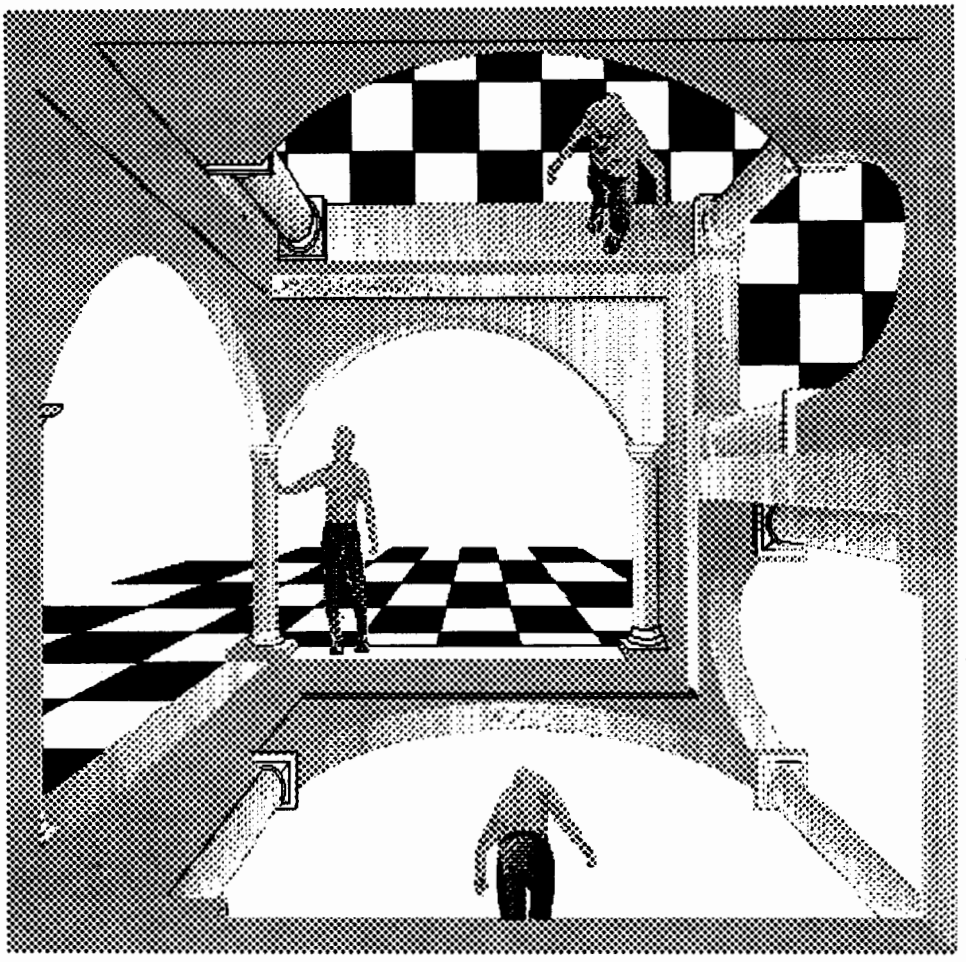
# SIMULACION DIGITAL

J. MACLOVIO SAUTTO VALLEJO



**DEPARTAMENTO DE INGENIERIA DE SISTEMAS**

**División de Estudios de Posgrado** **Facultad de Ingeniería**

EJ L 715034

F-DEPFI
MISC
0014
ej.6

BIBLIOTECA DE LA UNIVERSIDAD NACIONAL DE MEXICO
DEPFI

Como parte de las actividades del Departamento de Ingeniería de Sistemas de la División de Estudios de Posgrado, Facultad de Ingeniería de la UNAM, nos hemos propuesto el desarrollo de una serie de apuntes que sirvan de apoyo a los distintos cursos que se imparten y, desde luego, como material de referencia o de lectura para quienes así lo deseen.

En dichos apuntes se desea mantener un alto nivel académico de manera que contribuyan a una sólida formación teórica del alumno, pero al mismo tiempo se intenta mantenerlo cerca de las aplicaciones de tales conocimientos, como es en síntesis la aplicación general del Posgrado de Ingeniería.

Agradecemos al Programa de Apoyo a las Divisiones de Estudios de Posgrado (PADEP), por financiar éstos trabajos.

Departamento de Ingeniería de Sistemas
División de Estudios de Posgrado
Facultad de Ingeniería
UNAM

G(2) 23375

# PRESENTACION

Vivimos momentos de intensa producción en todas las áreas del conocimiento. Una de nuestras tareas urgentes (de los docentes) es insertar a las instituciones educativas en el centro del desarrollo científico y tecnológico. Debemos enfrentar con decisión los retos de nuestro tiempo, y dejar de esperar que lleguen las cosas que necesitamos, porque con seguridad se están haciendo en algún país desarrollado, debemos empezar a producir lo que requerimos, debemos combatir la política de "esperar a que lleguen las cosas y no producir".

En particular el desarrollo y aplicación de la simulación digital en la Ingeniería de Sistemas, es cada día mayor y la literatura al respecto es incipiente, esta es otra razón por la que se desarrolló este trabajo.

Estoy convencido de que cada vez haremos mejor las cosas, pero es necesario empezar a producirlas y poner nuestro mejor empeño en ello.

Quiero agradecer a la M.I. Idalia Flores y a las autoridades del Posgrado de Ingeniería de la UNAM la oportunidad que me dieron para la elaboración de este trabajo. A PADEP por su financiamiento y al M.C. Efrén Marmolejo director de la Facultad de Matemáticas de la UAG, por el apoyo que me ha brindado para poder realizar mis estudios doctorales

J. MACLOVIO SAUTTO VALLEJO[1]

[1]Profesor de tiempo completo de la facultad de matemáticas de la UAG. Estudiante del doctorado en Investigación de Operaciones en la DEPFI, UNAM.

# INDICE

# INTRODUCCION

*Es el acontecimiento central, la clave de la comprensión de los años inmediatamente venideros. Es un acontecimiento tan profundo como aquella primera ola de cambio desencadenada hace diez mil años por la invención de la agricultura, o la sísmica segunda ola de cambio disparada por la revolución industrial. Nosotros somos hijos de la transformación siguiente, la tercera ola.*
*Tratamos de encontrar palabras para describir toda la fuerza y el alcance de este extraordinario cambio. Algunos hablan de una emergente era espacial, Era de la información, Era electrónica o Aldea global. Zbigniew Brzezinski nos ha dicho que nos hallamos ante una " era tecnetrónica ". El sociólogo Daniel Bell describe el advenimiento de una " sociedad postindustrial ". Los futuristas soviéticos hablan de la RCT, la " revolución científico-tecnológica ". Yo mismo he escrito extensamente sobre el advenimiento de una " sociedad superindustrial ". Pero ninguno de estos términos incluido el mío, es adecuado.*

*Alvin Toffler "La tercera ola".*

La Investigación de Operaciones (I. de O.), forma parte de un conjunto de nuevas ciencias, producto directo de la revolución científico tecnológica, cuya oleada de transformación ha venido y continua trastocando nuestras formas de vida. A partir de mediados del presente siglo, hemos sido testigos de un impresionante cambio sustentado en la electrónica, más concretamente en la computación. A principios de los 60's, cuando las primeras grandes computadoras irrumpieron en las industrias, ministerios y universidades, se pensó que esta nueva tecnología, que ese gran "megacerebro" electrónico sería capaz de centralizar y controlar toda la información necesaria para conducir una institución, así se crearon una serie de mitos alrededor de las computadoras como "solo los superdotados pueden manipularlas", "solo manejan ceros y unos y es difícil interpretar los resultados", etc. La verdad era, que durante esa época, más que una superinteligencia se requería una paciencia de monje tibetano para programar, corregir y leer los resultados obtenidos, y sobre todo, obtener tiempo de máquina.

Una década después, en los años 70, los microprocesadores llegaron, los tiempos de jalarse los cabellos, arrastrar los pies y de ir a mendigar unos pocos

minutos de tiempo de computadora, llegaba a su fin. La revolución de las micros dio al traste con el monopolio de la información, los costos por el uso de computadoras se fue a tierra, y sobre todo desmitificó a la computación misma; ahora la computadora era accesible a todos y cada día aparecían nuevos programas que volverían más accesible el uso de los ordenadores, el microprocesador fue el sepulturero de la idea "solo los superdotados usan las computadoras".

En los últimos 10 años, palabras como software y hardware, se han convertido de uso corriente en nuestras universidades, empresas e instituciones públicas. El desarrollo tecnológico permite que el acceso a la información sobre el desarrollo científico y tecnológico sea más expedito, en la actualidad es posible recibir publicaciones científicas y de divulgación, editadas en cualquier parte del mundo, el mismo mes en que se publican.

El desarrollo del software especializado descansa sobre dos pilares, el primero constituido por herramientas de la computación tales como los lenguajes y los procedimientos de apoyo, y el segundo por los conocimientos, habilidad e ingenio de nuestros diseñadores y programadores. El acceso a las herramientas de programación, está al alcance de todos, de donde el desarrollo de software especializado es un problema práctico, donde nuestra pertenencia a un país tercer mundista no es pretexto para evadir este nuevo reto de nuestro tiempo.

Por primera vez, podemos plantearnos competir con cualquiera, en el desarrollo de software, que resuelva problemáticas de las diversas empresas y/o instituciones públicas.

Este es un texto de introducción a la simulación digital, las reflexiones anteriores, buscan generar en el lector la actitud de incursionar e impulsar esta forma de análisis, el terreno es virgen y existe una infinidad de posibilidades de desarrollo como por ejemplo:

La elaboración de modelos iterativos, con un usuario neófito en programación y

El desarrollo de modelos de simulación de grandes instituciones mediante redes.

... Y todas aquellas aplicaciones que Ud. considere importantes.

# CAPITULO I

# FUNDAMENTOS DEL MODELADO

*"...todo retrato pintado con emoción es un retrato del artista, no del modelo. Este no es más que el accidente, la ocasión. No es él el revelado por el pintor, sino mas bien éste quien, sobre el lienzo pintado, se revela a sí mismo".*

*Oscar Wilde: "El retrato de Dorian Gray"*

La simulación es una de las herramientas de análisis mas poderosas de que disponen quiénes se dedican al diseño y operación de procesos o sistemas complejos. El concepto de simulación es simple e intuitivamente atractivo, le permite al usuario experimentar con sistemas reales o ficticios en casos en los que de otra manera esto sería imposible o impráctico.

El modelado de simulaciones se basa principalmente en la computación, la matemática, la probabilidad y la estadística, es por esto que el propósito de éste primer capítulo es presentar las bases de la modelación en general y en particular la modelación de sistemas, así como la forma en que se puede utilizar la computadora para éste propósito; se presenta en la primera sección un breve desarrollo histórico de la computadora así como una tabla donde se muestran los avances mas significativos de las computadoras en los últimos veinte años. En la segunda sección se define un modelo y se describe la función de los modelos. En la tercera se da una clasificación de modelos analíticos y heurísticos y se describe cada uno de ellos, finalmente en la cuarta sección se desarrolla lo que es la modelación de sistemas.

## *1.1 LA COMPUTADORA.*

Así como la máquina de vapor sentó las bases materiales para el inicio de la Revolución Industrial, la computadora, la niña consentida de la electrónica, sentó las bases materiales para esta nueva revolución, que en pocos años ha transformado nuestras formas de vida. Poco a poco, sin darnos cuenta, el ***cambio*** se ha convertido en la única constante de nuestra forma de vida.

Esta nueva revolución, ha impactado todas las esferas de la sociedad. Dentro del conocimiento humano, ha facilitado el desarrollo de las viejas áreas del conocimiento y creado nuevas ciencias, tales como la informática, la cibernética y desde luego la Investigación de Operaciones. Basta dar un simple repaso a los hechos históricos más relevantes durante los últimos 40 años.

En 1946, se construye la primera computadora (ENIAC), y con ella la posibilidad real de poder realizar cálculos aritméticos miles o tal vez millones de veces más rápido que los seres humanos. Este hecho, permitió la concretización de métodos de solución iterativos que involucran un gran número de operaciones aritméticas. Un año después en el verano de 1947 George Dantzig desarrolló completamente el método simplex para resolver problemas de programación lineal. Sin embargo fue hasta enero de 1952 cuando se realizó la primera solución exitosa de un problema de programación lineal en una computadora electrónica de "alta velocidad", en la maquina SEAC del National Bureaus of Standards.

En la década de los 50's, se adelantó la teoría necesaria y se establecieron algoritmos que permitían resolver problemas de programación lineal, programación dinámica, líneas de espera, inventarios, etc.

En sus primeros 20 años, la Investigación de Operaciones (I. de O.) elaboró la teoría para enfrentar exitosamente, la mayoría de las problemáticas que se generan en la pequeña y mediana industria. Y para mediados de los años 70's, ocupaba ya un lugar importante en la Industria y Universidades, popularizándose el estudio de sus técnicas en los países desarrollados. En estos mismos 20 años la electrónica a partir del invento del transistor en 1948 y de los circuitos integrados en 1959 permitió el desarrollo de nuevas y mejores computadoras que año con año se mejoran, en rapidez, precisión y en el manejo de grandes volúmenes de información.

Los 70's, vieron nacer una nueva generación de computadoras, las

micros, que en pocos años, han alcanzado dimensiones verdaderamente impresionantes, en rapidez, en volúmenes de información que manejan, y sobre todo en abaratamiento de sus costos. Las microcomputadoras que hoy encontramos en cualquier Universidad, son superiores a las primeras grandes computadoras que se desarrollaron durante los años 50's, lo que ha permitido que la I. de O. las utilice con mayor frecuencia y desarrolle la simulación digital para realizar modelos cada vez más apegados a la realidad, cuya modelación matemática no es posible.

La tabla 1.1 nos da una idea de la rápida evolución de la industria de las microcomputadoras.

Durante la década de los 80, se empezó a hablar de la programación en paralelo y de la utilización de redes de computadoras, la programación en paralelo supone la existencia de nuevas arquitecturas de computadoras, con varios procesadores. Mientras que la utilización de redes de microcomputadoras posibilitan el uso de las viejas microcomputadoras. Cualquiera que sea el desarrollo de las computadoras, en la década de los 90's tendrá la posibilidad de programar en paralelo o simular dicha programación mediante redes de computadoras abriendo las puertas a nuevas épocas en el desarrollo de las técnicas de la I. de O. sobre todo de la simulación digital.

TABLA 1.1

| AÑO | ACONTECIMIENTO |
|---|---|
| 1948 | John Bardeen, Walter Brattain y William Shockley de los laboratorios Bell inventan el transistor. |
| 1959 | La Texas Instruments descubre el primer circuito integrado. |
| 1964 | John G. Kemeny y Thomas E. Kurtz desarrollan el lenguaje de programación BASIC en el Dartmouth College. |
| | La Digital Equipment Corp. anuncia la minicomputadora PDP-8 con un costo de $16, 200. |
| 1970 | Scientific American publica en la sección de Martin Gardner "Recreaciones Matemáticas" el juego de la vida de John Conway. |

1971 La Intel Corp. pone el microprocesador de 4-bits 4004 en un sólo chip, su precio inicial es de $200.

1972 La Intel Corp. introduce su primer microprocesador de 8-bits 8008.

1973 Scelbi Computer Consulting ofrece su computadora Scelbi- 8H basada en el 8008 con 1 k. byte de memoria en $565.

1975 OTOÑO. Sphere Corp. lanza la computadora Sphere I (6800, 4k bytes de RAM, monitor ROM, teclado, interfase de video, por $650)

SEPT. IBM anuncia la IBM , la primer computadora de bolsillo (con BASIC, 16 k bytes y sistema de almacenamiento en cartucho por $9 000).

1976 PRIMAVERA Texas Instruments anuncia su TMS 9000, el primer microprocesador de 16- bits

ABRIL Se conforma la Apple Computer Inc.

DICIEMBRE Shugart lanza su lectora de discos de 5 pulg. por $390.

1977 INVIERNO Ohio Scientific Instruments ofrece la primer microcomputadora con Microsoft (punto flotante) BASIC en ROM.

ABRIL Jim Warren organiza el Primer Congreso de Computación de la Costa Oeste en San Francisco, donde se presentan la Apple II y la Commodore PET.

JUNIO La Apple Computer Inc. lanza su primer anuncio (6502. 16k bytes de RAM, ROM y monitor integrado con 16 k bytes en ROM, teclado, interfase para casete, tarjeta principal con 8 ranuras de expansión, controles para juegos, interfase para gráficas-textos expandible a 48 k bytes en RAM) con un costo $ 2638

1978 MAYO Ken Bowles describe en la revista Byte el sistema operativo basado en lenguaje PASCAL UCSD.

DICIEMBRE Epson America Inc. lanza la impresora de matriz de

puntos MX-80, su alto perfeccionamiento y bajo precio apabulla a los competidores y los forza a bajar los precios en el mercado de las impresoras.

1979 Wayne Ratliff desarrolla el programa de base de datos Vulcan que mas tarde se convertirá en dBASE II.

Video juegos como Pac Man y Space Invaders se convierten en la gran locura.

Intel introduce el 8088 que se convertirá en el corazón de la IBM PC.

1980 FEBRERO Sinclair Research lanza su computadora ZX80, la primer microcomputadora con un costo menor a $200 (consta de 1k byte de RAM, 4k en ROM con BASIC integrado, teclado de membrana plástica). Su sucesor el ZX81, lo vendió mas tarde Timex en menos de $100 antes de que Timex dejara el mercado de las microcomputadoras.

Satellite Software International, mas tarde WordPerfect Corp., anuncia la primer versión del WordPerfect para computadoras de datos generales.

1981 AGOSTO IBM introduce la computadora personal IBM PC ( 8088, 64k bytes de RAM, 40 k bytes de ROM, lectora de discos de 5 pulg. con un costo de $ 3005) lo que legitima la industria de las microcomputadoras al resto del mundo y establece la preeminencia de la familia de procesadores Intel 8088 y el sistema operativo MS-DOS de Microsoft.

1982 VERANO El lenguaje de programación LOGO se hace disponible para varias computadoras, mas notablemente para Apple II y T1-99/4A.

OCTUBRE Lotus Development Corp. lanza su paquete de graficación y hoja de cálculo 1-2-3 con capacidad de manejo de listas para IBM PC. Su velocidad y capacidad permiten reemplazar al Visicalc y su combinación de varias funciones en un programa da comienzo al movimiento conocido como "software integrado" en microcomputadoras.

DICIEMBRE Volition Systems presenta su primer implantación del lenguaje Modula-2

1983 FEBRERO IBM lanza su IBM PC XT. Aumenta a 10 mega byte en disco duro, 3 ranuras de expansión extra, una interfase de serie para el BASIC creado para la IBM PC. Con 128 k bytes de RAM y una lectora de discos y un costo de $ 4 995

ABRIL Microsoft Corp, anuncia el Multi-Tool Word que mas tarde sólo se conocerá como Word.

MAYO AT&T Information Systems presenta su sistema operativo UNIX System V.

OCTUBRE Borland International Inc publica su Turbo Pascal para computadoras con procesadores CP/M y 8088. Su calidad, velocidad y bajo precio lo hacen inmediatamente estándar especialmente en el mundo de las IBM PC.

NOVIEMBRE Microsoft Corp. presenta Windows, el software de ventanas múltiples para la IBM PC, pero que se hace disponible hasta el verano de 1985.

1984 ENERO Apple Computer Inc. introduce la Macintosh a $ 2495.

FEBRERO Lotus anuncia su paquete Simphony a un precio de $695

MARZO Ashton-Tate lanza su paquete Framework por $695 en competencia con el Simphony

MAYO La Apple Computer Inc. lanza la Apple IIc.

AGOSTO La IBM presenta la IBM PC AT (80286, 256 k bytes en RAM, lectora de discos de 1.2 megas a un costo de $ 5 469) también lanza su red local PC.

Hewlett Packard introduce la LaserJet y la HP110 una laptop 80C86

Satellite Software International publica el WordPerfect para la IBMPC, Victor 9000, DEC Rainbow, Zenith Z-100 y Tandy 2000.

1985 Toshiba introduce su laptop T1100

Aldus introduce el original Page Maker para la MAC 512-k byte. Ansa introduce Paradox que mas tarde comprará Borland. IBM introduce su red Token Ring

1986 Sperry y Burroughs se unifican en UNISYS. Peter Norton anuncia su Norton Commander. Microsoft introduce Works para la Mac.

1987 Apple introduce la Mac II y la Mac SE. IBM introduce PS/2 y OS/2, la primer IBM 386 (modelo 80). Commodore anuncia la Amiga 2000 y 5000. Borland lanza el Quattro. IBM lanza la PS/2 modelo 25. Lotus firma un convenio con IBM para desarrollarle software por 10 años, comenzando con un 1-2-3/M.

1988 Intel anuncia su procesador 386SX. Ashton-Tate lanza el D-BASE IV. IBM y Microsoft lanzan su OS/2 1.1 con presentación de director.

1989 Novell anuncia su NetWare 386. Microsoft e IBM presentan su OS/2 1.2 Borland anuncia su Turbo Pascal 5.5 con extensiones orientadas a objetos. Hewlett-Packard lanza su New Wave. Después de muchos años Lotus lanza su hoja de calculo y procesador 3-D. Apple anuncia su Mac IIci y su laptop Mac Portable

1990 Microsoft anuncia su Windows 3.0. Dragon Systems lanza su Dragon-Dictate con un sistema de reconocimiento de voz para 30 000 palabras, corre en 386 y tiene un costo de $ 9000. Borland anuncia su Turbo C++. IBM anuncia la PS/1 que pretende romper el mercado de las PC para casa y oficina.

## *1.2 DEFINICION DE MODELO*

Un modelo es una representación de un problema o de una situación real. Esta representación podemos hacerla de diferentes maneras y utilizando distintos recursos.

Independientemente de cómo y con qué hagamos nuestro modelo, en cualquier caso involucra un proceso de abstracción, que consiste básicamente

en:

a) Seleccionar de la realidad, los elementos más importantes que intervienen en el problema y desechar aquellos que consideramos no juegan un papel determinante en el mismo.

b) Establecer con precisión las distintas relaciones que guardan entre si dichos elementos

Una vez realizado este proceso de abstracción estamos en condiciones de elaborar un modelo, dependiendo de ***cómo y con qué lo hagamos*** tomará distintas características. Construido el modelo, podemos manipular los elementos y sobre todo buscar posibles soluciones. Resolver el problema en el modelo significa haber contestado las siguientes preguntas:

a) ¿Existe solución? si la respuesta es negativa habremos terminado, el modelo construido no tiene solución, podemos replantearnos la pregunta y/o replantear el modelo. Si la respuesta es afirmativa la siguiente pregunta es:

b) ¿La solución es única? Si la respuesta es afirmativa habremos acabado, si resulta negativa, significa que existe más de una solución, y tendríamos que formularnos la tercera pregunta:

c) ¿Cuál de todas es la que más nos conviene? para contestar esta última muchas veces tenemos que volver a reflexionar sobre la realidad y/o sobre nuestro modelo, para establecer los criterios que nos permitan decir cual es mejor.

Después de resolver el problema en el modelo, podemos trasladar la solución encontrada a la realidad, este proceso recibe el nombre de ***aplicación***.

Aunque, como hemos dicho anteriormente, al investigador de operaciones no le corresponde ni elegir la mejor solución (proponemos todas las soluciones posibles y señalamos las mejores), ni llevar a cabo ese proceso de aplicación.

Lo que deseamos es que si se optó por una de nuestras soluciones propuestas al aplicarse a la realidad los resultados obtenidos correspondan a lo esperado.

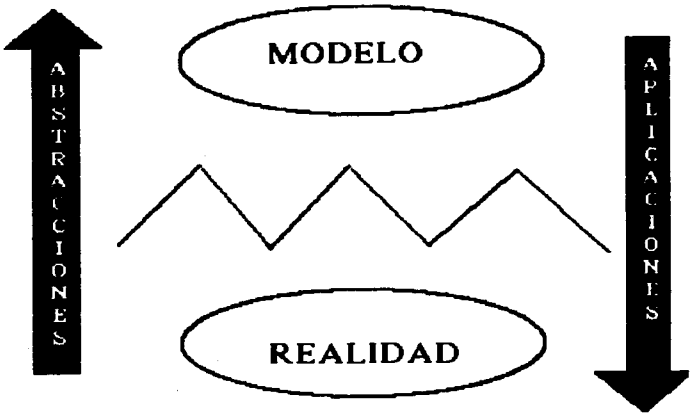



figura 1.1

La Figura 1.1 ilustra esta relación entre los modelos y la realidad donde se aprecia con claridad un rompimiento entre la realidad y el modelo, este rompimiento se da en el proceso de abstracción al seleccionar los "elementos más importantes", y en la determinación de las reglas o leyes de relación entre dichos elementos, esto puede ocurrir por:

a) Una mala selección de los "principales elementos" y/o una mala determinación de las reglas de relación. Dicho de otra forma, el modelo no corresponde a la realidad modelada.

b) Se elaboró un modelo adecuado, esto es, se hizo una selección adecuada de elementos, se determinaron con precisión las relaciones de interdependencia, pero su manipulación no fue del todo adecuada y las soluciones encontradas no resuelven el modelo, mucho menos la realidad.

c) Elaboración y manipulación correcta del modelo, sin embargo, la información para determinar los distintos parámetros, los elementos principales y/o las reglas de relacionales no corresponden a la realidad.

d) El modelo es adecuado y la manipulación fue correcta. La solución encontrada es satisfactoria, sin embargo, durante el tiempo de modelación y búsqueda de solución, la realidad ha cambiado. Elementos que eran importantes han dejado de serlo, relaciones de interdependencia entre elementos del problema se han modificado. La realidad es otra a la modelada por lo tanto el modelo no corresponde a la nueva realidad, a la nueva problemática.

En los primeros dos casos, se debe a errores de tipo humano, falta de destreza en la elaboración de modelos o de su manipulación, esto es, desconocimiento de la teoría necesaria para manejar el modelo. El tercer caso, se debe principalmente a que las bases de datos, de donde se obtiene la información, son alimentadas por sistemas muy rudimentarios que no verifican la información que se almacena, teniendo como resultado grandes bases de datos, parecidos a basureros de información donde hay que pepenar la información requerida. La labor de rescatar información de este tipo de bases de datos es muy laboriosa y no siempre es posible garantizar una alta confiabilidad de los datos. El último caso suele ser el menos frecuente, y es motivado por la dinámica propia de la realidad. Podría alegarse aquí, que un buen modelo debe considerar también esta propiedad cambiante de la realidad, al menos por hoy, esto no es posible, tal vez en el futuro se vuelva realidad la ficción de la psicohistoria novelada magistralmente por Issac Asimov en su trilogía trantoriana.

En cualquiera de estos casos, el decidor debe de tenerlos en cuenta y el investigador de operaciones debe de enfatizarlo. Es en el proceso de aplicación donde debe de manifestarse la experiencia y el sentido común de nuestros decisores.

## *1.3 MODELOS ANALITICOS Y HEURISTICOS.*

### *MODELOS ANALITICOS.*

Los modelos analíticos son aquellos que para su elaboración se utiliza el lenguaje proporcionado por las matemáticas y cuyas soluciones las buscamos apoyándonos en las teorías matemáticas.

En particular en la I. de O. el principal y sin lugar a dudas el modelo analítico más usado es el modelo de programación lineal.

Los conceptos matemáticos en que se apoya -las funciones lineales- son el resultado de un largo proceso de abstracción, aunque el origen de esta teoría está en fenómenos y problemáticas derivadas de la realidad, por su grado de abstracción, se ha despojado totalmente de las diversas características que le dieron origen.

En física por ejemplo, la expresión $d=v*t$ es un modelo analítico, que describe el comportamiento de un cuerpo que se mueve a velocidad constante durante un cierto intervalo de tiempo. Para llegar a esta expresión los físicos hicieron una serie de suposiciones sobre el modelo, dijeron: " esto es cierto bajo condiciones ***ideales***, esto es sin considerar fricción, resistencia del aire - al vacío- , etc." Así este modelo describe una "realidad", que sólo podemos encontrar en un buen laboratorio.

Por lo tanto para llegar al modelo $d=v*t$, los físicos modelaron una realidad, despojada de todas sus características particulares como forma, color, materia del cuerpo, velocidad a que se mueve, etc. y esta porción de la realidad la han aislado totalmente de la realidad que la rodea, al considerar "condiciones ideales".

Quiere decir esto que el modelo $d=v*t$ ¿no describe la realidad?, la respuesta es ***no***, a una realidad cotidiana, es decir que no nos enfrentemos con ella . Que describe una parte de la realidad ***si***. Una porción que no se presenta sola en la realidad, sino acompañada de más elementos. ***Esta parcialización de las descripciones de los modelos analíticos es su principal desventaja.***

Así en el caso del modelo de programación lineal, éste describe una situación ideal, determinista, continua y lineal, sin embargo la realidad dista mucho de ser una situación ideal, pero comprender y sobre todo predecir el comportamiento de una realidad, aunque sea bajo la suposición de condiciones ideales nos ayuda mucho.

La ventaja de los modelos analíticos, es su simplicidad y exactitud, los modelos analíticos -si están bien planteados desde luego- no dejan lugar a dudas en sus resultados.

No todos los modelos analíticos son deterministas, la probabilidad es la parte de las matemáticas que tuvo su origen en la descripción estocástica del

mundo. Las variables aleatorias, como toda función, describen o pretenden describir el comportamiento de un fenómeno de la realidad. Asociar el comportamiento de un ***ente*** con una variable aleatoria, que tenga una expresión analítica es un verdadero problema y no en todos los casos es posible. Sin embargo trabajar con una variable aleatoria que tenga una expresión analítica conocida, simplifica mucho nuestros cálculos. En la teoría de la probabilidad la mayor dificultad radica en saber interpretar los resultados a que se llegan.

La mayoría de los modelos analíticos son susceptibles de programarse en una computadora. En la actualidad existe un buen número de paquetes de cómputo que resuelven el modelo de programación lineal, a través de derivaciones del método simplex, lo que ha popularizado su utilización. Sin embargo actualmente se desarrollan nuevos métodos para la solución del modelo de programación lineal, como el algoritmo de Khachian que tiene la propiedad de que el número de operaciones nunca excede una cota polinomial fija, al menos en el peor de los casos. Este algoritmo mantiene una solución desconocida -el vector que minimiza o maximiza cx- dentro de una serie de elipsoides que se contraen en un punto. Ese punto es la solución. Otro algoritmo más reciente llamado el Método de Karmarkar también opera dentro de un conjunto factible - mientras que el método simplex opera en los extremos. La solución se encuentra en tiempo polinomial y se dice que es cincuenta veces mas rápido que el simplex.

La programación de los modelos analíticos ha dado una nueva connotación al concepto de algoritmo, cuyo origen lo encontramos en Euclides (año 300 antes de nuestra era). El concepto de algoritmo, podemos resumirlo como un proceso formado por:

a) Etapa de preparación.

b) Ciclo iterativo.

c) Criterio de terminación.

Un algoritmo asegura una solución óptima, en un número finito de pasos y debe ir acompañado de su correspondiente demostración.

Podemos concluir que en general, los modelos analíticos tienen las ventajas de la exactitud y la desventaja de la parcialización del modelo.

## *MODELOS HEURISTICOS.*

La heurística o heurética, o "ars inveniendi", tal es el nombre de esta ciencia que tiene por objeto el estudio de las reglas y de los métodos del descubrimiento y la invención. Algunos comentarios a cerca de la heurística los encontramos en ***Papus*** (siglo III de nuestra era), comentando a Euclides. Los ensayos más conocidos sobre la construcción de un sistema heurístico son debidos a ***Descartes (1596-1650) y Leibniz (1646-1716)***, ambos filósofos y matemáticos brillantes. Igualmente debemos a ***Bernard Bolzano (1781-1848)*** una exposición sobre heurística detallada y notable. Sin embargo, quien retoma la heurística y le da la connotación actual fue ***George Polya (1887- )***, quien en 1945 publicó un pequeño libro ***How to solve it*** (Cómo resolverlo), cuya traducción al español fue ***Cómo plantear y resolver problemas***. La portada del libro promete "Un sistema de pensar que puede ayudarle a resolver cualquier problema". La pretensión resulta excesiva, pero el libro es sumamente instructivo. Polya define su libro como heurístico, palabra que significa "que sirve para descubrir". Este libro considera cuestiones generales con los que se enfrentan los que resuelven problemas y propone métodos generales para tratarlos. Propone una serie de preguntas para cada etapa de su metodología de solución.

Dentro de la I. de O. se consideró inicialmente a la modelación heurística como los procedimientos que se siguen paso a paso y que aseguran, mediante un número finito de ellos, alcanzar una solución satisfactoria. Poco difiere esta acepción de la de algoritmo, ya que difieren exclusivamente en que ésta, no nos garantiza una solución óptima y desde luego no requiere de una demostración, además hay algoritmos que pueden garantizar un porcentaje alto de acercamiento al óptimo. Así la modelación heurística nos deja manos libres en la búsqueda de soluciones a diversas problemáticas y sobre todo nos permita modelar situaciones que analíticamente no pueden modelarse, o su modelación puede ser muy costosa.

Resumiendo la heurística nos da oportunidad de modelar problemáticas más complejas.

## *1.4 MODELACION DE SISTEMAS.*

Hasta aquí hemos hablado de la modelación de problemáticas, hemos hecho a un lado el concepto de sistema, sin embargo, la importancia misma del concepto nos obliga a darle un trato especial.

***"Sistema"***, es uno de los conceptos relevantes producto de esta revolución tecnológica. Su impacto y difusión ha inundado todas las áreas del conocimiento humano, se ha presentado como técnica, como metodología y finalmente como una filosofía. Más que dar una amplia discusión al respecto, el objetivo de estas líneas es delimitar algunas de sus características que para nuestros objetivos (la modelación de sistemas) resultan valiosos.

La definición más popular de sistema es "Un conjunto de elementos interrelacionados que constituyen una totalidad", esta definición es tan vaga que en ella tienen cabida desde un átomo, hasta el universo mismo, desde una célula, hasta el organismo vivo más complejo, etc. Así cualquier problemática de las que nos hemos referido arriba al hablar de modelación, nos estábamos refiriendo a sistemas. Sin embargo viene a constituir un marco teórico que nos ayuda a entender el rompimiento que se da entre la realidad y el modelo, a continuación daremos un resumen de sus principales características:

***COMPLEJIDAD.*** El enfoque de sistemas parte del reconocimiento de que la realidad es compleja. Esta complejidad se debe a que los elementos del objeto a modelar están íntimamente interelacionados y a que el objeto mismo interactúa en el medio ambiente con otros objetos. (fig.1.2 )

El reconocer esta complejidad, lleva consigo el aceptar que el funcionamiento adecuado del objeto, esta más allá del correcto desempeño de los elementos que lo componen, ya que también influye la manera en que estos interactúan entre sí y con su entorno.

Este argumento es válido para cualquier sistema, y su peso depende del problema mismo a modelar. La modelación de un sistema puede requerir el concurso de especialistas de diversas disciplinas, lo que demanda una visión interdisciplinaria.

Cualquiera que sea la realidad a modelar, por pequeña y simple que nos parezca, tiene cierto grado de complejidad. No esta por demás reconocer que este trabajo esta dirigido a resolver problemáticas modestas

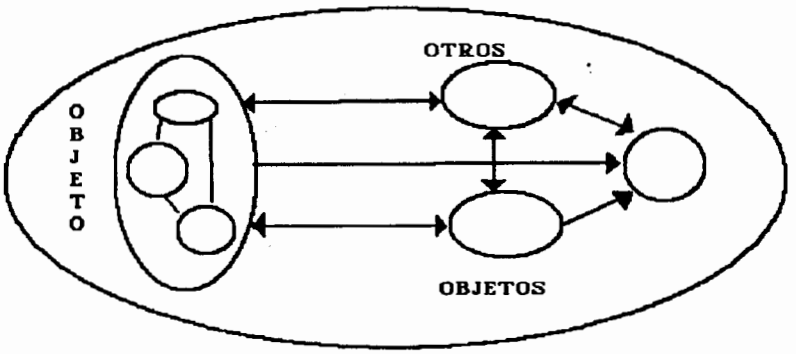



figura 1.2

***TOTALIDAD.*** El enfoque de sistemas reconoce la necesidad de tener una visión global o total de la problemática a modelar. Sin lugar a dudas este es siempre el primer paso y significa, además de definir el sistema y su problemática, establecer la finalidad del modelo. El no tener esta visión global del problema, acarrea pérdida de tiempo y recursos, es como "tirar palos de ciego".

Adquirir esta visión global del problema no es fácil, depende de la naturaleza misma de la problemática a modelar, algunas veces esto es tan difícil que nos parece imposible. Por ejemplo los problemas sociales.

***ANALISIS Y SINTESIS.*** El enfoque de sistemas, no ignora ni se contrapone a lo que algunos autores han denominado enfoque analítico el cual propone:

a) Aislar y dividir en partes el todo

b) Entender el funcionamiento de cada una de las partes.

c) Reunir el conocimiento de las partes para entender el comportamiento y propiedades del todo.

Para entender cada una de las partes, puede procederse de la misma forma tantas veces como sea necesario.

El enfoque sistémico, sostiene que cuando en el sistema a modelar tiene cierto grado de complejidad, como relaciones no lineales o expresiones no analíticas, este procedimiento se complica e invalida. Ante esta situación el enfoque sistémico añade el estudio de las partes que interaccionan como un todo. Así el enfoque sistémico propone las siguientes características al proceso de síntesis, entendido éste como el proceso de integrar el conocimiento de las partes para entender el todo.

a) Las propiedades o el comportamiento de cada elemento del conjunto tiene un efecto en las propiedades o comportamiento del todo.

b) Las propiedades o el comportamiento de cada elemento y la forma en que afecta al todo depende de las propiedades y del comportamiento de al menos otro elemento del conjunto.

c) Cada subgrupo posible exhibe las dos propiedades anteriores.

Así, el enfoque sistémico llama a la creación de nuevas técnicas y métodos, cuya característica básica radica en que el conjunto de elementos y relaciones deben estudiarse bajo esta perspectiva de totalidad.

***EL MODELO DIAMANTE.***

Durante la década de los 70's Ian I. Mitroff, junto con otros autores, desarrollaron un modelo cualitativo de solución de problemas, llamado modelo diamante, en el cual en forma un tanto esquemática divide el proceso de solución de problemas en cuatro fases, que tienen un orden de desarrollo y que requieren de retroalimentarse entre sí, me permito resumir dicho modelo porque nos sirva como marco de referencia en la definición de modelo de simulación.

A. ***Situación Problemática***: Consiste en el reconocimiento del problema, es el reconocimiento de que algo anda mal o el sentimiento de que algo puede mejorarse.

B. ***Modelo Conceptual***: Consiste en lo que hemos llamado definición del problema, esto es, establecer la estructura del problema, delimitar el área

de interés y establecer los elementos relevantes y las relaciones que guardan entre si.

C. ***Modelo Formal***: Generalmente consiste en un modelo matemático, un conjunto de ecuaciones, etc.

D. ***Solución***: Esta actividad es la integración de los distintos modos de acción, es la integración de las estrategias de implantar en la realidad el sistema.

Los modelos de simulación, se aproximan más a lo que, en el modelo diamante, se define como modelo conceptual y al que hemos denominado modelo heurístico, que permite integrar en él uno o varios modelos analíticos (o formales).

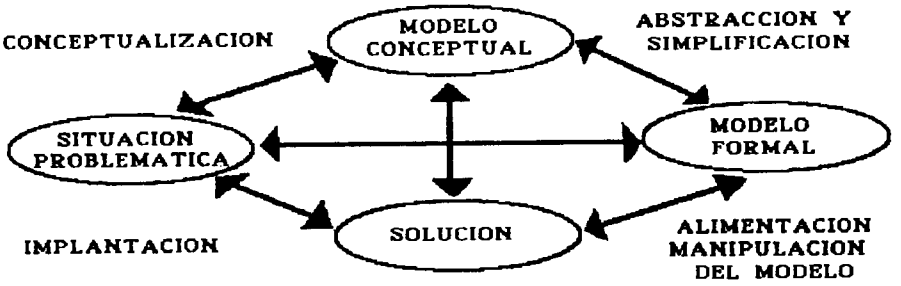



figura 1.3

# CAPITULO 2.

# LA SIMULACION DIGITAL.

***No es digno de hombres excelentes perder como esclavos, horas y horas haciendo cálculos que, con facilidad, un cualquiera podría hacer por medio de máquinas.***

***Barón Gottfried Wilhelm Von Leibnitz.***

La simulación digital, es la modelación de cierto tipo de sistemas utilizando un computador digital. Inicialmente empieza a utilizarse en la solución de problemas de líneas de espera, cuya solución analítica no es posible, por lo que podemos afirmar que vino a apoyar a la modelación analítica.

Una de las características de estos primeros modelos simulados es que son aleatorios y dinámicos. Tal es el caso de los modelos de líneas de espera que se desarrollan a lo largo de un intervalo de tiempo dado. Por ejemplo, el modelo clásico consiste en una serie de clientes que llegan a intervalos de tiempo con una determinada distribución a un modulo de servicio en donde si no se está atendiendo a nadie pasa a ser atendido, en caso contrario espera formando una línea de espera, el tiempo de servicio también es una variable aleatoria. La pretensión usual del modelo es estimar los tiempos medios de permanencia de los clientes, el tamaño máximo de la línea de espera y el porcentaje de tiempo de ocupación del modulo de servicio, para que con ésta información se puedan hacer los ajustes convenientes y mejorar por ejemplo el tiempo de servicio.

El primer problema que tuvieron que enfrentar estos modelos fue la generación de números aleatorios con una función de distribución determinada, este problema se reduce al de generar una serie de números aleatorios con

distribución uniforme en el intervalo cero - uno y calcular la inversa de la función acumulada de probabilidad de la distribución de números deseada, esta técnica se ilustra en el anexo 1 y tiene la ventaja de poder generar series de números con distribuciones empíricas.

La técnica de simulación consiste básicamente en reproducir varios ciclos de vida del sistema y sacar las estadísticas respectivas, por ejemplo en el modelo clásico que se dio en el párrafo anterior, un ciclo de vida del sistema sería un día de labores, de donde resolverlo por simulación seria simular varios días de servicio y sacar estadísticas de los comportamientos observados.

Este Capítulo se desarrolla como sigue: En la primera sección se da una breve introducción histórica del desarrollo de la simulación y se dan algunas definiciones. En la segunda se presentan algunas definiciones y conceptos básicos de simulación. En la tercera se describe en términos generales como se desarrolla una simulación, dentro de la cual se resaltan dos puntos: la validación del modelo simulado así como su análisis de sensibilidad e implantación. En la cuarta se presentan algunas ventajas y desventajas de hacer simulación. En la quinta sección se describen los lenguajes de propósito especial así como una clasificación de ellos.

## *2.1 INTRODUCCION.*

Durante la década de los 60's, la técnica de simulación sirvió básicamente como apoyo a las técnicas analíticas en la solución de modelos de subsistemas. Su aplicación resultaba costosa en términos del tiempo de máquina requerido por cada corrida. A pesar de esto fue en esta época cuando se desarrolló la primera versión del GPSS, lenguaje de programación de propósito especifico, para simulación de sistemas discretos.

La simulación digital, desde sus inicios dejó ver un enorme potencial en la solución de problemas, su principal limitación durante sus primeros 20 años de desarrollo, como ya se ha dicho fueron los altos costos de los equipos de cómputo. Con la aparición de las microcomputadoras, su posterior desarrollo y abaratamiento, se sentaron las bases materiales para un gradual y pujante desarrollo de la simulación digital.

En la década de los 70's, la simulación digital, empezó a ser usada a mayor escala y da aportaciones en:

El desarrollo de la teoría de la I. de O.

En la obtención de información en cuanto al funcionamiento de un sistema y

En la validación de modelos analíticos.

Un cuarto uso de la simulación es en optimización. Ya en 1980 Hanssmann et al, da una discusión detallada de la utilización de modelos de simulación en optimización. Este autor distingue cuatro métodos de optimización. Cada método optimiza primero un modelo analítico aproximado y después utiliza los valores encontrados como parámetros en el modelo de simulación.

Los cuatro métodos difieren en la forma en que se usan los parámetros derivados del modelo analítico. Estos son:

a) Para determinar resultados de los parámetros trabajados.

b) Como punto de inicio de un proceso de búsqueda en un procedimiento de solución por simulación.

c) Para determinar resultados de los parámetros trabajados que retroalimentarán el procedimiento de solución del modelo de simulación para su uso en reoptimización del modelo analítico. El nuevo conjunto de valores solución se usa entonces como nuevos parámetros en el modelo de simulación y así se continúa iterando entre ambos modelos hasta obtener resultados esperados.

d) Igual que en c), con la diferencia de que en cada iteración en el modelo de simulación se aplica un procedimiento de búsqueda como en b).

Los métodos c) y d) permiten que la iteración de soluciones comience con cualquiera de los dos modelos. Si la sucesión comienza con el modelo de simulación, el analista debe seleccionar los parámetros iniciales en el simulador.

Fetter and Thompson reportaron el uso de este tipo de modelación (4.c), en un estudio para la asignación y operación de un hospital, iterando entre el uso de un modelo de programación lineal y un modelo de simulación. Con el modelo de programación ubicaban las camas en las diferentes unidades del

hospital y con el simulador determinaban la política del uso de las camas, esto es, las reglas operacionales de funcionamiento del hospital.

Con todo lo referido anteriormente podemos intentar dar una definición de simulación como un método usado para estudiar la dinámica de sistemas. Donde por sistema entendemos un grupo de unidades que operan en forma interrelacionada. A menudo el propósito del estudio de los sistemas es entender de manera más exacta la operación total de las unidades que lo conforman. La simulación proporciona una descripción del funcionamiento del sistema en un período de tiempo.

Otra definición es la dada por Shannon:

Simulación es el proceso de diseñar un modelo de un sistema real y realizar experimentos con él para entender el comportamiento del sistema o evaluar varias estrategias (dentro de los límites impuestos por un criterio o por un conjunto de criterios) para la operación del sistema.

En consecuencia entendemos que el proceso de simulación incluye tanto la construcción del modelo como su uso posterior para estudiar un problema.

Es entonces de ésta forma que no restringimos nuestra definición de simulación a los experimentos realizados sobre modelos de computadoras. Muchas simulaciones útiles pueden realizarse y se realizan con sólo lápiz y papel o con la ayuda de una calculadora. El modelado de la simulación es, por tanto, una metodología aplicada y experimental que intenta:

a) Describir el comportamiento de sistemas.

b) Postular teorías o hipótesis que expliquen el comportamiento observado.

c) Usar estas teorías para predecir un comportamiento futuro, es decir, los efectos que se producirán mediante cambios en el sistema o en su método de operación.

A diferencia de la mayoría de las tecnologías, las cuales pueden clasificarse de acuerdo con la disciplina ( por ejemplo, física o química) de la que se originan, la simulación es aplicable a todas las disciplinas. La simulación como la conocemos recibió su impulso original de los programas aeroespaciales,

pero incluso un examen superficial de la literatura sobre este tema indica el amplio campo de sus aplicaciones actuales. Por ejemplo, en administración, economía, educación, política, transporte e innumerables áreas.

## *2.2 DEFINICIONES Y CONCEPTOS BASICOS DE SIMULACION.*

A continuación se presentan una serie de definiciones que, a pesar de las limitaciones que se puedan encontrar en ellas, nos permitirán trabajar con precisión las técnicas de simulación. No se pretende dar una discusión de los conceptos aquí definidos, se trata de dejar claros algunos rasgo de dichos conceptos.

Definición 1 ***Simulación*** es una técnica experimental para resolver problemas que nos permite tener un modo efectivo de probar, manejar y evaluar un sistema propuesto sin tener acción directa sobre el sistema real.

Definición 2 ***Sistema*** es un conjunto de elementos unidos entre si por medio de una interacción o interdependencia regular Como se ilustra en la figura 2.1

SISTEMA DE PRODUCCION

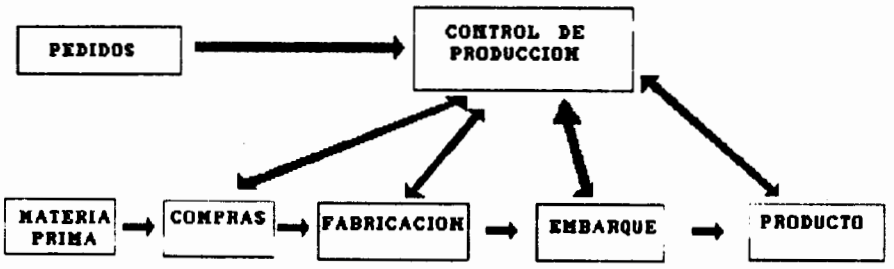



figura 2.1

Definición 3 ***Entidad*** es un elemento que nos interesa en el sistema.

Definición 4 ***Atributo*** es una descripción o definición de las propiedades de las entidades.

Definición 5 ***Actividad*** proceso que causa cambios en el sistema.

La tabla 1 ilustra las definiciones anteriores.

Definición 6 ***Estado del sistema*** es la descripción de entidades, atributos y actividades del sistema en un momento dado (es una fotografía).

| Sistema | Entidad | Atributo | Actividades |
|---|---|---|---|
| Banco | Cliente, ventanilla, terminal de computadora | Edo. de cta, tamaño de la cola, disponi-bilidad | Depósito, consulta, retiro, etc. |
| Red de tránsito urbano | carro, camiones, semáforo, calle, esquina, etc. | tamaño, velocidad, duración de verde, rojo, longitud, carriles, etc. | arrancar, parar, cambio de luz del semáforo, vuelta a la derecha,etc.. |
| Metro | pasajeros, estaciones, trenes, etc. | longitud entre estaciones, capacidad,.. | llegadas a estación de trenes y pasajeros, .. |
| Super | cajas, cuentas, clientes, | cuantos compran, cuantos hay,.. | pagar llegada de clientes,.. |

Tabla 1

Definición 7 ***Modelo*** es una representación del sistema o parte del sistema que queremos estudiar (entidades, atributos, actividades).

Definición 8 ***Actividad Endógena*** es aquella que se genera dentro del mismo sistema.

Definición 9 ***Actividad Exógena*** es aquella que proviene del medio exterior.

Definición 10 ***Un sistema es cerrado*** si no tiene actividades exógenas y es ***abierto*** si al menos tiene una.

Ejemplo 1

Los tiempos de llegada a un estacionamiento son endógenos si con base en sus atributos los generamos dentro del sistema. Son exógenos si es la información histórica, la que ocupamos para la asignación de atributos.

Definición 11 ***Un sistema continuo*** es aquel en el que los cambios son suaves (smooth) ***es discreto,*** si los cambios son puntuales en el tiempo. Se trabajará con sistemas discretos, discretizando si es necesario.

Definición 12 ***Modelo*** es una representación de un objeto, un sistema, o idea, de forma diferente a la de la identidad misma.

Los modelos dependiendo de la finalidad que se persiguen los podemos clasificar en:

***Descriptivo***.- Representa el sistema sin tener acción sobre el.

***Predictivo***.- Representa el comportamiento del sistema.

***Explicativo***.- Sirve para tratar de entender el comportamiento.

Otra clasificación, ahora atendiendo la constitución del modelo es:

***Icónicos***.- Las característica o propiedades se representan por si mismas (también llamados físicos).

***Analógicos***.- Las propiedades son representadas por otras equivalentes (modelos hidráulicos para representar flujo de tránsito).

***Simbólicos***.- Las propiedades se representan por símbolos y las relaciones entre ellas por relaciones entre los símbolos (modelos matemáticos).

***Lógico.-*** Esta dado por elementos lógicos que al irse siguiendo representan el comportamiento del sistema (diagramas de flujo).

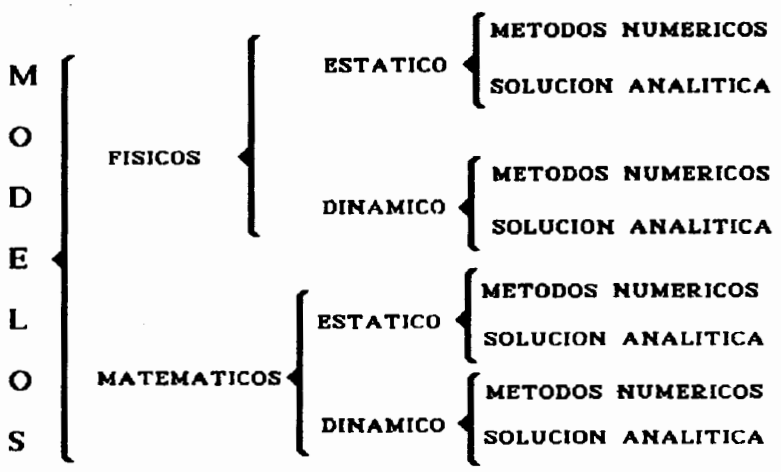



figura 2.2

La figura 2.2 es una subclasificación de los modelos físicos (icónicos) y de los matemáticos o simbólicos, atendiendo a las características de sus variables.

Definición 13 Un modelo es ***estático*** si no esta en función del tiempo.

Nos interesan las relaciones entre los atributos cuando el sistema esta en equilibrio. Por ejemplo fórmulas para determinar cargas, modelos econométricos que no dependen del tiempo, etc..

Definición 14 Un sistema ***dinámico*** representa cambios en el tiempo.

Por ejemplo, considere f(x,y,z,t) una ecuación de movimiento, si f es sencilla podemos encontrar una solución analítica, de lo contrario obtenemos una solución numérica que da los valores de x, si en ciertos tiempos, en cualquier caso, se trata de un modelo dinámico.

Definición 15 ***Actividad determinística***: aquella cuyo resultado está totalmente determinado por los datos de entrada.


Definición 16 ***Modelo determinístico***: formado exclusivamente por actividades determinísticas. se puede seguir el modelo siguiendo las actividades que lo forman.

Definición 17 ***Variable estocástica***: es una variable que corresponde a una actividad aleatoria. No se conoce la secuencia de los valores pero si el rango de ellos y su probabilidad.

## *2.3 PROCESO DE SIMULACION DE SISTEMAS.*

### *2.3.1 SISTEMAS DISCRETOS Y CONTINUOS.*

Hemos visto que un modelo es una representación de una porción de la realidad. Un modelo puede consistir es una serie de ecuaciones matemáticas, generalmente ecuaciones diferenciales. Cuando tenemos un modelo de este tipo podemos intentar sacar información relevante del modelo a través de la solución analítica del sistema de ecuaciones. Pocos son los casos en que podemos realmente encontrar soluciones analíticas. La mayoría de las veces tenemos que utilizar métodos numéricos de solución, de ahí la rica variedad de métodos numéricos de solución de ecuaciones que se han desarrollado y aún hoy en día se continúan desarrollando. Cuando estos modelos son dinámicos, una técnica específica que se ha llegado a identificar como de simulación de sistemas es aquella en que se resuelven simultáneamente todas las ecuaciones del modelo con valores continuamente incrementados del tiempo. Este tipo de sistemas, generalmente modelado mediante un conjunto de ecuaciones diferenciales, generan cambios suaves en el sistema. Las simulaciones basadas en este tipo de modelos, reciben el nombre de ***simulaciones continuas***. Ejemplo de este tipo de simulaciones es cuando simulamos un proceso químico ó físico, como los desarrollados en la ***NASA***, movimiento de cohetes, etc. La forma en que se realiza este tipo de simulación es incrementando el tiempo y calculando los cambios producidos en el estado del sistema.

Los ***sistemas discretos***, en los que centraremos nuestro interés, se rigen por

ecuaciones lógicas que expresan condiciones para que un evento ocurra. La simulación ***discreta***, consiste en seguir los cambios en el estado del sistema resultado de cada uno de los eventos que se realizan. Por regla general este tipo de simulación se realiza siguiendo la secuencia de ocurrencia de eventos, es decir avanzamos el tiempo de la simulación al tiempo de ocurrencia del siguiente evento.

### *2.3.2 PASOS EN LA SIMULACIÓN.*

A pesar de lo limitante de dar una serie de pasos a seguir al resolver problemas de simulación, lo haremos dejando claro, que estos pasos pueden y deben irse modificando de acuerdo con la experiencia que vayamos adquiriendo.

1.- Definición del problema.
2.- Planeación del estudio.
3.- Revisión y obtención de datos.
4.- Formulación y evaluación del modelo matemático.
5.- Formulación del programa de cómputo
6.- Validación.
7.- Diseño de experimentos.
8.- Ejecución de la simulación.

1.- ***Definición del problema:-*** Parece obvio, pero lo primero que se debe hacer es definir que es lo que se quiere estudiar, ¿Que problemática pretende resolverse?, ¿Que se persigue?. En no pocas ocasiones se nos invita a participar en proyectos, cuya problemática ni siquiera esta bien definida, de aquí que el primer paso consista en definirla.

2.- ***Planeación del estudio:-*** Una vez superado el punto anterior, hay que determinar: el tiempo con el que contamos, los recursos de que se dispone, gente, computadora, dinero, etc. Una vez determinados los recursos hay que definir los alcances y planear el tiempo de realización del trabajo.

3.- ***Revisión y obtención de datos:-*** ¿ Con que información contamos?. Hasta hace pocos años, el principal problema consistía en que no existía información almacenada, había que diseñar estrategias para su obtención y sobre todo ser lo suficientemente creativos para buscar fuentes alternas de información. Para ejemplificar lo anterior basta recordar cómo se

construyó la matriz origen-destino para el estudio de la línea 2 del metro, (tengo entendido que para la línea 1, no se realizó un estudio tan detallado como para las subsecuentes líneas). Hay que recordar que a principios de la década de los 70, no se realizaban estudios sistemáticos sobre el movimiento de los capitalinos. Planear un estudio que nos proporcionara la información necesaria para su elaboración, involucraba una serie de encuestas, cuyo diseño, aplicación, depuración y procesamiento resultaba sumamente costoso. El problema se resolvió, acudiendo a los archivos del IMSS: los trabajadores se registran en la clínica más cercana a su hogar y entre los datos que proporciona es el de su domicilio y la empresa en que labora. El que propuso esa idea, provocó un ahorro de no pocos millones de pesos, éste es un bonito ejemplo de una idea productiva. En la actualidad y al menos durante algunos años más, el principal problema consiste en depurar la información almacenada en las computadoras de la mayoría de nuestras empresas, ya que, la mayoría de los sistemas no verifican la información que reciben, lo que ocasiona verdaderos basureros de información, de donde hay que rescatar lo rescatable, un trabajo de pepenadores de información, que requiere una gran habilidad y sobre todo creatividad para realizar esta labor.

4.- ***Formulación y evaluación del modelo.***- Estructurar el modelo: definir variables, interrelación de variables, distribuciones de probabilidad, etc. En esta etapa juega un papel importante, el que conoce el sistema a simular. Con frecuencia al ir desarrollando el punto 5, se vuelve a este punto.

5.- ***Formulación del programa de cómputo:*** Programar y probar el modelo. Los objetivos del modelo están muy ligados al lenguaje de programación que se usará. Este punto descansa sobre todo en el programador.

6.- ***Validación y calibración:***- ¿ El programa que representa el modelo, refleja adecuadamente el sistema?. Alimentar al modelo con ciertas condiciones conocidas del sistema, comparar los resultados obtenidos con los esperados y mover los parámetros de calibración del sistema, hasta obtener los resultados esperados, es un proceso de pruebas y ajustes, que nos conduce con frecuencia a un ciclo, donde volvemos al punto 4, es decir a formular nuevamente el modelo, reprogramar los cambios y validar el modelo. Así hasta que el modelo corresponda con la realidad.

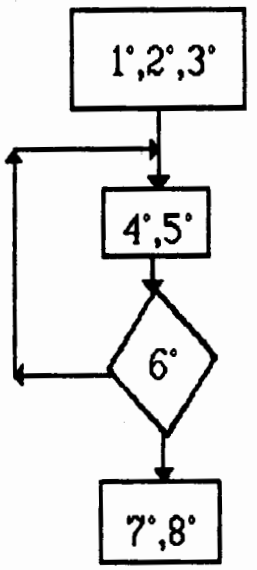



figura 2.3

7.- ***Diseño de experimentos:***- diseñar las diferentes corridas que se quieran hacer con el modelo, para obtener ciertos resultados que al analizarlos den luz de lo que se quiere.

8.- ***Ejecución de la simulación:*** corrida en computadora.

Estos pasos los podemos agrupar y representar mediante el diagrama de flujo de la figura 2.3, que nos ayudará a entender como se relacionan:

Con los pasos anteriores se supone que el problema puede resolverse de una mejor manera mediante la simulación. Como ya se indicó, ésta quizá no sea la forma mas efectiva. A menudo se ha dicho que la simulación es un planteamiento aproximado o un último recurso para resolver problemas. En realidad, es cierto que cuando un problema puede reducirse a un modelo simple y resolverse analíticamente, la simulación no es necesaria. Debería investigarse todas las herramientas disponibles para manejar cada problema y optimizar entre los resultados y el costo. También un debería convencerse de que un modelo analítico simple es inadecuado, antes de considerar la posibilidad de una simulación

Debido a que es necesario y conveniente ajustar la herramienta al problema, las decisiones referentes a qué herramienta o método usar deben seguir la formulación del problema. La decisión de usar la simulación no debe considerarse como irrevocable. Conforme se obtienen más datos y se entiende más el problema, la validez del uso de la simulación debe revaluarse. Debido a que con frecuencia intervienen grandes computadoras y muestras grandes, el costo de una simulación es casi siempre alto comparado con la resolución de pequeño problema analítico. El costo probable y el tiempo de la simulación siempre deberán compararse con el valor de la información que producirán.

### *2.3.3 VALIDACION DEL MODELO*

Un modelo debería crearse exclusivamente para un propósito específico, y su veracidad o validez debería evaluarse solamente en términos de ese propósito. Nuestra meta es generar un modelo que cree los mismos problemas y características de comportamiento que las del proceso o sistema que se está estudiando. Evaluar un modelo significa desarrollar un nivel aceptable de confianza de modo que las inferencias obtenidas del comportamiento del modelo sean correctas y aplicables al sistema del mundo real.

Así la validación es el proceso de llevar a un nivel aceptable la confianza del usuario referente a que cualquier inferencia acerca de un sistema que se derive de la simulación sea correcta. Es imposible probar que cualquier simulador es un modelo correcto o verdadero del sistema real. Por fortuna, rara vez nos interesamos en probar la "verdad" de un modelo. En vez de esto, principalmente nos importa la validación de los discernimientos que hemos obtenido u obtendremos a partir de la simulación. Por lo tanto, nos importa la utilidad operativa del modelo y no la verdad de su estructura.

La validación del modelo es sumamente importante debido a que los simuladores se ven reales y tanto los modeladores como los usuarios los encuentran fácilmente creíbles. Lamentablemente, con frecuencia los simuladores ocultan sus suposiciones del observador casual y algunas veces del modelador. Por lo tanto, si la validación no se lleva a cabo cuidadosa y detalladamente, pueden aceptarse resultados erróneos con consecuencias desastrosas.

No existe una prueba de validación, en lugar de eso, el experimentador debe realizar pruebas a lo largo del proceso de desarrollo del modelo a fin de crear confianza. Se pueden usar tres pruebas para validar un modelo:

Primera: debemos cerciorarnos de que el modelo tenga validez en forma general, por ejemplo: ¿Es posible que el modelo dé respuestas absurdas si se lleva a los parámetros a valores extremos?. O también podemos preguntar si los resultados del modelo parecen razonables, esto se puede hacer mediante la creación de un panel de expertos de dos tipos: en software y en el sistema real de lo que se esté simulando.

El segundo método de validación es la prueba de suposición y el tercero es la prueba de transformaciones de entrada-salida. Estas dos últimas conllevan el uso de pruebas estadísticas de medias y varianzas, regresión, análisis de factores, análisis espectral, autocorrelación, ji-cuadrada, y pruebas no paramétricas. Debido a que cada una de estas pruebas estadísticas hacen suposiciones acerca del proceso fundamental, el uso de éstas da origen a las preguntas de validez. Algunas pruebas estadísticas requieren menos suposiciones que otras, pero en general, la potencia de la prueba disminuye conforme se atenúan las suposiciones.

Fishman y Kiviat dividen la evaluación de las simulaciones en tres categorías:

1. Verificación — Asegurándose de que el modelo se comporta de la manera que el experimentador desee.

2. Validación — Probando la concordancia entre el desempeño del modelo y el desempeño del sistema real y

3. Análisis — Del problema. Dibujando las inferencias estadísticamente significativas a partir de los datos generados por la simulación de computadoras.

Sin embargo, la realización de esta prueba padece de los problemas estándar de la investigación empírica: pequeñas muestras debido al alto costo de los datos, datos que están muy agrupados y datos de validez dudosa.

Por lo tanto la pregunta acerca de la validación tiene dos facetas: la determinación de que el modelo se comporte igual que el sistema del mundo real; y la validación de que las inferencias que se obtienen de los experimentos con el modelo sean válidas o correctas. Conceptualmente, ambos puntos se transforman en el problema estándar de decisión referente al equilibrio que debe existir entre el costo de cada acción contra el valor de la información aumentada y las consecuencias de obtener conclusiones erróneas.

### *2.3.4 ANALISIS DE SENSIBILIDAD E IMPLANTACION*

El análisis de sensibilidad es uno de los conceptos más importantes en los modelos de simulación, por esto queremos decir, determinar la sensibilidad de nuestras respuestas finales a los valores de los parámetros que se usaron. Usualmente el análisis de sensibilidad consiste en la variación sistemática de los valores de los parámetros sobre algún intervalo de interés y en la observación del efecto en la respuesta del modelo. En casi cualquier modelo de simulación, gran parte de las variables establecidas se basan en datos sumamente dudosos. En la mayoría de los casos, estos valores se pueden determinar sólo con base en las conjeturas del personal experimentado o en un análisis muy superficial de datos mínimos. En consecuencia, es vital determinar el grado de sensibilidad de los resultados para los valores que se usaron. Si la respuesta cambia enormemente con pequeñas variaciones en los valores de algunos de estos parámetros, esto puede proporcionar la motivación y justificación para asignar más tiempo y dinero para obtener estimaciones más precisas. Además si los resultados no cambian sobre las amplias fluctuaciones en los valores del parámetro, no se necesita un esfuerzo adicional.

En la simulación, el modelador tiene un control absoluto sobre su modelo de aquí que el análisis de sensibilidad sea importante ya que se pueden manipular fácilmente los parámetros del modelo.

Finalmente se necesitan otros dos elementos a incluirse en cualquier proyecto de simulación y son la implantación y la documentación. Es importante que se implante el modelo ya que una de las principales causas de fracaso en la investigación de operaciones y en proyectos de ciencias administrativas se debe al mal entendimiento de los resultados por parte del usuario y a la falta de implantación.

La documentación se relaciona estrechamente con la implantación. La documentación cuidadosa y completa del desarrollo y la operación del modelo pueden incrementar notablemente su vida útil y sus oportunidades para una exitosa implantación. La buena documentación facilita la modificación y asegura que el modelo pueda usarse a pesar de que los servicios de los que se encargan del desarrollo ya no estén disponibles.Así como poder usarse en caso de realizar proyectos en el futuro que la requieran.

## *2.4 VENTAJAS Y DESVENTAJAS DE LA SIMULACION.*

Todos los modelos de simulación se llaman modelos de entrada-salida. Es decir, ellos producen la salida del sistema si se les da la entrada a sus subsistemas interactuantes. Por tanto, los modelos de simulación se "corren" en vez de "resolverse" a fin de obtener la información o los resultados deseados. Son incapaces de generar una solución por sí mismos en el sentido de los modelos analíticos; sólo pueden servir como herramienta para el análisis del comportamiento de un sistema en condiciones especificadas por el experimentador. Por tanto, la simulación no es una teoría, sino una metodología de resolución de problemas. Además la simulación es sólo uno de varios planteamientos valiosos para resolver problemas que están disponibles para el analista de sistemas. Debido a que es necesario y deseable ajustar la herramienta o técnica al problema en vez de lo contrario, esto hace que surja la pregunta ¿Cuándo es útil la simulación?

Hemos definido la simulación como la experimentación con un modelo de un sistema real. Un problema experimental aparece cuando surge la necesidad de cierta información específica acerca de un sistema, la cual no está disponible en las fuentes existentes conocidas . Sin embargo algunas veces las desventajas de la experimentación directa son muy grandes.

1. Puede interrumpir las operaciones de la compañía.
2. Si la gente es parte integral del sistema, pueden verse afectados los resultados, el hecho de que se observe a la gente puede modificar su comportamiento.
3. Mantener las mismas condiciones operativas para cada repetición o corrida del experimento, puede ser muy difícil.
4. El obtener el mismo tamaño de la muestra ( y por tanto importancia estadística) puede requerir más tiempo y ser más costoso.
5. Quizás no pueda ser posible explorar muchos tipos de alternativas en la experimentación del mundo real.

Por lo tanto, el analista debe considerar el uso de la simulación cuando existan una o más de las siguientes condiciones:

1. No existe una completa formulación matemática del problema o los métodos analíticos para resolver el modelo matemático no se han desarrollado aún. Muchos modelos de líneas de espera corresponden a ésta categoría.

2. Los métodos analíticos están disponibles, pero los procedimientos matemáticos son tan complejos y difíciles, que la simulación proporciona un método más simple de solución.
3. Las soluciones analíticas existen y son posibles, pero están más allá de la habilidad matemática del personal disponible. El costo del diseño, la prueba y la corrida de una simulación debe entonces evaluarse contra el costo de obtener ayuda externa.
4. Se desea observar el trayecto histórico simulado del proceso sobre un período, además de estimar ciertos parámetros.
5. La simulación puede ser la única posibilidad, debido a la dificultad para realizar experimentos y observar fenómenos en su entorno real - por ejemplo, estudios de vehículos espaciales en sus vuelos interplanetarios.
6. Se requiere la aceleración del tiempo para sistemas o procesos que requieren de largo tiempo para realizarse. La simulación proporciona un control total sobre el tiempo, debido a que un fenómeno se puede acelerar o retardar según se desee. El análisis de los problemas de deterioro urbano corresponde a ésta categoría.

Una ventaja adicional de la simulación radica en su poderosa aplicación educativa y de entrenamiento. El desarrollo y uso de un modelo de simulación le permite al experimentador observar y jugar con el sistema. Esto, a su vez, le ayudará a entender y adquirir experiencia sobre el problema, por lo que auxiliará al proceso de innovación.

Estas razones entre otras han hecho que la simulación sea una de las técnicas cuantitativas de más uso, para resolver problemas de administración. La mayoría de los administradores y analistas están interesados principalmente en obtener una respuesta a sus problemas inmediatos, pero el fin justifica los medios. Es precisamente esta preocupación por los medios la que nos hace preguntarnos si éstos pueden alcanzarse de la manera mas eficiente y efectiva mediante la simulación. Con frecuencia, la respuesta es no, por las siguientes razones:

1. El desarrollo de un buen modelo de simulación es costoso y requiere de mucho tiempo, ya que demanda un alto grado de talento que no se puede encontrar disponible fácilmente.
2. Puede parecer que una simulación refleja con precisión una situación del mundo real cuando, en verdad no lo hace. Varios problemas intrínsecos de la simulación pueden producir resultados erróneos si no se resuelven correctamente.

3. La simulación es imprecisa, y no podemos medir el grado de su imprecisión. El análisis de sensibilidad del modelo para cambiar valores de los parámetros, sólo puede superar parcialmente esta dificultad.
4. Usualmente, los resultados de simulación son numéricos, y dados a cualquier número de puntos decimales que el experimentador seleccione. Por tanto surge el peligro de la "deificación de los números", atribuyéndoles un mayor grado de validez y de precisión del que se puede justificar.

Lo anterior indica que aunque la simulación sea un planteamiento extremadamente valioso y útil para resolver problemas, en realidad no es una panacea para todos los problema administrativos. El uso y desarrollo de modelos de simulación se refiere en mayor grado a las artes que a las ciencias. En consecuencia, al igual que con otras artes, no es tanto la técnica la que determina el éxito o el fracaso, sino cómo se aplica. Aunque la simulación es un arte, aquellos que poseen la capacidad necesaria de ingenuidad, perspicacia e ingenio pueden dominarla con facilidad.

## *2.5 LENGUAJES DE PROPOSITO ESPECIAL.*

Los primeros esfuerzos en el estudio de simulación se interesaron en definir el sistema que se iba a modelar y en describirlo en términos de diagramas de flujo lógicos y relaciones funcionales. Sin embargo, eventualmente uno se enfrenta al problema de describir el modelo en un lenguaje que sea aceptable para la computadora. La mayoría de las computadoras operan en un lenguaje binario de representación de datos o en algún múltiplo del binario. Debido a que estos lenguajes son inconvenientes para la comunicación, los lenguajes de programación han evolucionado para que los usuarios conversen con la computadora. Lamentablemente se han desarrollado tantos lenguajes de programación de propósito general y de propósito especial a lo largo de los años que es difícil decidir que lenguaje se ajusta mejor o casi se ajusta a una aplicación particular. En consecuencia, el procedimiento usual es adoptar un lenguaje conocido por el analista, no porque sea el mejor sino porque se conoce. Cualquier lenguaje algorítmico general puede expresar el modelo deseado; sin embargo, uno de los lenguajes de simulación especializados puede tener ventajas muy diferentes en términos de facilidad, eficiencia y efectividad.

Las principales diferencias entres los lenguajes de simulación de propósito especial son, por lo general:

1. La organización del tiempo y las actividades
2. La designación y la estructuración de las entidades dentro del modelo.
3. La prueba de actividades y condiciones entre los elementos.
4. Los tipos de pruebas estadísticas que son posibles sobre los datos.
5. La facilidad para cambiar la estructura del modelo.

### *2.5.1 VENTAJA DE LOS LENGUAJES DE SIMULACION.*

El desarrollo evolutivo de los lenguajes de simulación comenzó a finales de los años cincuenta. Al principio, los lenguajes que se usaron en simulación eran lenguajes de una máquina específica o de propósito general. Emshoff y Sisson reconstruyen los antecedentes históricos de los lenguajes de simulación como un proceso en el que los analistas pasan por una secuencia de etapas muy parecida a la siguiente:

a) El analista analiza mentalmente el problema.
b) Puede exponer la situación en algún idioma nativo, tal como el inglés aumentado por una terminología técnica.
c) Describe la situación con tanta precisión como le sea posible en este lenguaje.
d) La descripción es usada por un programador ( el cual puede ser el mismo analista) para preparar un programa de cómputo en un lenguaje de propósito general.

Después de repetir esta secuencia muchas veces, los analistas observaron que en gran cantidad de situaciones, siendo simuladas, podían clasificarse ampliamente como sistemas que implicaban el flujo de datos a través de procesos. Debido a que un gran número de los programas tenían procesos funcionalmente similares, la idea de crear lenguajes de propósito especial evolucionó casi simultáneamente dentro de varios grupos de investigadores a principios de los sesenta. Estos lenguajes cambiaron gradualmente de programas de lenguajes ensambladores con características especiales, por medio de lenguajes orientados a problemas comercialmente disponibles, hasta los lenguajes sofisticados de simulación de propósito especial. Cualquier lenguaje de programación algorítmico puede usarse para la modelación de simulación, pero aquellos lenguajes diseñados específicamente para el propósito de simulación por computadora proporcionan ciertas características útiles:

a) Reducen la tarea de programación.
b) Proporcionan una guía conceptual
c) Ayudan a definir las clases de entidades dentro del sistema
d) Proporcionan flexibilidad de cambios.
e) Suministran un medio de diferenciación entre entidades de la misma clase mediante características o propiedades.
f) Describen la relación de las entidades entre si y con su ambiente común.
g) Ajustan el número de entidades conforme varían las condiciones dentro del sistema.

Emshoff y Sisson creen que todas las simulaciones requieren de ciertas funciones comunes que diferencian un lenguaje de simulación de un lenguaje algebraico general o de uno de programas de aplicación comercial o administrativa. Entre éstas se encuentra la necesidad de:

a) Crear números aleatorios.
b) Crear variables aleatorias.
c) Variar el tiempo, ya sea por una unidad fija o hasta que ocurra el siguiente evento.
d) Registrar datos para salida.
e) Efectuar análisis estadísticos sobre datos registrados.
f) Arreglar salidas en formatos específicos.
g) Detectar e informar inconsistencias lógicas y otras condiciones de error.

Además, ellos indican que para las simulaciones en las que se procesan datos discretos mediante operaciones específicas, los siguientes procesos comunes están presentes por añadidura:

a) Determinado el tipo de evento (después de la recuperación de una lista de eventos).
b) Llamar subrutinas para que ajusten las variables de estado, como resultado del evento.
c) Identificar las condiciones de estado específicas.
d) Almacenar y recuperar datos de las listas (tablas o arreglos), incluyendo la lista de eventos y aquellas que representen el estado.

Algunos de los lenguajes de simulación son lenguajes, en el sentido más general, que, además de unir al usuario con la computadora como medio de conversación, le proporcionan ayuda para formular el problema. Al tener un

vocabulario y una sintaxis, dichos lenguajes son descriptivos, en consecuencia, los usuarios tienden, después de utilizarlos algún tiempo a pensar en términos de ellos. Por lo tanto, las dos razones mas importantes para utilizar los lenguajes de simulación en lugar de los de propósito general, son la ventaja de la programación y la articulación de conceptos. La articulación de conceptos es importante en la fase de modelado y en el planteamiento en general dedicados a la experimentación del sistema. La utilidad del programa se vuelve importante durante la escritura real del programa de cómputo. Otra ventaja de los lenguajes de simulación es su uso como dispositivos de comunicación o documentación. Cuando están escritos en lenguajes parecidos al inglés las simulaciones pueden explicarse más fácilmente a los gerentes de proyectos y a otros usuarios que no están familiarizados con la programación.

Una desventaja muy importante del uso de los lenguajes de simulación es que, debido a que la mayoría fueron desarrollados por organizaciones particulares para su propio uso y puestos a disposición del público, más como un gesto intelectual y de conveniencia que como un producto comercial, la mayoría de los usuarios, acostumbrados a que los fabricantes de computadoras hagan el trabajo de apoyo de compilación como un servicio, no están dispuestos a realizar este trabajo por si mismos. Sin embargo, cada vez hay más lenguajes nuevos de simulación bien documentados.

A continuación se presenta una tabla donde se resumen las ventajas y desventajas de unos y otros lenguajes.

### *2.5.2. CLASIFICACION DE LENGUAJES.*

En general se puede decir que existen tres técnicas de cómputo para la simulación. Estas son: la digital, la analógica y la híbrida. La mayoría de éstos lenguajes tienen varias versiones y dialectos. Por lo tanto, hemos decidido usar nombres de familia o genéricos en lugar de listar todas las diversas versiones.

Las técnicas de simulación digital incluyen el uso de lenguajes de propósito general y de propósito especial. Las ventajas y desventajas relativas a cada una de estas clases ya las presentamos en la tabla 1. Sin lugar a dudas, el uso de un lenguaje de propósito general le ofrece al programador la máxima flexibilidad en cuanto al diseño, implantación y uso de su modelo. Sin embargo, esta flexibilidad se obtiene a expensas de mayor dificultad en programación. Gran parte de esta dificultad se deriva de los problemas de ordenamiento, cronometraje y control sin embargo muchos programadores habilidosos y

experimentados prefieren los lenguajes de propósito general por la flexibilidad del formato de informe de salida y la habilidad para escribir sus propias subrutinas.

Los lenguajes de simulación digital de propósito especial incluyen dos grupos o tipos diferentes de simulación, los cuales se desarrollaron por separado. Estos grupos (la simulación de procesos de cambio discreto y de cambio continuo) se tratan individualmente, junto con su desarrollo histórico. Hablaremos primero de las técnicas de simulación de cambio continuo. Una técnica incluida en esta categoría representa el final del espectro híbrido, en el cual la hibridación ocurre en un nivel de conceptualización (es decir, la computadora digital imita a la computadora híbrida). La categoría de los lenguajes de simulación de cambio continuo puede dividirse en tres tipos de lenguajes: los lenguajes de simulación analógica; el lenguaje dirigido a la solución de ecuaciones incrementales que representan un sistema cerrado determinístico de lazo, y los lenguajes de simulación de ecuaciones. Los dos primeros tipos son lenguajes orientados a bloques y el tercero se construye directamente a partir de las ecuaciones.

Los lenguajes de simulación de cambio continuo se derivan del trabajo de Selfridge en 1955. Su programa sin nombre ha sido la inspiración para el gran campo de lenguajes de simulación de tipo analógico para las computadoras digitales. Entre los lenguajes de simulación analógica más útiles están MIDAS, PACTOLUS, SCADS, MADBLOC, COBLOC y 1130 CSMP. Estos lenguajes emulan el comportamiento de computadoras analógicas e híbridas sobre una base de componente- por - componente. Por ejemplo, un sumador es reemplazado por un código operacional de suma, un integrador por un código operacional de integración, etc. Todos los lenguajes de simulación analógica se inspiraron y motivaron en el diagrama de bloques analógicos, como un medio simple y conveniente para describir los sistemas continuos.

Tres lenguajes que se ajustan a la representación de un lenguaje útil para modelar variables continuas en valor, pero discretas en tiempo. El DYNAMO (Dynamic Models) se desarrolló en el M. I. T., utilizando ecuaciones incrementadas de primer orden para aproximar el proceso continuo, debido a que son más prácticas que las ecuaciones diferenciales para representar las secuencias de tiempo de la entrada y salida del sistema. Las variables esenciales de un estudio de sistemas dinámicos, las variables de estado y las variables de salida se describen en el DYNAMO mediante ecuaciones de niveles y ecuaciones de velocidad, respectivamente. Las variables de estado (niveles) describen el estado o condición del sistema en un momento determinado; las

variables de proporción de flujo (RATE) describen como cambian los estados con el paso del tiempo. Las ecuaciones auxiliares, componentes de las ecuaciones de velocidad, describen totalmente la función de las ecuaciones de velocidad. Aunque el sistema está descrito por las ecuaciones de niveles y velocidad, las ecuaciones auxiliares son muy importantes para representar el control de retroalimentación, debido a que forman la base de control de las velocidades. Si el DYNAMO no está disponible también puede usarse el MIMIC o el CSMP.

TABLA 1

| LENGUAJE DE PROPOSITO GENERAL | |
|---|---|
| VENTAJAS | DESVENTAJAS |
| 1. Número mínimo de restricciones impuestas al formato de salida | 1. Tiempo de programación más largo. |
| 2. A menudo muy bien informado en el lenguaje | 2. La depuración de los términos de simulación no es una característica. |
| **LENGUAJES DE PROPOSITO ESPECIAL** | |
| VENTAJAS | DESVENTAJAS |
| 1. Requiere menos tiempo de programación | 1. Debe apegarse a los requerimientos del formato de salida del lenguaje. |
| 2. Proporciona técnicas de comprobación de errores | 2. Flexibilidad reducida de los modelos |
| 3. Ofrece un medio conciso y directo para expresar los conceptos que surgen en un estudio de simulación. | |
| 4. Tiene la habilidad de construir y proporcionar las subrutinas del usuario | |
| 5. Genera automáticamente ciertos datos que se necesitan en las corridas de la simulación. | |
| 6. Facilita la recopilación y el despliegue de los datos producidos. | |
| 7. Controla la administración y la asignación del almacenamiento de la computadora, durante la corrida de la simulación | |

Existe una separación en dos escuelas de la primer teoría de simulación de cambio discreto: las introducidas por IBM con su lenguaje GPSS, que usa símbolos de diagrama de flujo como descriptores de modelo básico, y las escuelas orientadas a instrucciones. En general los lenguajes enfocados al diagrama de flujo son más fáciles de aprender, aunque los lenguajes orientados a instrucciones son comúnmente mas flexibles. La mayoría de los lenguajes mas recientes son lenguajes de instrucciones, a pesar de que el lenguaje de diagrama de flujo es atractivo, y aparte del GPSS, ha sido utilizado el SIMCOM y BOSS. En nuestro esquema de clasificación, hemos usado cuatro subcategorías: orientaciones de flujo a actividades, de eventos, de procesos y de transacciones. Los lenguajes orientados a flujo de transacciones son realmente lenguajes de proceso, debido a que toman una perspectiva sinóptica de los sistemas, pero los hemos considerado en una categoría individual debido a su orientación de diagrama de flujo. Los lenguajes de eventos, actividades y procesos (excepto el SIMCOM) usan instrucciones de programación para describir las relaciones de causa y efecto entre los elementos del sistema.

Los lenguajes orientados a actividad representan actos pendientes de tiempo como ocurrencias simultáneas en tiempo simulado. Al usar estos lenguajes, no asignamos un calendario o itinerario a los sucesos dentro de un programa, sino que especificamos las condiciones en las cuales éstas pueden ocurrir. Ninguna instrucción de itinerario de actividad programada aparece en estos lenguajes, pero éstos contienen programas ejecutivos que pueden analizar conjuntos de condiciones, antes de cada incremento de tiempo de simulación, para determinar si cualesquiera de las actividades pueden llevarse a cabo. En este tipo de lenguaje, el programa consiste de una sección de prueba y de una sección de acción. Cuando se adelanta el tiempo de simulación, se examinan todos los programas de actividades para determinar su posible rendimiento. Todas las condiciones de prueba deben cumplirse para que las instrucciones de cambio de estado y de tiempo de calentamiento de la sección de acción se lleven a cabo. Si alguna de las condiciones de prueba no se cumple las instrucciones de acción se omiten Mediante una exploración cíclica de los programas con base en actividades, nos aseguramos de que todas las posibilidades tengan oportunidad de llevarse a cabo y de que se responda por todas las interacciones. Entre los lenguajes orientados a actividades están el CSL, ESP, FORSIMIV, GSP y el MILITRAN.

Puede ser que el problema disponible se escriba mas eficientemente en un lenguaje orientado a eventos. Cada evento debe representarse como una ocurrencia instantánea en tiempo simulado, programada para ocurrir cuando la dinámica del modelo perciba que existen las condiciones adecuadas para que

ocurra. Los programas separados se requieren, para digamos, un hombre, una máquina y una parte que interactúa. Un programa ejecutivo ordena automáticamente los eventos programados, de tal manera que ocurran de manera adecuada en el tiempo simulado. Entre los lenguajes orientados a eventos se encuentran el SIMSCRPT, GASP, SIMCOM y SIMPAC.

Los lenguajes orientados a procesos intentan combinar la notación concisa de los lenguajes orientados a actividades con las habilidades de los lenguajes orientados a eventos. Un conjunto de eventos asociado a la descripción del comportamiento de un sistema es un proceso. Un proceso existe a través del tiempo y puede tener un comportamiento dinámico. Los procesos son flexibles, pueden programarse para ocurrir, pueden interrumpirse, pueden tener subprocesos que los obedezcan, pueden estar tan programados que son capaces de retrasarse a si mismos y a otros procesos hasta que cumplan ciertas condiciones, etc. Una característica muy importante es que los lenguajes orientados a procesos hacen que un sólo programa actúe como si fueran varios programas, independientemente de que sean controlados por las exploraciones orientadas a actividades o por la formación de itinerarios orientados a eventos. La característica de programación que hace posible este concepto es el punto de reactivación -esencialmente un apuntador que le señala a una rutina de proceso, dónde empezar la ejecución después de que se ha llevado a cabo un comando de demora. El programa ejecutivo tiene más actividad en estos lenguajes que en los lenguajes orientados a actividades u orientados a eventos. E esta categoría se incluyen el SIMULA, OPS, y el SOL.

A pesar de que los diagramas de flujo son a menudo una herramienta vital para describir la lógica y la interrelación de los elementos de los sistemas modelados en los tres tipos de cambio discretos descritos, éstos no son fundamentales para su programación o teoría. En ellos, el usuario construye un modelo de simulación escribiendo instrucciones que definen las condiciones que deben mantenerse para que sucedan ciertas acciones, describen los resultados de dichas acciones y especifican las relaciones de tiempo entre los elementos y actividades del sistema en el cual participan. En el último tipo de lenguaje que se describió, se aplican los conceptos de los lenguajes orientados a procesos, pero los sistemas se modelan mediante la investigación de los flujos de transacciones a través de bloques de actividades. El tiempo de simulación avanza conforme las transacciones pasan a través de los bloques, y las decisiones se toman conforme se reproduce la lógica del sistema simulado. Los lenguajes que se ajustan a esta categoría se conocen como lenguajes de flujo de transacciones. Una persona que conoce los conceptos del diagrama de flujo no tiene problema alguno para aprender a modelar en estos lenguajes. Los

bloques especializados se ensamblan para formar estructuras que representan la lógica y el flujo del sistema que se está modelando. El sistema se representa en términos de bloques, el programa crea transacciones, dirigiéndolas a los bloques específicos, y ejecuta las acciones asociadas con cada bloque. Debido a que estos bloques de modelación especializados también son las instrucciones básicas de programación, la construcción de un modelo de diagrama de flujo equivale a elaborar un programa. El precio que se paga por esta facilidad de aprender a usar estos lenguajes es una pérdida de flexibilidad. Los lenguajes en esta categoría son el GPSS y el BOSS.

Cabe mencionar aquí, que a la par del avance de los lenguajes de propósito general, los lenguajes de propósito especial también lo han hecho, y a últimas fechas se tienen paquetes como el MODSIM II que es un lenguaje de programación y simulación orientado a objetos. Nuevas versiones del GPSS, SIMULA, así como nuevos paquetes tales como el MODEL - C que es un programa de simulación con uso de diagramas de bloques o el ACSL que sirve para simular modelos de sistemas de control dinámicos, se encuentran entre los programas comerciales que se difunden en universidades y revistas especializadas. Finalmente, con el desarrollo de la programación orientada a objetos, la cual descansa sobre tres ideas principales, clases, objetos y la herencia de sistemas estructurados, se está desarrollando simulación con el C + +, un lenguaje orientado a objetos.

# CAPITULO 3.

# CONCEPTOS DE PROBABILIDAD USADOS EN SIMULACION.

*" Para que los hombres no malicien que tu relato es falso mantén la probabilidad a la vista. "*

*John Gay*

## *3.1 GENERACION DE VARIABLES ALEATORIAS.*

Los modelos de simulación discretos que vamos a estudiar tienen la característica de ser estocásticos. El primer problema a que nos vamos a enfrentar una vez diseñado el modelo, es la generación o descripción de las variables aleatorias que se vayan a utilizar.

A continuación discutimos este problema, empezando por el caso discreto y después el continuo. En este último, analizamos las distintas posibilidades que se presentan en el modelaje de simulación.

***3.1.1 CASO DISCRETO.*** Si la variable es discreta y solo puede tomar n valores dados que son $x_i$, $1 \leq i \leq n$, cuya probabilidad es $p_i$, sabemos que:

$$\sum_{i=1}^{n} p_i = 1$$

Estas probabilidades pueden ser dadas de antemano, o bien determinarse mediante una serie de observaciones, a partir de las cuales se establecen las diferentes probabilidades, como se muestra en el siguiente:

Ejemplo 1: En el crucero de las calles A y B, se realizó la siguiente observación sobre los vehículos que circulaban sobre la calle A:

| $x_i$ | observaciones | probabilidad (pi) | probabilidad acumulada |
|---|---|---|---|
| a la derecha | 28 | 0.28 | 0.28 |
| a la izquierda | 23 | 0.23 | 0.51 |
| no dan vuelta | 49 | 0.49 | 1.00 |

La variable estocástica $x_i$ puede tomar los siguientes valores son tres a saber: dar vuelta a la derecha, dar vuelta a la izquierda y no dar vuelta (observe que los posibles valores no necesariamente deben ser numéricos, pueden ser como en este caso, acciones. Lo que se exige para que sea discreta es que el número de posibles resultados sea finito) con las siguientes probabilidades:

| VALOR | derecha | izquierda | no da vuelta |
|---|---|---|---|
| PROBABILIDAD | 0.04 | 0.21 | 0.25 |

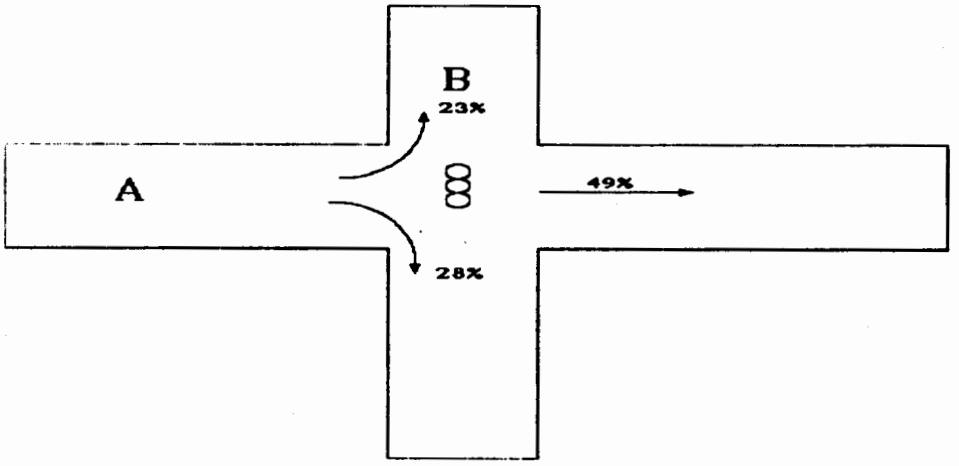



figura 3.1

Para generar números aleatorios con estas probabilidades, podemos hacerlo, considerando la gráfica de la probabilidad acumulada, como se muestra en la figura 3.2, generamos una sucesión ri, de números aleatorios con distribución uniforme entre cero y uno, y dependiendo el intervalo en que este el número aleatorio será, el valor que le asociemos, por ejemplo:

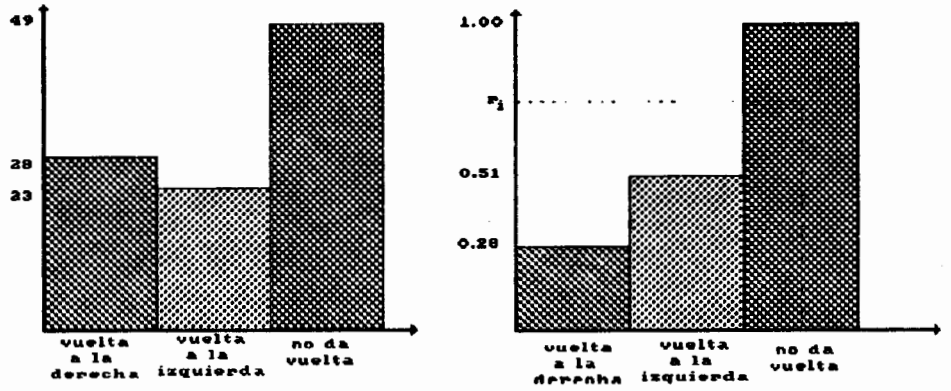



figura 3.2

| $r_i$ intervalo valor de $x_i$ | 0.274 (0,.28) derecha | 0.911 (.51,1) no da vuelta | 0.046 (0,.28) derecha | 0.466 (.28,.51)iz quierda | 0.4976 (.28,.51) izquierda |
|---|---|---|---|---|---|

***3.1.2 CASO CONTINUO.***- Teóricamente la variable puede tomar un número infinito de valores, y la probabilidad de cada uno de estos valores (teórica) es cero, y la podemos describir a través de una función de distribución de probabilidad (f.d.p.) f(x), que debe cumplir:

$$i) \quad f(x) \geq 0 \quad para\ toda\ x \in R$$
$$ii) \quad \int_{-\infty}^{\infty} f(x)\,dx = 1$$

y la probabilidad de que x tome un valor entre $x_1$ y $x_2$.

$$P(x_1 \leq x \leq x_2) = \int_{x_1}^{x_2} f(x)\,dx$$

Discretización de funciones continuas. Algunas veces resulta conveniente discretizar funciones continuas, en otras ocasiones no tenemos más que esa alternativa, veamos pues los dos casos más frecuentes.

a) La variable aleatoria tiene una función de densidad conocida y continua $y = f(x)$. De donde $F(x)$ la podemos evaluar para algunos xi, esto siempre es posible por métodos numéricos, esto es:

$$F(x) = \int_{-\infty}^{\infty} f(x)\ dx$$

Una vez encontrado una sucesión de puntos $(x_i, F(x_i))$, los unimos por medio de segmentos de rectas, como se ilustra en la figura 3.3

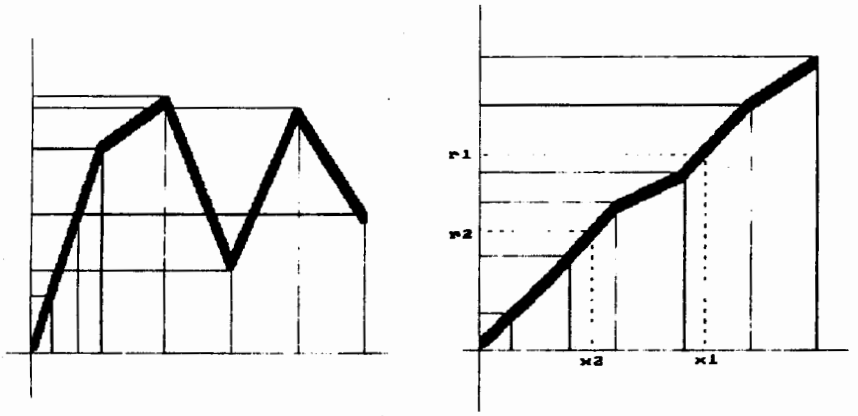



figura 3.3

Para generar números aleatorios que sigan esa distribución generamos primero una sucesión de números aleatorio con distribución uniforme $r_i$, cada uno de estos valores, caerá en alguno de los intervalos determinados por $F(x_i)$, $F(x_{i-1})$, para alguna i, mediante interpolación obtendremos el valor $y_i$ buscado. Veamos un ejemplo:

Suponga que la función de densidad esta dada de la siguiente forma:

$$f(x) = \begin{cases} 2(1-X) & si 0 \le x \le 1 \\ 0 & en\ otro\ caso \end{cases}$$

Lo primero que se hace es verificar si efectivamente se trata de una función de densidad, para lo que integramos f(x):

$$\int_{-\infty}^{\infty} f(x)\,dx = \int_{-\infty}^{0} dx + \int_{0}^{1} 2(x-1)\,dx + \int_{1}^{\infty} 0\,dx$$

*de donde:*

$$\int_{-\infty}^{\infty} f(x)\,dx = 0 + \int_{0}^{1} 2(x-1)\,dx + 0 = 2x - x^2 \Big|_0^1 = 1$$

Como $f(x) \geq 0$ para toda x, entonces se f es una función de densidad, y además $F(x) = 2x - x^2$ si $x \epsilon (0,1)$, 1 si $x \geq 1$ y 0 si $x \leq 0$, de ahí que la sucesión de $x_i$ la debemos de tomar en el intervalo (0,1), esta sucesión no tiene que estar necesariamente a intervalos regulares en x, aquí los tomaremos equidistantes por comodidad, tomemos por ejemplo 10 valores de x y evaluemos para esos valores F(x), como se indica en la siguiente tabla:

| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| f(x) | 0.19 | 0.36 | 0.51 | 0.64 | 0.75 | 0.84 | 0.91 | 0.96 | 0.99 | 1 |

Para fines didácticos, esta división es buena, observe que las variaciones en x, son constantes, mientras que las variaciones de F(x) cada vez son menores, una mejor selección sería empezar con valores más próximos a 0 e ir incrementando la variación de los intervalos.

Una vez que hemos evaluado puntualmente nuestra función, podemos considerarla como lineal entre cada par de puntos, como se muestra en la figura 8, para generar números con esta distribución, generamos una sucesión ri, de números aleatorios con distribución uniforme entre cero y uno, y dependiendo el intervalo en que este, le asignamos un valor interpolando entre los correspondientes valores de las F(x). Como se muestra en la tabla siguiente.

b) No conocemos la expresión analítica de la función de densidad, pero sabemos que es continua y tenemos -o podemos hacer- observaciones de su comportamiento.

| $r_i$ | Intervalo de F(x) | Intervalo de x | Interpolación $y_i = x_{i-1} + [(x_i-x_{i-1})/(F(x_i)-F(x_{i-1}))]*[r_i-x_{i-1}]$ | $y_i$ |
|---|---|---|---|---|
| 0.517 | (.51,.64) | (.3,.4) | 3+[.1/(.64-.51)]*(.517-.51) | 0.305 |
| 0.240 | (.19,.36) | (.1,.2) | .1+[.1/(.36-.19)]*(.240-.19) | 0.129 |
| 0.459 | (.36,.51) | (.2,.3) | .2+[.1/(.51-.36)]*(.459-.36) | 0.266 |
| 0.305 | (.19,.36) | (.1,.2) | .1+[.1/(.36-.19)]*(.305-.19) | 0.167 |
| 0.035 | (0.0,.19) | (0,0.1) | 0+[.1/(.19-0.0)]*(.035-0.0) | 0.018 |
| 0.649 | (.64,.75) | (.4,.5) | .4+[.1/(.75-.64)]*(.649-.64) | 0.408 |
| 0.156 | (0.0,.19) | (0,0.1) | 0+[.1/(.19-0.0)]*(.156-0.0) | 0.082 |
| 0.094 | (0.0,.19) | (0,0.1) | 0+[.1/(.19-0.0)]*(.094-0.0) | 0.049 |
| 0.216 | (.19,.36) | (.1,.2) | .1+[.1/(.36-.19)]*(.216-.19) | 0.115 |
| 0.910 | (.91,.96) | (.7,.8) | .7+[.1/(.96-.91)]*(.910-.91) | 0.7 |

Ejemplo 2: Considere la variable aleatoria de duración de una llamada telefónica, en una caseta pública y suponga que tenemos 100 observaciones, entonces procedemos a agruparlas en intervalos, la siguiente tabla muestra las observaciones ya agrupadas, en la primera columna se indica el intervalo de tiempo considerado, estos intervalos son cerrados superiormente y abiertos en el limite inferior del mismo, la segunda columna muestra el conteo de llamadas que caen en el, la tercera columna corresponde a la frecuencia relativa de las llamadas en el intervalo, es decir número de llamadas del intervalo entre el número total de llamadas registradas y finalmente la última columna corresponde a la frecuencia acumulada hasta el intervalo en cuestión.

| Intervalo | Conteo | Frecuencia Relativa | Frecuencia Acumulada |
|---|---|---|---|
| (0,30] | 2 | 0.02 | 0.02 |
| (30,60] | 5 | 0.05 | 0.07 |
| (60,90] | 7 | 0.07 | 0.14 |
| (90,120] | 24 | 0.24 | 0.38 |
| (120,150] | 25 | 0.25 | 0.63 |
| (150,180] | 18 | 0.18 | 0.81 |
| (180,210] | 10 | 0.10 | 0.91 |
| (210,240] | 1 | 0.01 | 0.92 |
| (240,270] | 3 | 0.03 | 0.95 |
| (270,300] | 2 | 0.02 | 0.97 |
| (300,420] | 3 | 0.03 | 1.00 |

De la tabla anterior, podemos graficar la última columna, esto es la frecuencia acumulada, haciendo corresponder el límite superior del intervalo a la frecuencia acumulada del mismo, y posteriormente unimos mediante segmentos de rectas estos puntos, como se muestra en la figura 3.3, de esta forma para generar números aleatorios con esta distribución, nos encontramos en este momento, en las mismas condiciones que en el caso anterior. De donde basta generar una serie de números aleatorios ri con distribución uniforme entre cero y uno y asociarle una serie yi mediante interpolación.

Hasta ahora hemos reducido el problema de generar números aleatorios con una distribución dada, en forma analítica o empírica a través de series de números aleatorios con distribución uniforme, con seguridad si resolvemos el problema de generar números aleatorios con distribución uniforme, salvaremos el problema general, para cualquier distribución.

## *3.2 GENERACION DE NUMEROS ALEATORIOS CON DISTRIBUCION UNIFORME, ENTRE CERO Y UNO.*

En los ejemplos anteriores, obtuvimos las series de números aleatorios uniformemente distribuidos, de tablas de números con esta distribución que suelen venir al final de los libros de estadística, esta es la primera de las formas de obtener este tipo de números, otra forma es por métodos físicos y una tercera mediante métodos matemáticos.

3.2.1 ***MÉTODOS FÍSICOS.*** Existe una gran variedad de este tipo de métodos para generar números aleatorios con distribución uniforme. La más socorrida de todas es meter en un saco 10 papelitos, o bolas con números del 0 al 9, si deseamos generar números con 3 cifras, entonces sacamos un primer papel o bola observamos el número marcado y lo escribimos, volvemos a meter el papel o la bola al saco, agitamos y repetimos la misma operación, así hasta completar el número de cifras deseadas. Lo importante es resaltar que una vez observado y anotado el número, regresamos el papelito o la bola al saco.

***3.2.2 TABLAS.*** Localice una tabla de números aleatorios uniformemente distribuidos (generalmente los libros de estadística traen una), elija al azar un número y a partir de el, empiece a seleccionar los demás números determinando la forma de irlos seleccionando, esta puede ser hacia abajo, hacia arriba, en diagonal, etc. Lo único que no vale es irlos tomando salteados o sin ningún orden.

***3.2.3 MÉTODOS MATEMÁTICOS.*** Existe una gran variedad de esta forma de generación y dan lugar a generación de estos números, mediante computadora, con frecuencia son llamados también seudoaleatorios, debido a que realmente la sucesión de números a que da lugar forma ciclos, esto es, a partir de un momento vuelve a repetirse la serie de números. A continuación dos formas de generación:

***Cuadrado del Medio.*** Tome cualquier número de cuatro dígitos y elévelos al cuadrado, del resultado quédese con las 4 cifras intermedias. El resultado vuelve elevarse al cuadrado y de nuevo tome las cuatro cifras intermedias, repita esta operación tantas veces como números aleatorios desee generar. Por ejemplo:

| número | cuadrado del número. | nuevo número |
|---|---|---|
| 3715 | 13801225 | 8012 |
| 8012 | 64192144 | 1921 |
| 1921 | 3690241 | 9024 |
| 9024 | 81432576 | 4325 |
| 4325 | 18705625 | 7056 |
| 7056 | 49787136 | 7871 |
| 7871 | 61952641 | 9526 |

Debe estar claro, que en un momento, llega a generarse un número aleatorio que ya se haya generado. Lo que formaría un ciclo, que volvería a repetirse.

***Métodos congruenciales o residuales.*** Este método al igual que el anterior requiere de un primer número, a partir del cual se genera la serie de números aleatorios mediante la aplicación recursiva de la siguiente fórmula:

$$X_{i+1} = (aX_i + a) \text{ modulo } p$$

Los valores , y p dependen del sistema y equipo a emplearse para la generación de estos números aleatorios, generalmente p es el máximo valor entero que soporte el sistema. Si =1 el método recibe el nombre de aditivo, si =0 el método se dice multiplicativo, el valor inicial xo se le denomina núcleo.

Otra forma más sencilla aun, es utilizar el random con que cuentan los diferentes lenguajes de programación, es fácil ver que el método del cuadrado

del medio, la longitud máxima del ciclo de números aleatorios es de 10,000. En el caso de los congruenciales su longitud depende del módulo elegido. Estuve una tarde generando números seudoaleatorios, probando con distintas semillas y módulos, en los dos métodos aquí descritos y en ninguno de mis intentos genere ciclos mayores a 3,000, de donde concluir que el mejor generador es el random del lenguaje.

## *3.3 GENERACION DE NUMEROS ALEATORIOS DE DISTRIBUCIONES FRECUENTES EN SIMULACION.*

En este apartado supondremos que el problema de generar números aleatorios con distribución uniforme en el intervalo cero-uno ha sido resuelto satisfactoriamente. Se han separado en tiempos de servicio, que incluyen las distribuciones, uniforme en un intervalo, normales y exponenciales y en tiempos de llegada, donde únicamente se expone el caso de llegadas poissonianas. En todos estos casos damos una pequeña descripción y a continuación damos un algoritmo en pascal.

### *3.3.1 PARA TIEMPOS DE SERVICIO.*

***Uniformes en un intervalo:***

1) ***Caso entero***: Sea $a<b$, dos números enteros y deseamos generar números aleatorios enteros comprendidos entre a y b, incluyendo a ambos.

   Sea $y_i$, una serie de números aleatorios con distribución uniforme en (0,1), entonces la serie $x_i$ definida por:

   $x_i$ = mayor entero contenido en $[a + y_i*(b-a+1)]$.

Tiene distribución uniforme, para los posibles valores de $(a,a+1,..,b)$, observe que $y_i$, al tomar valores próximos a cero convierte al factor $y_i*(b-a+1)$, en un valor cercano al cero, de donde $x_i$ tomará el valor de a. Por otro lado $y_i$, no toma el valor de uno, de donde por muy cercano que sea a uno, el factor $y_i*(b-a+1)$, será menor que $b-a+1$, de donde $x_i$ tomará el valor de b.

```
FUNCTION UNIFORME (A,B:INTEGER):INTEGER;
BEGIN
  UNIFORME: = TRUNC(A + RANDOM*(B-A + 1)
END;
```

2). ***Caso continuo:*** Sean a< <b dos números reales cuales quiera, y queremos generar números aleatorios con distribución uniforme en ese intervalo, entonces si la serie $y_i$ tiene distribución uniforme en (0,1), la serie definida por:

$$x_i = y_i * (b-a);$$

tiene distribución uniforme en el intervalo (a,b).

```
FUNCTION UNIFORME_CONTINUA (A,B:REAL):REAL;
BEGIN
     UNIFORME_CONTINUA: = RANDOM  * (B-A)
 END;
```

***Normales con media M y desviación D.***

Primero consideremos el caso de los normales, con media cero, y desviación uno. Si $x_i$, es una serie de números aleatorios distribuidos uniformemente en el intervalo (0,1), entonces la serie definida por:

$$z_i = \frac{\sum_{i=1}^{k} x_i - \frac{k}{2}}{(\frac{k}{12}) * (1/2)}$$

tiene distribución normal (0,1). En la medida que k crece, la aproximación es mejor. Una buena aproximación es para k = 12. de donde la forma anterior se reduce a:

$$z_i = \sum_{i=1}^{12} x_i - 6$$

de donde si la serie $y_i$, definida por:

$$y_i = M + z_i * D$$

donde $z_i$ tiene distribución normal (0,1), tiene distribución normal de media M y desviación D.

Aquí hay que tener cuidado, ya que el resultado teóricamente y mediante el algoritmo puede resultar negativo, cosa que para nuestros fines no puede ocurrir y generaría problemas, en el modelo de simulación.

```
FUNCTION NORMAL (M,D:REAL):REAL;
 VAR
AUX : REAL;
 I  : INTEGER;
BEGIN
 REPEAT
  AUX:=0;
   FOR I:=1 TO 12 DO
       AUX:=AUX+RANDOM;
   AUX:=AUX-6;
   AUX:= M + AUX*D;
 UNTIL AUX>=0;
 NORMAL:=AUX;
END;
```

***Tiempos con distribución exponenciales, de media A.***

La distribución teórica está dada por f(t)=A exp(-At ), de donde la acumulada de probabilidad esta dada por:

$$F(t) = 1-exp(-At)$$

y por lo tanto la imagen inversa de $x_i \epsilon F(t)$, esta dado por:

$$t_i = Ln(1-x_i)/(-A).$$

De donde si $x_i$ tiene distribución uniforme en (0,1), $t_i$ tendrá distribución exponencial de media A. Es claro que si $x_i$ es uniformemente distribuida en (0,1), también lo será, la serie $1-x_i$. y por lo tanto la expresión anterior puede ser resumida en:

$$t_i = Ln(x_i)/(-A).$$

Observe que como xi, el logaritmo resulta negativo y al dividirlo por una cantidad negativa, obtenemos una cantidad positiva.

```
FUNCTION SERVICIO_EXPO (MEDIA : REAL):REAL;
 BEGIN
  SERVICIO_EXPO: = LN(RANDOM)/(-MEDIA)
 END;
```

### *3.3.2 PARA TIEMPOS DE LLEGADA.*

Llegadas poissonianas (o exponenciales), con tiempo medio entre llegadas TA.

Basta observar que si TA es el tiempo medio entre llegadas, entonces A = (1/TA) es el tiempo promedio entre llegadas, siguiendo el razonamiento de la última distribución, y sustituyendo A = (1/TA) obtenemos:

$$t_i = -TA * LN(x_i)$$

```
FUNCTION LLEGADA_EXPO (TA:REAL):REAL;
 BEGIN
   LLEGADA_EXPO = -TA*LN(RANDOM)
 END;
```

## *3.4 CONCEPTOS DE TEORIA DE COLAS.*

En los modelos de simulación existen una serie de situaciones en donde "alguien" o "algo" requieren de un determinado tipo de servicio para lo cual en ocasiones tienen que hacer una cola o esperar en línea ante "alguien" o "algo" que es quien proporciona dicho servicio. Es en este sentido que la teoría de colas ofrece elementos importantes de decisión en la solución a problemas tales como : disminuir o eliminar la congestión de un determinado servicio, minimizar pérdidas de tiempo debidas a operaciones deficientes, minimizar excesos de capacidad, etc.

La naturaleza de ésta situación de espera puede analizarse matemáticamente si se conocen las leyes que gobiernan las llegadas para un servicio, el orden en que son atendidos, y los tiempos para dar el servicio.

En la figura 3.4 se muestra la estructura cualitativa de proceso básico de líneas de espera.

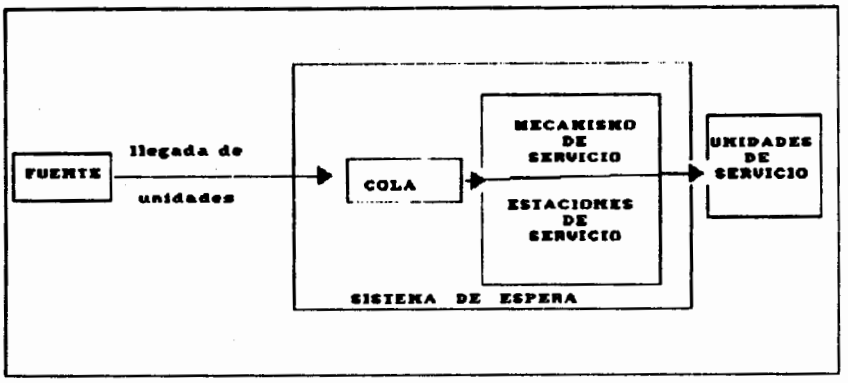



figura 3.4

Existe una fuente de la cual provienen las unidades que requieren un determinado servicio, mismas que llegan al sistema de espera formando una cola. El mecanismo de servicio, a través de estaciones de servicio selecciona en el tiempo a una de las unidades en la cola para prestar el servicio solicitado. Una vez terminado éste, la unidad deja el sistema de espera.

La gran diversidad de situaciones en cada una de las fases del proceso descritas origina una multiplicidad de situaciones de espera, por ejemplo:

| TIPOS DE ENTRADAS | NATURALEZA DEL SERVICIO | PRESTADORES DE SERVICIO |
|---|---|---|
| Clientes | Venta de artículos | Dependientes |
| Barcos | Carga y descarga | Muelles |
| Maquinaria | Reparación | Mecánicos |
| Pacientes | Consulta | Médicos |

De igual forma, se pueden señalar como ejemplos de situaciones de espera los siguientes:

| SERVICIO | LINEAS DE ESPERA | ESTACION DE SERVICIO |
|---|---|---|
| Consultorio | 1 | 1 |
| Peluquería | 1 | muchas |
| Gasolinera | 2 | 1 |
| Banco | muchas | muchas |

Las características mas frecuentes del proceso de espera se mencionan a continuación:

### *3.4.1 CARACTERISTICAS DEL PROCESO DE ESPERA*

1. ***Tipo de Llegadas.*** Una característica de la fuente es el tamaño del número total de unidades que solicitan servicio pudiendo ser finito o infinito. Otra característica es la forma en que las unidades llegan al sistema de espera, pudiendo distinguirse los siguientes casos:

   - Llegadas de unidades al sistema a intervalos iguales de tiempo.
   - Llegadas de unidades al sistema a intervalos desiguales de tiempo pero perfectamente conocidos.
   - Llegadas de unidades al sistema a intervalos desiguales de tiempo cuyas probabilidades son conocidas (intervalos aleatorios).
   - Llegadas de unidades al sistema a intervalos desiguales de tiempo con probabilidades desconocidas y en cuyo caso no puede ser estimado.

Los casos mas típicos de distribución de llegadas con su correspondiente distribución de tiempo entre llegadas, son los siguientes:

| DISTRIBUCION DE LLEGADAS | DISTRIBUCION DE TIEMPO ENTRE LLEGADAS |
|---|---|
| Llegadas que ocurren a intervalos iguales de tiempo | Constante |
| Llegadas poissonianas | Exponencial |
| Tipo poissoniano es intermedia entre las dos anteriores | Erlang (dist. Gama) donde: si $k = 1$ exponencial si $k = \alpha$ constante |
| Caso General | distribución general |

En la práctica existen tipos de llegadas mas complejos: llegadas programadas a ciertos instantes de tiempo pero sujetas a variación; llegadas en grupos, colas cíclicas en las cuales un pequeño número de unidades recicla, etc.

2. ***Disciplina de la línea de espera.*** Existen diferentes disciplinas en la línea de espera, siendo el caso mas sencillo y el que normalmente se considera en modelos de espera: primero en llegar es a quien se le da primero servicio (FIFO). Otros tipos de disciplina se establecen de acuerdo con un cierto orden preferencial para otorgar servicio.

3. ***Mecanismo de servicio.*** El mecanismo de servicio puede constar de una o mas unidades de servicio, conteniendo cada unidad una o mas estaciones de servicio.

   Igual que las llegadas de unidades al sistema, el tiempo de servicio (comprendido desde que la unidad entra al servicio hasta que termina) sigue la misma clasificación que en el punto 1.

| DISTRIBUCION DE SALIDAS | DISTRIBUCION DEL TIEMPO DE SERVICIO |
|---|---|
| Llegadas que ocurren a intervalos iguales de tiempo | Constante |
| Llegadas poissonianas | Exponencial $\mu$ exp(-$\mu$t) |
| Tipo poissoniano es intermedia entre las dos anteriores | Erlang<br>$f_k(t) = t_{k-1}(\mu k)k \exp(-\mu kt)/(k-1)!$<br>si k = 1 exponencial<br>si k = $\sigma$ constante |
| Caso General | distribución general |

Otros factores que pueden influir en el servicio son:

Servicio por grupo: colas en serie en las cuales la salida de una unidad de una estación de servicio es la entrada en la siguiente estación, etc.

En general, soluciones analíticas son difíciles de obtener para problemas muy complejos debiendo usarse la simulación.

### *3.4.2 NOTACIÓN.*

Una forma de describir un proceso de espera en forma sencilla, es a través de una serie de símbolos y diagonales tales como A/B/X/Y,Z/, notación debida a Kendall (1953), en donde A indica la distribución de tiempo entre llegadas, B la distribución de probabilidades de tiempo entre servicios, X el número de estaciones de servicio paralelas, Y la restricción en la capacidad del sistema y Z la disciplina de espera.

Normalmente se utilizan los primeros tres símbolos, omitiéndose los símbolos Y y Z. Así M/D/2 representa un sistema de espera con tiempo entre llegadas exponencial, servicio determinístico, dos estaciones de servicio, ningún límite en la capacidad de servicio y como disciplina primero en llegar primero en servicio.

| CARACTERISTICA | SIMBOLO | EXPLICACION |
|---|---|---|
| Distribución de tiempo entre llegadas (A) | M | exponencial o poisson |
| | D | determinística |
| | Ek | Erlang |
| | G1 | General independiente |
| Distribución de tiempo de servicio (B) | M | exponencial |
| | D | determinística |
| | Ek | Erlang |
| | G | General |
| Número de estaciones de servicio paralelas (X) | 1,2..X | |
| Restricción en la capacidad del sistema (Y) | 1,2..Y | |
| Disciplina de espera (Z) | FIFO | primero en entrar primero en salir |
| | LIFO | último en entrar primero en salir |
| | SIRO | servicio en forma aleatoria |
| | PRI | prioridad |
| | GD | disciplina general |

/M/M/2. Llegadas poissonianas- servicio poissoniano - 2 estaciones de servicio.

/M/G/8. Llegadas poissonianas - servicio con distribución general de tiempo entre servicios - 8 estaciones de servicio.

En la descripción de los modelos de espera se usa la siguiente nomenclatura:

$E_n$ : Estado en el cual hay n unidades en el sistema.

$\lambda$ : Relación media de llegadas = número esperado de llegadas por unidad de tiempo.

$\mu$ : Relación media de servicios = número esperado de unidades servidas por unidad de tiempo (de una estación de servicio)

$1/\lambda$ : Tiempo esperado entre llegadas.

$1/\mu$ : Tiempo esperado entre salidas - tiempo medio esperado de servicio.

s : Número de estaciones de servicio

$\lambda_n$ : Relación media de llegadas de nuevas unidades cuando hay n unidades en el sistema.

n : Relación media de servicios cuando hay n unidades en el sistema.

$q_j(t)$ : Probabilidad de que existan j unidades en el sistema en el instante t.

$r$ : X/s Porcentaje de utilización del servicio = intensidad de tráfico = probabilidad de que el servicio esté ocupado.

# CAPITULO 4.

# SIMULACION DE SISTEMAS DISCRETOS.

" *La realidad no se encuentra ni en el mundo ni en nuestros modelos, sino en el proceso de trabajar alternativamente entre el mundo y el modelo.* "

E.A. Singer.

## *4.1 MODELACION DE SISTEMAS DISCRETOS*

### *4.1.1 IMAGEN DEL SISTEMA.*

La simulación de un sistema la podemos ver, como una secuencia de fotografías del estado del sistema, que llamaremos ***Imagen del sistema***, al conjunto de atributos de las distintas entidades que conforman el modelo. Así una simulación la podemos ver como una película, en el caso de los sistemas continuos, se apega más esta concepción de simulación, ya que las fotos en las películas las tomamos a intervalos regulares de tiempo.

En la simulación de sistemas discretos, donde los cambios en la imagen del sistema no se producen suave y continuamente, sino bruscamente, a intervalos generalmente aleatorios de tiempo, no tiene mucho sentido "tomar varias fotos" de la imagen del sistema, si no se han producido cambios en él. Además puede ocurrir que en un intervalo de tiempo, se produzcan varios cambios y perderíamos así parte de la descripción del comportamiento del sistema que estamos analizando.

Simular un sistema discreto, significa llevar un registro detallado de todos y cada uno de los cambios que se producen en el sistema a lo largo de el período de tiempo que deseamos simular. En el capítulo 2, definimos el concepto de evento, como todo aquello que provoca un cambio en la imagen del sistema. Una simulación es el seguimiento y registro de los tiempos de eventos que secuencialmente se van presentando y que a su vez generan nuevos tiempos de eventos futuros conforme se desarrolla la simulación. Para llevar este seguimiento se requiere de una variable de control que representa el tiempo de simulación.

### *4.1.2 REPRESENTACIÓN DEL TIEMPO.*

La variable de control para la representación del tiempo, la denominamos ***tiempo reloj***, o simplemente ***reloj***. Generalmente al inicio de la simulación tiene el valor de cero y en cualquier momento su valor nos indica el tiempo que se ha simulado. El movimiento del ***reloj***, esta determinado por la secuencia en que se van produciendo los eventos. Este ***tiempo reloj***, no tiene nada que ver con el tiempo en que se realiza la simulación. Esto significa que podemos simular años de un sistema en pocos minutos reales y también podemos simular instantes de la vida de un sistema, en minutos de la realidad.

Existen dos métodos de actualizar el ***reloj***. El primero consiste en avanzar el reloj a la hora en que se producirá el siguiente evento. El segundo método consiste en avanzar el reloj a intervalos regulares de tiempo y analizar los distintos eventos que se han producido durante ese intervalo de tiempo. Este segundo método es el más utilizado cuando se realizan simulaciones de sistemas continuos. El primer método se conoce como orientado a eventos y es el que se utiliza en la simulación de sistemas discretos.

### *4.1.3 GENERACIÓN DE PATRONES DE LLEGADAS.*

La generación de llegadas al sistema pueden ser exógenas. Por ejemplo tomadas de registros históricos del sistema que se va a simular. Este tipo de generación de llegadas, no es muy utilizado en simulación, aunque en algunos casos, nos permite realizar pruebas de validación del modelo realizado.

La generación de llegadas que generalmente se utiliza, son generadas por el mismo sistema de simulación, obedeciendo a una distribución de tiempo entre llegadas. Una llegada es un evento, que provoca un cambio en la imagen del

sistema, al producirse una llegada, generamos el siguiente tiempo de llegada, es decir producimos un evento futuro, que llamaremos ***siguiente llegada***, al llegar el ***reloj*** de la simulación a ese tiempo se produce una llegada al sistema, generando a su vez un nuevo tiempo de llegada y por lo tanto un nuevo evento futuro. Este tipo de generación de llegadas se conoce con el nombre de *boot-stripping* (cordón de bota).

La entidad que llega generalmente requiere de la asignación de ciertos atributos, si estos atributos dependen de las condiciones que guarde el sistema al momento de la llegada, deberán de generarse al momento de la llegada, si la asignación de atributos no dependen de las condiciones del sistema, pueden generarse en el tiempo en que se genera su tiempo de llegada o cuando se produzca su arribo al sistema. Normalmente generamos sus atributos al momento de llegar al sistema.

### *4.1.4. MODELACIÓN*

Las entidades de un sistema, las podemos clasificar en pasivas y dinámicas, ambas entidades tienen atributos, sin embargo son las entidades dinámicas las únicas que producen cambios en el sistema. El primer paso que tenemos que hacer es identificar todas las entidades que conforman el sistema y determinar cuales de ellas son pasivas y cuales dinámicas, definir los atributos de cada una de ellas. El conjunto de entidades y atributos la imagen del sistema.

El establecer con precisión las leyes de comportamiento de cada una de ellas y los mecanismos de interacción nos permitirá definir los distintos eventos que conformarán el algoritmo de la simulación. Definir la imagen del sistema y los distintos eventos que se pueden producir se conoce con el nombre de ***Generación o creación del modelo***. Otro aspecto importante, antes de proceder con la simulación es el establecimiento del criterio de terminación. El algoritmo de simulación consiste en determinar el siguiente evento a realizar, una vez seleccionado determinar si puede ejecutarse, si se ejecuta, modificar la imagen del sistema, recabar estadísticas y preguntar si el criterio de terminación se satisface, si el criterio de terminación es satisfecho generar un reporte de la simulación. En caso contrario volver a determinar el siguiente evento a realizar. Este proceso se ilustra en el diagrama 4.1

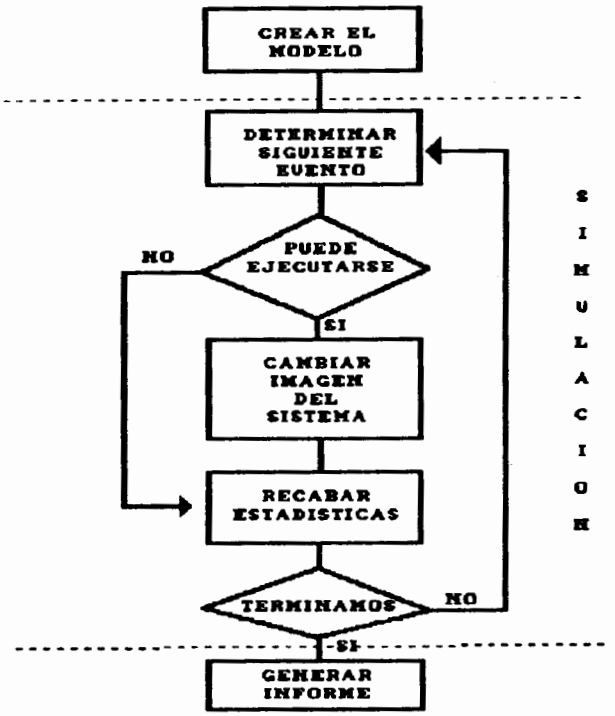



diagrama 4.1

## *4.2 SIMULACION DE UNA TIENDA.*

Para ilustrar el proceso de simulación de los sistemas discretos, lo haremos mediante el siguiente ejemplo:

### *4.2.1 EL PROBLEMA A SIMULAR.*

La tienda.

> Los clientes llegan a una tienda, con un tiempo medio entre llegadas de 10 segundos, las llegadas son poissonianas. Los clientes compran de uno a cuatro artículos, con los siguientes porcentajes:

| articulo | porcentaje |
|---|---|
| 1 | 50 % |
| 2 | 20 % |
| 3 | 20 % |
| 4 | 10 % |

Al llegar a la caja, si esta desocupada pasan a pagar, sino observan la cola, si en la cola hay 3 clientes, dejan la compra y se van, si por el contrario hay menos de tres clientes, se forman y esperan su turno para pagar. El cajero tarda en despachar a sus clientes 10 seg por cada artículo que compra, más 10 segundos en saludar al cliente. Al iniciar la simulación la tienda esta vacía.

### *4.2.2 CONFORMACIÓN DEL MODELO.*

***Determinación de entidades.*** En forma evidente podemos determinar las siguientes entidades, con los siguientes atributos:

| | | |
|---|---|---|
| a) | Cliente. | Número de artículos a comprar. |
| b) | Caja. | Ocupada o desocupada. |
| c) | Cola de la caja. | Tamaño de la cola. |

Otra entidad no tan evidente, es la selección de artículos por parte de los clientes, esta la podemos ver como una cola, que a diferencia de la cola de la caja, que funciona con política FIFO, en esta cola cada vez que llega un cliente se inserta en la cola dependiendo de el tiempo que se vaya a tardar en seleccionar sus artículos. Puede darse el caso de que un cliente que llegue después que otro, termine su selección primero, de donde:

| | | |
|---|---|---|
| d) | Cola Selección de artículos. | Tamaño. |

***Determinación de eventos.***

| | | |
|---|---|---|
| 1) | Llegadas de clientes. | Es evidente que cuando se produce una llegada el sistema va a cambiar de imagen. Se incrementa y modifica la cola de selección de artículos. |
| 2) | Selección de artículos. | Decrementa la cola de selección, si la caja esta desocupada la ocupa. Si por el contrario esta ocupada y la cola de la caja es menor que 3, se forma a esperar y por lo tanto incrementa la cola de caja. Si la cola es de 3 deja sus artículos y abandona la tienda, modificando así las estadísticas. |
| 3) | Libera caja. | Cuando un cliente termina de ser atendido, si hay alguien en cola, pasa a ser atendido. Si no hay nadie en cola, la caja se declara desocupada. |

A cada uno de los eventos, se encuentra ligado un tiempo de realización del evento. Observe como a partir de la primera llegada se genera una lista de eventos futuros. Al llegar el primer cliente se genera la llegada del siguiente cliente, así habrá siempre un evento futuro llamado llegada. Y genera también otro evento futuro el de selección de artículos que tendrá asociada la hora en que termine de hacer su selección. Al producirse el evento futuro selección de artículos, si la caja esta desocupada, la ocupa y genera con ello un evento futuro de tipo 3, liberar caja. Si la caja se encuentra ocupada no generará ningún evento futuro, simplemente modificará la imagen del sistema, decrementando la cola de los clientes que están seleccionando artículos, si la cola es menor que tres, la incrementará. Si la cola es de tres, modificará nuestras estadísticas al abandonar el sistema. Finalmente en el evento futuro tipo 3: liberar caja, si no hay clientes en cola modifica el estado de la caja, poniéndola desocupada. Si por el contrario hay clientes en cola, genera un nuevo evento tipo 3, cuyo tiempo de realización depende del número de artículos que halla seleccionado el cliente en turno.

La Imagen del sistema al iniciarse la simulación, se ilustrar en la figura 4.1

Un diagrama de flujo de las distintas actividades que se realizan en la

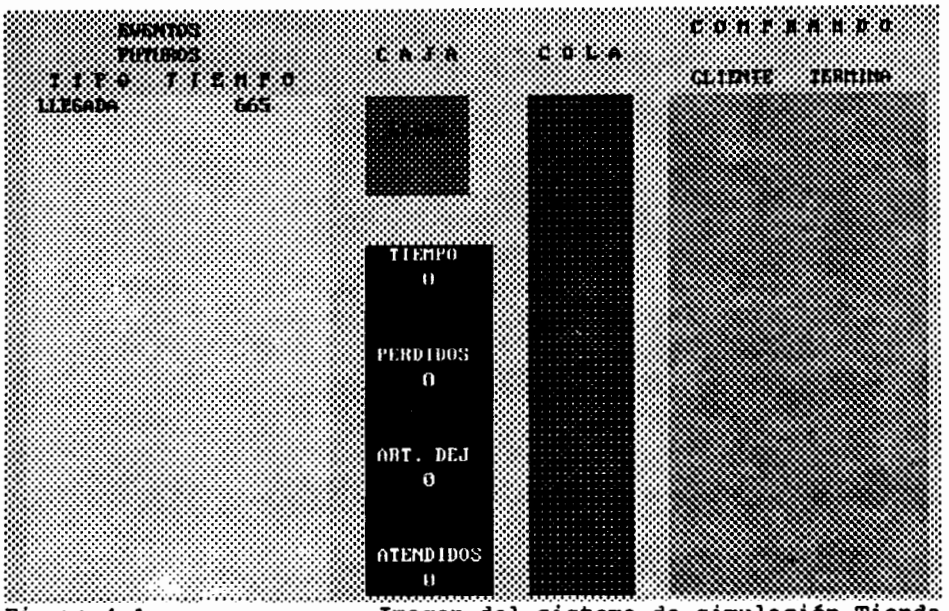



Figura 4.1 Imagen del sistema de simulación Tienda

simulación de la tienda se muestra en el diagrama 4.2

Al inicializarse el sistema, los distintos contadores se ponen en cero y se genera la primera llegada. Al llegar el primer cliente, se genera el evento futuro llegada al generar un nuevo tiempo de llegada. Y se procede a determinar el número de artículos a comprar generando así, otro evento futuro en este caso de tipo 2, cuyo tiempo de realización esta dado por el tiempo actual más el número de artículos a comprar multiplicado por 10.

La figura 4.2, muestra la Imagen del sistema al tiempo reloj de 443, la lista de eventos futuros, que como puede apreciarse son 4 de tipo 2, que corresponde a los clientes que están seleccionando artículos (al lado derecho de la figura), hay tres clientes en cola, en la figura se indica el número de cliente que esta en cola. Hemos ido numerando secuencialmente de acuerdo al orden en que han ido llegando. Hasta este momento (reloj = 443), se han perdido 4 clientes, dejado 7 artículos y sido atendidos 19 clientes. La caja será liberada al minuto 463. Puede apreciarse, que los siguientes tres clientes que terminen de seleccionar su compra, abandonarán el sistema sin comprar.

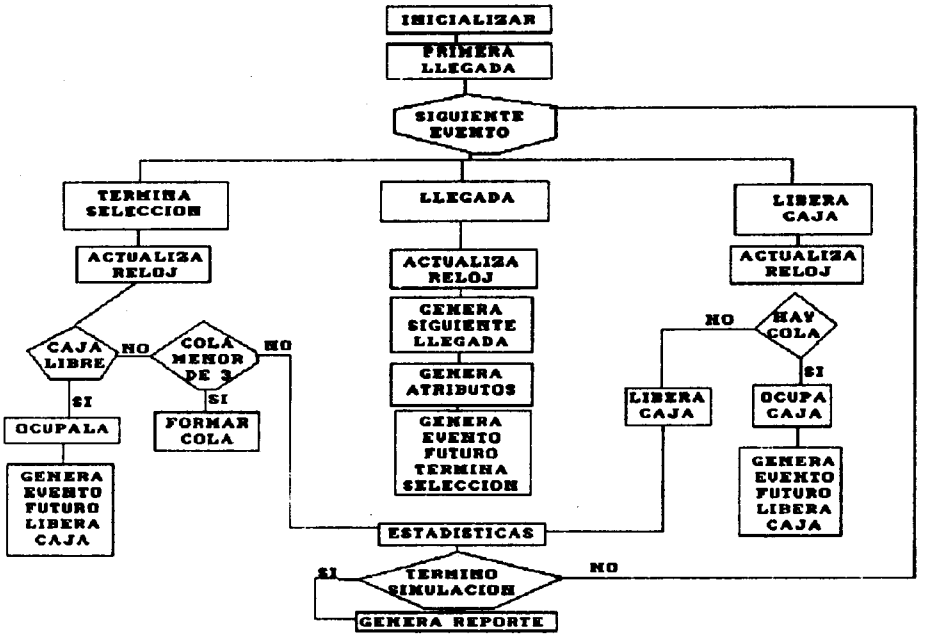



diagrama 4.2

***Los resultados.*** Diseñar las corridas del modelo e interpretar los resultados, es la finalidad última de la simulación. Los resultados de un modelo donde están involucradas variables aleatorias, necesariamente tiene como resultado valores estocásticos. Por ejemplo si perseguimos pronosticar el resultado de una nueva política de funcionamiento en un sistema productivo, mediante un modelo de simulación. Debemos determinar el número de repeticiones que debe de ejecutarse la simulación. Cada una de las corridas tendrá un resultado diferente de donde en la interpretación de los resultados tendremos que hablar de valores medios esperados y de desviaciones.

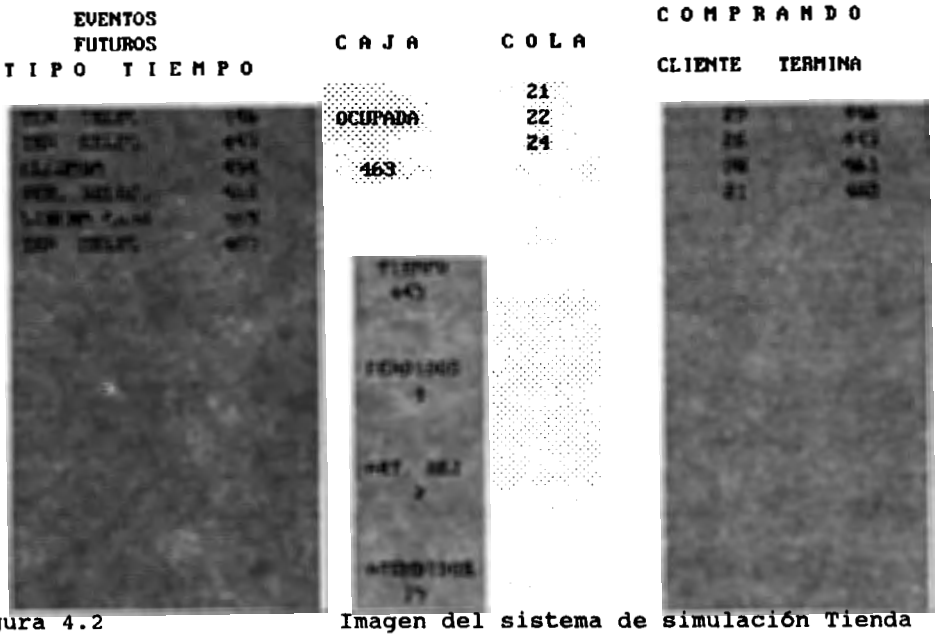



Figura 4.2 Imagen del sistema de simulación Tienda

# CAPITULO 5

# SIMULACION CON PASCAL.

*" Nunca se intentará algo si se piensa en superar primero todas las posibles objeciones."*

*Samuel Johnson*

## *5.1 GENERALIDADES DEL PASCAL.*

Suponemos que el estudiante tiene conocimientos mínimos del lenguaje Pascal, o de algún otro lenguaje de programación. Esta sección esta dirigida para los que no están familiarizados con pascal.

Un programa de Pascal, tiene la siguiente estructura:

a) Definición de tipos.
b) Definición de Variables.
c) Definición de procedimientos y funciones.
d) Cuerpo del programa.

### *5.1.1 TIPOS.*

El pascal tiene predefinidos varios tipos de variables, tanto alfanuméricas como numéricas, las cuales son:

Alfanuméricas:

| | |
|---|---|
| Char : | Corresponde a un carácter, y consume un byte de memoria. |
| String: | Corresponde a una cadena de hasta 256 caracteres. |

Numéricas:

byte : Corresponde a un entero entre 0 y 255, consume un byte de memoria.

Integer:

Word :

Real :

A partir de estos tipos predefinidos, el usuario puede definir nuevos tipos, que pueden utilizar las siguientes estructuras.

ARRAY : Lo podemos ver, en su forma elemental, como un vector de cualquier tipo, ya sea de los predefinidos o definidos por el usuario con anterioridad. El arreglo en formas más complicadas podemos verlo como una matriz, cuyos elementos pueden ser de cualquier tipo definido con anterioridad, ejemplo:

```
type            { le avisamos al compilador que vamos a definir tipos
                  de variables}

Vector = array[1..30] of real;
                {es un vector de 30 componentes, cada una de las
                cuales es un real.}

matriz = array[1..30,1..30] of integer;
                {es una matriz de 30 por 30 componentes, cada una
                de las cuales es un entero}.
```

RECORD : Registro, lo podemos ver como un conjunto de variables de distintos tipos agrupadas en una sola. Un registro es lo que se utiliza para definir una base de datos, lo podemos ver también como una hoja de cardex, de pacientes, alumnos, maestros, etc. Ejemplo:

```
type            {solo es necesario escribir una sola vez type, para
                definir todo un conjunto de tipos    }

Caja  = Record
ocupada : boolean;
se_desocupa : word;
end;            {con end; le indicamos al compilador que hemos
```

terminado de definir nuestro registro }

Apuntadores: (Pointer) Es una estructura dinámica, la podemos ver como una variable que apunta o señala a una estructura, es de gran utilidad en la simulación, generalmente se la asociamos a las entidades dinámicas en un modelo de simulación,

Ejemplo 1

```
type
xevento = ^even;
                {xevento apunta a una estructura llamada even que se
                define posteriormente }

even = record    {definimos a even como un record }
tiempo : word;
tipo  : byte;    {tenemos numerados los distintos posibles eventos
                futuros diferentes}
siguiente : xevento;
                {la estructura even tiene un campo que a su ves es un
                apuntador que apunta a otra estructura de tipo even}

end;             {termina la definición de even }
```

Ejemplo 2

```
Xcliente = ^Cliente;
                {Xcliente apunta a una estructura llamada Cliente que
                definiremos posteriormente }
Cliente = Record
llego  : word;   {hora de llegada}
h_cola : boolean; {verdadero si hizo cola, falso si no hizo}
salió: word;     { hora de salida del sistema }
Arti : byte;     {número de artículos que compró}
t_selec: word;   {tiempo reloj en que termina su selección }
No_compra : boolean;
                {true se retira sin comprar, false termina su compra}
sig: Xcliente;   {este campo es un apuntador que apunta a una
                estructura de tipo cliente }
end;             {termina la definición de cliente }
```

## *5.2 SIMULACIÓN CON PASCAL.*

Cuando vamos a programar, el problema del modelaje del sistema debe estar resuelto, es decir, ya contamos con un diagrama de flujo, las entidades, la imagen del sistema, definición de los distintos tipos de eventos que se pueden producir, cada una de las actividades que genera cada uno de los eventos, las repercusiones de estas actividades en la imagen del sistema, etc.

El primer paso para programar un simulador en pascal, es definir el tipo de estructuras que se van a utilizar. La estructura puede ser algo muy sencillo, para lo cual no será necesario definir ningún tipo de estructura, bastara con definir una simple variable, de algunos de los tipos predefinidos por pascal, como por ejemplo entero, byte, real, booleano, etc.

Generalmente, utilizamos Record para definir los diferentes tipos de estructura que tengan asociados dos o más atributos, sin importar los tipos de los atributos.

Para las distintas actividades que tengan lugar en la simulación, hacemos procedimientos, un procedimiento es lo que en fortran y basic se conoce como subrutina, que no es más que un pequeño programa en pascal.

El tipo de procesos que vamos simular, requieren que manejemos colas, las entidades con las que se requiere este manejo, son en forma natural por sus características, entidades dinámicas, para lo cual emplearemos apuntadores y haremos un breve paréntesis sobre su utilización en colas.

### *5.2.1 MANEJO DE COLAS EN PASCAL.*

El manejo de colas requiere resolver los siguientes problemas:

a) Insertar a alguien en cola.
b) Sacar a alguien de cola.
c) Inicializar una cola.

Los dos primeros incisos, tienen varias modalidades, pueden ser al inicio o final de la cola o a media cola dependiendo de alguna de las características de la variable que estemos insertando o sacando de la cola.

Generalmente cuando modelamos un sistema para su simulación, aparecen varias colas, una de ellas es la de eventos futuros. Para ilustrar el manejo de colas, programaremos el modelo de la tienda, discutido en el capítulo anterior.

### *5.2.2 UN EJEMPLO.*

La tienda.

Los clientes llegan a una tienda, con un tiempo medio entre llegadas de 10 segundos, las llegadas son poissonianas. Los clientes compran de uno a cuatro artículos, con los siguientes porcentajes:

| articulo | porcentaje |
|---|---|
| 1 | 50 % |
| 2 | 20 % |
| 3 | 20 % |
| 4 | 10 % |

Al llegar a la caja, si esta desocupada pasan a pagar, sino observan la cola, si en la cola hay 3 clientes, dejan la compra y se van, si por el contrario hay menos de tres clientes, se forman y esperan su turno para pagar. El cajero tarda en despachar a sus clientes 10 segundos por cada artículo que compra, más 10 segundos en saludar al cliente. Al iniciar la simulación la tienda esta vacía.

Observe que los eventos que producen cambios en la imagen del sistema son:

a) Cuando llega un cliente.- Se incrementa el número de clientes en el sistema y se genera un evento futuro que depende de la hora en que termine su selección de artículos. Aquí puede suceder que un cliente que llegue después que otro, termine de seleccionar primeros sus artículos, por ser éstos menos.

b) Cuando un cliente termina de seleccionar su mercancía. Aquí, si la caja esta libre, procede a ocuparla y genera un evento futuro el de desocupar la caja. Si esta ocupada y la cola es menor de tres, se integra a la cola a esperar su turno. Si por el contrario la cola es tres el cliente deja su mercancía y abandona la tienda sin comprar.

c) Cuando termina de pagar y abandona el sistema. Si hay clientes en la cola, pasa el primero y genera un evento futuro, el de terminar de pagar. Si no hay clientes en cola, la caja se declara libre, lista para atender al siguiente cliente que termine su selección.

```
program tienda;
uses crt;

{Este programa simula la problematica anterior y toma como parámetros: el
tiempo medio entre llegadas, cola máxima permitida y el número de clientes
atendidos que se desean simular}

(* definición de tipos *)

type

{este tipo de registro lo utilizaremos para representar la caja (o al despachador,
según sea su interpretación) de la tienda}

cajaa = record
libre : boolean;     {true si la caja esta desocupada}
t_desocupar : word;
                     {tiempo reloj de la simulación, en que se desocupa la caja}}
end;                 { de caja }

even = ^evento;      {El apuntador even, apunta a un registro de tipo evento que
                     se define a continuación y que nos permitirá llevar una cola
                     de eventos futuros }
evento = record
tipo  : byte;        {tipo de evento, que puede ocurrir durante la simulación}
t_real: word;        {tiempo reloj en que se realizara el evento}
sig : even;          { Apuntador, que apunta a otra estructura de tipo evento}
end;                 {de definición de evento }

clie = ^clientes;    {El apuntador Clie, apunta a un registro de tipo clientes}
clientes = Record
número,              {vamos a numerar a los clientes que van llegando, en este
                     campo guardaremos el número que le corresponde a cada
                     uno de ellos }
h_llegada,           { tiempo reloj en que llega el cliente }
```

```
h_salida,          {tiempo reloj en que abandona la tienda }
t_termina : word;
                   {tiempo reloj en que termina de seleccionar su compra}
artículos : byte;  {Número de artículos a comprar }
siguiente : clie;  {apuntador, que apunta a otra estructura de tipo clie}
end;               { de definición de clientes  }

Var
```

{Las primeras tres variables, que a continuación se definen, se utilizarán como parámetros de la simulación y que serán proporcionados por el usuario al inicio de la simulación}

```
cuantos,           {Número de clientes a simular}
tiempo_entre_llegadas,
                   {Tiempo medio entre llegadas de los clientes}
max_cola : integer;
                   {Número máximo de clientes en cola}
```

{La siguiente variable, se utiliza para representar la unidad de servicio (caja, despachador ).}

```
caja : cajaa;
```

{los apuntadores cabeven y fineven lo utilizamos para formar la cola de eventos futuros}

```
cabeven,
fineven  : even;
```

{Las variables que a continuación definimos de tipo clie-apuntadores, se utilizarán para inicializar y controlar la cola que se forme en caja}

```
cab_cola,
fin_cola,
```

{Con las siguientes dos variables, llevaremos el control de los clientes en la selección de artículos}

```
cab_compra,
fin_compra  : clie;
```



BIBLIOTECA DE LA UNIVERSIDAD NACIONAL DE MEXICO
DEPFI

{La siguientes dos variables las utilizaremos para}

*t_llegada,* {El tiempo reloj de la llegada del siguiente cliente}
*t_reloj : word;* {El tiempo reloj de la simulación}

{Las siguientes variables, las utilizaremos para llevar estadísticas de la simulación, el número de estas variables depende de lo que queramos observar durante el desarrollo de la simulación, dependiendo de nuestros intereses, el número de estas variables puede incrementarse o disminuirse}

*art_dejados,* {número de artículos que se dejan por los clientes que abandonan el sistema sin comprar}
*clientes_perdidos,*
{número de clientes que abandonan el sistema sin comprar}
*cola,* {El tamaño actual de la cola}
*clientes_comprando,*
{Número de clientes que están seleccionando artículos}
*clientes_en_el_sistema,*
{Número de clientes en todo el sistema}
*num_clientes,* {número de clientes que han llegado al sistema }
*clientes_atendidos : integer;*
{ Número de clientes que han concluido sus compras con exito.}

{A continuación pasamos a la definición de procedimientos y funciones que utilizaremos, son de dos tipos, unos nos sirven para realizar la simulación y otros para visualizarla, empezaremos por los de visualización}

***procedure pon_pantalla;***
{Este procedimiento se utiliza en una sola ocasión, limpia la pantalla y enuncia en las primeras líneas la imagen del sistema}
*begin*
*TextBackground(Lightgreen);*
{pone el fondo azul claro}
*TextColor(black);* { usa color blanco para las letras }
*ClrScr;* { limpia la ventana, con el color de fondo }
*window(1,1,80,4);*
*gotoxy(11,2); write('EVENTOS');*
*gotoxy(11,3); write('FUTUROS');*
*gotoxy(5,4); write('T I P O');*
*gotoxy(15,4); write('T I E M P O');*

```
  gotoxy(33,3); write('C A J A');
  gotoxy(47,3); write('C O L A');
  gotoxy(60,2); write('C O M P R A N D O');
  gotoxy(60,4); write('CLIENTE');
  gotoxy(70,4); write('TERMINA');
end;                { pon_pantalla }

procedure pon_imagen;
{Este procedimiento lo utilizamos cada vez que se produce un cambio en la
imagen del sistema}

var
 evento_futuro  : even;
 cleo           : clie;
 cch            : char;
 posy           : integer;
                  {esta variable local, la utilizaremos para escribir en las ventas,
                  donde se despliega la información}
begin
{escribimos eventos futuros }
TextBackground(12);
                  {Color de fondo el 12}
TextColor(black); { usa color negro para las letras }
window(3,5,28,24);
                  {Definimos la parte de la pantalla donde vamos a escribir los
                  eventos futuros}
ClrScr;           {Limpia la región anterior, con el color de fondo }

{Para escribir los eventos futuros, utilizaremos la variable local evento_futuro,
para empezar la hacemos igual al primero de la lista, es decir al que apunta
cabeven}

evento_futuro:=cabeven^.sig;

{inicializamos la escritura en el primer renglón de la ventana, haciendo posy
igual a uno}

posy:=1;
```

```
while evento_futuro <> fineven do
                    {mientras no termine la lista de eventos futuros
                    (evento_futuro <> fineven), has lo siguiente:}
begin
gotoxy(2,posy);  {posiciona el cursor, en la segunda columna y en el renglón
                    correspondiente}
case evento_futuro^.tipo of

{Dependiendo del valor que tenga el tipo del evento futuro, escribe, en nuestro
modelo, solo hay tres tipos de actividades que provocan cambios en el sistema,
el número que aparece a la izquierda es el que se le ha asignado}

3 : write('LLEGADA');      {llega cliente}
2 : write('TER. SELEC.'); {Termina de seleccionar sus artículos el cliente}
4 : write('LIBERA CAJA'); {Terminan de atender a un cliente en caja}
end;                 {del case}


gotoxy(15,posy); {mueve el cursor a la columna 15 en el renglón posy}
write(evento_futuro^.t_real:7);
                    {escribe el tiempo reloj, en que se producirá este evento
                    futuro}

{Con lo anterior acabamos de escribir un evento futuro, las siguientes dos
instrucciones son para prepararnos a escribir el siguiente evento futuro}

inc(posy);          {incrementa el número de renglón, en uno}
evento_futuro:=evento_futuro^.sig;
                    {convierte e evento futuro al evento que sigue de la lista}
end;                {del while}

{ Pasamos a describir en la pantalla el estado actual de la caja }

TextBackground(5);
                    {color de fondo el número 5}
TextColor(black); {usa color negro para las letras }
window(32,5,40,8);
                    {definimos la región de la pantalla donde escribiremos la
                    información}
ClrScr;             {limpia la ventana, con el color de fondo }
```

```
gotoxy(2,2);        {ponemos el cursor en la segunda columna, segundo renglón
                     de la ventana}
if caja.libre then  {si la caja esta libre escribe LIBRE}
write(' LIBRE ') else  {si esta ocupada}
    begin
     write('OCUPADA');{escribe OCUPADA}
     gotoxy(4,4);     {mueve el cursor al renglón 4, columna 4}
     write(caja.t_desocupar);
                     {escribe el tiempo en que se desocupara}
    end;             {del if}

{ Escribimos los clientes que están en cola }

TextBackground(6);  {color de fondo el número 6}
TextColor(black); {usar color negro para las letras }
window(46,5,54,24);
                     {definimos la región de la pantalla donde escribiremos la
                     información}
ClrScr;              { limpia la ventana, con el color de fondo }

{Utilizaremos la variable local cleo, para ir escribiendo a los clientes que están
en la cola}

cleo:=cab_cola^.siguiente;
                     {hacemos a cleo igual al primero de la lista}
posy:=1;             {inicalizamos posy en el primer renglón de la ventana}
while cleo <> fin_cola do
                     {mientras no hallamos escrito a todos los clientes en cola
                     has}

begin
gotoxy(1,posy);   {pon el cursor en la columna 1 y renglón posy}
write(cleo^.numero:5);
                     {Escribe que número de cliente es}
inc(posy);           {incrementa el número de renglón posy}
cleo:=cleo^.siguiente;  {has a cleo igual al cliente que sigue}
end;                 {del while }

{Escribimos los clientes que están comprando, en el orden en que van a
terminar de seleccionar sus artículos }
```

```
TextBackground(7);
                    {color de fondo el número 6}
TextColor(black); {usar color negro para las letras }
window(58,5,79,24);
                    {definimos la región de la pantalla donde escribiremos la
                    información}
ClrScr;             {limpia la ventana, con el color de fondo }
posy:=1;            {inicalizamos posy en el primer renglón de la ventana}

{volvemos a utilizar la variable local cleo, para escribir a los clientes que están
comprando}

cleo:=cab_compra^.siguiente;
                    {cleo igual al primero de la lista}
while cleo <> fin_compra do
                    {mientras no acabemos de escribir a todos los clientes que
                    están comprando vamos a escribirlos}
begin
gotoxy(1,posy);
write(cleo^.numero:7); {primero el No. de cliente }
gotoxy(11,posy);
write(cleo^.t_termina:8);{ el tiempo en que termina}
inc(posy);
cleo:=cleo^.siguiente;
end; {del while}
      { escribimos el reloj }
TextBackground(8);
TextColor(white);       {usa color blanco para las letras }
window(32,11,42,24);
                    {definimos la ventana a utilizar }
ClrScr;             {limpia la ventana, con el color de fondo }
gotoxy(2,1);
write(' TIEMPO');
gotoxy(2,2);
write(t_reloj:5);   {escribimos el tiempo de la simulación}
gotoxy(2,5);
write('PERDIDOS');
gotoxy(2,6);        {Escribimos el No. de clientes perdidos}
write(clientes_perdidos:5);
gotoxy(2,9);        {Escribimos el No. de artículos dejados }
```

```
write('ART. DEJ');
gotoxy(2,10);
write(art_dejados:5);
gotoxy(2,13);      {Escribimos el No. de clientes atendidos}
write('ATENDIDOS');
gotoxy(2,14);
write(clientes_atendidos:5);
cch:=readkey;      {esperamos cualquier teclazo para continuar}
end; {pon_imagen}
```

{Con el procedimiento anterior, concluimos los que utilizaremos para describir el modelo en la pantalla}
{La siguiente serie de procedimientos son para manejo de colas, tanto de eventos futuros como de clientes (manejaremos dos colas de clientes, la de la caja que es de tipo FIFO y la compra, que la tendremos ordena de acuerdo al tiempo en que el cliente termina de seleccionar su compra}

```
procedure mete_cola ( var xx : clie );
{ mete al cliente al final de la cola de caja }
var
  aux1,
  aux2  : clie;
begin
  inc(cola);
  aux1:=cab_cola;
  repeat
    aux2:=aux1^.siguiente;
    if aux2=fin_cola then
     begin
       xx^.siguiente:=fin_cola;
       aux1^.siguiente:=xx;
     end
     else
       aux1:=aux2;
  until aux2=fin_cola;
end; {mete_cola}
```

```
procedure saca_cola ( var xx : clie);
{ este procedimiento saca al primero de la cola }
{ supone que la cola no es vacía}
begin
  dec(cola);
  xx:=cab_cola^.siguiente;
  cab_cola^.siguiente:=xx^.siguiente;
end;


procedure saca_cola_compra ( var xx : clie);
{ este procedimiento saca al primero de la cola_compra }
{ supone que la cola no es vacía }
begin
  xx:=cab_compra^.siguiente;
  cab_compra^.siguiente:=xx^.siguiente;
end;


procedure mete_cola_compra (var xx : clie);
{ este procedimiento mete en cola dependiendo de la hora de  }
{ terminacion de la compra}
var
  aux1,
  aux2   : clie;
begin
    aux1:=cab_compra;
    repeat
      aux2:=aux1^.siguiente;
      if aux2 =fin_compra then
        begin
          xx^.siguiente:=aux2;
          aux1^.siguiente := xx;
        end
        else
        begin
          if xx^.t_termina < aux2^.t_termina
             then
                begin
                    xx^.siguiente:=aux2;
                    aux1^.siguiente := xx;
```

```
                aux2:=fin_compra;
            end
        else
            aux1:=aux2;
    end;
  until aux2=fin_compra;
end;
```

***procedure mete_cola_evento(var xxx : even);***
{Este procedimiento, inserta un evento futuro, en la cola de eventos futuros, dependiendo de la hora en que deba de ocurrir}

```
var
 aux1, aux2 : even;
begin
   aux1:=cabeven;
   repeat
    aux2:=aux1^.sig;
    if aux2 =fineven then
     begin
       xxx^.sig:=aux2;
       aux1^.sig := xxx;
     end
     else
     begin
      if xxx^.t_real < aux2^.t_real
         then
            begin
                xxx^.sig:=aux2;
                aux1^.sig := xxx;
                aux2:=fineven;
            end
        else
            aux1:=aux2;
     end;
  until aux2=fineven;
end;
```

```
function num_de_artículos  : byte;
```
{Esta función, cada vez que se invoque, nos proporciona el número de artículos que va a seleccionar un cliente, con la probabilidad señalada en el enunciado del problema}

```
var
  r : real;
begin
 r:=random;
 if r < 0.5 then num_de_artículos:=1 else
 if r < 0.7 then num_de_artículos:=2 else
 if r < 0.9 then num_de_artículos:=3 else
 num_de_artículos:=4;
end;
```

{Los siguientes dos procedimientos, se utilizan en los procedimientos que describen los distintos tipos de eventos futuros}

```
procedure siguiente_llegada;
```
{genera el siguiente tiempo de llegada y el evento futuro de llegada, que lo insertamos en la cola de eventos futuros al llamar al procedimiento mete_cola_evento}

```
var
  evento_futuro  : even;
begin
  t_llegada:=t_reloj - trunc(tiempo_entre_llegadas * Ln(random));
   new(evento_futuro);
   with evento_futuro^ do
   begin
     tipo:=3;   { el evento tipo 3 es llegada}
     t_real:=t_llegada;
   end;
   mete_cola_evento(evento_futuro);
end;
```

```
procedure ocupa_caja (var xx : clie);
{ ocupa la caja y genera un evento futuro (libera caja)}

var
  evento_futuro : even;
begin
      caja.libre:=false;
      caja.t_desocupar:=t_reloj+10*(xx^.artículos+1);
      new(evento_futuro);
      with evento_futuro^ do
      begin
          tipo:=4;   { el evento tipo 4 es libera_caja}
          t_real:=caja.t_desocupar;
      end;
      mete_cola_evento(evento_futuro);
end; { ocupa_caja}
```

{ empezamos con los procedimientos asociados a los distintos eventos que se presentan en el modelo}

```
procedure llega_cliente;   { evento tipo 3 }
```

{genera al cliente, llena sus campos y lo mete en cola de compradores genera el siguiente tiempo de llegada, actualiza el t_reloj el evento futuro de llegada, determina numero de artículos a comprar por el cliente, y el evento futuro de termina de comprar y actualiza contadores }

```
var
   xx  : clie;
   evento_futuro : even;
begin
    t_reloj:=t_llegada;     { actualizo reloj }
    siguiente_llegada;      { genero tiempo de siguiente llegada }
    inc(num_clientes);
    inc(clientes_en_el_sistema);
    inc(clientes_comprando);
    new(xx);
    with xx^ do
     begin
       numero  := num_clientes;
       h_llegada := t_reloj;
```

```
      h_salida  := 0;
      artículos := num_de_artículos;
      t_termina := t_reloj + 10*artículos;
    end; {de with}
    mete_cola_compra(xx);
   new(evento_futuro);
   with evento_futuro^ do
   begin
     tipo:=2; {el evento tipo 2 es termina de comprar}
     t_real:=xx^.t_termina;
   end;
   mete_cola_evento (evento_futuro);
   pon_imagen;
end;
```

***procedure termina_compra; { evento tipo 2 }***

{ El cliente acaba de hacer su selección de artículos, si la caja esta libre la ocupa, si esta ocupada y hay menos de dos en cola, forma cola y si hay más en cola abandona la tienda}

```
var
  xx : clie;
begin
  saca_cola_compra(xx);
  dec(clientes_comprando);
  if caja.libre then { si la caja esta desocupada }
  ocupa_caja(xx)
    else { la caja esta ocupada }
    begin
      if cola=max_cola then {si hay mucha cola, el cliente se va}
                      begin
                        inc(art_dejados,xx^.artículos);
                        inc(clientes_perdidos);
                        dec(clientes_en_el_sistema);
                      end
              else {hay pocos en cola}
                mete_cola(xx);
    end;
  pon_imagen;
end; {de termina_compra}
```

```
procedure libera_caja;  { evento tipo 4 }
```
{El cliente acaba de efectuar su pago en caja y abandona el sistema, si hay clientes en cola, saca al primero de la cola, decrementa la cola y vuelve a ocupar la caja, si no hay cola libera la caja}

```
var
  xx  : clie;
begin
  inc(clientes_atendidos);
  dec(clientes_en_el_sistema);
  if cola > 0 then { si hay clientes en cola }
     begin
       saca_cola(xx);
       ocupa_caja(xx);
       end
      else
        caja.libre: = true;
  pon_imagen;
end; { de libera_caja}
```

{ Los siguientes dos procedimientos son para inicializar la simulación}

```
procedure toma_entrada;
```
{Este procedimiento solicita al usuario, los parámetros del modelo. El tiempo medio entre llegadas, cola máxima permitida, y el número de clientes a simular, aquí si se comete un error en el ingreso de los datos, como por ejemplo teclear una letra en vez de un número, el sistema se cae}

```
begin
  TextBackground(LightBlue); { pone el fondo azul claro }
  TextColor(white);          { usa color blanco para las letras }
  ClrScr;    { limpia la ventana, con el color de fondo }
  gotoxy(19,11);
  write('TIEMPO MEDIO ENTRE LLEGADAS DE CLIENTES: ');
  gotoxy(22,13); write('          C O L A   M A X I M A : ');
  gotoxy(17,15);
  write('CUANTOS CLIENTES ATENDIDOS, DESEA SIMULAR:');
  gotoxy(60,11); read(tiempo_entre_llegadas);
  gotoxy(60,13); read(max_cola);
  gotoxy(60,15); read(cuantos);
end;
```

***procedure inicia;***
{Este procedimiento, solicita al usuario los parámetros de la simulación invocando el procedimiento anterior. Inicia colas y contadores importantes de la simulación, pone la pantalla que describirá el comportamiento de la simulación y genera la primera llegada al sistema}

```
begin
  toma_entrada;
  t_reloj:=0;
  cola:=0;
  caja.libre:=true; {caja desocupada}
  art_dejados:=0;
  clientes_perdidos:=0;
  clientes_en_el_sistema:=0;
  clientes_comprando:=0;
  num_clientes:=0;
  clientes_atendidos:=0;
  new(cab_cola);
  new(fin_cola);
  new(cab_compra);
  new(fin_compra);
  new(cabeven);
  new(fineven);
  with cabeven^ do
  begin
    tipo:=0;
    t_real:=0;
    sig:=fineven;
  end;{del with cabeven}
  with fineven^ do
  begin
    tipo:=0;
    t_real:=0;
    sig:=nil;
  end;{del with cabeven}
  with cab_cola^ do
  begin
    h_llegada:=0;
    h_salida:=0;
```

```
      siguiente:=fin_cola;
      t_termina:=0;
      articulos:=0;
    end;{del with cab_cola}
    with fin_cola^ do
    begin
      h_llegada:=0;
      h_salida:=0;
      siguiente:=nil;
      t_termina:=0;
      articulos:=0;
    end;{del with cab_cola}
    with cab_compra^ do
    begin
      h_llegada:=0;
      h_salida:=0;
      siguiente:=fin_compra;
      t_termina:=0;
      articulos:=0;
    end;{del with cab_compra}
    with fin_compra^ do
    begin
      h_llegada:=0;
      h_salida:=0;
      siguiente:=nil;
      t_termina:=0;
      articulos:=0;
    end;{del with cab_cola}
    siguiente_llegada;
    pon_pantalla;
    pon_imagen;
  end; {de inicia}
```

{ Aquí comienza el programa principal, se inicializa y se simula mientras el número de clientes atendidos sea menor que los que deseamos simular. Al completarse el número de clientes deseados, la simulación concluye}

```
begin
  inicia;
  while clientes_atendidos < cuantos do
  begin
    evento_futuro:=cabeven^.sig;
    cabeven^.sig:=evento_futuro^.sig;
    { saco al evento futuro de la lista de eventos futuros }
    case evento_futuro^.tipo of
       2  : termina_compra;
       3  : llega_cliente;
       4  : libera_caja;
    end; {del case}
  end;
end.
```

# CAPITULO 6.

# SIMULACIÓN CON GENERAL PORPUSE SIMULATION SYSTEM

# (GPSS-PC).

***Es fácil ver que cualquier tarea que pueda hacer un robot no es apta para que la haga un ser humano. Para expresarlo de otra manera, si un ser humano realiza trabajo que pueda hacer un robot, dicho trabajo acabará por hacer un robot del ser humano.***

***Isaac Asimov. "La Receta del Tiranosaurio"***

## *6.1 INTRODUCCIÓN.*

Existe una gran cantidad de lenguajes de simulación, todos ellos tienen el propósito de facilitar el arduo trabajo de programación que se requiere. Estos lenguajes los podemos clasificar en forma similar que los distintos tipos de simulación : continuos y discretos.

En forma natural, cada vez que se elabora un modelo de simulación, se pretende que todo el esfuerzo computacional que significo trasienda al modelo mismo y podamos utilizarlo de nuevo y si es posible que otras gentes puedan hacerlo también. Este es el origen de los lenguajes de simulación, de ahí que no resulte extraño que cada día aparezcan nuevos lenguajes de éste tipo.

Por la brevedad del curso (quince semanas aproximadamente), he optado por un solo lenguaje de simulación y el uso de lenguajes de propósito general en particular del "C" o Pascal, por ser estos los de mayor uso en nuestras universidades (en el caso del Pascal) y por ser el "C" el lenguaje que empieza a convertirse en estándar dentro y fuera de nuestras instituciones de educación, No hay que olvidar que las versiones de "UNIX", que utilizan las grandes y medianas computadoras están hechas en "C" y este tipo de máquinas son las

que se utilizan en los sectores de servicios e industriales.

El GPSS-PC, fue seleccionado, porque a pesar de la gran cantidad de lenguajes de propósito específico que existen, este es un estándar dentro de la simulación discreta. Es difícil encontrar hoy en día (1993) algún lugar donde se hable de los lenguajes de simulación discreta y no se mencione al GPSS, más aún es difícil encontrar una universidad donde se enseñen lenguajes de simulación y el GPSS no este incluido. Otro factor es que en la Investigación de Operaciones, este tipo de simulación es la de mayor frecuencia. Otro argumento más es que no se exige a los que llevan este curso que manejen con soltura un lenguaje de propósito general, exigencia que con toda seguridad se solicitará antes de que termine esta década. Pienso que el GPSS, es un instrumento de análisis que puede serles de gran utilidad en un futuro próximo, dentro de su desempeño profesional.

Finalmente el último argumento que tengo - y válido solamente para mis alumnos- es que lo mínimo que solicitó para acreditar mi curso, es el manejo satisfactorio de un paquete de simulación.

## *6.2 GENERALIDADES DEL GPSS-PC.*

GPSS esta construido de abstracciones elementales llamadas entidades. A partir de estas entidades podemos crear modelos complejos de simulación. El estudiante debe de adquirir y entender estas entidades y las reglas con las cuales se pueden manipular.

Asociados a cada una de estas entidades, existen "bloques", que realizan funciones específicas con la entidad relacionada. El GPSS define sobre los bloques y entidades un ***S***istema ***N***umérico de ***A***tributos, que nos proporciona el estado (valores) de las distintas variables asociadas a cada bloque o entidad. Finalmente además de los bloques, existen los comandos del GPSS que nos permite realizar la programación e interactuar con la simulación misma, son comandos de edición y control de la simulación.

La idea central de este lenguaje es describir el sistema a simular mediante un diagrama de bloques, cada uno de ellos representan distintas actividades y las líneas que los unen la secuencia en que se ejecutan las actividades. Para lo cual el ***GPSS-PC*** define con precisión distintos tipos de bloques específicos, cada uno de los cuales representa una acción determinada en el sistema. La restricción consiste en que solo se pueden utilizar los bloques definidos en

GPSS.

### *6.3 ENTIDADES DEL GPSS*

a) Dinámicas.

Denominadas transacciones.

b) Equipo.

Son entidades que son utilizadas por las transacciones y son:
***Instalaciones***. Utilizadas por una sola transacción.
***Almacenes***. Usadas por varias transacciones simultáneamente.
***Switches Lógicos***. Entidad que puede estar prendida o apagada. Puede ser consultada y/o modificada por las transacciones.

c) Estadísticas.

Son entidades que nos sirven para recabar estadísticas y conteos que se consideren necesarios al realizar una simulación y son:
***Colas***, (Queue) que cuentan el número de transacciones que se encuentran entre dos puntos del diagrama de flujo.
***Matrices***, (Matrix) que nos facilita contar sobre un arreglo matricial.
***Guarda valores***, variables donde podemos almacenar valores que podemos modificar a lo largo de la simulación
***Tablas***, nos permite recabar datos agrupados, para elaborar tablas de frecuencias.
***Qtablas***, igual que la anterior pero específicamente sobre colas.

d) Operacionales.

***Funciones***. Que cuentan con dos campos, el argumento y la salida, existen 5 tipos diferentes de funciones, que se discutirán más adelante.
***Variables***. existen tres tipos, cuando son llamadas realizan una serie de cálculos predefinidos por el usuario y nos proporciona un valor entero.
***Aleatorios***. Existen varios generadores de números aleatorios.
***Cadena de Usuarios***. El Gpss permite manipular cadenas de transacciones a voluntad del usuario.

e) Comandos de control.

***Edición***. Los referidos a la edición de un programa.
***Interacción***. Los que permiten al usuario interaccionar con una corrida de simulación.

La mayoría de las entidades GPSS se crean automáticamente cuando las necesitamos. Por ejemplo, una referencia a una instalación si no existe con anterioridad se crea automáticamente en ese momento. Esta conveniencia puede en algunos casos salirse de la memoria debido a un defecto en la estructura de su programa GPSS.

Algunas entidades se deben declarar específicamente antes de que puedan ser usadas. Generalmente estas tienen un atributo como tamaño el cual debe hacerse conocer a GPSS/PC. El nombre en el campo de etiquetas se usa entonces para referir la entidad.

Las siguientes entidades se deben declarar al inicio del programa.

- Las entidades de almacenaje STORAGE.
- Variables aritméticas VARIABLE.
- Variables de " punto flotante " FVARIABLE.
- Variables booleanas BVARIABLE.
- Las matrices MATRIX.
- Las tablas TABLE.
- Tablas Q, QTABLE.
- Las funciones FUNCTION.

Los parámetros se deben definir previamente antes de ser requeridos a través de cualquiera de los bloques especificados a continuación.

- Parámetros de transacción se generan al utilizar los bloques ASSIGN, MARK.

## *6.4 ENTIDADES DINAMICAS*

Las transacciones, son las entidades dinámicas del sistema, se mueven de bloque a bloque en una simulación. Una vez que una transacción empieza a moverse en el modelo, continuará entrando a bloques tanto como pueda. La transacción que esta intentando moverse a través del modelo en cualquier instante se llama transacción activa (solo hay una sola transacción activa en un momento dado). Si una transacción falla en encontrar condiciones favorables para entrar un bloque puede venir a reposo. Entonces otra transacción se elige para empezar a moverse a través del modelo hasta que llega al reposo.

Las transacciones se numeran secuencialmente a través de una sesión empezando con el numero uno. EL bloque de control CLEAR comienza la numeración de las transacciones otra vez en uno.

El comportamiento de una transacción esta determinado de alguna forma por varios atributos o variables de estado. Los atributos principales de las transacciones son:

***Parámetros***.- Los parámetros de una transacción son un conjunto de números enteros. Cada transacción puede tener un número cualquiera de parámetros. Cada parámetro tiene un número de parámetro por el cual es referido y un valor. Ambos de estos números son enteros con valor ilimitado y precisión. El valor de cualquier parámetro de la transacción activa puede ser retornado por el SNA.

Los parámetros de la transacción se deben ser crear y sus valores asignados antes de que ellos puedan ser referenciados. Los bloques ASSIGN y MARK crean un parámetro de transacción si no existe uno.

***Prioridad***.- La prioridad de una transacción determina la preferencia que ella recibe cuando ella y otras transacciones están esperando por el mismo recurso. Las filas de prioridad mas importantes en el modelo en cuestión son: Las cadenas de retardo en instalaciones y las cadenas de retardo de almacenaje. El efecto de la prioridad es que una transacción será elegida al frente de las transacciones de menor prioridad, cuando una transacción activa nueva o una nueva propietaria de instalación o almacenaje deba ser elegida. Las transacciones dentro de una prioridad son usualmente catalogadas por primero en llegar, primero en ser servido, conocido como política FIFO (First In First Out).

***Marca de tiempo***.- El tiempo absoluto de reloj que la transacción primero introduce en la simulación o al entrar en un bloque MARK sin ningún operador A.

***Familia***.- Cada transacción generada en un GENERATE, tiene un número de identificación. El bloque SPLIT se utiliza para crear, a partir de una transacción varias transacciones, todas ellas copia de la primera y por tanto pertenecientes a la misma familia.

***Bloque actual***.- El número del bloque que contiene a la transacción.

***Siguiente bloque***.- El numero del bloque que contiene la transacción que va a entrar enseguida.

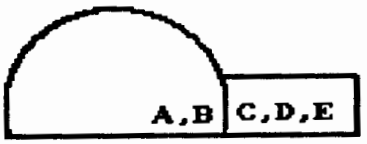
***Estado de la transacción.*** La transacción puede estar en cualquiera de los siguientes estados:

| | |
|---|---|
| ***Activa.*** | Es la transacción activa en el diagrama. |
| ***Suspendida.*** | Cuando la transacción se encuentra en una cadena de eventos futuros. |
| ***Terminada.*** | La transacción ha concluido con su viaje a través del diagrama. |
| ***Interrumpida.*** | La transacción ha sido interrumpida en alguna instalación. |

***Atributos numéricos estándar***, asociados a las transacciones.

| | |
|---|---|
| ***MB***ent. | Si el bloque "ent" se encuentra en estado Match. Es decir, si en el bloque referido, se encuentra una o más transacciones esperando reunir a un grupo de su familia retorna el valor de 1 y cero en caso contrario. |
| ***MP***ent. | Nos proporciona el valor del tiempo actual menos el valor almacenado en el parámetro "ent". |
| ***M1*** | Nos proporciona el tiempo de tránsito de la transacción. (Tiempo actual menos hora de nacimiento). |
| ***P***ent. | Valor del parámetro "ent", de la transacción activa. |
| ***PR*** | Nos proporciona el valor de la prioridad de la transacción. |
| ***XN1*** | Nos retorna el número de la transacción activa. |

### *6.4.1 BLOQUES REFERIDOS A TRANSACCIONES.*




***GENERATE A,B,C,D,E.*** El bloque generate crea transacciones para futuras entradas dentro de la simulación. Tiene cinco campos de operación.

***A*** Tiempo medio de generación. El operador debe ser nombre de función o variable. Entero positivo, sna o parámetro sna.

***B*** Modificador o función modificadora. Opcional. El operador puede ser nulo, name, entero positivo, sna o parámetro sna. Si el operador es entero el tiempo entre llegadas será uniforme entre lo indicado en el campo A más menos el valor entero de B. Si el operador B es una función el tiempo entre llegadas será: lo indicado en el campo A multiplicado por la función indicada en este campo.

***C*** Tiempo de generación de la primera transacción. Opcional el operador puede ser nulo, name, entero positivo, sna o parámetro sna. Por omisión el valor es cero.

***D*** Limite de creación. Por omisión no tiene limite. Opcional.

***E*** Nivel de prioridad. (Opcional) Por omisión es el cero.

***Ejemplo*** ***GENERATE 10,5***

Este bloque crea transacciones con prioridad cero cada 10 ± 5 unidades de tiempo.

***Acción***

Cuando una simulación esta empezando, un bloque generate el cual no esta siendo utilizado es llamado para programar su primera transacción. Así las transacciones son programadas dentro del bloque generate y colocados en la cadena de eventos futuros si ellos tienen un incremento de tiempo positivo. El campo ***C*** puede ser usado para especificar un incremento de tiempo positivo para la primera transacción. De otra manera, el incremento de tiempo de la primera transacción es calculado por los operandos a y b.

Antes de que una nueva transacción sea creada, el campo ***D*** es evaluado para ver si todas las transacciones deseadas han sido generadas. Si el limite de

creación no ha sido excedido, continua procesando. El bloque generate entonces crea la nueva transacción asignándole el siguiente número de transacción, la prioridad indicada en el campo ***E***, y el tiempo de marca de transacción es asignado el valor en el reloj del sistema relativo.


***ADVANCE A,B.*** Este bloque demora el paso de una transacción. Tiene dos campos.

***A.*** El tiempo medio de detención. El operando tiene que ser: ***función, entero positivo, sna, o parámetro de sna.***

***B.*** Modificador. Opcional. El operando tiene que ***ser nulo, name, entero positivo, sna, o parámetro de sna.*** Estos campos actúan de la misma forma que en bloque GENERATE.

***Ejemplo 1: ADVANCE 100,50***

Este bloque genera un número aleatorio uniformemente distribuido entre 50 y 150, y detiene a la transacción ese período de tiempo.

***Ejemplo 2: ADVANCE 50,FN$EXPO***

Este bloque evalúa la función llamada ***EXPO***, y lo multiplica por 50, cantidad resultante es el tiempo que se detendrá la transacción en ese bloque.


***TERMINATE A.*** Este bloque retira la transacción activa de la simulación y reduce, lo indicado en el campo ***A***, la cuenta de terminación.

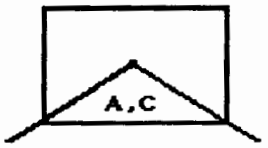
Por omisión, el valor de ***A*** es cero.

***Ejemplo 1. TERMINATE 1***

En este ejemplo, cuando una transacción entra el bloque terminate es retirada de la simulación. Y la cuenta de terminación es disminuida en una unidad.

***Atributos numéricos estándar.***

***Tg1***.- Retorna el valor actual de la cuenta de terminación.




***SPLIT A, B, C.*** Este bloque genera transacciones a partir de una. Este bloque tiene tres campos

***A.*** Número de transacciones relacionadas para ser creadas (Obligatorio).
***B.*** Destino para las nuevas transacciones (Opcional).
***C.*** Número de parámetro. Las transacciones generadas son numeradas empezando por 1, ese número de serie se almacena en el parámetro indicado en este campo (opcional).

***Ejemplo 1.*** ***SPLIT 1***

Al llegar una transacción a este bloque, se genera una copia de ella con los mismos atributos (parámetros, prioridad, etc.). Y continua sobre el diagrama desde ese punto, inmediatamente después de la transacción que la generó.

***Ejemplo 2.*** ***SPLIT 3,P6,32***

Al llegar una transacción a este bloque, se generan tres copias exactamente iguales, excepto el parámetro 32, en el cual cada copia tendrá su número de serie (en este caso 1,2,3). Cada copia es trasladada al bloque indicado en el parámetro número 6, de la transacción generadora.

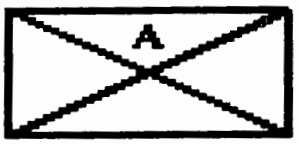

***ASSEMBLE A*** Este bloque reúne el número indicado en el campo A de (destruye) transacciones de una misma familia.

***A.*** Número de transacciones a reunir.

***Ejemplo*** ***ASSEMBLE 3***

Al llegar la primera transacción a este bloque, es detenida y espera la llegada de otras dos de la misma familia (sumando tres en total), una vez

reunidas, la primera continua sobre el diagrama y las dos restantes son destruidas.



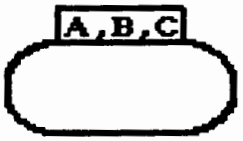

***GATHER*** ***A*** Este bloque sincroniza el paso de transacciones.

***A.*** Número de transacción a sincronizar.

***Ejemplo*** ***GATHER 2***

Este es el modo mas simple de usar un bloque gather. La primera transacción de una familia es forzado a esperar la llegada de otra de la misma familia a este bloque. Cuando esto ocurre ambas transacciones relacionadas son liberadas y puestas en la cadena de eventos actuales.

Un bloque gather se diferencia de un bloque assemble en que las transacciones no son destruidas en bloque gather




***ASSIGN*** ***A,B,C*** Este bloque asigna o modifica el valor de un parámetro.

***A*** Número del parámetro a modificar. Además del número del parámetro puede estar seguido por +, - o nada.

***B*** Valor a ser asignado, sumado o restado.

***C*** Opcional. Modificador de B

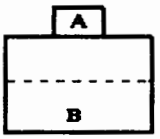
Ejemplo assign 20,150

Al llegar una transacción a este bloque, al parámetro 20 le asigna el valor de 150, si el parámetro no existe lo crea.

***Ejemplo*** ***ASSIGN 200+,20,10***

En este ejemplo, el [ + ] seguido al operador que indica el número del parámetro a modificar, indica que el valor del operador b será sumado al valor del parámetro original (si fuera [-] se restaría).

Al llegar la transacción a este bloque (si el parámetro 200 no existe es creado e inicializado a 0), se le suma un número aleatorio entre 10 y 30.




***PRIORITY A,B.*** El bloque priority modifica la prioridad de la transacción al pasar por este bloque.

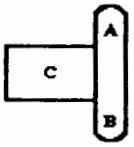
***A.*** Nuevo valor de la prioridad.
***B.*** Opción buffer. Este operador puede ser BU (buffer) o Nulo.

***Ejemplo*** ***PRIORITY 10***

Al pasar la transacción por este bloque, le asigna prioridad de 10 y pasa al siguiente bloque.

***Atributos numéricos estándar.***

PR. Proporciona el valor de la prioridad de la transacción activa.




***LINK*** ***A, B, C*** El bloque link permite la utilización de cadenas de usuario (colas manejadas por el usuario). Este bloque coloca a la transacción en una cadena de usuario.

***A*** Número de cadena de usuario, a donde va a ir la transacción.

***B*** Orden en la cadena, (LIFO, FIFO, etc).

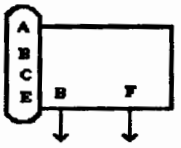
***C*** Siguiente bloque, si la transacción encuentra la cadena fuera de servicio. (opcional).

***Ejemplo*** ***LINK COLA1,FIFO***

Al llegar la transacción a este bloque es mandada a la cadena de espera llamada COLA1, con orden FIFO, donde permanecerá hasta que alguna otra transacción entre a un bloque UNLINK y ordene su salida.

***Atributos numéricos estándar.***

***CA***entnum Tamaño promedio de la cadena entnum.
***CC***entnum Número de entradas a la cadena entnum.
***CH***entnum Longitud actual de la cadena entnum.
***CM***entnum Tamaño máximo de la cadena entnum.
***CT***entnum Tiempo promedio de permanencia en la cadena entnum




***UNLINK O*** ***A, B, C, D, E, F*** El bloque UNLINK retira una transacción de una cadena de usuario.

***O*** Operador Relacional. Criterio de comparación entre los campos D y E. Puede ser: vació, E, G, GE, L, LE, o NE.

***A*** Número o nombre de la cadena de la cuál una o más transacciones serán retiradas. Obligatorio.

***B*** Número del Bloque destino para las transacciones salidas de la cadena. Obligatorio.

***C*** Límite de Remoción. El número máximo de transacción a ser retiradas. Opcional. Por default son todas.

***D*** Si el campo O es vació, forma de liberación de la cadena, vació libera de acuerdo a la organización de la cadena, BACK libera al revés. Si el campo O en no vació este campo corresponde al primer valor de comparación. Opcional.

***E*** Segundo valor de la prueba. Opcional.

***F*** Número o nombre del bloque alterno para la transacción. Opcional.

***Ejemplo. UNLINK COLA1, CASA2, 1.***

Este es el modo más simple de utilizar el bloque UNLINK. Al llegar la transacción saca de la cadena de usuarios llamada cola1, al primero (si existe) de la cola y es mandado al bloque CASA2.

***Atributos numéricos estándar.***

Las mismas de Link

## *6.5 BLOQUES REFERIDOS A EQUIPOS.*

### *6.5.1.- INSTALACIONES.*

Las instalaciones, pueden ser ocupadas por una sola transacción, puede estar en tres estados: Libre o desocupada, ocupada e interrumpida.


***SEIZE A*** Este bloque, sirve para ocupar la instalación indicada en el campo A

***A*** Nombre de la instalación a ocupar.

***Ejemplo: SEIZE 1***

Al llegar la transacción a este bloque, si la instalación 1 esta desocupada, la ocupa, si esta ocupada, espera en cola tipo fifo se desocupe, para ocuparla.


***RELEASE A*** Este bloque sirve para desocupar la instalación indicada en el campo A

***A*** Nombre de la instalación a desocupar.

*Ejemplo* ***RELEASE P3***

Al llegar la transacción a este bloque, la transacción libera la instalación indicada en su parámetro 3


***PREEMPT*** ***A,B,C,D,E*** Este bloque se utiliza para interrumpir la instalación indicada en el campo A.

***A*** Nombre de la instalación a interrumpir.
***B*** Opción prioridad. [PR] con prioridad, vació no.
***C*** Siguiente bloque para la transacción interrumpida.
***D*** Número de parámetro de la transacción interrumpida.
***E*** Opción de remoción. [PE], ya no espera.

***Ejemplo PREEMPT 7,,OTRO,6***

Al llegar la transacción a este bloque, si la instalación 7 esta desocupada la ocupa, si esta ocupada la interrumpe. La transacción interrumpida se va al bloque llamado OTRO, si estaba en un bloque advance, el tiempo que le restaba de espera se le almacena en el parámetro 6.


***RETURN*** ***A*** Este bloque sirve para terminar una interrupción

***A*** Nombre de la instalación a desinterrumpir.

***Ejemplo*** ***RETURN 7***

Al llegar la transacción a este bloque, deja de interrumpir la instalación 7, dejándola en estado libre.

***Atributos numéricos estándar referido a instalaciones.***

***F***entnum Si la facilidad entnum esta ocupada, ***F***entnum es 1, de otra forma toma el valor de cero.

***FC***entnum Número de veces que la instalación entnum ha sido ocupada.

***FL***entnum Si la facilidad entnum esta interrumpida FLentnum regresa el valor de 1, de otra manera FLentnum regresa a cero.

***FR***entnum Porcentaje de utilización de la facilidad entnum FRentnum es expresado en partes por mil y posteriormente regresa un entero entre 0 y 1000.

***FT***entnum Promedio de tiempo de ocupación por transacción de la instalación entnum.

***FV***entnum Si la instalación entnum esta en estado libre FVentnum regresa a 1 y 0 en cualquier otro caso.

### *6.5.2. ALMACENES.*

Los almacenes pueden estar en los siguientes estados: Vacío, lleno, no lleno y no vacío.


***ENTER*** ***A, B*** Este bloque intenta meter un número específico de unidades a un almacén.

***A*** Nombre o numero del almacén a utilizar.

***B*** Número de unidades a meter al almacén. (Optativo), si es nulo la cantidad a ingresar es una unidad.

***Ejemplo. ENTER TIENDA, 2***

Al llegar la transacción a este bloque, si el almacén llamado ***TIENDA***, tiene capacidad de 2 unidades, la transacción las mete y pasa al siguiente bloque. Si no hay capacidad en el almacén espera haciendo cola FIFO.


***LEAVE*** ***A,B*** Este bloque saca B unidades del almacén A

***A*** Nombre del almacén.

***B*** Número de unidades a sacar del almacén A. El valor por omisión es 1. Opcional.

***Ejemplo: LEAVE REPARADORES, 10***

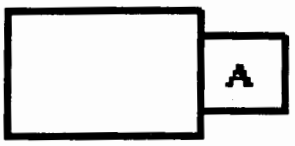
en este ejemplo, cuando una transacción entra al bloque "leave", saca 10 unidades de almacén REPARADORES, incrementando la capacidad disponible en 10 unidades. Si hay menos de 10 unidades disponibles se produce paro de error.

***Atributos numéricos estándar referido a almacenes.***

***R***entnum Capacidad del almacén entnum.
***S***entnum Contenido actual del almacén entnum.
***SA***entnum Promedio de contenido en el almacén entnum.
***SC***entnum El número total de unidades por entrar ( o que intentan entrar al) almacén entnum.
***SE***entnum Booleana. Seentnum regresa 1 si el almacén entnum esta vació, 0 de otra manera.
***SF***entnum Booleana. Regresa 1 si el almacén entnum esta completamente ocupado (lleno), 0 de otra manera.
***SR***entnum Uso del almacén. Uso promedio del almacén entnum. Srentnum esta expresado en partes por mil y por lo tanto regresa un entero 0 - 1000.
***SM***entnum Almacenamiento máximo del almacén entnum.
***ST***entnum Tiempo promedio de permanencia por unidad en el almacén entnum.
***SV***entnum Booleana. Regresa 1 si el almacén esta en estado disponible. 0 de otra manera.

### *6.5.3. SWITCHES LOGICOS.*

Los switches lógicos sirven para administrar el paso de las transacciones por el diagrama de flujo. Pueden estar en dos estados prendido (ON = SI) o apagados (OFF = NO).




***LOGIC X A*** Este bloque sirve para cambiar el estado de un switch.

***X*** Tiene tres valores posibles, ***I*** invierte el estado del switch, si esta prendido lo apaga, si esta apagado lo prende. ***R*** apaga el switch. ***S*** prende el switch.

***A*** Nombre del switch lógico a modificar.

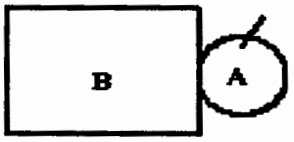
***Atributo numérico estándar referido a switches lógicos.***

***LS***entnum Si el switch lógico entnum esta prendido toma el valor de 1, cero si esta apagado.

## *6.6 BLOQUES ESTADÍSTICOS, COLAS Y CONTADORES.*

### *6.6.1 LINEAS DE ESPERA (COLAS).*

Los siguientes bloques, sirven para llevar el conteo del tamaño de cualquier línea de espera.



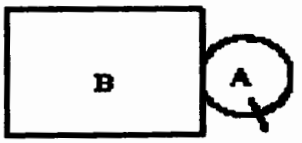

***QUEUE A B*** Este bloque incrementa en la cola A.

***A*** Nombre o número de la cola a incrementar

***B*** Número de unidades a incrementar en la cola. El valor por omisión es 1. Opcional.

***Ejemplo: QUEUE TIENDA***

Al llegar una transacción a este bloque, incrementa en una unidad la cola llamada TIENDA y continua al siguiente bloque.




***DEPART A B*** Este bloque descuenta en la cola A.

***A*** Nombre o número de la cola a descontar.

***B*** Número de unidades a descontar a la cola. El valor por omisión es 1. Opcional.

***Ejemplo: DEPART TIENDA***

Al llegar una transacción a este bloque, descuenta en una unidad la cola llamada TIENDA y continua al siguiente bloque.

***Atributos numéricos estándar referido a almacenes.***

***Q***entnum Tamaño actual de la cola entnum.
***QA***entnum Tamaño promedio de la cola entnum.

***QC***entnum Suma de todas las entradas a la cola entnum.
***QM***entnum Tamaño máximo de la cola entnum.
***QT***entnum Tiempo promedio de estancia en la cola entnum.
***QX***entnum Tiempo promedio de estancia en la cola entnum sin considerar entradas de tiempo cero.
***QZ***entnum La cantidad de entradas a la cola entnum de tiempo cero.

### *6.6.2 CONTADORES.*

Son entidades que nos sirven para guardar algún valor o modificarlo. Para utilizar el siguiente bloque en el diagrama de flujo, debió haber sido declarado al inicio del programa la matriz en cuestión.


***MSAVEVALUE A B C D*** Este bloque modifica el valor de uno de los elementos de una matriz.

***A*** Nombre de la matriz y opcionalmente el signo + para incrementar, - para decrementar y nada para sustituir el valor.

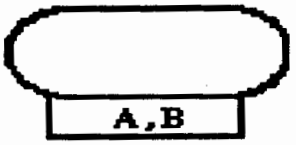
***B*** Renglón del valor a modificar.

***C*** Columna del valor a modificar.

***D*** Valor de modificación.

***ejemplo: MSAVEVALUE CONTROL+, 7,10,15***

Al llegar una transacción a este bloque, va a modificar el valor de la matriz de nombre CONTROL, añadiendo a la componente del renglón 7, columna 10, el valor de 15.




***SAVEVALUE A B*** Este bloque sirve para modificar un contador.

***A*** Nombre del contador a modificar y opcionalmente + para incrementar, - para decrementar y vació para sustituir el valor.

***B*** Cantidad de modificación.

***ejemplo: SAVEVALUE CUENTA3-, 15***

Al llegar una transacción a este bloque, decrementa en 15 unidades al contador de nombre CUENTA3.

***Atributos numéricos estándar referido a contadores.***

***X***entnum Proporciona el valor del guardavalor entnum.
***MX***entnum(m,n) Proporciona el valor (m,n) de la matriz entnum.

### *6.6.3 TABLAS ESTADISTICAS.*

Sirven para obtener información en forma de tablas de frecuencia, de diferentes entidades del sistema en ciertos puntos del diagrama. Las tablas hay que definirlas al inicio del programa -existen cuatro tipos de tablas- mediante el uso del siguiente comando:

***TABLE***

***NOM_TABLA TABLE A,B,C,D***

NOM_TABLA es el identificador (nombre) de la tabla, ***TABLE***, indica que se esta definiendo una tabla estadística y los campos:

***(Modo normal)***

***A*** Atributo numérico estándar a tabular.

***B*** Límite superior de la 1ar. clase de frecuencia.

***C*** Longitud de los intervalos de frecuencia.

***D*** No. de clases de frecuencia.

***Ejemplo. ALFA TABLE P6,100,50,8***

Estamos definiendo una tabla con el nombre de ALFA, para tabular el parámetro 6 (***P6***), donde el límite superior de la primera clase es 100, la longitud de los intervalos es de 50 y consta de 8 intervalos.

***(Modo Diferencia)***

Lo que se tabula en este caso es la diferencia entre el atributo numérico estándar del actual con el último registrado, la primera transacción no se registra, se indica el modo agregando un signo menos (-).

***Ejemplo. 7 TABLE C1-,0,10,100***

Nombre de la tabla 7, atributo a tabular el tiempo reloj en modo diferencia, es decir la diferencia de tiempo entre transacciones.

***(modo Tiempo entre llegadas)***

Campo A = IA, registra el tiempo entre llegadas.

***Ejemplo. ABC TABLE IA,20,10,50***

Definimos la tabla ABC, como de diferencia entre tiempos de llegada (IA),

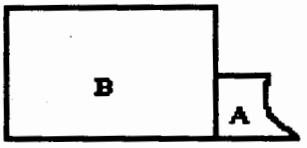
esta definición es equivalente a la anterior.

***(Modo velocidad de llegada)***

Campo A = RT, esta tabla tiene un campo más el E, de intervalo de tiempo.

***Ejemplo XYZ TABLE RT,0,10,10,3600***

Lo que registra es el número de unidades que llegan al bloque tabulate, en el intervalo de tiempo indicado en el campo E. En este caso cada 3600.




***TABULATE A, B*** Este bloque sirve para construir la tabla de frecuencia.

***A*** Nombre de la tabla.

***B*** Número de unidades, vació igual a uno.

***Ejemplo TABULATE ALFA,2***

Al llegar una transacción a este bloque, dependiendo del valor de P6, determina el intervalo de frecuencia a incrementar y lo hace con dos unidades (campo b).

***QTABLE A,B,C,D*** Este bloque es de definición de tabla de frecuencia, para una cola específica.

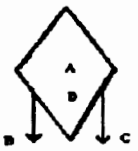
***A*** Nombre de la línea de espera (cola).
***B,C y D*** Igual que en TABLE.

Observación. Esta tabla hace los registros sobre los bloques de la cola QUEUE y DEPART, no requiere de TABULATE.

***Atributos numéricos estándar referido a tablas.***

***TB***nombre Valor medio de la tabla nombre (truncado).
***TC***nombre Número de unidades que han entrado a la tabla nombre.
***TD***nombre Desviación estándar de la tabla nombre (truncado).

## *6.7 BLOQUES DE ALTERACIÓN Y CONTROL DE FLUJO EN EL DIAGRAMA.*




***TRANSFER A, B, C, D*** Este bloque altera el flujo de la transacción en diversas formas.

***A*** Opcional. Modo de Transferencia. El operando puede ser vació, fracción, BOTH, ALL, PICK, FN, P, SBR, o SIM.
***B*** Ubicación o numero de Bloque destino. Opcional.
***C*** número o ubicación de Bloque alterno. Incrementa el valor en modo de FN. Opcional.
***D*** Incrementa el número de Bloque para el modo ALL. El valor por DEFAULT es 1.

***MODO INCONDICIONAL.*** (Campo A vació)

***Ejemplo. TRANSFER , NUEVO_LUGAR***

Al llegar una transacción a este bloque es transferido al bloque NUEVO_LUGAR.

***MODO FRACCIONAL.*** (Campo A igual a una fracción).

***Ejemplo. TRANSFER .75, NUEVO_LUGAR***

Al llegar la transacción a este bloque se genera un número aleatorio, si es menor de .75 pasa al bloque NUEVO_LUGAR, en caso contrario como el campo C, para el destino alterno es vació procede al siguiente bloque secuencial.

***MODO BOTH*** (Campo A igual a BOTH)

***Ejemplo. TRANSFER BOTH, LUGAR_1, LUGAR_2.***

Al llegar una transacción a este bloque, el bloque LUGAR_1 es probado. Si la transacción puede entrar, lo hace. Si no, prueba en el bloque Lugar_2, si esta disponible entra, de otra forma la transacción permanece en el bloque TRANSFER hasta que pueda pasar a alguno de los bloques anteriores.

***MODO ALL*** (Campo A igual a ALL).

***Ejemplo. TRANSFER ALL, LUGAR_1, LUGAR_FIN, 2***

Al llegar una transacción a este bloque, el bloque llamado LUGAR_1 es probado. Si la transacción puede entrar, lo hace. Si no los bloques con ubicación más alta son probado, cada dos lugares (ya que el campo D=2, indica que se prueben cada dos lugares). La transacción entra si puede. Si todo bloque probado rehúsa, los extremos probadores LUGAR_1 y LUGAR_FIN o con el bloque justamente antes de el, la transacción permanece en el bloque transfer hasta que sea aceptado en cualquiera de los bloques anteriores.

***MODO PICK*** (Campo A igual a PICK).

***Ejemplo. TRANSFER PICK, LUGAR_1, LUGAR_FIN***

Cuando una transacción llega a este bloque se elige aleatoriamente un bloque entre LUGAR_1 y LUGAR_FIN inclusive y a ese bloque se traslada la transacción.

***MODO FUNCTION*** (Campo A igual a FN).

***Ejemplo. TRANSFER FN, FUNC1, 5***

Cuando una transacción llega a este bloque, la función de nombre FUNC1 es evaluada, y se le sumara 5, para determinar el bloque destino.

***MODO PARAMETRO*** (Campo A igual a P).

***Ejemplo TRANSFER P, 5,1.***

Al llegar una transacción a este bloque, se manda al siguiente bloque del indicado en el parámetro número 5 de la transacción.

***MODO SUBRUTINA*** (Campo A igual a SBR).

***Ejemplo. TRANSFER SBR, LUGAR1, 3***

Cuando una transacción llega a este bloque de TRANSFERENCIA, es trasladada inmediatamente al bloque LUGAR1. La ubicación del bloque TRANSFER es almacenado en el parámetro 3. Si no existe el parámetro, es creado.

***MODO SIMULTANEO*** (Campo A igual a SIM).

***Ejemplo. TRANSFER SIM, LUGAR_SR, LUGAR_RET***

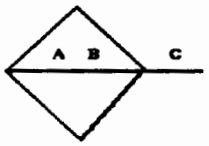
Cuando una transacción llega a este bloque, es mandado a LUGAR_SR si el indicador de demora esta activo, si el indicador de demora esta apagado la transacción es mandada a LUGAR_RET. Después de la transferencia, el indicador de demora siempre se apaga.

RESTRICCIONES ESPECIALES.

En modo ALL, el operando C tiene que ser mayor que el operando B.

Todos los destinos de transacción calculados tienen que ser ubicaciones de bloque válidas.

Tanto en modo BOTH como ALL es posible desperdiciar una gran cantidad de tiempo de computadora en pruebas infructuosas. Usted puede querer colocar transacciones en una cadena de usuario hasta que la prueba sea probablemente exitosa.




***TEST O A,B,C*** Este bloque compara dos valores, normalmente atributos numéricos estándar y determina el destino de la transacción en base al resultado.

***O*** Obligatorio. Tipo de comparación a realizar entre los campos ***A*** y ***B***. El operando tiene que ser E, G, GE, L, LE, o NE.

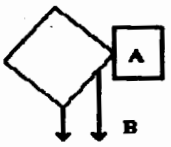
***A*** Obligatorio. Primer valor de comparación.
***B*** Obligatorio. Segundo valor de comparación.
***C*** Opcional. Bloque destino. Si es vació trabaja en modo rechazo.

***Ejemplo. TEST L Q$COLA1,7, FUERA***

Al llegar la transacción a este bloque, pregunta si el tamaño de la cola1 es menor que 7, si la respuesta es afirmativa continua al siguiente bloque, si en negativa pasa al bloque ***FUERA***.

Operandos de comparación.

***E*** El valor del operando A tiene que ser igual al valor del operando B.
***G*** El valor del operando A tiene que ser mayor que el valor del operando B.
***GE*** El valor de operando A tiene que ser mayor o igual al valor del operando B.
***L*** El valor de operando A tiene que ser menor que el valor del operando B.
***LE*** El valor de operando A tiene que ser menor o igual a el valor del operando B.
***NE*** El valor de operando A tiene que ser diferente del valor del operando B.




***GATE O A,B*** Este bloque altera el flujo de una transacción basado en el estado de una entidad.

O - operador condicional. Condición requerida en una entidad a probar por una prueba exitosa. El operador puede ser, dependiendo a la entidad que se elige:

| Almacenes | Instalaciones | Switch Lógicos | Bloques |
|---|---|---|---|
| SE Vació | NI No Interrumpida | LS Prendido | M Edo. Match |
| SF LLeno | I Interrumpida | LR Apagado | NM Edo. No Match |
| SNE no Vació | NU Desocupada | | |
| SNF no Lleno | U En uso | | |

***A*** Nombre o número de la entidad a ser probada. El tipo de entidad esta implícito por el operador condicional.

***B*** Nombre del bloque destino cuando la prueba no tiene éxito. Opcional. ( Si no se utiliza opera en modo rechazo).

Ejemplos

GATE SNE PELUQUE, BARBAS

Al llegar la transacción a este bloque, si el almacén llamado "peluque" no esta vacía (i. e. Si al menos una unidad del almacén esta en uso) la transacción procede al siguiente bloque. Si el almacén esta vació (no tiene éxito la prueba), la transacción activa es llevada al bloque en la dirección llamada "barbas".

## *6.8 ENTIDADES DE CALCULO.*

Funciones y variables. Estas entidades se definen generalmente al inicio del programa, para posteriormente ser invocadas para generar un valor y utilizarlo en alguno de los bloques.

Funciones. Dada una variable independiente, nos permite calcular el valor de una variable dependiente (función) definida por una serie de puntos.

### *6.8.1 FUNCIONES DISCRETAS*

Forma general:

***IDEN FUNCTION A,B***

Donde ***IDEN*** es el nombre de la función, el campo ***A*** es el argumento de la función o variable independiente. El campo ***B*** declara el tipo y tamaño de la función, este campo puede iniciar con la letra ***D*** para el caso discreto, o bien, con la letra ***C*** para el caso continuo. Posteriormente a la letra sigue un número que indica el número de puntos que definen la función.

Ejemplo 1.

***COMPRA FUNCTION RN1, D5***
***0.2,3/0.5,7/0.73,4/1.0,8***

Definimos una función discreta de 5 puntos, que tiene como argumento la serie de números aleatorios RN1 y que se denomina COMPRA. Se llama a la función de la forma ***FN$COMPRA***, en el campo de algún bloque, en ese momento se genera un numero aleatorio, entre 0 y 0.9999 y dependiendo del valor generado, si es o igual o menor que 0.2 la función tendrá un valor de 3,

si es mayor pero menor o igual a 0.5 la función valdrá 7, si es mayor pero menor o igual que 0.73 la función valdrá 4, finalmente si es mayor a este valor la función valdrá 8. La función compra puede ser vista como una función escalón como se ilustra en la figura 6.1

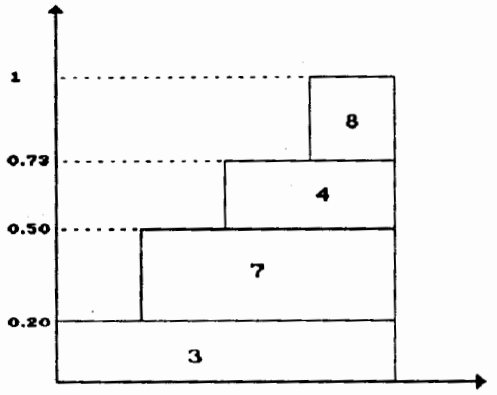



figura 6.1

Ejemplo 2

***7 FUNCTION P3,D4***
***1,3/3,6/15,8/23,9***

Definimos la función discreta de 4 puntos, que tiene como argumento, el valor del parámetro 3 de la transacción activa, el nombre de la función es 7. Se llama a la función de la forma ***FN7.***
Cuando es llamada, si el valor del parámetro 3 de la transacción activa es menor o igual a 1, la función tendrá un valor de 3, si es mayor pero menor o igual a 3, el valor de la función será 6, si el parámetro es mayor a 3 pero menor o igual que 15 la función valdrá 8, si es mayor que 15 la función valdrá 9. El valor de 9 lo tendrá la función también para cualquier valor del parámetro mayor de 23

Ejemplo 3

***DIST FUNCTION RN3, C4***
***0,0/0.2,3/0.6,10/1,15***

Definimos una función continua de 4 puntos, que tiene como argumento la serie de números aleatorios RN3 y que se denomina DIST. Se llama a la función de la forma ***FN$DIST***, en el campo de algún bloque, en ese momento se genera un numero aleatorio, entre 0 y 0.9999 y dependiendo del valor generado, si es o igual o menor que 0.2 la función interpola entre 0 y 3, si es mayor pero menor o igual a 0.6 la función interpola entre 3 y 10, si es mayor pero menor o igual que 1, la función interpola entre 10 y 15, finalmente si es mayor a este valor la función valdrá 15. La función dist puede ser vista como una función poligonal como se ilustra en la figura 6.2. El valor proporcionado por una función continua siempre es entero, obteniendo este valor mediante truncamiento.

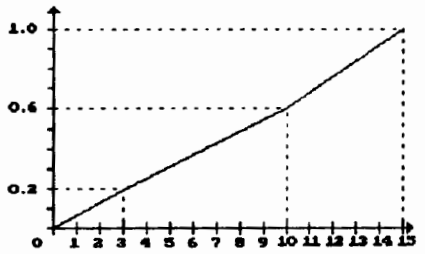



figura 6.2

***Atributos numéricos estándar referido a funciones***

***FN***entum Avalúa la función entum y regresa ese valor.

## *6.8.2 VARIABLES*

Nos permite relacionar en forma lógica y/o algebraica, diferentes entidades del programa.

Las operaciones aritméticas que se tienen son:

+ suma; - resta, # multiplicación, / división, @ módulo entero.

Las operaciones de multiplicación , división y módulo entero tienen preferencia sobre la de suma y resta. Se evalúa de izquierda a derecha, según aparezcan. Se permite la utilización de paréntesis para agrupar. Depende de la versión de GPSS que se utilice, el número de paréntesis que soporte (la primera versión soportaba hasta 5 paréntesis de profundidad).

ejemplo de definición de una variable.

***COST VARIABLE P3#FN$PRE+V3-8***

La forma de llamar a una variable es con la letra ***V*** y a continuación su número si le asignamos por nombre un número o con el símbolo ***$*** y el nombre a continuación. Así podemos utilizar ***V$COST***, en un campo de algún bloque, en ese momento se determina el valor de la variable, utilizando el valor del parámetro 3 de la transacción activa, multiplicarlo por lo que valga la función de nombre PRE, sumándole el valor de la variable 3 y restandole 8. En cada uno de estos pasos, si el valor resultante es fraccionario se trunca y se continúan las operaciones.

### *6.8.3 VARIABLES FLOTANTES.*

La diferencia entre este tipo de variables y las anteriores consiste, en que aquí, se ejecutan todas las operaciones que definen a la variable, ***sin truncar, el truncamiento se efectúa al final.***

Ejemplo:

***5 FVARIABLE FN3*V4 - 7*Q1***

Para llamar a la variable se hace de la misma forma que una variable normal. Aquí al escribir ***V5***, en el campo de un bloque, al llegar la transacción se evalúa ***FN3*** y se multiplica por ***V4***, al resultado se le resta el producto de ***7*** por el tamaño de la cola 1.

***Atributos numéricos estándar referido a variables.***

*V*entum Evalúa la variable (flotante o no) entum y regresa ese valor.

### *6.8.4 VARIABLES LÓGICAS O BOOLEANAS*

Los posibles valores de este tipo de variables es falso o verdadero. Nos proporciona el valor de 1 si es verdadero y 0 si es falso. Puede utilizar como argumentos cualquiera de los valores lógicos asociados a los equipos y manipularlos mediante los operadores lógicos.

Operadores lógicos:

* operación AND; + operación OR

Ejemplo:

*7 BVARIABLE (C1 'GE' 3600) * SF3*

Para llamar a esta variable se escribe ***BV7***, en el campo de algún bloque, al llegar la transacción , si el tiempo reloj de la simulación (***C1***) es menor que 3600 y el almacén 3 esta lleno, la variable regresa el valor de 1 en cualquier otro caso regresa el valor de cero.

***Atributos numéricos estándar referido a variables lógicas.***

***BV***entum Evalúa la variable booleana entum y regresa ese valor.

ejemplos:

*500 ASSIGN 3,V5*

Al llegar la transacción a este bloque, evalúa la variable 5 y le asigna ese valor a su parámetro n° 5.

*600 ASSIGN 4,FN$DIST*

Al llegar la transacción a este bloque, evalúa la función de nombre DIST y le asigna ese valor a su parámetro n° 4.

En los ejemplos anteriores, asumimos que la variable y la función fueron definidos de antemano.

Un ejemplo en GPSSPC. Considere el siguiente modelo:

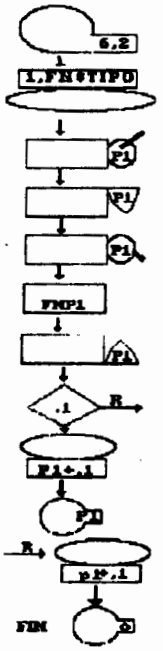
Una fábrica produce artículos cada 6 ± 2 min. El 90% de ellos son tipo A y tardan en su revisión 6 ± 2 min. El 10 % restante son de tipo B, cuya inspección se realiza en 20 ± 10 min. Existe un inspector para cada uno de los tipos de artículos, si el inspector esta ocupado y llega un artículo a revisión, este aguarda en cola tipo FIFO, hasta que le toque su turno y sea revisado. El porcentaje de artículos rechazados es del 10%. Simule hasta una producción de 100 artículos de tipo A aceptados.

Existen diversas formas de modelar este problema a continuación exponemos dos diagramas que resuelven el problema y el listado de uno de ellos.

```
; GPSS/PC Program file UNO              06-08-1993   13:15:03

30          SIMULATE
40 *
50 *   DEFINICION DE FUNCIONES
70 *  90% TIPO 1, 10% TIPO 2
80 TIPO     FUNCTION    RN1,D2   ;DETERMINA TIPO DE ARTICULO
.9,1/1,2
85 * ESTA FUNCION DETERMINA EL TIEMPO DE REVISION DE ART. TIPO 1
90 UNO      FUNCTION    RN2,C2
0,2/1,5
95 * ESTA FUNCION DETERMINA EL TIEMPO DE REV. DE LOS ART. TIPO 2
100 DOS     FUNCTION    RN2,C2
0,10/1,31
105 * DEPENDIENDO DEL TIPO DE ART. SE DETERMINA TIEMPO DE
REVISION
110 TRES    FUNCTION    P1,E2
1,FN$UNO/2,FN$DOS
112 CUA     FUNCTION    P1,D2  ; SELEC GUARDAVALOR DE RECHAZADOS
1,3/2,4
115 * COMIENZA 1er SEGMENTO
120         GENERATE    6,2      ;GENERA ART CADA 6 ± 2 MIN
130         ASSIGN      1,FN$TIPO ;DETERMINA EL TIPO DE ARTICULO
140         QUEUE       P1       ;INC. COLA DE SU TIPO DE ART
150         SEIZE       P1       ;ENTRA A REVISION
160         DEPART      P1       ;DESCUENTA EN COLA
170         ADVANCE     FN$TRES  ;TIEMPO DE REVISION
180         RELEASE     P1       ;LIBERA AL INSPECTOR
190         TRANSFER    .1, RECHAZ;10%  DE RECHAZADOS
200         SAVEVALUE   P1+,1    ;INCREMENTO ACEPTADOS
```

```
210       TEST L     P1 2 FIN  ;MANDO A FIN A LOS TIPO DOS
220       TERMINATE  1         ;LOS TIP 1 DESCUENTA CUENTA DE T
230 *  EMPIEZA SEGMENTO DOS DE RECHAZADOS
240 RECHAZ SAVEVALUE  FN$CUA+,1 ;INCREMENTO RECHAZADOS POR
TIPO
250 FIN    TERMINATE           ;SALEN SIN DESCONTAR
START 100
```




GENERA ART. CADA
6 MAS MENOS 2 MIN.

DETERMINAMOS TIPO
DE ART. Y LO GUARDAMOS
EN EL PARAMETRO 1

INCREMENTA COLA ART.
INDICADO EN P1

INSPECCION DE ART.
INDICADO EN P1

DESCUENTO COLA ART.
INDICADO EN P1

TIEMPO DE INSPECCION
DADO POR LA FUNCION
INDICADA EN P1

LIBERA INSPECTOR
INDICADO EN P1

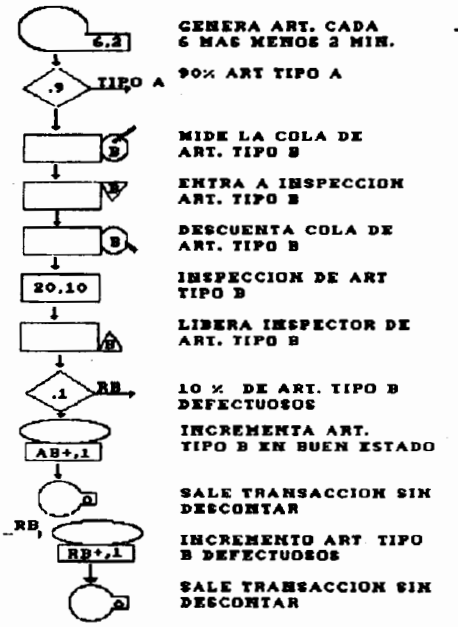
A R LOS ARTICULOS
RECHAZADOS (10%)

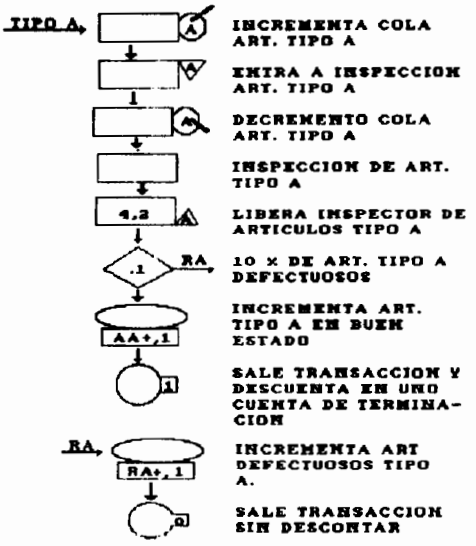
INCREMENTA ART.
TIPO

SALE TRANSACCION SIN
DESCONTAR

INCREMENTO ART. TIPO
P1 DEFECTUOSOS

SALE TRANSACCION SIN
DESCONTAR


6,2
GENERA ART. CADA 6 MAS MENOS 2 MIN.
.9
TIPO A
90% ART TIPO A
B
MIDE LA COLA DE ART. TIPO B
ENTRA A INSPECCION ART. TIPO B
DESCUENTA COLA DE ART. TIPO B
20,10
INSPECCION DE ART TIPO B
LIBERA INSPECTOR DE ART. TIPO B
.1
RB
10 % DE ART. TIPO B DEFECTUOSOS
AB+,1
INCREMENTA ART. TIPO B EN BUEN ESTADO
SALE TRANSACCION SIN DESCONTAR
RB
RB+,1
INCREMENTO ART TIPO B DEFECTUOSOS
SALE TRANSACCION SIN DESCONTAR


TIPO A
INCREMENTA COLA ART. TIPO A
ENTRA A INSPECCION ART. TIPO A
DECREMENTO COLA ART. TIPO A
INSPECCION DE ART. TIPO A
4,2
LIBERA INSPECTOR DE ARTICULOS TIPO A
.1
RA
10 % DE ART. TIPO A DEFECTUOSOS
AA+,1
INCREMENTA ART. TIPO A EN BUEN ESTADO
SALE TRANSACCION Y DESCUENTA EN UNO CUENTA DE TERMINACION
RA
RA+,1
INCREMENTA ART DEFECTUOSOS TIPO A.
SALE TRANSACCION SIN DESCONTAR

## *BIBLIOGRAFIA*


1.- COATS, P.K. "Combining an expert system with simulation to enhance planning for banking networks" Simulation, Vol.54, Num 6, junio 1990.

2.- COSS BU, R. "Simulación de Sistemas" ed. Diana 1986.

3.- COX, J. " Object Oriented Programing. An Evolutionary Approach", Addison Wesley, 1987

4.- ELDREDGE, D. MC. GREGOR, J.D. Y SUMMERS, M.K. "Applying the object- orientes paradigm to discrete event simulations using C++ language". Simulation, Vol 54, Num 6, junio 1990.

5.- EZZELL, B. " Programming the IBM User Interface Using Turbo Pascal", Addison Wesley Publishing Company, 1989.

6.- FUENTES ZENON A. "El enfoque de sistemas en la solución de problemas", Cuadernos de planeación y sistemas, DEPFI. UNAM. 1990

7.- FUENTES Z. A. y SANCHEZ G. G. "Metodología de la Planeación Normativa", Cuadernos de planeación y sistemas, DEPFI. UNAM 1990.

8.- GORDON, G." Simulación de Sistemas", Editorial Diana, 1988.

9.- GPSS-PC "Manual de usuario", Minuteman Software.1990

10.- HILLIER, y LIEBERMAN. "Introducción a la Investigación de Operaciones" Ed. Mc. Graw Hill. 1989.

11.- HOFFMAN, P. y NICOLOFF, T. "Sistema Operativo MS-DOS. Guia del Usuario", Osborne Mc Graw Hill, 1984.

12.- JAMSA, K. y NAMEROFF, S. "Turbo Pascal: Biblioteca de Programas", Mc. Graw Hill, 1986.

13.- KIVIAT, P. "Development of Discrete Digital Simulation Languages", Simulation, Vol 8, Num 2, febrero 1967.

14.- LUCAS, H. " Análisis, Diseño e Implementación de Sistemas d e Información"

15.- MC. MILLAN, C. Y GONZALEZ, R.F. "Análisis de Sistemas" ed. Trillas, 1986.

16.- MEYER, B. "Object-Oriented Software Construction". Prentice- H a l l International Series in Computer Science, 1988.

17.- MILLER, A. "El ABC del MS-DOS. Versión 3.3", Ventura Ediciones 1988.

18.- PAYNE, J.A. "Introduction to Simulation: Programming Techniques and Methods of Analysis" ed. Mc. Graw-Hill, 1988.

19.- P. VAN GIGCH, J. "Teoría General de Sistemas" ed Trillas 1987.

20.- SAATY, T.H. "Elementos de la Teoría de Colas". ed Aguilar, 1967.

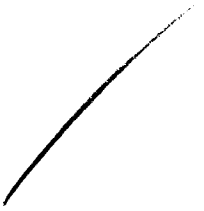
21.- SHANNON, R.E. "Simulación de Sistemas" ed. Trillas, 1988.

22.- SMARTE, G. y REINHARDT, A. "15 Years of Bits, Bytes and Other Great Moments", Byte, vol 15, No. 9, sept. 1990.

23.- Stephen O'Brien "Turbo Pascal 6 Manual de referencia".

24.- WILLIAMS, G. y WELCH, M. "A Microcomputing Timeline", Byte vol 10, No. 9, sept. 1985.

ESTA OBRA SE TERMINO DE IMPRIMIR EN LOS TALLERES GRAFICOS DE: **TESIS ECONOMICAS PROFESIONALES**, CAMPECHE 156, COL ROMA. **TELS: 564-3954 Y 584-8153** BAJO LA SUPERVISION DE LA MAESTRA IDALIA FLORES DE LA MOTA. EL TIRAJE DE ESTA EDICION FUE DE 300 EJEMPLARES.



BAR
EJ. 2

F-DEPFI/MISC 0014/Ej.6
*720875*

F-DEPFI
MISC
0014
Ej.6

G