potenciales de cambios pronosticados en las areas operacionales.

F9

Transformación de los Requerimientos de Información

Identificación de los elementos Data

organice los elementos data en listas genericas
resuelva redundancia y definición

Identifique las tareas operacionales

tarea: una unidad unica de trabajo

Consiste de una secuencia de pasos dirigidos hacia una meta comun.

todos los pasos crean o usan los mismos datos.

Compare las entrevistas repetidas

Documente

+ asigne un identificador de la tarea
+ defina la tarea a través de un verbo-objeto
+ clasifique las tareas como operacionales de control o planeación
+ la frecuencia de la tarea
+ relacione cada tarea con el proceso de que es parte

Identifique las tareas de Control y Operacionales

Identifique las políticas operativas actuales y futuras.

Desempeño de la Estructura logica de la Base de Datos

Accesos a registros logicos

$$LRA_i = \sum_{j=1}^{N} LRA_{ij}$$

Aplicación i

numero de tipos de Registro N

Accesos por unidad de tiempo

$$LRA = \sum_{i=1}^{M} \sum_{j=1}^{N} LRA_{ij} \times F_i$$

M = numero total de aplicaciones

F = frecuencia de ejecución de la aplicación

Volumen de Transporte

$$TRVOL_i = \sum_{j=1}^{N} LRA_{ij} \times RECSIZE_j$$

Volumen de Transporte por unidad de tiempo

$$TRVOL = \sum_{i=1}^{M} \sum_{j=1}^{N} LRA_{ij} \times RECSIZE \times F_i$$

Espacio de Almacenamiento 1.1.4

$$DSTOR = \sum_{j=1}^{N} RECSIZE_j \times NREC_j$$

Almacenamiento de Apuntadores

$$PTRSTOR = \sum_{j=1}^{N} NREC_j \times PS \times NPTR_j$$

PS = Tamaño del apuntador en bytes

$NPTR_j$ = numero promedio de apuntadores almacenados con registro tipo j

Tipos de Sistemas operativos

Batch
Multiprogramación
Tiempo compartido
Procesamiento de transacciones

Organización de un Sistema de Archivos

(14)

Características Básicas deseadas

+ Acceso rápido para recuperación
+ Actualización conveniente
+ Economía de almacenamiento

Características Secundarias

+ Capacidad para representar estructuras de información del mundo real
± Confiabilidad
+ Protección de Privacidad
+ Mantenimiento de la Integridad

1. Orden

Número de Orden
Clave de Vendedor
Nombre del Vendedor
No. de Cliente
Nombre del Cliente
Fecha Orden
Clave del que toma la Orden
Cantidad Ordenada
Número de Referencia
Descripción del Producto

2. Catálogo de Productos

Número de Referencia
Descripción del Producto

3. Llamada a Bodega

Número de Referencia
Cantidad Ordenada
Hay Existencia
Fecha esperada por el Embarque
Precio Unitario

4. Llamada a Vendedor

Prod. que no hay Existencia
Núm. de Ref. del Prod. Alternativo
Descripción del Prod. Alternativo

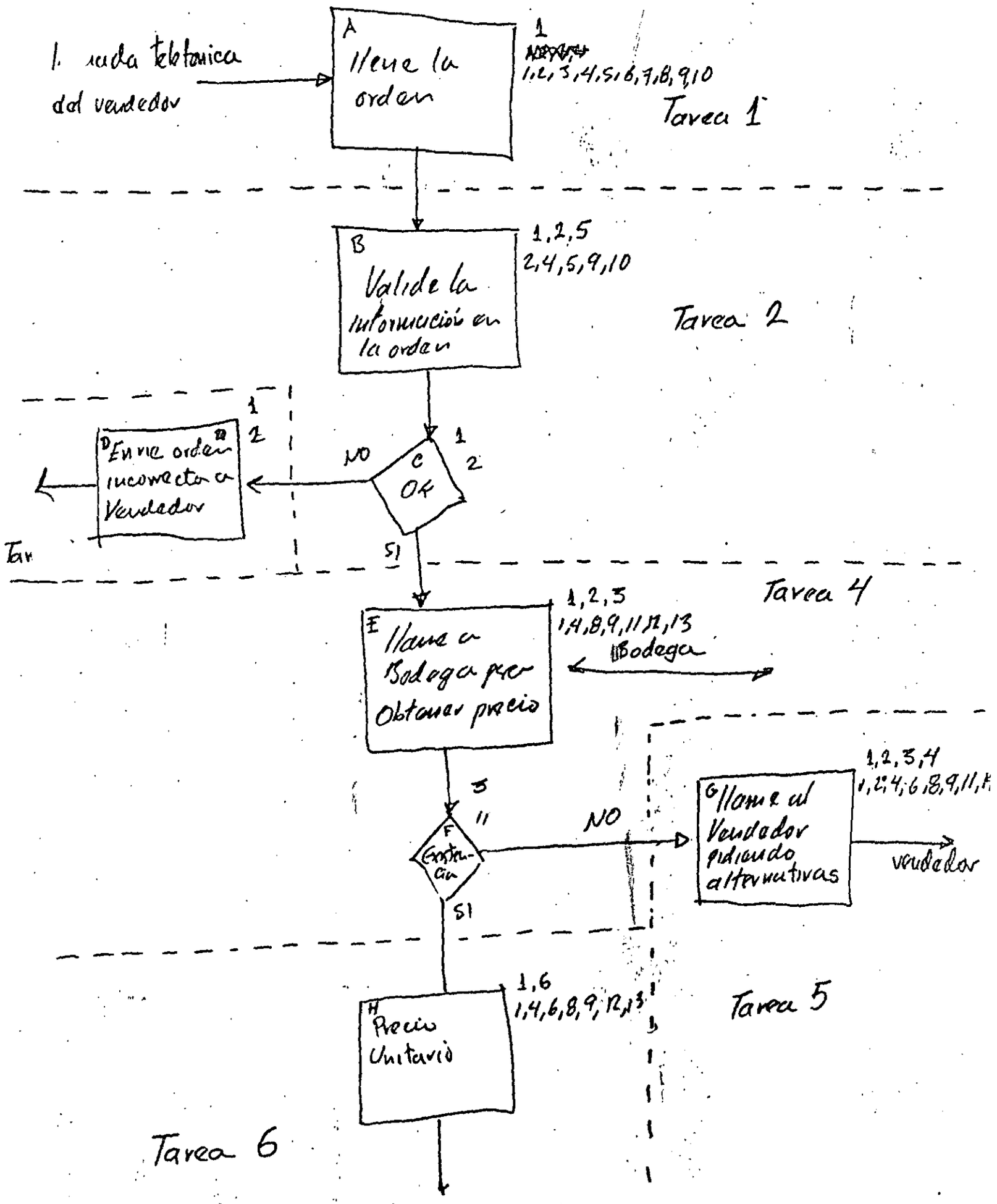
5. Kardex del Cliente

No. de Cliente
Nombre del Cliente
Clave del Vendedor

6. Orden de Precio

Número de Orden
Fecha
Cantidad Ordenada
Número de Referencia
Precio Unitario
Fecha de Embarque

		1	2	3	4	5	6
1.	Número de Orden	X					X
2.	Clave del Vendedor	X				X	
3.	Nombre del Vendedor	X					
4.	No. de Cliente	X				X	
5.	Nombre del Cliente	X				X	
6.	Fecha Orden	X					X
7.	Clave del que tomar Orden	X					
8.	Cantidad Ordenada	X		X			X
9.	Número de Referencia	X	X	X	X		X
10.	Descripción del Prod.	X	X				
11.	Hay Existencia			X			
12.	Fecha para Embarque			X			X
13.	Precio Unitario			X			X
14.	Prod. Alternativo				X		
15.	Descr. Prod. Alternativo				X		


1. cada tebtanica del vendedor
A
Llene la orden
1
1,2,3,4,5,6,7,8,9,10
Tarea 1
B
Valide la información en la orden
1,2,5
2,4,5,9,10
Tarea 2
C
OK
NO
SI
D
Envie orden incorrecta a Vendedor
1
2
Tarea 4
E
Llame a Bodega por Obtener precio
1,2,3
1,4,8,9,11,12,13
Bodega
F
Existencia
NO
SI
G
Llame al Vendedor pidiendo alternativas
1,2,3,4
vendedor
Tarea 5
H
Precio Unitario
1,6
1,4,6,8,9,12,13
Tarea 6

TAREA	ELEMENTOS DATO
1	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
2	2, 4, 5, 9, 10
3	2
4	1, 4, 8, 9, 11, 12, 13
5	1, 2, 4, 6, 8, 9, 11, 14, 15
6	1, 4, 6, 8, 9, 12, 13

MATRIZ DE VINCULOS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1		2	1	3	1	2	1	3	3	1	2	1	1	1	1
2	2		1	3	2	2	1	2	3	2	1			1	1
3	1	1		1	1	1	1	1	1	1					
4	3	3	1		2	2	1	3	4	2	2	1	1	1	1
5	1	2	1	2		1	1	1	2	2					
6	2	2	1	2	1		1	2	2	1	1			1	1
7	1	1	1	1	1	1		1	1	1					
8	3	2	1	3	1	2	1		3	1	2	1	1	1	1
9	3	3	1	4	2	2	1	3		2	2	1	1	1	1
10	1	2	1	2	2	1	1	1	2						
11	2	1		2		1		2	2			1	1	1	1
12	1			1				1	1		1		1		
13	1			1				1	1		1	1			
14	1	1		1		1		1	1		1				1
15	1	1		1		1		1	1		1			1	

VECTOR DE CUENTA DE USO POR TAREA

1	4
2	4
3	1
4	5
5	2
6	3
7	1
8	4
9	5
10	2
11	2
12	2
13	2
14	1
15	1

VECTOR DE TOTAL DE VINCULOS

1	14
2	12
3	9
4	14
5	9
6	12
7	9
8	14
9	14
10	9
11	10
12	6
13	6
14	8
15	8

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1		.5	.25	.75	.25	.5	.25	.75	.75	.25	.5	.25	.25	.25	.25
2	.5		.25	.75	.5	.5	.25	.5	.75	.5	.25			.25	.25
3	1	1		1	1	1	1	1	1	1					
4	.6	.6	.2		.4	.4	.2	.6	.8	.4	.4	.2	.2	.2	.2
5	.5	1	.5	.5		.5	.5	.5	1	1					
6	.66	.66	.33	.66	.33		.33	.66	.66	.33	.33			.33	.33
7	1	1	1	1	1	1		1	1	1					
8	.75	.5	.25	.75	.25	.5	.25		.75	.25	.5	.25	.25	.25	.25
9	.6	.6	.2	.8	.4	.4	.2	.6		.4	.4	.2	.2	.2	.2
10	.5	1	.5	1	1	.5	.5	.5	1						
11	1	.5		1		.5		1	1			.5	.5	.5	.5
12	.5			.5				.5	.5		.5		.5		
13	.5			.5				.5	.5		.5	.5			
14	1	1		1		1		1	1		1				1
15	1	1		1		1		1	1		1			1	

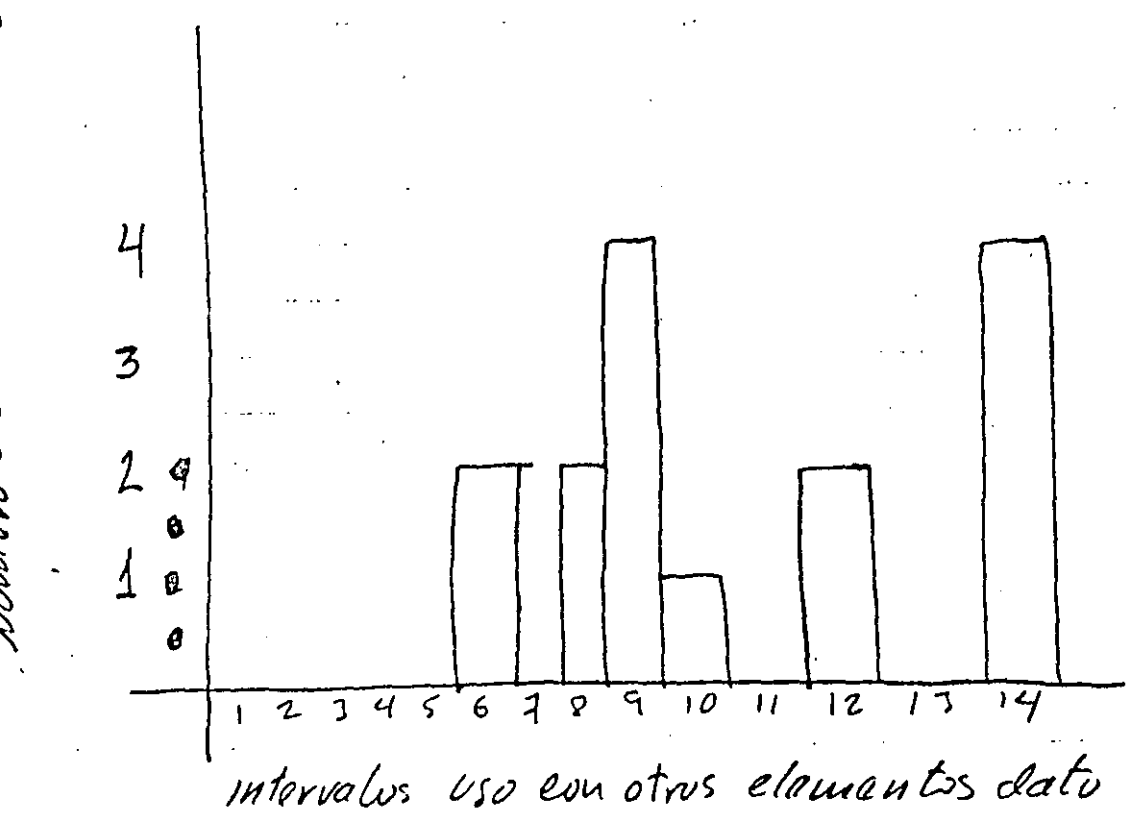
vector de alto uso .70

1	3
2	2
3	9
4	1
5	3
6	0
7	9
8	3
9	1
10	4
11	4
12	0
13	0
14	8
15	8

vector de radio de uso

1	21.42
2	16.66
3	100.00
4	7.14
5	33.33
6	0.0
7	100.00
8	21.42
9	7.14
10	44.44
11	40.00
12	0.0
13	0.0
14	100.00
15	100.00

HISTOGRAMA USO CON OTROS ELEMENTOS DATO



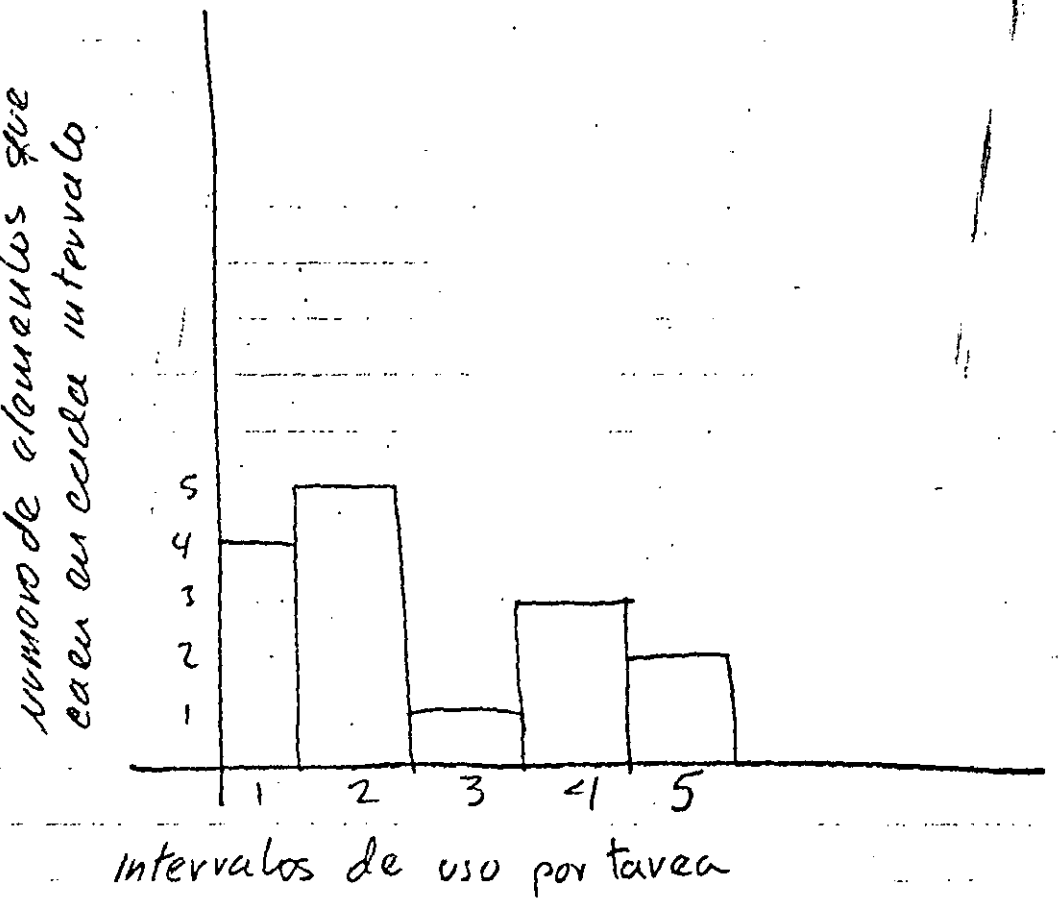
VÉCTOR DE TOTAL DE VÍNCULOS

1	14
2	12
3	9
4	14
5	9
6	12
7	9
8	14
9	14
10	9
11	10
12	6
13	6
14	0
15	8

FRECUENCIAS

0	
1	
2	
3	
4	
5	
6	2
7	
8	2
9	4
10	1
11	0
12	2
13	0
14	4

HISTOGRAMA USO A TRAVEZ DE TAREAS



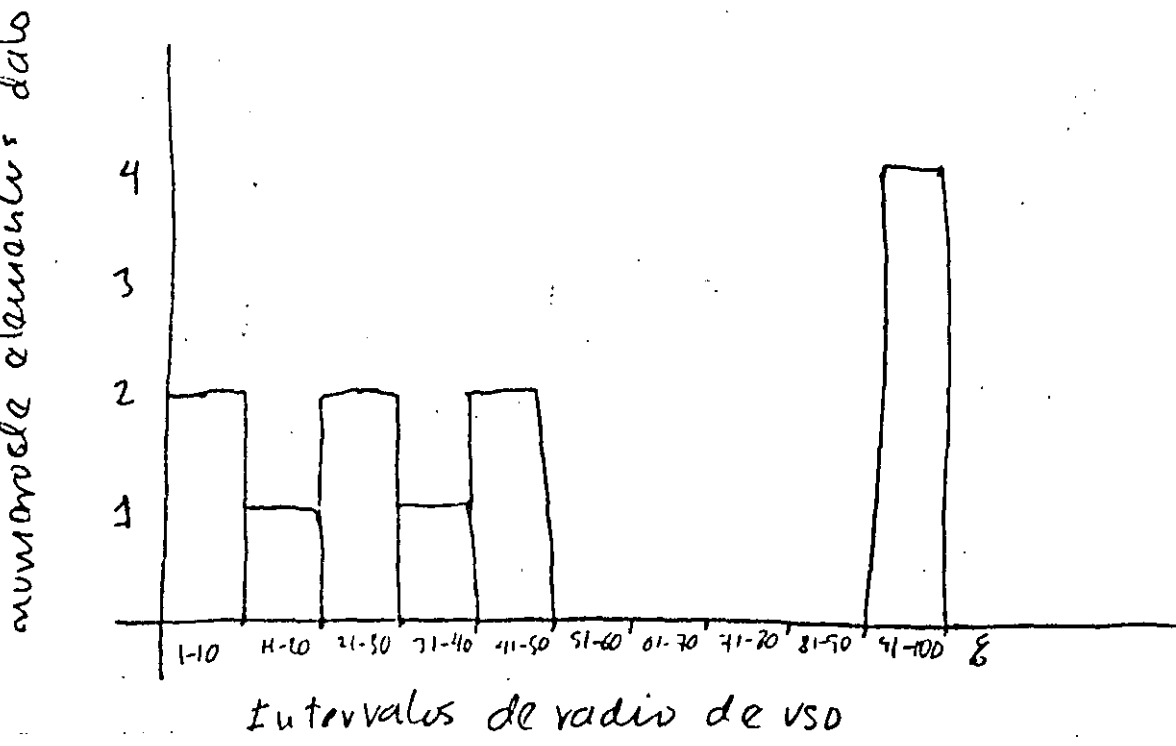
VECTOR DE CUENTA DE USO POR TAREA

1	4
2	4
3	1
4	5
5	2
6	3
7	1
8	4
9	5
10	2
11	2
12	2
13	2
14	1
15	1

FRECUENCIAS

0	0
1	4
2	5
3	1
4	3
5	2

HISTOGRAMA RADIO DE USO



VECTOR DE RADIO DE USO

1	21.42
2	16.66
3	100.0
4	7.14
5	33.33
6	0.0
7	100.0
8	21.42
9	7.14
10	44.44
11	40.0
12	0.0
13	0.0
14	100.0
15	100.0

FRECUENCIA

1-10	2
11-20	1
21-30	2
31-40	1
41-50	2
51-60	
61-70	
71-80	
81-90	
91-100	4

	USO A TRAVEZ DE LAS TAREAS	USO JUNTO CON OTROS ELEMENTOS DATO	RADIO DE USO
ALTO			
PROM			
BAJO			

	USO A TRAVES	USO CON OTROS DATOS	RADIO DE USO
ENTIDAD	ALTO	ALTO	BAJO
ATRIBUTO REFERENCIADO POR ENTIDAD NO UNICA	PROM	PROM	PROM
ATRIBUTO REFERENCIADO POR ENTIDAD UNICA	BAJO	BAJO	ALTO

Elemento dato	Uso a través de las tareas	Uso con otros datos	Radio de uso	Clasif.
1	4	14	21	Entidad
2	4	12	16	Entidad
3	1	9	100	
4	5	14	7	Entidad
5	2	9	33	
6	3	12	0	—
7	1	9	100	
8	4	14	21	Entidad
9	5	14	7	Entidad
10	2	9	44	
11	2	10	40	
12	2	6	0	—
13	2	6	0	—
14	1	8	100	
15	1	8	100	

ENTIDADES POR TAREA

TAREA	ENTIDADES
1	1, 2, 4, 8, 9
2	2, 4, 9
3	2
4	1, 4, 8, 9
5	1, 2, 4, 8, 9
6	1, 4, 8, 9

Candidatos a entidades no unicas	numero de apariciones juntos	Frecuencia
1, 2	2	33
1, 4	4	60
1, 8	4	60
1, 9	4	60
2, 4	2	33
2, 8	2	33
2, 9	3	50
4, 8	4	60
4, 9	5	83
1, 2, 4	2	33
1, 2, 8	2	33
1, 2, 9	2	33
1, 2, 4, 8	2	33
1, 2, 4, 9	2	33
1, 2, 4, 8, 9	2	33

Elementos dato usados por candidatos a ser
no unicas

Candidatos	Tareas en que aparece	Otros elementos
1, 4	1	3, 5, 6, 7, 10
	4	11, 12, 13
	5	6, 11, 14, 15
	6	6, 12, 13
1, 8	1	3, 5, 6, 7, 10
	4	11, 12, 13
	5	6, 11, 14, 15
	6	6, 12, 13
1, 9	1	3, 5, 6, 7, 10
	4	11, 12, 3
	5	6, 11, 14, 15
	6	6, 12, 13
4, 8	1	3, 5, 6, 7, 10
	4	11, 12, 13
	5	6, 11, 14, 15
	6	6, 12, 13
4, 9	1	3, 5, 6, 7, 10
	2	5, 10
	4	11, 12, 13
	5	6, 11, 14, 15
	6	6, 12, 13

Identificación de Vinculos

Politicas

1= Un cliente es asignado a un vendedor

2= el que toma la llamada es responsable del proceso de la v

3= Se aceptan ordenes solo del vendedor que maneja a el c

DIRECTORIO DE ALUMNOS DEL CURSO "DISEÑO DE BASE DE DATOS"
IMPARTIDO EN ESTA DIVISION DEL 17 DE MAYO AL 14 DE JUNIO 1985.

1.- ACOSTA GODINEZ VICTOR
S. A. R. H.
ANALISTA
REFORMA No. 133-6o. PISO
COL. SAN RAFAEL
DELEGACION CUAUHTEMOC
566-87-24

MORELOS No. 510 DEPTO. 304
DELEGACION IZTAPALAPA
566-89-24

2.- BARRUETA AVILA JOSE ALEJANDRO
GRUPO IPESA
ING. DE PROYECTOS
AV. SNA LORENZO No. 153
COL. DEL VALLE
DELEGACION BENITO JUAREZ
03100 MEXICO, D.F.
575-40-77

AV. SAN BERBABE No. 210
COL. M. CONTRERAS
10100 MEXICO, D.F.
595-21-45

3.-- CALLEJAS CASTRO MANUEL
INDUSTRIAS METALICA INTEGRADO
JEFE DE SISTEMAS
KM. 10.5 CARR. OCOYOACAN
TIANGUISTENGO ED_. DE MEXICO

AV. UNIVERSIDAD No. 1900
EDIF. 34-201
COL. COPILCO UNIVERSIDAD
DELEGACION COYOACAN
658-25-20

4.- CAÑEDO SARABIA MIGUEL ANGEL
DIREC. DE INFORMATICA I.B.M.
ANALISTA
CALLE SATURNO COL. LINDAVISTA
DELEGACION GUSTAVO A. MADERO

TRIPOLI No. 507-118
COL. PORTALES
DELEGACION BENITO JUAREZ
03300 MEXICO, D.F.
539-65-51

5.- CARRANZA ECHEVERRIA LOURDES
I.S.S.S.T.E.
JEFE DE DEPARTAENTO
AV. JUAREZ No. 140-11o. PISO

RAYA OLA VERDE No. 379
COL. IZTACALCO
08840 MEXICO, D.F.
579-17-46

6.- CASTILLO GUERRERO ANGEL ENRIQUE
KANAN No. 1417
PROGRAMADOR COBOL
AÑIL No. 109 ESQ. ROSINA
COL. GRANJAS DE MEXICO
657-88-22

AV. TAXQUEÑA No. 4303
DELEGACION COYOACAN
04200 MEXICO, D.F.
549-20-74

7.- CELIS DIAZ HELMER
POLYGRAM DISCOS
GERENTE DE SISTEMAS
MIGUEL ANGEL DE QUEVEDO No. 531
DELEGACION COYOACAN
658-84-00

MIGUEL ANGEL DE QUEVEDO No. 531
DELEGACION COYOACAN
658-84-00

8.- CERVANTES OLVERA CECILIA
COMISION FEDEFAL DE ELECTRICIDAD
ACTUARIO
LEON TOLSTOY No. 29-14o. PISO
COL. ANZURES
545-91-32

COLINA DE LA GACELA No. 47
SATELITE, EDO. DE MEXICO
53140 MEXICO, D.F.
572-06-83

9.- CIPRIAN AVILA JORGE ROMEO
COMISON FEDERAL DE ELECTRICIDAD
TECNICO EN INVEST. ECONOMICA
TOLSTOY No. 29-14o.PISO
COL. ANZURES
DELEGACION MIGUEL HIDALGO
545-91-32

ANDADOR 37 EDIF. 33 A DEPTO. 402
DELEGACION GUSTAVO A. AMDERO
391-35-38

10.- CORTEZ CAMACHO FRANCISCO
IPESA
ANALISTA DE SISTEMAS
SAN LORENZO No. 153-4o. PISO
COL. DEL VALLE
575-09-49

VENTURINA No. 156
COL. ESTRELLA
DELEGACION GUSTAVO A. MADERO
07810 MEXICO, D.F.
577-24-55

11.- DAVIGHI HERRERA ALDO CLAUDIO
KANAN HIT
GERENTE DE INFORMATICA
AÑIL No. 109 ESQ. RESINA
COL. GRANAJAS MEXICO
657-88-22

AV. TAXQUEÑA No. 1303
DELEGACION COYOACAN
04200 MEXICO, D.F.
549-20-74

12.- DE GANTE SANCHEZ MARCOS RAUL
S. C. T. DIREC. GRAL. ING, DE SIST.
ANALISTA DE SISTEMAS
AV. MICHOACAN S/N
COL. TEPALCATES
DELEGACION IZTAPALAPA
09201 MEXICO, D.F.
592-00-77 ext. 391

AV. PERIFERICO EDIF. 6 ENT. 1 DEPTO.
COL. U. VICENTE GUERRERO
DELEGACION IZTPALAPA
04200 MEXICO, D.F.

13.- DE LA ROSA GARCIA LUCIA
CENTRO PANAMERICANO DE ECOLOGIA
HUMANA Y SALUD D.P.S.
RANCHO DE GUADALUPE
APARTADO POSTAL 37-473
06696 MEXICO, D.F.

CERRO DE LA ESTRELLA No. 167
COL. CAMPESTRE CHURUBUSCO
DELEGACION COYOACAN
04200 MEXICO, D.F.
544-50-26

14.- DEL MORAL GARGIA MA. TERESA
COMISION FEDERAL DE ELECTRICIDAD
ANALISTA PROGRAMADOR
L. TOLSTOI No. 29-12o. PISO
COL. ANZURES
DELEGACION CUAUHTEMOC
593-71-33

CENTENO No. 858 H-303
COL. GRANJAS MEXICO
DELEGACION IZTACALCO
650-67-08

15.- DURAN ACOSTA ALBERTO
TELEFONOS DE MEXICO
INGENIERO DE SOFTWARE
ERNESTO PUGIBET No. 2-3er. PISO
COL. CENTRO
DELEGACION CUAUHTEMOC
706-00-02

AV. AVILA CAMACHO No. 467
COL. PERIODISTA
DELEGACION MIGUEL HIDALGO
11220 MEXICO, D.F.
557-30-28

16.- ESCUTIA ACOSTA RAUL
S. A. R. H.
PROGRAMADOR ANALISTA
AV. INSURGENTES SUR No. 30-32
COL. JAUREZ
DELEGACION CUAUHTEMOC
06600 MEXICO, D.F.
591-18-30

ROSA VENUS No. 23 MOLINO DE ROSAS
DELEGACION ALVARO OBREGON
01470 MEXICO, D.F.
680-74-43

17.- ESPINOSA GONZALEZ CARLOS
POLYGRAM DISCOS
SUBGERENTE SISTEMAS
MIGUEL ANGEL DE QUEVEDO No. 531
DELEGACION COYOACAN
658-84-00

CASCADA No. 320-A
DELEGACION BENITO JUAREZ
03300 MEXICO, D.F.

18.- FERREIRA TREVIÑO HECTOR H.
SERVICIOS Y ASESORIAS EMPRESARIALES
CONSULTOR ASOCIADO
NICOLAS SAN JUAN No. 225
COL. DEL VALLE
DLELGACION BENITO JUAREZ
03100 MEXICO, DF.
536-50-83

DIVISION DEL NORTE No. 24-12
COL. DEL VALLE
DELEGACION BENITO JUAREZ
03100 MEXICO, D.F.

19.- FLORES FLORES FRANCISCO JAVIER
DIREC. GRAL. SERV. DEL D. D.F.
JEFE DE INFORMATICA
TEPIC No. 37
COL. ROMA SUR
564-17-41

RINCON DEL AMOR No. 26
COL. BOSQUE RESIDENCIAL DEL SUR
DELEGACION XOCHIMICLO
16010 MEXICO, D.F.
676-67-29

20.- GARCIA CRUZ MARIA TERESA
DIREC. GRAL. OBRAS
OPERADOR DE SISTEMAS
AV. REVOLUCION No. 2045
DIUDAD UNIVERSITARIA
DELEGACION COYOACAN
550-57-83

DR. TERRES No. 88-16
DELEGACION CUAUHTEMOC
06720 MEXICO, D.F.
588-53-55

21.- GARCIA ESPINOSA J. JESUS
DIREC. GRAL. OBRAS UNAM
OPERADOR
AV. REVOLUSION No. 2045
CIUDAD UNIVERSITAIRA
DELEGACION COYOACAN
550-52-15 ext. 4113

SUR 111 A No. 711
COL. SECTOR POPULAR
DELEGACION IZTAPALAPA
09060 MEXICO, D.F.
582-76-84

22.- GARCIA CARRANZA MA. TERESA
POLIGRAM DISCOS
ANALISTA PROGRAMADOR
MIGUEL ANGEL DE QUEVEDO No. 531
DELEGACION COYOACAN
658-84-00

SUIZA No. 16-2
DELEGACION BENITO JUAREZ
03300 MEXICO, D.F.
532-60-05

23.- GARCIA JUAREZ JOSE DE JESUS
D.G.T. S. C. T.
ANALISTA ESPECIALIZADO
EJE CENTRAL LAZARO CARDENAS 567
COL. NARVARTE
530-30-60

AV. SAN FELIPE LOTE 20 MAN 521
COL. URSULA COAPA
DELEGACION COYOACAN
04918 MEXICO, D. F.
677-90-82

24.- GARNICA HUITRON GONZALO
S. C. T.

25.- GONZALEZ DURAN NORBERTO
SEGUROS DE MEXICO, S.A.
JEFE DE SISTEMAS BAÑOS
AV. UNIVERSIDAD No. 1200
DELEGACION COYOACAN
534-00-34

AÑO 30 DE MANUELA SAENZ No. 221-B
DELEGACION COYOACAN
04480 MEXICO, D.F.
552-23-62

26.- IBARRA BLANCAS GUILLERMO
ATISA-ATKINS
LIDER DE PROYECTO
BAHIA DE CORRIENTES No. 77
COL. ANZURES
DELEGACION MIGUEL HIDALGO
11300 MEXICO, D.F.
250-82-11

VELAZQUEZ DE LEON No. 14-6
DELEGACION CUAUHTEMOC
06470 MEXICO, D.F.
546-69-76

27.- JUAREZ REYES RAFAEL
TERMOS Y TRANSPORTES
PROGRAMADOR
MANUEL SALAZAR No. 130
COL. SAN JUAN THIHUACA
DELEGACION AZCAPOTZALCO
382-70-76

GRAL. MARIANO SALAS No. 78-A
COL. MARTIN CARRERA
DELEGACION GUSTAVO A. MADERO
781-30-67

28.- JUAREZ TELLEZ JUAN MANUEL
DIREC. GRAL. OBRAS UNAM
OPERADOR DE SISTEMAS
AV. REVOLUCION No. 2045
COL. PEDREGAL
DELEGACION COYOACAN
550-52-15 ext. 4113

MANZ. I EDIF. 235 DEPTO. 101 SEC. 9
UNIDAD C.T.M. CULHUACAN
DELEGACION COYOACAN
04480 MEXICO, D.F.

29.- LACROIX MARTINEZ GUSTAVO

MOTOZINTLA No. 92
COL. NARVARTE
DELEGACION BENITO JUAREZ
03600 MEXICO, DF.
539-63-01

30.- LANDEROS AVILES CARLOS FLORENTINO
UNIVERSIDAD AUTONOMA DE CHAPINGO
INVESTIGADOR
TEZCOCO, MEXICO

ZACATECAS No. 178-3
564-83-23

31.- LOPEZ CERVANTES MIGUEL ANGEL
SUBDIREC. PROCESAMIENTO DATOS
U. S. E. D. XALAPA
TECNICO MANTO. SIST. COMPUTACION
CARRETERA XALAPA, VERACRUZ KM. 4.5
XALAPA, VER. 91000

JORULLO No. 16
COL. AGUACATAL
91134 XALAPA, VER.

32.- LOPEZ LOPEZ ROBERTO
TELEFONOS DE MEXICO
INGENIERO DE SOFTWARE
ERNESTO PUGIBET No. 12 EDIF. AXE
3er. PISO
COL. CENTRO
DELEGACION CUAUHTEMOC
521-68-14

EDIF. 20 DEPTO. 401
UNIDAD LOMAS DE BECERRA
DELEGACION ALVARO OBREGON
563-98-38

33.- LOPEZ LOPEZ ROBERTO
ACEROS FORTUNA

34.- LOPEZ MENDIZABAL VICTOR
S. A. R. H.
ANLAISTA
AV. INSURGENTES SUR No. 30-2o. PISO
COL. JUAREZ
DELEGACION CUAUHTEMOC
591-18-55

ING. JOSE J. REYNOSOS No. 68
COL. CONSTITUYENTES DE 1917
DELEGACION IZTAPALAPA
09260 MEXICO, D.F.
691-37-79

35.- LUIS AGUILAR ALBERTO

MANZANA 3 EDIFICIO J DEPTO. 01
U. CANDELARIA DE LOS PATOS
539-65-51

36.- MARTINEZ CAMACHO JOSE LUIS
PEMEX
JEFE DE ANALISTAS
AV. MARINA NACIONAL No. 329
COL. ANAHUAC

AV. 508 No. 185
COL. SAN J. DE ARAGON
DELEGACION GUSTAVO A. AMDERO
551-21-79

37.- MIRELES CISNEROS MA. GUADALUPE
DIREC. GRAL. OBRAS UNAM
PROGRAMADOR
AV. REVOLUCION No. 2045
DELEGACION COYOACAN
550-57-83

JABONERO No. 8
COL. PROGRESISTA
DELEGACION VENUSTIANO CARRANZA
15370 MEXICO, D.F.
789-03-53

38.- MOLINA VILCHIS MARIA AURORA
DIRE. GRAL. TELECOMUNICACIONES
ANALISTA ESP. SIST. DE COMPUTACION
EJE CENTRAL LAZARO CARDENAS
COL. NARVARTE
530-60-09

SUR 119 MZ. 40 LOTE 39
DELEGACION IZTACALCO
4001 MEXICO, D.F.
650-54-30

39.- MONTERRUBIO ESCOBAR MANUEL
PEMEX
ANALISTA PROGRAMAS
MARINA NACIOANL No. 329
COL. ANAHUAC
250-26-11 ext. 21902

RET. 36 DE CECILIO ROBELO EDIF. 890-B-6
DELEGACION COYOACAN
15900 MEXICO, D.F.

40.- MORENO GUTIERREZ ARTURO CARLOS
ICI DE MEXICO, S.A.
QUIMICIO INVESTIGADOR
CALLE BELLO S/N
COL. SAN JUNA IXHUATEPEC
TLALNEPANTLA EDO. DE MEXICO
754-44-77

CONVENTO DE LA MERCED No. 39
54050 EDO. DE MEXICO
398-58-21

41.- NAVA GUERRA RAFAEL
S. A. R. H.
ANALISTA DE SISTEMAS
INSRUGENTES SUR No. 30
COL. PORTALES
DELEGACION CUAUHTEMOC
591-18-35

INSURGENTES No. 30
COL. JAUREZ
DELEGACION CUAUHTEMOC
591-18-35

42.- PARRAS VALDOVINOS ENRIQUE
PEMEX
JEFE DE ANLAISTAS
AV. MARINA NACIONAL No. 329
TORRE 14o. PISO
DELEGACION CUAUHTEMOC

COTINGAS No. 72
ATIZAPAN DE ZARAGOZA
54500 MEXICO, D.F.
574-66-36

43.- PEREZ RAMOS EVELIA
S. C. T.
ANALISTA TECNICO
AV. XOLA Y UNIVERSIDAD
COL. NARVARTE
DELEGACION BENITO JUAREZ
519-51-34

XAVIER SORONGO No. 260
DELEGACION BENITO JUAREZ
03530 MEXICO, DF.
519-51-34

44.- PICAZO SALINAS JORGE
S. C. T.

45.- RAMIREZ MAGAÑA ROBERTO
S. C. T. OBRAS MARITIMAS
JEFE DE OFICINA
PROVIDENCIA No. 807-4o. PISO
COL. DEL VALLE
DELEGACION BNEIOT JUAREZ
523-45-38

AV. CLAVERIA No. 224-106
COL. CLAVERIA
DELEGACION AZCAPOTZALCO
02080 MEXICO, D.F.
523-45-38

46.- RODRIGUEZ MALDONADO MARTIN B.
SECRETARIA DE COMUNICACIONES
Y TRANSPORTE
ANALISTA DE SISTEMAS
530-60-09

MISANTLA No. 38
DELEGACION CUAUHTEOC

47.- ROMANO DIAZ JUAN
PETROLEOS MEXICANOS
JEFE DE DEPTO. DE SISTEMAS
MARINA NACIONAL No. 329
COL. ANAHUAC

DELFINES C
FUENTES DE SATELITE
572-69-89

48.- ROMERO RUIZ RUBEN
ENEP ACATLAN
PROFESOR DE ASIGNATURA "A"
AV. ALCANFORES S/N
COL. ACATLAN

PASEO DE ITALIA No. 69
LOMAS VERDES III
53120 NAUCALPNA DE JUAREZ
562-98-41

49.- ROSALES PADILLA JORGE LUIS
INSTITUTO MEXICNAO DEL PETROLEO
AV. LAZARO CARDENAS No. 152
COL. SAN BARTOLO ATEPEHUACAN
DELEGACION GUSTAVO A. MADERO
567-68-00 ext. 20414

AV. SAN ISIDRO No. 696-C-304
DELEGACION ATZCAPOTZALCO
02710 MEXICO, D.F.
567-80-15

50.- SANCHEZ ALVAREZ SERGIO A.
ASESORA LENIAL S. C.
GERENTE DE SISTEMAS
FERROCARRIL No. 17
FRAC. ALCE BLANCO
NAUCALPNA DE JUAREZ
576-53-55

TURIN No. 41-10
COL. JAUREZ
06600 MEXICO? D.F.
552-02-24

51.- SANCHEZ ZARATE MIGUEL A.
S. A. R. H.

52.- SERAFIN CANDELARIO ILDEFONSO
S. C. T. INGENIERIA DE SISTEMAS
ANALISTA PROGRAMADOR
MICHOACAN S/N
COL. TEPALCATES
DELEGACION IZTAPALAPA

AGAPANDO No. 18
COL. LOS ANGELES
DELEGACION IZTAPALAPA
09710 MEXICO, D.F.

53.- SOTO LOPEZ EDUARDO
C. B. S.
SOPORTE TECNICO
AVE. 16 DE SEPTIEMBRE No. 275
58000 NAUCALPAN DE JUAREZ
576-03-33

CRUZ GALVEZ No. 195
COL. NUEVA SANTA MARIA
556-68-60

54.- TEJERO ANDRADE LUCAS RICARDO
COMISION FEDERAL DE ELECTRICIDAD
RODANO No. 14
COL. CUAUTEMOC
DELEGACION CUAUHTEMOC
06598 MEXICO, D.F.
553-71-33 ext. 2183

TAJIN No. 679
DELEGACION BENITO JUAREZ
03600 MEXICO, D.F.

55.- TREJO FLORES LIDIA
DIREC. GRAL. INGENIERIA DE SISTEMAS
ANALISTA PROGRAMADOR
AV. MICHOACAN S/N
COL. TEPALCATES
DELEGACION IXTAPALAPA
09201 MEXICO, D.F.
691-70-56

GONZALEZ CAMARENA No. 9
COL. JACARANDAS
DELEGACION IXTAPALAPA
09280 MEXICO, D.F.
691-77-69

56.- TREJO GUTIERREZ RAUL
DIREC. GRAL. TELECOMUNICACIONES
JEFE DEPTO. TRANSMISION DATOS
LAZARO CARDENAS No. 567
COL. NARVARTE
538-81-01

JUPITER No. 46
DELEGACION CUAUHTEMOC
583-40-18

57.- VALENCIA NAVARRETE EDUARDO
DIREC. GRAL. ING. DE SISTEMAS
LIDER DE PROYECTO
AV. TELECOMUNICACIONES S/N
COL. DEL MORAL
IZTAPALAPA
697-00-77

AV. TELECOMUNICACIONES S/N
COL. DEL MORAL
IZTAPALAPA

58.- VALDES ALMANZA GONZALO
INSTITUTO MEXICANO DEL PETROLEO