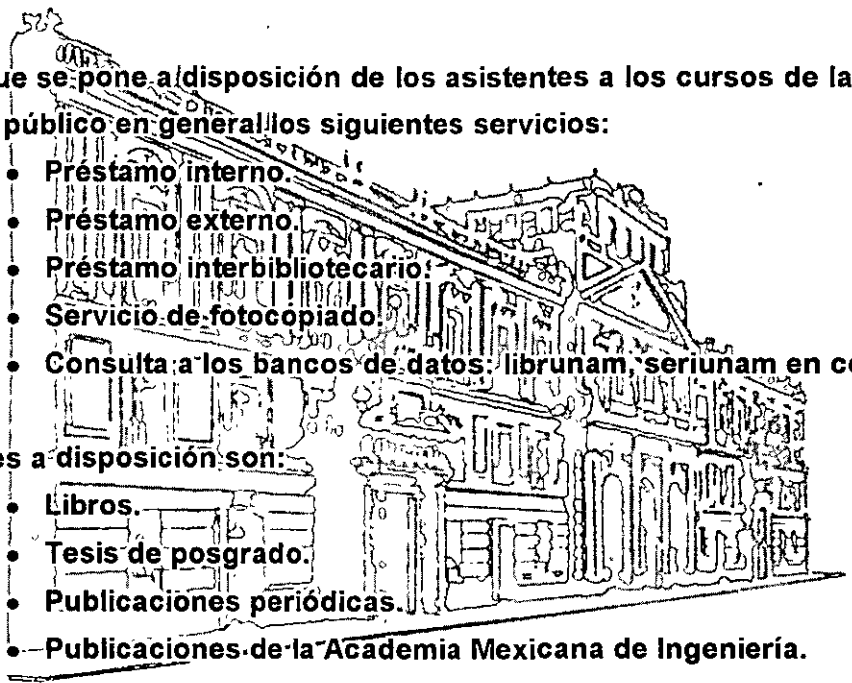
FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN
" ING. BRUNO MASCANZONI "

El Centro de Información y Documentación Ing. Bruno Mascanzoni tiene por objetivo satisfacer las necesidades de actualización y proporcionar una adecuada información que permita a los ingenieros, profesores y alumnos estar al tanto del estado actual del conocimiento sobre temas específicos, enfatizando las investigaciones de vanguardia de los campos de la ingeniería, tanto nacionales como extranjeras.



Es por ello que se pone a disposición de los asistentes a los cursos de la DECFI, así como del público en general los siguientes servicios:

- **Préstamo interno.**
- **Préstamo externo.**
- **Préstamo interbibliotecario.**
- **Servicio de fotocopiado.**
- **Consulta a los bancos de datos: librunam, seriunam en cd-rom.**

Los materiales a disposición son:

- **Libros.**
- **Tesis de posgrado.**
- **Publicaciones periódicas.**
- **Publicaciones de la Academia Mexicana de Ingeniería.**
- **Notas de los cursos que se han impartido de 1988 a la fecha.**

En las áreas de ingeniería industrial, civil, electrónica, ciencias de la tierra, computación y, mecánica y eléctrica.

El CID se encuentra ubicado en el mezzanine del Palacio de Minería, lado oriente.

El horario de servicio es de 10:00 a 14:30 y 16:00 a 17:30 de lunes a viernes.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

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

A LOS ASISTENTES A LOS CURSOS

Las autoridades de la Facultad de Ingeniería, por conducto del jefe de la División de Educación Continua, otorgan una constancia de asistencia a quienes cumplan con los requisitos establecidos para cada curso.

El control de asistencia se llevará a cabo a través de la persona que le entregó las notas. Las inasistencias serán computadas por las autoridades de la División, con el fin de entregarle constancia solamente a los alumnos que tengan un mínimo de 80% de asistencias.

Pedimos a los asistentes recoger su constancia el día de la clausura. Estas se retendrán por el periodo de un año, pasado este tiempo la DECFI no se hará responsable de este documento.

Se recomienda a los asistentes participar activamente con sus ideas y experiencias, pues los cursos que ofrece la División están planeados para que los profesores expongan una tesis, pero sobre todo, para que coordinen las opiniones de todos los interesados, constituyendo verdaderos seminarios.

Es muy importante que todos los asistentes llenen y entreguen su hoja de inscripción al inicio del curso, información que servirá para integrar un directorio de asistentes, que se entregará oportunamente.

Con el objeto de mejorar los servicios que la División de Educación Continua ofrece, al final del curso deberán entregar la evaluación a través de un cuestionario diseñado para emitir juicios anónimos.

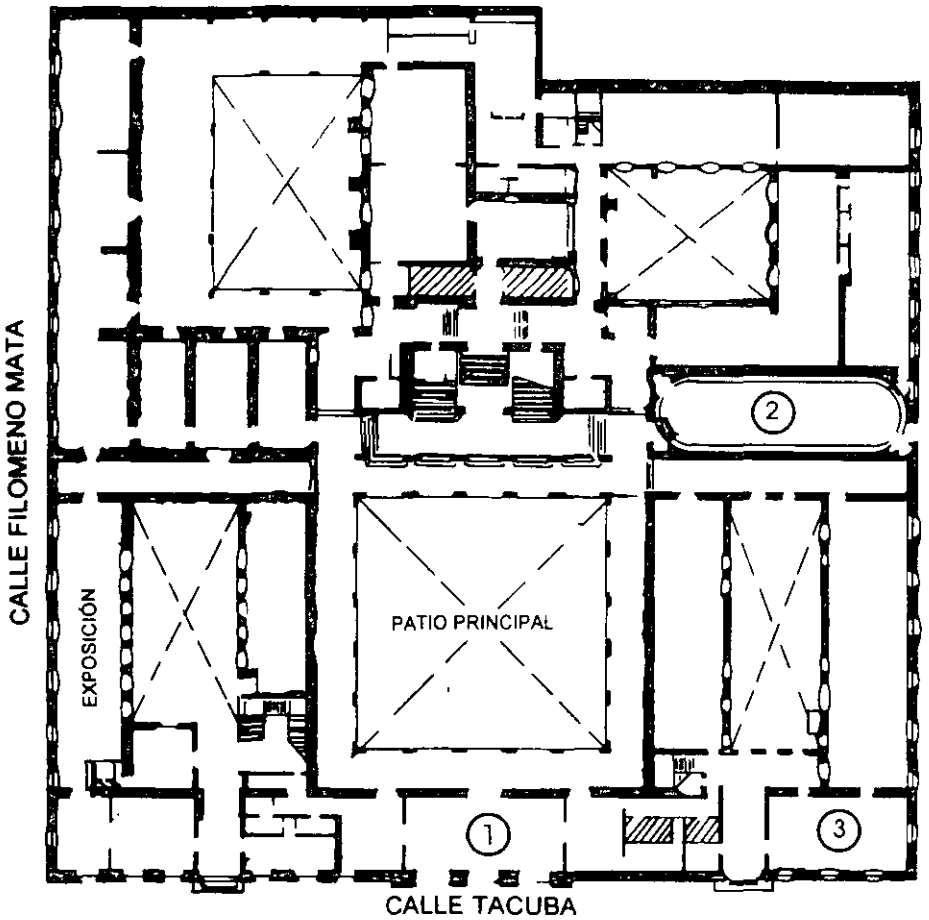
Se recomienda llenar dicha evaluación conforme los profesores impartan sus clases, a efecto de no llenar en la última sesión las evaluaciones y con esto sean más fehacientes sus apreciaciones.

Atentamente

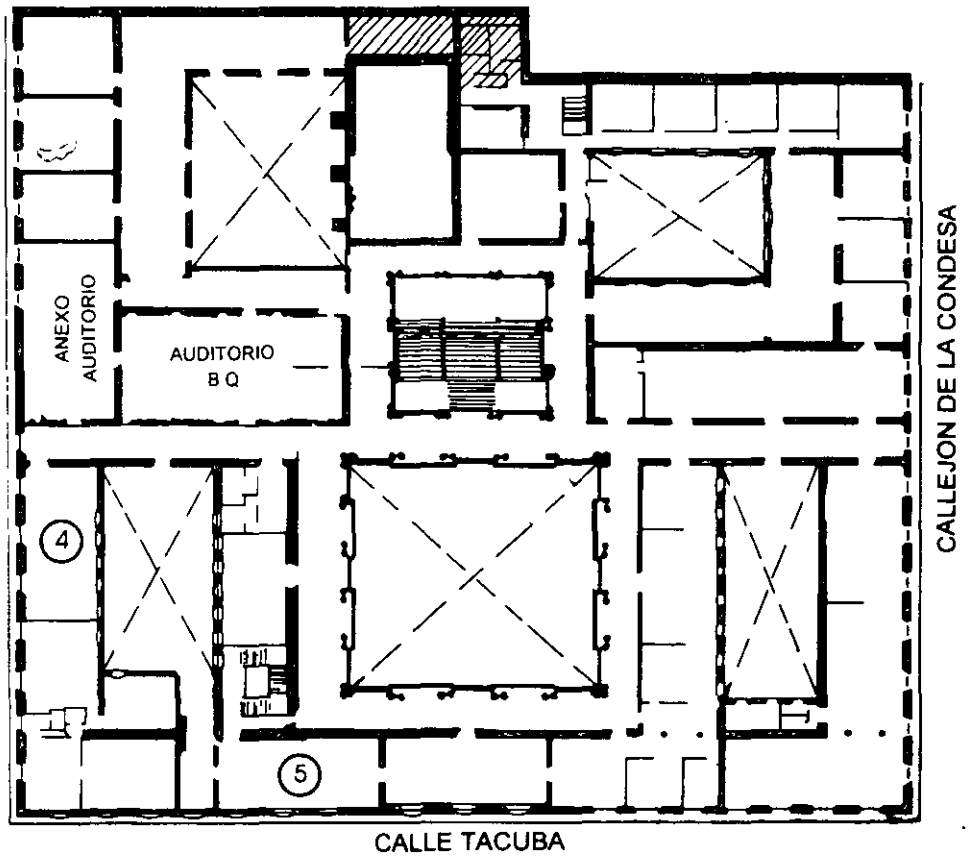
División de Educación Continua.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 5512-8955 5512-5121 5521-7335 5521-1987 Fax 5510-0573 5521-4021 AL 25

PALACIO DE MINERIA

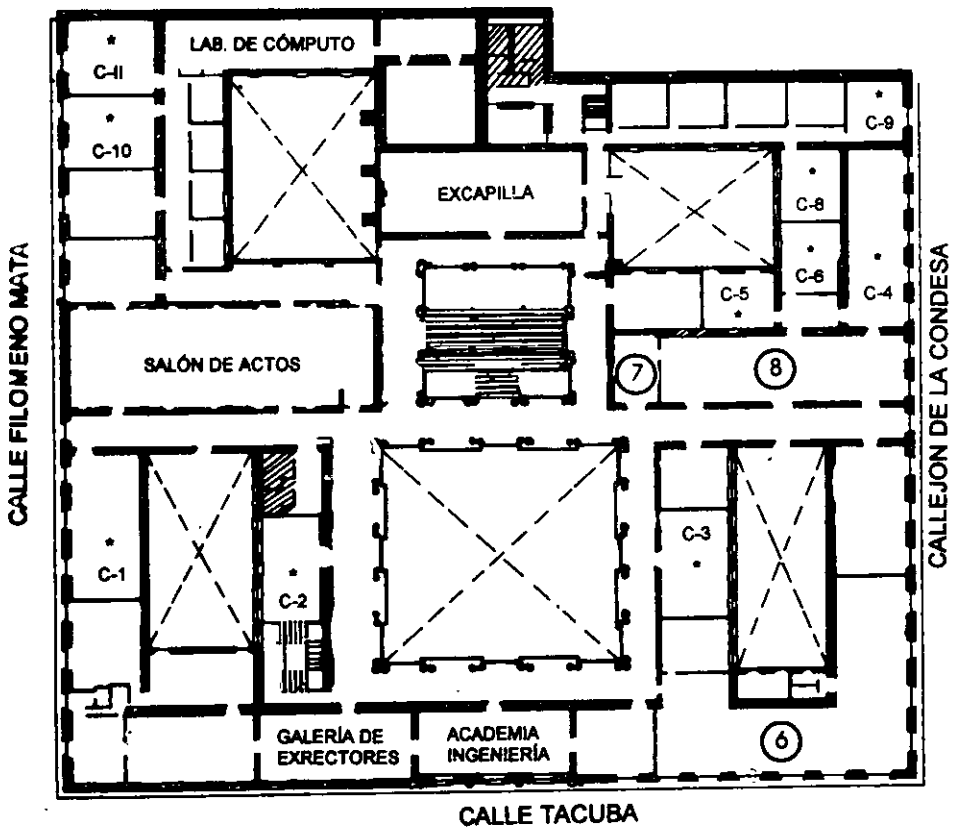


PLANTA BAJA



MEZZANINNE

PALACIO DE MINERIA



1er. PISO

GUÍA DE LOCALIZACIÓN

1. ACCESO
2. BIBLIOTECA HISTÓRICA
3. LIBRERÍA UNAM
4. CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN "ING. BRUNO MASCANZONI"
5. PROGRAMA DE APOYO A LA TITULACIÓN
6. OFICINAS GENERALES
7. ENTREGA DE MATERIAL Y CONTROL DE ASISTENCIA
8. SALA DE DESCANSO

SANITARIOS

* AULAS

DIVISIÓN DE EDUCACIÓN CONTINUA
FACULTAD DE INGENIERÍA U.N.A.M.
CURSOS ABIERTOS

DIVISIÓN DE EDUCACIÓN CONTINUA
DEC

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

MATERIAL DIDACTICO DEL CURSO

ANALISIS Y DISEÑO ORIENTADO A OBJETOS CON APOYO DE HERRAMIENTAS CASE

ABRIL, 2002

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos 5512-8955 5512-5121 5521-7335 5521-1987 Fax 5510-0573 5521-4021 AL 25

Índice

Introducción al modelado orientado a objetos

El desarrollo de proyectos software ha sufrido una evolución desde los primeros sistemas de calculo, implementados en grandes computadores simplemente ayudados mediante unas tarjetas perforadas donde los programadores escribían sus algoritmos de control, hasta la revolución de los sistemas de información e Internet. Han existido dos grandes cambios desde aquellos sistemas meramente algorítmicos donde todo el esfuerzo de desarrollo se centraba en la escritura de programas que realizaran algún tipo de calculo. El primero de ellos es la aparición del modelo relacional, un modelo con fuerte base matemática que supuso el desarrollo las bases de datos y propició la aparición de los grandes sistemas de información. El segundo cambio es sobre los lenguajes de programación, la aparición de los *Lenguajes Orientados a Objetos* (aunque los primero lenguajes con características de orientación a objetos aparecieron en la década de los setenta, por ejemplo *Simula 67*) supuso una revolución en la industria software. El problema entonces radicaba en poder sacarle partido a los lenguajes orientados a objetos por lo que aparecieron numerosas metodologías para el diseño orientado objetos, hubo un momento en el que se podía decir que el concepto de orientación a objetos estaba "de moda" y todo era orientado a objetos, cuando realmente lo que ocurría es que las grandes empresas que proporcionaban los compiladores y lenguajes de programación "lavaban la cara" a sus compiladores, sacaban nuevas versiones que adoptaran alguno de los conceptos de orientación a objetos y los vendían como orientados a objetos.

Para poner un poco de orden, sobre todo en lo que respecta a la modelización de sistemas software, aparece UML (Unified Modeling Languaje, *Lenguaje Unificado de Modelado*) que pretende unificar las tres metodologías más difundidas (OMT, Bootch y OOSE) e intentar que la industria software termine su maduración como *Ingeniería* . Y lo consigue en tal manera que lo que UML proporciona son las herramientas necesarias para poder obtener los *planos del software* equivalentes a los que se utilizan en la construcción, la mecánica o la industria aeroespacial. UML abarca todas las fases del ciclo de vida de un proyecto, soporta diferentes maneras de visualización dependiendo de quién tenga que interpretar *los planos* y en que fase del proyecto se encuentre. Lo que describiéremos en este curso es una *introducción al diseño orientado a objetos* y que solución aporta UML, explicando sus características principales.

Modelado

Para producir software que cumpla su propósito hay que obtener los requisitos del sistema, esto se consigue conociendo de una forma disciplinada a los usuarios y haciéndolos participar de manera activa para que no queden "cabos sueltos". Para conseguir un software de calidad, que sea duradero y fácil de mantener hay que idear una sólida base arquitectónica que sea flexible al cambio. Para desarrollar software rápida y eficientemente, minimizando el trabajo de recodificación y evitando crear miles de líneas de código inútil hay que disponer, además de la gente y las herramientas necesarias, de un enfoque apropiado.

Para conseguir, que a la hora de desarrollar software de manera industrial se obtenga un producto de calidad, es completamente necesario seguir ciertas pautas y no abordar los problemas de manera somera, con el fin de obtener un modelo que represente lo suficientemente bien el problema que hemos de abordar. El modelado es la espina dorsal del desarrollo software de calidad. Se construyen modelos para poder comunicarnos con otros, para explicar el comportamiento del sistema a desarrollar, para comprender, nosotros mismos, mejor ese sistema, para controlar el riesgo y en definitiva para poder atacar problemas que sin el modelado su resolución seria imposible, tanto

desde el punto de vista de los desarrolladores (no se pueden cumplir los plazos estimados, no se consigue ajustar los presupuestos...) como desde el punto de vista del cliente, el cual, si finalmente se le entrega el producto del desarrollo, se encontrará con infinidades de problemas, desde que no se cumplen las especificaciones hasta fallos que dejan inutilizado el sistema.

Por todas estas razones es inevitable el uso de modelos. Pero, ¿qué es un modelo?. La respuesta es bien sencilla, ***un modelo es una simplificación de la realidad***. El modelo nos proporciona los planos de un sistema, desde los más generales, que proporcionan una visión general del sistema, hasta los más detallados. En un modelo se han de incluir los elementos que tengan más relevancia y omitir los que no son interesantes para el nivel de abstracción que se ha elegido. A través del modelado conseguimos cuatro objetivos:

- Los modelos nos ayudan a visualizar cómo es o queremos que sea un sistema.
- Los modelos nos permiten especificar la estructura o el comportamiento de un sistema.
- Los modelos nos proporcionan plantillas que nos guían en la construcción de un sistema.
- Los modelos documentan las decisiones que hemos adoptado.

Principios básicos del modelado

Existen cuatro principios básicos, estos principios son fruto de la experiencia en todas las ramas de la ingeniería.

a) *La elección de qué modelos se creen influye directamente sobre cómo se acomete el problema*. Hay que seleccionar el modelo adecuado para cada momento y dependiendo de que modelo se elija se obtendrán diferentes beneficios y diferentes costes. En la industria software se ha comprobado que un modelado orientado a objetos proporciona unas arquitecturas más flexibles y readaptables que otros por ejemplo orientados a la funcionalidad o a los datos.

b) Todo modelo puede ser expresado a diferentes niveles de precisión. Esto es, es necesario poder seleccionar el nivel de detalle que se desea ya que en diferentes partes de un proyecto y en diferentes etapas se tendrán unas determinadas necesidades.

c) Los mejores modelos están ligados a la realidad. Lo principal es tener modelos que nos permitan representar la realidad lo más claramente posible, pero no sólo esto, tenemos que saber, exactamente cuando se apartan de la realidad para no caer en la ocultación de ningún detalle importante.

d) *Un único modelo no es suficiente*. Cualquier sistema que no sea trivial se afronta mejor desde pequeños modelos casi independientes, que los podamos construir y estudiar independientemente y que nos representen las partes más diferenciadas del sistema y sus interrelaciones.

Orientación a Objetos

La programación estructurada tradicional se basa fundamentalmente en la ecuación de Wirth:

Algoritmos + Estructuras de Datos = Programas

Esta ecuación significa que en la programación estructurada u orientada a procedimientos los datos y el código se trata por separado y lo único se realiza son funciones o procedimientos que tratan esos datos y los van pasando de unos a otros hasta que se obtiene el resultado que se desea. La *Programación Orientada a Objetos, POO* (OOP, Object Oriented Programming, en ingles), es una técnica de programación cuyo soporte fundamental es el ***objeto***. Un objeto es una extensión de un *Tipo Abstracto de Datos (TAD)*, concepto ampliamente utilizado desde la década de los setenta. Un TAD es un tipo definido por el usuario, que encapsula un conjunto de datos y las operaciones sobre estos datos.

A la hora de definir TAD's (u objetos) se usa un concepto que nos ayuda a representar la realidad mediante modelos informáticos, la ***abstracción,*** que es un proceso mental por el que se evitan los detalles para centrarse en las cosas más genéricas de manera que se facilite su comprensión. De hecho la abstracción no sólo se utiliza en la informática, un arquitecto al que le han encargado realizar los planos de un edificio no comenzará por diseñar los planos con máximo nivel de detalle, sino que comenzará a realzar ciertos esbozos en un papel para posteriormente ir refinando. Por supuesto que cuando está realizando los esbozos no se preocupa de por dónde van a ir las líneas eléctricas ni las tuberías de saneamiento, abstrae esos detalles para atacarlos posteriormente cuando tenga clara la estructura del edificio. La diferencia entre el concepto de TAD y el de ***objeto*** radica en que además del proceso de abstracción que se utiliza para su definición, existen otros dos con los que se forma el núcleo principal de la programación orientada a objetos, estos son la *herencia* y el *polimorfismo*.

Ventajas de la orientación a objetos

Las ventajas más importantes de la programación orientada a objetos son las siguientes:

- *Mantenibilidad* (facilidad de mantenimiento). Los programas que se diseñan utilizando el concepto de orientación a objetos son más fáciles de leer y comprender y el control de la complejidad del programa se consigue gracias a la ocultación de la información que permite dejar visibles sólo los detalles más relevantes.

- *Modificabilidad* (facilidad para modificar los programas). Se pueden realizar añadidos o supresiones a programas simplemente añadiendo, suprimiendo o modificando objetos.

- *Resusabilidad*. Los objetos, si han sido correctamente diseñados, se pueden usar numerosas veces y en distintos proyectos.

- *Fiabilidad*. Los programas orientados a objetos suelen ser más fiables ya que se basan en el uso de objetos ya definidos que están ampliamente testados.

Conceptos básicos de la orientación a objeto

Como ya hemos dicho la orientación a objetos se basa en conceptos como clase, objeto, herencia y polimorfismo, pero también en otros muchos. En esta sección se intenta, sin entrar en detalles, realizar una breve descripción de los conceptos más importantes que existen en el modelado orientado a objetos. Estos conceptos serán explicados y ampliados posteriormente desde la perspectiva de UML.

Clase: Es una descripción de un conjunto de objetos similares. Por ejemplo la clase *Coches*. Una clase contiene los atributos y las operaciones sobre esos atributos que hacen que una clase tenga la entidad que se desea.

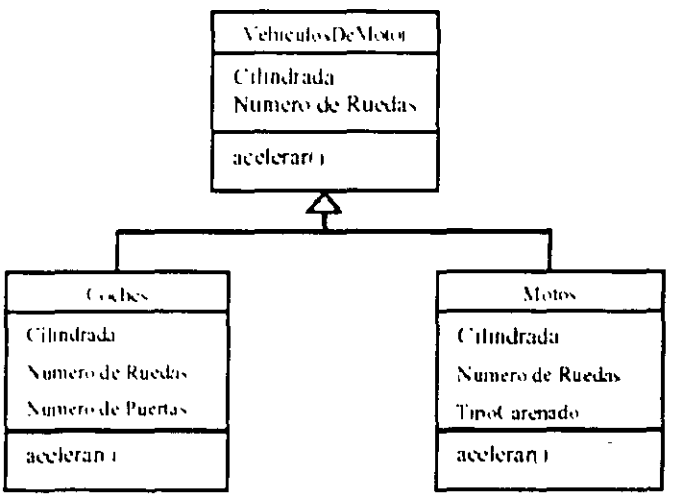
Objeto: Un objeto es una cosa, generalmente extraída del vocabulario del espacio del problema o del espacio de la solución. Todo objeto tiene un nombre (se le puede identificar), un estado (generalmente hay algunos datos asociados a él) y un comportamiento (se le pueden hacer cosas a objeto y él puede hacer cosas a otros objetos). Un objeto de la clase *Coches* puede ser un *Ford Mustang*.

Atributo: Es una característica concreta de una clase. Por ejemplo atributos de la clase *Coches* pueden ser el *Color*, el *Numero de Puertas*...

Método: Es una operación concreta de una determinada clase. Por ejemplo de la clase *Coches* podríamos tener un método *arrancar()* que lo que hace es poner en marcha el coche.

Instancia: Es una manifestación concreta de una clase (un objeto con valores concretos). También se le suele llamar *ocurrencia*. Por ejemplo una instancia de la clase *Coches* puede ser: Un Ford Mustang, de color Gris con 3 puertas

Herencia: Es un mecanismo mediante el cual se puede crear una nueva clase partiendo de una existente, se dice entonces que la nueva clase hereda las características de la case existentes aunque se le puede añadir más capacidades (añadiendo datos o capacidades) o modificar las que tiene. Por ejemplo supongamos que tenemos la *VehiculosDeMotor*. En esta clase tenemos los siguientes atributos: *Cilindrada* y *Numero de Ruedas*, y el método *acelerar()*. Mediante el mecanismo de herencia podemos definir la clase *Coches* y la clase *Motos*. Estas dos clases heredan los atributos *Cilindrada* y *Numero de Ruedas* de la clase *VehiculosDeMotor* pero a su vez tendrán atributos propios (como hemos dicho antes el *Numero de Puertas* es un atributo propio de la clase *Coches* que no tienen sentido en la clase *Motos*). Se puede decir que *Coches* extiende la clase *VehiculosDeMotor*, o que *VehiculosDeMotor* es una ***generalización*** de las clases *Coches* y *Motos*.



Polimorfismo: Hace referencia a la posibilidad de que dos métodos implementen distintas acciones, aun teniendo el mismo nombre, dependiendo del objeto que lo ejecuta o de los parámetros que recibe. En el ejemplo anterior teníamos dos objetos que heredaban el método *acelerar()* de la

clase *VehiculosDeMotor*. De hecho en clase *VehiculosDeMotor* al ser general no tiene sentido que tenga una implementación concreta de este método. Sin embargo, en las clases Coches y Motos si que hay una implementación clara y distinta del método *acelerar()*, lo podemos ver en el código fuente 1 y 2. De este modo podríamos tener un objeto *VehiculosDeMotor*, llamado *vdm*, en el que residiera un objeto Coche. Si realizáramos la llamada *vdm.acelerar()* sabría exactamente que ha de ejecutar el método *Coches::acelerar()*.

Introducción al lenguaje unificado de modelado, UML

UML es un lenguaje estándar que sirve para escribir los *planos del software*, puede utilizarse para visualizar, especificar, construir y documentar todos los artefactos que componen un sistema con gran cantidad de software. UML puede usarse para modelar desde sistemas de información hasta aplicaciones distribuidas basadas en Web, pasando por sistemas empotrados de tiempo real. UML es solamente un lenguaje por lo que es sólo una parte de un método de desarrollo software, es independiente del proceso aunque para que sea optimo debe usarse en un proceso dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental.

UML es un lenguaje por que proporciona un vocabulario y las reglas para utilizarlo, además es un lenguaje de modelado lo que significa que el vocabulario y las reglas se utilizan para la representación conceptual y física del sistema.

UML es un lenguaje que nos ayuda a interpretar grandes sistemas mediante gráficos o mediante texto obteniendo modelos explícitos que ayudan a la comunicación durante el desarrollo ya que al ser estándar, los modelos podrán ser interpretados por personas que no participaron en su diseño (e incluso por herramientas) sin ninguna ambigüedad. En este contexto, UML sirve para *especificar*, modelos concretos, no ambiguos y completos.

Vista general de UML

El lenguaje UML se compone de tres elementos básicos, los bloques de construcción, las reglas y algunos mecanismos comunes. Estos elementos interaccionan entre sí para dar a UML el carácter de completitud y no-ambigüedad que antes comentábamos.

Los **bloques de construcción** se dividen en tres partes: **Elementos**, que son las abstracciones de primer nivel, **Relaciones**, que unen a los elementos entre sí, y los **Diagramas**, que son agrupaciones interesantes de elementos. Existen cuatro tipos de elementos en UML, dependiendo del uso que se haga de ellos: *elementos estructurales, elementos de comportamiento, elementos de agrupación y elementos de anotación.*

Las relaciones, a su vez se dividen para abarcar las posibles interacciones entre elementos que se nos pueden presentar a la hora de modelar usando UML, estas son: *relaciones de dependencia, relaciones de asociación, relaciones de generalización y relaciones de realización.*

Se utilizan diferentes diagramas dependiendo de qué, nos interese representar en cada momento, para dar diferentes perspectivas de un mismo problema, para ajustar el nivel de detalle..., por esta

razón UML soporta un gran numero de diagramas diferentes aunque, en la practica, sólo se utilicen un pequeño número de combinaciones.

UML proporciona un conjunto de reglas que dictan las pautas a la hora de realizar asociaciones entre objetos para poder obtener modelos bien formados, estas son reglas semánticas que afectan a los **nombres**, al **alcance** de dichos nombres, a la **visibilidad** de estos nombres por otros, a la **integridad** de unos elementos con otros y a la **ejecución**, o sea la vista dinámica del sistema.

UML proporciona una serie de mecanismos comunes que sirven para que cada persona o entidad adapte el lenguaje a sus necesidades, pero dentro de un marco ordenado y siguiendo unas ciertas reglas para que en el trasfondo de la adaptación no se pierda la semántica propia de UML. Dentro de estos mecanismos están las **especificaciones**, que proporcionan la explicación textual de la sintaxis y semántica de los bloques de construcción. Otro mecanismo es el de los **adornos** que sirven para conferir a los modelos de más semántica, los adornos son elementos secundarios ya que proporcionan más nivel de detalle, que quizá en un primer momento no sea conveniente descubrir.

Bloques de construcción de UML

A continuación se van a describir todos los elementos que componen los bloques estructurales de UML, así como su notación, para que nos sirva de introducción y se vaya generando un esquema conceptual sobre UML. En temas sucesivos se tratará con más profundidad cada uno de los bloques.

Elementos Estructurales

Los elementos estructurales en UML, es su mayoría, son las partes estáticas del modelo y representan cosas que son conceptuales o materiales.

Clases

Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Una clase implementa una o más interfaces. Gráficamente se representa como un rectángulo que incluye su nombre, sus atributos y sus operaciones (figura 3).

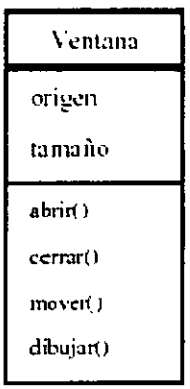


Figura 3. Clases

Interfaz

Una interfaz es una colección de operaciones que especifican un servicio de una determinada clase o componente. Una interfaz describe el comportamiento visible externamente de ese elemento, puede mostrar el comportamiento completo o sólo una parte del mismo. Una interfaz describe un conjunto de especificaciones de operaciones (o sea su signatura) pero nunca su implementación. Se representa con un circulo, como podemos ver en la figura 4, y rara vez se encuentra aislada sino que más bien conectada a la clase o componente que realiza.

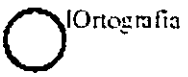


Figura 4. Interfaz

Colaboración

Define una interacción y es una sociedad de roles y otros elementos que colaboran para proporcionar un comportamiento cooperativo mayor que la suma de los comportamientos de sus elementos. Las colaboraciones tienen una dimensión tanto estructural como de comportamiento. Una misma clase puede participar en diferentes colaboraciones. Las colaboraciones representan la implementación de patrones que forman un sistema. Se representa mediante una elipse con borde discontinuo, como en la figura 5.

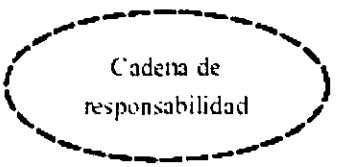


Figura 5 Colaboración

Casos de Uso

Un caso de uso es la descripción de un conjunto de acciones que un sistema ejecuta y que produce un determinado resultado que es de interés para un actor particular. Un caso de uso se utiliza para organizar los aspectos del comportamiento en un modelo. Un caso de uso es realizado por una colaboración. Se representa como en la figura 6, una elipse con borde continuo.

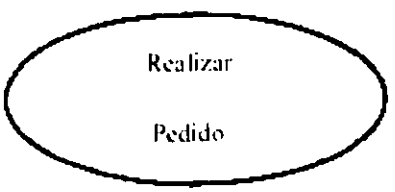


Figura 6. Casos de Uso

Clase Activa

Es una clase cuyos objetos tienen uno o más procesos o hilos de ejecución por lo y tanto pueden dar lugar a actividades de control. Una clase activa es igual que una clase, excepto que sus objetos representan elementos cuyo comportamiento es concurrente con otros elementos. Se representa igual que una clase (figura 3), pero con líneas más gruesas (figura 7).

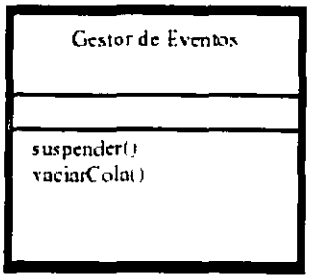


Figura 7. Clases Activas

Componentes

Un componente es una parte física y reemplazable de un sistema que conforma con un conjunto de interfaces y proporciona la implementación de dicho conjunto. Un componente representa típicamente el empaquetamiento físico de diferentes elementos lógicos, como clases, interfaces y colaboraciones.

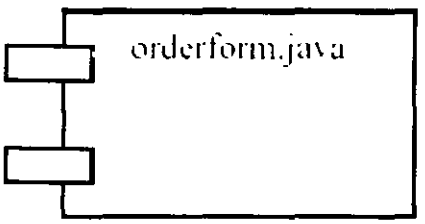


Figura 8. Componentes

Nodos

Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional que, por lo general, dispone de algo de memoria y, con frecuencia, de capacidad de procesamiento. Un conjunto de componentes puede residir en un nodo.

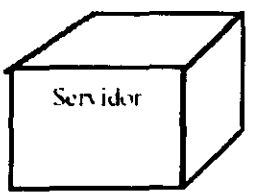


Figura 9. Nodos

Estos siete elementos vistos son los elementos estructurales básico que se pueden incluir en un modelo UML. Existen variaciones sobre estos elementos básicos, tales como actores, señales, utilidades (tipos de clases), procesos e hilos (tipos de clases activas) y aplicaciones, documentos, archivos, bibliotecas, páginas y tablas (tipos de componentes).

Elementos de comportamiento

Los elementos de comportamiento son las partes dinámicas de un modelo. Se podría decir que son los verbos de un modelo y representan el comportamiento en el tiempo y en el espacio. Los principales elementos son los dos que siguen.

Interacción

Es un comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos, dentro de un contexto particular para conseguir un propósito específico. Una interacción involucra otros muchos elementos, incluyendo mensajes, secuencias de acción (comportamiento invocado por un objeto) y enlaces (conexiones entre objetos). La representación de un mensaje es una flecha dirigida que normalmente con el nombre de la operación.

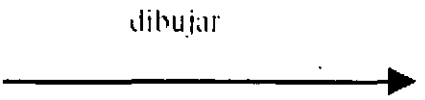


Figura 10. Mensajes

Maquinas de estados

Es un comportamiento que especifica las secuencias de estados por las que van pasando los objetos o las interacciones durante su vida en respuesta a eventos, junto con las respuestas a esos eventos. Una maquina de estados involucra otros elementos como son estados, transiciones (flujo de un estado a otro), eventos (que disparan una transición) y actividades (respuesta de una transición)

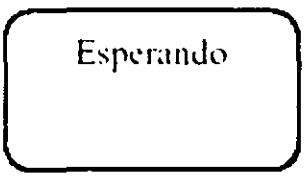


Figura 11. Estados

Elementos de agrupación

Forman la parte organizativa de los modelos UML. El principal elemento de agrupación es el **paquete**, que es un mecanismo de propósito general para organizar elementos en grupos. Los elementos estructurales, los elementos de comportamiento, incluso los propios elementos de agrupación se pueden incluir en un paquete. Un paquete es puramente conceptual (sólo existe en tiempo de desarrollo). Gráficamente se representa como una carpeta conteniendo normalmente su nombre y, a veces, su contenido.

Figura 12. Paquetes

Elementos de anotación

Los elementos de anotación son las partes explicativas de los modelos UML. Son comentarios que se pueden aplicar para describir, clasificar y hacer observaciones sobre cualquier elemento de un modelo. El tipo principal de anotación es la **nota** que simplemente es un símbolo para mostrar restricciones y comentarios junto a un elemento o un conjunto de elementos.

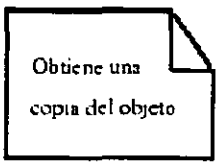


Figura 13. Notas

Relaciones

Existen cuatro tipos de relaciones entre los elementos de un modelo UML. *Dependencia*, *asociación*, *generalización* y *realización*, estas se describen a continuación:

Dependencia

Es una relación semántica entre dos elementos en la cual un cambio a un elemento (el elemento independiente) puede afectar a la semántica del otro elemento (elemento dependiente). Se representa como una línea discontinua (figura 14), posiblemente dirigida, que a veces incluye una etiqueta.

Figura 14. Dependencias

Asociación

Es una relación estructural que describe un conjunto de enlaces, los cuales son conexiones entre objetos. La agregación es un tipo especial de asociación y representa una relación estructural entre un todo y sus partes. La asociación se representa con una línea continua, posiblemente dirigida, que a veces incluye una etiqueta. A menudo se incluyen otros adornos para indicar la multiplicidad y roles de los objetos involucrados, como podemos ver en la figura 15.

Figura 15. Asociaciones

Generalización

Es una relación de especialización / generalización en la cual los objetos del elemento especializado (el hijo) pueden sustituir a los objetos del elemento general (el padre). De esta forma, el hijo

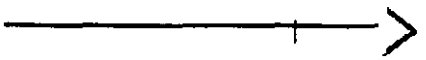


Figura 16. Generalización

comparte la estructura y el comportamiento del padre. Gráficamente, la generalización se representa con una línea con punta de flecha vacía (figura 16).

Realización

Es una relación semántica entre clasificadores, donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Se pueden encontrar relaciones de realización en dos sitios: entre interfaces y las clases y componentes que las realizan, y entre los casos de uso y las colaboraciones que los realizan. La realización se representa como una mezcla entre la generalización (figura 16) y la dependencia (figura 14), esto es, una línea discontinua con una punta de flecha vacía (figura 17).

Diagramas

Los diagramas se utilizan para representar diferentes perspectivas de un sistema de forma que un diagrama es una proyección del mismo. UML proporciona un amplio conjunto de diagramas que normalmente se usan en pequeños subconjuntos para poder representar las cinco vistas principales de la arquitectura de un sistema.

Diagramas de Clases

Muestran un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Estos diagramas son los más comunes en el modelado de sistemas orientados a objetos y cubren la vista de diseño estática o la vista de procesos estática (sí incluyen clases activas).

Diagramas de Objetos

Muestran un conjunto de objetos y sus relaciones, son como fotos instantáneas de los diagramas de clases y cubren la vista de diseño estática o la vista de procesos estática desde la perspectiva de casos reales o prototípicos.

Diagramas de Casos de Usos

Muestran un conjunto de casos de uso y actores (tipo especial de clases) y sus relaciones. Cubren la vista estática de los casos de uso y son especialmente importantes para el modelado y organización del comportamiento.

Diagramas de Secuencia y de Colaboración

Tanto los diagramas de secuencia como los diagramas de colaboración son un tipo de diagramas de interacción. Constan de un conjunto de objetos y sus relaciones, incluyendo los mensajes que se

pueden enviar unos objetos a otros. Cubren la vista dinámica del sistema. Los diagramas de secuencia enfatizan el ordenamiento temporal de los mensajes mientras que los diagramas de colaboración muestran la organización estructural de los objetos que envían y reciben mensajes. Los diagramas de secuencia se pueden convertir en diagramas de colaboración sin perdida de información, lo mismo ocurren en sentido opuesto.

Diagramas de Estados

Muestran una maquina de estados compuesta por estados, transiciones, eventos y actividades. Estos diagramas cubren la vista dinámica de un sistema y son muy importantes a la hora de modelar el comportamiento de una interfaz, clase o colaboración.

Diagramas de Actividades

Son un tipo especial de diagramas de estados que se centra en mostrar el flujo de actividades dentro de un sistema. Los diagramas de actividades cubren la parte dinámica de un sistema y se utilizan para modelar el funcionamiento de un sistema resaltando el flujo de control entre objetos.

Diagramas de Componentes

Muestra la organización y las dependencias entre un conjunto de componentes. Cubren la vista de la implementación estática y se relacionan con los diagramas de clases ya que en un componente suele tener una o más clases, interfaces o colaboraciones

Diagramas de Despliegue

Representan la configuración de los nodos de procesamiento en tiempo de ejecución y los componentes que residen en ellos. Muestran la vista de despliegue estática de una arquitectura y se relacionan con los componentes ya que, por lo común, los nodos contienen uno o más componentes.

Arquitectura

El desarrollo de un sistema con gran cantidad de software requiere que este sea visto desde diferentes perspectivas. Diferentes usuarios (usuario final, analistas, desarrolladores, integradores, jefes de proyecto...) siguen diferentes actividades en diferentes momentos del ciclo de vida del proyecto, lo que da lugar a las diferentes vistas del proyecto; dependiendo de qué interese más en cada instante de tiempo. La arquitectura es el conjunto de decisiones significativas sobre:

La organización del sistema
Selección de elementos estructurales y sus interfaces a través de los cuales se constituye el sistema.
El Comportamiento, como se especifica las colaboraciones entre esos componentes.
Composición de los elementos estructurales y de comportamiento en subsistemas progresivamente más grandes.
El estilo arquitectónico que guía esta organización: elementos estáticos y dinámicos y sus interfaces, sus colaboraciones y su composición.

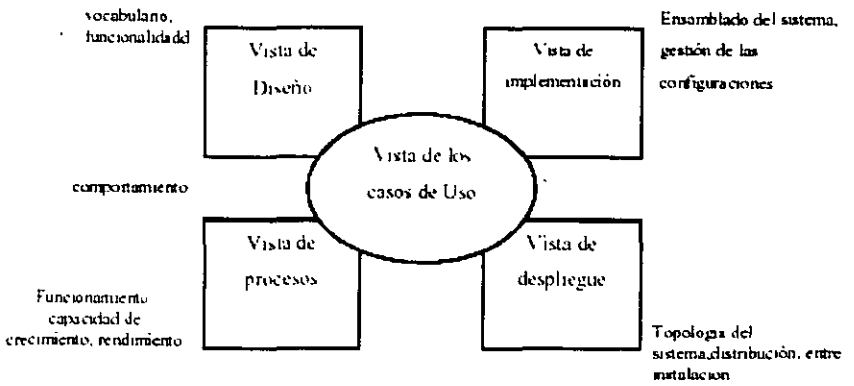


Figura 18 Modelado de la arquitectura de un sistema

La una arquitectura que no debe centrarse únicamente en la estructura y en el comportamiento, sino que abarque temas como el uso, funcionalidad, rendimiento, capacidad de adaptación, reutilización, capacidad para ser comprendida, restricciones, compromisos entre alternativas, así como aspectos estéticos. Para ello se sugiere una arquitectura que permita describir mejor los sistemas desde diferentes vistas, figura 18, donde cada una de ellas es una proyección de la organización y la estructura centrada en un aspecto particular del sistema.

La *vista de casos de uso* comprende la descripción del comportamiento del sistema tal y como es percibido por los usuarios finales, analistas y encargados de las pruebas y se utilizan los diagramas de casos de uso para capturar los aspectos estáticos mientras que los dinámicos son representados por diagramas de interacción, estados y actividades.

La *vista de diseño* comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y de la solución. Esta vista soporta principalmente los requisitos funcionales del sistema, o sea, los servicios que el sistema debe proporcionar. Los aspectos estáticos se representan mediante diagramas de clases y objetos y los aspectos dinámicos con diagramas de interacción, estados y actividades.

La *vista de procesos* comprende los hilos y procesos que forman mecanismos de sincronización y concurrencia del sistema cubriendo el funcionamiento, capacidad de crecimiento y el rendimiento del sistema. Con UML, los aspectos estáticos y dinámicos se representan igual que en la vista de diseño, pero con el énfasis que aportan las clases activas, las cuales representan los procesos y los hilos.

La *Vista de implementación* comprende los componentes y los archivos que un sistema utiliza para ensamblar y hacer disponible el sistema físico. Se ocupa principalmente de la gestión de configuraciones de las distintas versiones del sistema. Los aspectos estáticos se capturan con los

diagramas de componentes y los aspectos dinámicos con los diagramas de interacción, estados y actividades.

La *vista de despliegue* de un sistema contiene los nodos que forman la topología hardware sobre la que se ejecuta el sistema. Se preocupa principalmente de la distribución, entrega e instalación de las partes que constituyen el sistema. Los aspectos estáticos de esta vista se representan mediante los diagramas de despliegue y los aspectos dinámicos con diagramas de interacción, estados y actividades.

Modelado Estructural

A la hora de modelar un sistema es necesario identificar las cosas más importantes eligiendo un nivel de abstracción capaz de identificar todas las partes relevantes del sistema y sus interacciones. Por ejemplo, si estuviésemos trabajando en una tienda que vende ordenadores como proveedor final y nuestro jefe nos pidiera que montáramos un ordenado, directamente la división que tendríamos es la siguiente: necesitamos un monitor, un teclado, un ratón y una CPU, sin importarnos (inicialmente, claro) que la CPU este compuesta por varios elementos más. Una vez que tenemos claro que partes forman un ordenador nos daríamos cuenta que tanto el teclado como el ratón y el monitor son parte más o menos atómicas ya que, aunque estos tres objetos están compuestos por un montón de componentes electrónicos, la composición de estos no nos interesa para al nivel de abstracción que estamos trabajando (el de proveedor final). Al darnos cuenta de esto prepararíamos el monitor, el teclado y el ratón, pero en nuestro almacén tenemos monitores de 15 y 17 pulgadas, teclados ergonómicos y estándares y ratones de diferentes tamaños, colores, etc. Una vez que hemos determinado las propiedades de cada uno de ellos pasaríamos a preocuparnos por la CPU, ahora es cuando veríamos que para montar la CPU nos hacen falta una placa base, un microprocesador, una disquetera, un disco duro y un CD-ROM, cada uno de estos elementos con sus propiedades, disquetera de 1.44Mb, disco duro de 4Gb, microprocesador a 500Mhz y un CD-ROM 32x. Una vez que tenemos todo el material en el banco de trabajo tendríamos que montar el ordenador sabiendo que cada una de las partes interactúa con el resto de alguna manera, en la CPU la disquetera, el CD-ROM y el disco duro van conectados al bus de datos, este además está conectado a la placa base y el micro tiene un lugar bien definido también en la placa base, después conectaríamos el teclado, el monitor y el ratón a la CPU y ¡ya esta!, nuestro ordenador está montado. En este primer acercamiento no debemos contar con los detalles particulares de las cosas que hemos identificado como importantes, estos detalles atacarán posteriormente en cada una de las partes elegidas.

El modelado estructural es la parte de UML que se ocupa de identificar todas las partes importantes de un sistema así como sus interacciones. Para modelar las partes importantes del vocabulario del sistema se utilizan las clases, que nos permiten: identificación y representación de sus propiedades y sus operaciones mediante el mecanismo de abstracción. Las relaciones se utilizan para poder modelar las interacciones entre las clases.

Representación de las Clases en UML

Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Hay que hacer una especial diferenciación para no confundir el concepto de clase con el de objeto. Un objeto es una instancia individual de una clase, así podríamos tener una clase que fuera los monitores y un objeto de la clase monitores que fuera un monitor marca "A" de 17 pulgadas, con la pantalla plana. En UML se usan un rectángulo para representar las clases, con distintos compartimentos para indicar sus atributos y sus operaciones.

Una clase es identifica por un nombre que la distingue del resto, el nombre es una cadena de texto. Ese nombre sólo se denomina nombre simple; un nombre de camino consta del nombre de la clase precedido del nombre del paquete en el que se encuentra incluida. En las figuras 20 y 21 podemos ver la estructura de una clase y distintos nombres de camino y nombres simples.

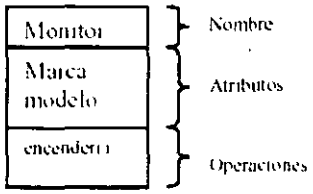


Figura 20. Representación de Clases

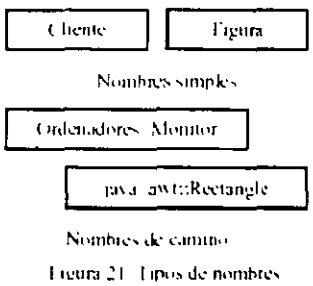


Figura 21. Tipos de nombres

Un atributo es una propiedad de una clase identificada con un nombre. Describe un rango de valores que pueden tomar las instancias de la propiedad. Los atributos representan propiedades comunes a todos los objetos de una determinada clase, por ejemplo todos los monitores tienen la propiedad dimensión, que se suele medir en pulgadas. Un cliente tiene las propiedades nombre, apellidos, dirección, teléfono... Los atributos son propiedades interesantes de las clases. Una instancia de una determinada clase tendrá valores concretos para sus atributos, mientras que las clases tienen rangos de valores que pueden admitir sus atributos.

El nombre de los atributos es un texto que normalmente se escribe en minúsculas si sólo es una palabra o la primera palabra del nombre del atributo en minúsculas y el resto de las palabras con la primera letra en mayúsculas, por ejemplo, nombrePropietario.

Una operación es una implementación de un servicio que puede ser requerido a cualquier objeto de la clase para que muestre su comportamiento. Una operación representa algo que el objeto puede hacer. Por ejemplo, un comportamiento esperado por cualquier usuario de un monitor es que este se pueda encender y apagar. El nombre de las operaciones sigue las mismas reglas de notación que los atributos pero suelen ser verbos cortos que indican una acción o comportamiento de la clase.

Como norma general, cuando se dibuja una clase no hay que mostrar todos sus atributos ni todas sus operaciones, sólo se deben mostrar los subconjuntos de estos más relevantes y una buena práctica es utilizar *estereotipos* para organizar los atributos y las operaciones.

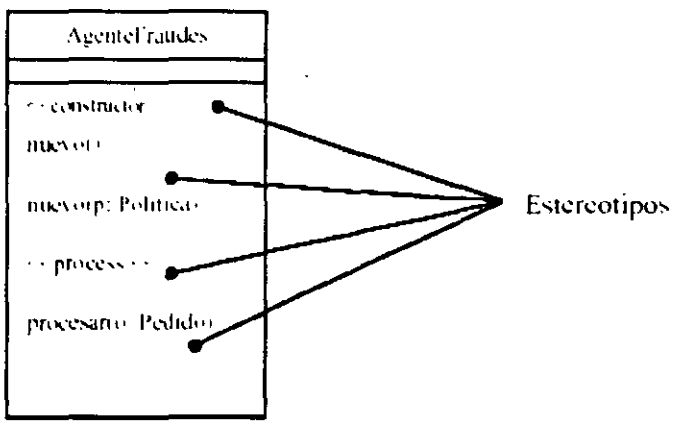


Figura 22. Estereotipos para características de las clases

Responsabilidades

Una responsabilidad es un contrato u obligación de una clase, ósea, el fin para el que es creada. Cuando creamos una clase se está expresando que todos los objetos de la clase tienen el mismo tipo de estado y el mismo tipo de comportamiento. Los atributos y las operaciones son características de la clase que le permiten llevar a cabo sus responsabilidades. Al iniciar el modelado de clases es una buena practica por iniciar especificando las responsabilidades de cada clase. Una clase puede tener cualquier numero de responsabilidades pero en la práctica el numero se reduce a una o a unas pocas. Cuando se van refinando los modelos las responsabilidades se van convirtiendo en atributos y operaciones, se puede decir que las responsabilidades de una clase están en un nivel más alto de

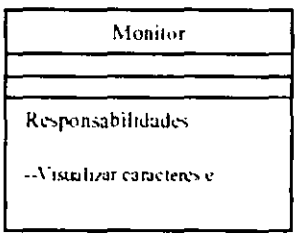


Figura 23. Responsabilidades

abstracción que los atributos y las operaciones. Para representar responsabilidades se utiliza un cuarto compartimiento en el bloque de construcción de la clase, en el cual se especifican mediante frases cortas de texto libre las responsabilidades que ha de cumplir la clase.

Relaciones

Como ya hemos comentado en otras ocasiones, las relaciones son la manera de representar las interacciones entre las clases. Siguiendo con el ejemplo del montaje del ordenador, cada pieza interactúa con otra de una determinada manera y aunque por si solas no tienen sentido todas juntas forman un ordenador, esto es lo que se denomina una relación de asociación, pero además hay unas que no pueden funcionar si no están conectadas a otras como por ejemplo un teclado, el cual, sin estar conectado a una CPU es totalmente inútil, además si la CPU cambiase su conector de teclado este ya no se podría conectar a no ser que cambiase el también, esto se puede representar mediante una relación de dependencia. Es más, tenemos una disquetera de 1,44Mb, un disco duro, un CD-ROM. Para referirnos a todos estos tipos de discos podríamos generalizar diciendo que tenemos una serie de discos con ciertas propiedades comunes, como pueden ser la capacidad, la tasa de transferencia en lectura y escritura... esto es lo que se denomina una relación de generalización. La construcción de relaciones no difiere mucho de la distribución de responsabilidades entre las clases. Si se modela en exceso se obtendrán diagramas con un alto nivel de dificultad para poder leerlos debido principalmente al lío que se forma con las relaciones, por el contrario, si se modela insuficientemente se obtendrán diagramas carentes de semántica.

Para poder representar con UML cómo se conectan las cosas entre sí, ya sea lógica o físicamente, utilizamos las relaciones. Existen tres tipos de relaciones muy importantes: dependencias, generalizaciones y asociaciones. Una relación se define como una conexión entre elementos. A continuación describimos los tres tipos más importantes de relaciones:

Relación de Dependencia

Es una relación de uso entre dos elementos de manera que el un cambio en la especificación del elemento independiente puede afectar al otro elemento implicado en la relación. Determinamos el elemento dependiente aquel que necesita del otro elemento implicado en la relación (el independiente) para poder cumplir sus responsabilidades. Por ejemplo supongamos que tenemos una clase que representa un aparato reproductor Vídeo, con sus funciones y sus propiedades. Bien, para utilizar el método grabar() de la clase video, dependemos directamente de la clase Canal ya que grabaremos un canal u otro dependiendo de cual tenga seleccionado aparato de vídeo. A su vez la clase Televisión también depende de la clase Canal para poder visualizar un determinado canal por la Televisión.

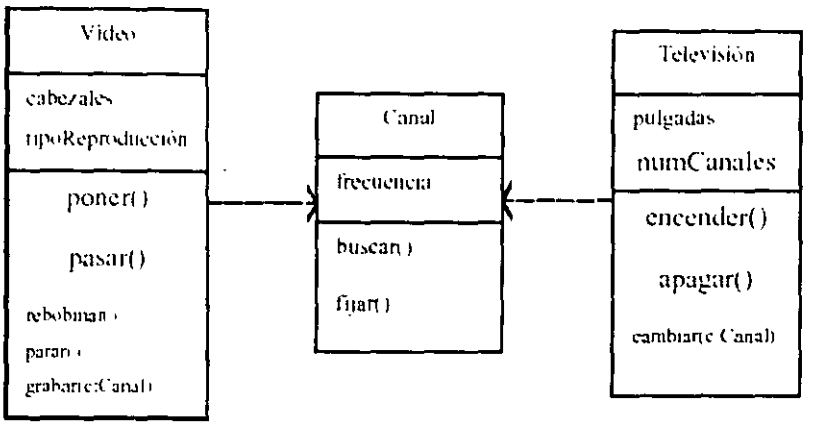


Figura 24 Ejemplo de Dependencias

En la figura 24 podemos observar un ejemplo de relación de dependencia. Si en algún momento la clase Canal cambiara (se modificara o añadiera nueva funcionalidad) las clases Video y Televisión. (que dependen de ella) podrían verse afectadas por el cambio y dejar de funcionar. Como podemos observar en esta figura tanto al método grabar() de la clase video, como al método cambiar() de la clase Televisión se le ha añadido más semántica representando el parámetro *c* de tipo Canal. Este es un mecanismo común en UML de diseño avanzado de clases. Cuando queremos aportar más información sobre una clase y sus métodos existe una nomenclatura bien definida para determinar más los atributos y métodos de la clase indicando si son ocultos o públicos, sus tipos de datos, parámetros que utilizan y los tipos que retornan.

Relación de Generalización

Es una relación entre un elemento general (llamado superclase o padre) y un caso más específico de ese elemento (llamado subclase o hijo). La generalización a veces es llamada relación "*es-un-tipo-de*", ósea, un elemento (por ejemplo, una clase Rectángulo) es-un-tipo-de un elemento más general (por ejemplo, la clase figura). La generalización implica que los objetos hijo se pueden utilizar en cualquier lugar donde aparece el padre, pero no a la inversa. La clase hijo siempre hereda todos los atributos y métodos de sus clases padre y a menudo (no siempre) el hijo extiende los atributos y operaciones del padre. Una operación de un hijo puede tener la misma signatura que en el padre pero la operación puede ser redefinida por el hijo; esto es lo que se conoce como polimorfismo. La generalización se representa mediante una flecha dirigida con la punta hueca. Una clase puede tener ninguno, uno o varios padres. Una clase sin padres y uno o más hijos se denomina clase raíz o clase base. Una clase sin hijos se denomina clase hoja. Una clase con un único padre se dice que utiliza herencia simple y una clase con varios padres se dice que utiliza herencia múltiple. UML utiliza las

relaciones de generalización para el modelado de clases e interfaces, pero también se utilizan para establecer generalizaciones entre otros elementos como por ejemplo los paquetes.

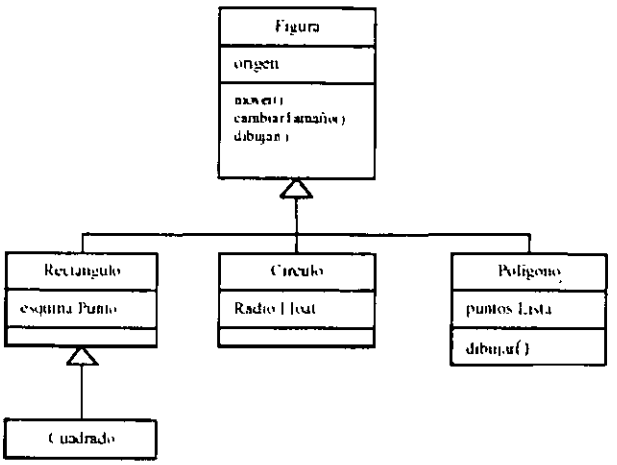


Figura 25. Ejemplo de Generalización

Mediante las relaciones de generalización podemos ver que un Cuadrado es-un-tipo-de Rectángulo que a su vez es-un-tipo-de figura. Con esta relación podemos representar la herencia, de este modo, la clase cuadrado, simplemente por herencia sabemos que tiene dos atributos, *esquina* (heredado de su padre Rectángulo) y *origen* (heredado del padre de Rectángulo, la clase figura, que se puede decir que es su abuelo). Lo mismo ocurre con las operaciones, la clase Cuadrado dispone de las operaciones mover(), cambiarTamaño() y dibujar(), heredadas todas desde figura.

Normalmente la herencia simple es suficiente para modelar los problemas a los que nos enfrentamos pero en ciertas ocasiones conviene modelar mediante herencia múltiple aunque vistos en esta situación se ha de ser extremadamente cuidadoso en no realizar herencia múltiple desde padres que solapen su estructura o comportamiento. La herencia múltiple se representa en UML simplemente haciendo llegar flechas (iguales que las de la generalización) de un determinado hijo a todos los padres dé los que hereda. En el siguiente ejemplo podemos observar como se puede usar la herencia múltiple para representar especializaciones cuyos padres son inherentemente disjuntos pero existen hijos con propiedades de ambos. En el caso de los *Vehículos*, estos se pueden dividir dependiendo de por donde circulen, así tendremos Vehículos aéreos, terrestres y acuáticos. Esta división parece que nos cubre completamente todas las necesidades, y así es. Dentro de los vehículos terrestres tendremos especializaciones como coches, motos, bicicletas, etc. Dentro de los acuáticos tendremos, por ejemplo, barcos, submarinos, etc. Dentro de los aéreos tendremos por ejemplo, aviones, globos, zeppelines... En este momento tenemos una clasificación bastante clara de

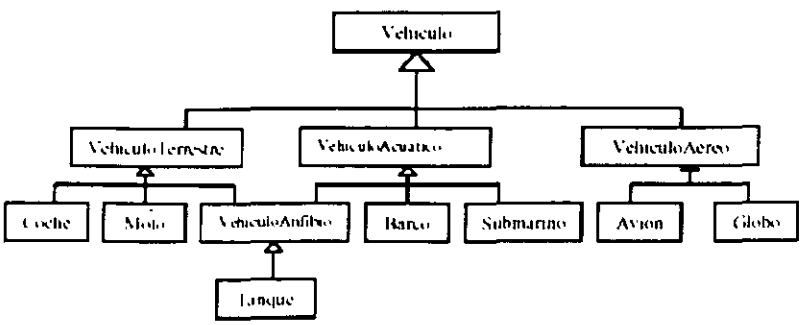


Figura 26. Ejemplo de Herencia Múltiple

los vehículos, pero que ocurre con los vehículos que pueden circular tanto por tierra como por agua, ósea vehículos anfibios, como por ejemplo un tanque de combate preparado para tal efecto, en este caso podemos pensar en utilizar un mecanismo de herencia múltiple para representar que dicho tanque reúne capacidades, atributos y operaciones tanto de vehículo terrestre como acuático.

Relación de Asociación

Una asociación es una relación estructural que especifica que los objetos de un elemento están conectados con los objetos de otro. Dada una asociación entre dos clases, se puede navegar desde un objeto de una de ellas hasta uno de los objetos de la otra, y viceversa. Es posible que la asociación se dé de manera recursiva en un objeto, esto significa que dado un objeto de la clase se puede conectar con otros objetos de la misma clase. También es posible, aunque menos frecuente, que se conecten más de dos clases, estas se suelen llamar asociaciones n-arias. Las relaciones de asociaciones se utilizan cuando se quieren representar relaciones estructurales. A parte de la forma básica de representar las asociaciones, mediante una línea continua entre las clases involucradas en la relación, existen cuatro adornos que se aplican a las asociaciones para facilitar su comprensión:

- ***Nombre:*** Se utiliza para describir la naturaleza de la relación. Para que no exista ambigüedad en su significado se le puede dar una dirección al nombre por medio de una flecha que apunte en la dirección que se pretende que el nombre sea leído.

- ***Rol:*** Cuando una clase participa en una asociación esta tiene un rol especifico que juega en dicha asociación. El rol es la cara que dicha clase presenta a la clase que se encuentra en el otro extremo. Las clases pueden jugar el mismo o diferentes roles en otras asociaciones.

- ***Multiplicidad:*** En muchas situaciones del modelado es conveniente señalar cuantos objetos se pueden conectar a través de una instancia de la asociación. Este "cuantos" se denomina multiplicidad del rol en la asociación y se expresa como un rango de valores o un valor explicito. Cuando se indica multiplicidad en un extremo de una asociación se está indicando que, para cada objeto de la clase en el extremo opuesto debe haber tantos objetos en este extremo. Se puede indicar una multiplicidad de exactamente uno (1), cero o uno (0..1), muchos (0..*), uno o más (1..*) e incluso un numero exacto (por ejemplo, 5).

- ***Agregación:*** Una asociación normal entre dos clases representa una relación estructural entre iguales, es decir, ambas clases están conceptualmente al mismo nivel. A veces interesa representar relaciones del tipo "todo / parte", en las cuales una cosa representa la cosa grande (el "todo") que consta de elementos más pequeños (las "partes"). Este tipo de relación se denomina de agregación la cual representa una relación del tipo "tiene-un".

  Una agregación es sólo un tipo especial de asociación, esta se especifica añadiendo simplemente un rombo vacio en la parte del todo.

- ***Composición:*** Es una variación de la agregación simple que añade una semántica importante. La composición es una forma de agregación, con una fuerte relación de pertenencia y vidas coincidentes de la parte del todo. Las partes con una multiplicidad no fijada puede crearse después de la parte que representa el todo (la parte compuesta), una vez creadas pertenecen a ella de manera que viven y mueren con ella. Las partes pueden ser eliminadas antes que el todo sea destruido pero una vez que este se elimine todas sus partes serán destruidas. El todo, además, se ha de encargar de toda la gestión de sus partes, creación, mantenimiento, disposición... En la figura 29 vemos una relación de composición

donde un marco pertenece a una ventana, pero ese marco no es compartido por ninguna otra ventana, esto contrasta con la agregación simple en la que una parte puede ser compartida por varios objetos agregados. Por ejemplo una pared puede estar compartida por varias habitaciones.

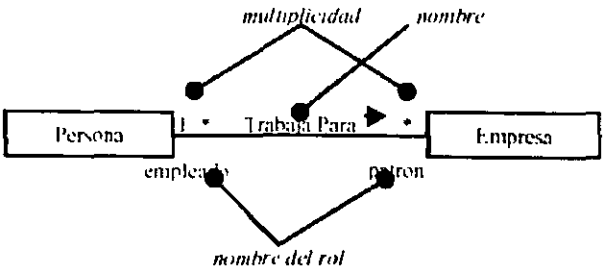


Figura 27. Ejemplo de asociacion y sus partes

Figura 28. Ejemplo de agregación

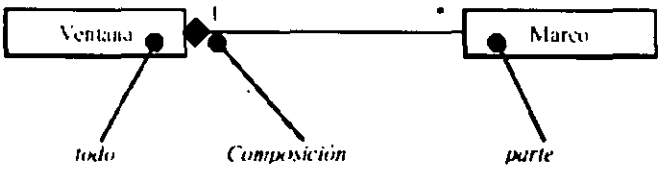


Figura 29. Ejemplo de Composicion

Interfaces

Una interfaz es una colección de operaciones que sirven para especificar el servicio que una clase o componente da al resto de las partes involucradas en un sistema. Al declarar una interfaz, se puede enunciar el comportamiento deseado de una abstracción independientemente de su implementación. Los clientes trabajan con esa interfaz de manera independiente, así que si en un momento dado su implementación cambia, no seremos afectados, siempre y cuando se siga manteniendo su interfaz intacta cumpliendo las responsabilidades que tenia.

UML utiliza las interfaces para modelar las líneas de separación del sistema. Muchos lenguajes de programación soportan el concepto de interfaces, como pueden ser Java, Visual Basic y el IDL de CORBA. Además, las interfaces no son sólo importantes para separar la especificación y la implementación de clases o componentes, sino que al pasar a sistemas más grandes, se pueden usar para especificar la vista externa de un paquete o subsistema.

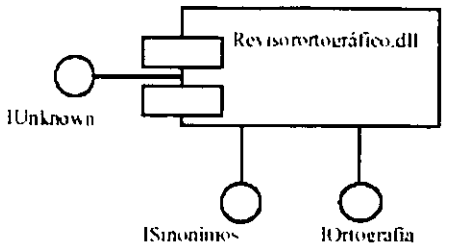


Figura 30 Interfaces

Existen dos formas de representar un interfaz en UML, la primera es mediante una piruleta conectada a un lado de una clase o componente. Esta forma es útil cuando queremos visualizar las líneas de separación del sistema ya que por limitaciones de estilo no se pueden representar las operaciones o las señales de la interfaz. La otra forma de representar una interfaz es mostrar una clase estereotipada que permite ver las operaciones y otras propiedades, y conectarla mediante una relación de realización con la componente o el clasificador que la contiene. Una realización se representa como una flecha de punta vacía con la línea discontinua.

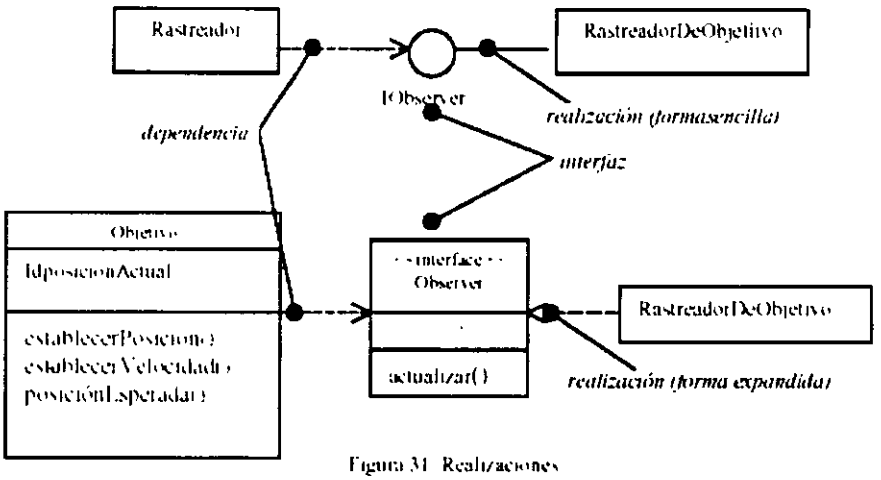


Figura 31 Realizaciones

La figura 31 muestra un ejemplo sobre las dos formas de representar un interfaz con UML.

Roles

Según hemos dicho, una clase puede implementar varios interfaces, por lo tanto una instancia de esa clase debe poder soportar todos esos interfaces, no obstante en determinados contextos, sólo un o más interfaces tendrán sentido. En este caso cada interfaz representa un rol que juega el objeto. Un rol denota un comportamiento de una entidad en un contexto particular, dicho de otra manera un rol es la cara que una abstracción presenta al mundo.

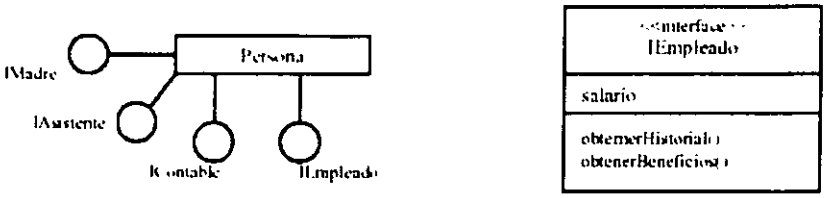


Figura 32. Roles

Para ilustrar este concepto supongamos la clase *Persona* y los roles que una persona puede desempeñar dependiendo del contexto en que se encuentre, como ejemplos de roles podemos tener: Madre, Asistente, Contable, Empleado, Directivo, Piloto, Cantante, etc. La cantidad de roles que pueda soportar una clase es en principio indefinido, solamente depende de ámbito de actuación del sistema que estemos modelando. Cada uno de los roles que se han definido tendrá una correspondencia con un interfaz concreto.

Como se puede observar en la figura 32, el interfaz IEmpleado tiene unas operaciones específicas para una persona que desempeñe el rol de Empleado. También podemos observar los diferentes interfaces que tiene la clase *Persona*, aunque lo usual es utilizar la notación en forma de piruleta para denotar líneas de separación del sistema que comúnmente serán necesario para componentes más que para clases y para estas utilizar la notación expandida de los interfaces con relaciones de realización.

Paquetes

Visualizar, especificar, construir y documentar grandes sistemas conlleva manejar una cantidad de clases, interfaces, componentes, nodos, diagramas y otros elementos que puede ser muy elevada. A medida que el sistema va creciendo hasta alcanzar un gran tamaño, se hace necesario organizar estos elementos en bloques mayores. En UML el paquete es un mecanismo de propósito general para organizar elementos de modelado en grupos. La visibilidad de los elementos dentro de un paquete se puede controlar para que algunos elementos sean visibles fuera del paquete mientras que otros permanezcan ocultos. Los paquetes también se pueden emplear para presentar las diferentes vistas de la arquitectura de un sistema. Todos los grandes sistemas se jerarquizan en niveles para facilitar su comprensión. Por ejemplo, cuando hablamos de un gran edificio podemos hablar de estructuras simples como paredes, techos y suelos, pero debido al nivel de complejidad que supone hablar de un gran edificio utilizamos abstracciones mayores como pueden ser zonas públicas, el área comercial y las oficinas. Al mismo tiempo, estas abstracciones pueden ser que se agrupen en otras mayores como la zona de alquileres y la zona de servicios del edificio, estas agrupaciones puede ser que no tengan nada que ver con la estructura final del edificio sino se usan simplemente para organizar los planos del mismo.

Términos y conceptos

En UML las abstracciones que organizan un modelo se llaman *paquetes*. Un paquete es un mecanismo de propósito general para organizar elementos en grupos. Los paquetes ayudan a organizar los elementos en los modelos con el fin de poder comprenderlos más fácilmente. Los paquetes también permiten controlar el acceso a sus contenidos para controlar las líneas de separación de un sistema. UML proporciona una representación gráfica de los paquetes como se muestra en la figura 33, esta notación permite visualizar grupos de elementos que se pueden manipular como un todo y en una forma permite controlar la visibilidad y el acceso a los elementos individuales.

Cada paquete ha de tener un nombre que lo distinga de otros paquetes, el nombre puede ser un nombre simple o un nombre de camino que indique el paquete donde está contenido. Al igual que las clases, un paquete, se puede dibujar adornado con valores etiquetados o con apartados adicionales para mostrar sus detalles.

Elementos Contenidos

Un paquete puede contener otros elementos, incluyendo clases, interfaces, componentes, nodos, colaboraciones, casos de uso, diagramas e incluso otros paquetes. La posesión es una relación compuesta, lo que significa que el elemento se declara en el paquete.

Si el paquete se destruye, el elemento es destruido. Cada elemento pertenece exclusivamente a un paquete. Un paquete forma un espacio de nombres, lo que quiere decir que los elementos de la misma categoría deben tener nombres únicos en el contexto del paquete contenedor. Por ejemplo no se pueden tener dos clases llamadas *Cola* dentro de un mismo paquete, pero si se puede tener la clase *Cola* dentro del paquete *P1* y otra clase (diferente) llamada *Cola* en el paquete *P2*. Las clases *P1::Cola* y *P2::Cola* son, de hecho, clases diferentes y se pueden distinguir por sus nombres de camino.

Diferentes tipos de elementos dentro del mismo paquete pueden tener el mismo nombre, por ejemplo, podemos tener dentro de un paquete la clase *Temporizador* y un componente llamado *Temporizador*. En la práctica, para evitar confusiones, es mejor asociar cada elemento a un nombre único para todas las categorías.

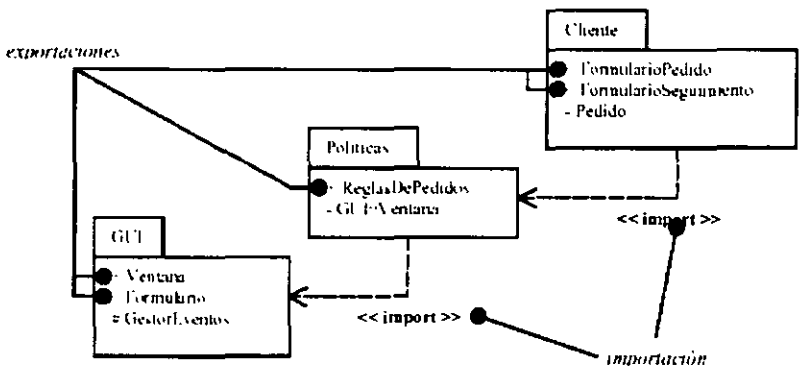


Figura 34 Importación y Exportación

Instancias

Los términos "instancia" y "objeto" son en gran parte sinónimos y, por ello, la mayoría de las veces pueden intercambiarse. Una instancia es una manifestación concreta de una abstracción a la que se puede aplicar un conjunto de operaciones y que puede tener un estado que almacena los efectos de la operación. Las instancias se utilizan para modelar cosas concretas o prototípicas del mundo real. Casi todos los bloques de construcción de UML participan de esta dicotomía clase / objeto. Por ejemplo, puede haber casos de uso e instancias de casos de uso, nodos e instancias de nodos, asociaciones e instancias de asociaciones, etc.

Una abstracción denota la esencia ideal de una cosa; una instancia denota una manifestación concreta. En UML se pueden representar abstracciones y sus instancias. Casi todos los bloques de construcción de UML pueden modelarse en términos de su esencia o en términos de sus instancias.

Una *instancia* es una manifestación concreta de una abstracción a la que se puede aplicar un conjunto de operaciones y posee un estado que almacena el efecto de las operaciones. Gráficamente, una instancia se representa subrayando su nombre.

Operaciones

Un objeto no sólo es algo que normalmente ocupa espacio en el mundo real, también es algo a lo que se le pueden hacer cosas. Las operaciones que se pueden ejecutar sobre un objeto se declaran en la abstracción del objeto (en la clase). Por ejemplo, si la clase *Transacción* define la operación *commit*, entonces, dada una instancia *t : Transacción*, se pueden escribir expresiones como *t.commit()*. La ejecución de esta expresión significa que sobre *t* (el objeto) opera *commit* (la operación).

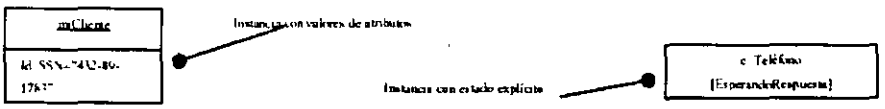


Figura 35. Estado de un objeto

Estado

Un objeto también tiene estado, en este sentido incluye todas las propiedades (normalmente estáticas) del objeto más los valores actuales (normalmente dinámicos) de estas propiedades. Estas propiedades incluyen los atributos del objeto, así como sus partes agregadas. El estado de un objeto, pues, dinámico, deforma que al visualizar su estado se está especificando su estado en un momento dado tiempo y del espacio. Es posible mostrar el cambio de estado de un objeto representándolo muchas veces en el mismo diagrama de interacción, pero con cada ocurrencia se está representando un estado diferente. Cuando se opera sobre un objeto normalmente cambia su estado, mientras que cuando se consulta un objeto su estado no cambia. En la figura 35 podemos observar dos instancias (dos objetos) donde se muestra por un lado sus atributos y por otro su estado de manera explicita.

Diagramas de clases y de objetos

Los diagramas de clases son los más utilizados en el modelado de sistemas orientados a objetos. Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Los diagramas de clases se utilizan para modelar la vista de diseño estática de un sistema. Principalmente, esto incluye modelar el vocabulario del sistema, modelar las colaboraciones o modelar esquemas. Los diagramas de clases también son la base para un par de diagramas relacionados, los diagramas de componentes y los diagramas de despliegue. Los diagramas de clases son importantes no sólo para visualizar, especificar y documentar modelos estructurales, sino que también para construir sistemas ejecutables aplicando ingeniería directa e inversa. Por otro lado, los diagramas de objetos modelan las instancias de los elementos contenidos en los diagramas de clases. Un diagrama de objetos muestra un conjunto de objetos y sus relaciones en un momento concreto. Se utilizan para modelar la vista de diseño estática o la vista de procesos estática, esto conlleva el modelado de una instantánea del sistema en un momento concreto y la representación de un conjunto de objetos con su estado y con sus relaciones.

Diagramas de Clases

Un diagrama de clases es un diagrama que muestra un conjunto de clases, interfaces, colaboraciones y sus relaciones. Al igual que otros diagramas los diagramas de clases pueden contener notas y

restricciones. También pueden contener paquetes o subsistemas, los cuales su usan para agrupar los elementos de un modelo en partes más grandes. A veces se colocarán instancias en los diagramas de clases, especialmente cuando se quiera mostrar el tipo (posiblemente dinámico) de una instancia. Los diagramas de componentes y de despliegue son muy parecidos a los de clases, simplemente que muestran componentes y nodos respectivamente en vez de clases.

Usos comunes

Los diagramas de clases se utilizan para modelar la vista de diseño estática de un sistema. Esta vista soporta principalmente los requisitos funcionales de un sistema, los servicios que el sistema debe proporcionar a los usuarios finales. Cuando se modela la vista de diseño estática de un sistema, normalmente se utilizarán los diagramas de clases de unas de estas tres formas:

1. **Para modelar el vocabulario de un sistema.** El modelado del vocabulario de un sistema implica tomar decisiones sobre qué abstracciones son parte del sistema en consideración y cuáles caen fuera de sus límites. Los diagramas de clases se utilizan para especificar estas abstracciones y sus responsabilidades.
2. **Para modelar colaboraciones simples.** Una colaboración es una sociedad de clases, interfaces y otros elementos que colaboran para proporcionar un comportamiento cooperativo mayor que la suma de todos sus elementos. Por ejemplo, cuando se modela la semántica de una transición en un sistema distribuido, no se puede observar simplemente a una clase aislada para comprender qué ocurre. Más bien, esta semántica se lleva a cabo por un conjunto de clases que colaboran entre sí. Los diagramas de clases se emplean para visualizar y especificar este conjunto de clases.

3. **Para modelar el esquema lógico de una base de datos** Se puede pensar en un esquema como en un plano para el diseño conceptual de una base de datos. En muchos dominios se necesitará almacenar información persistente en una base de datos relacional o en una base de datos orientada a objetos. Se pueden modelar esquemas para estas bases de datos mediante diagramas de clases.

Modelado de colaboraciones simples

Ninguna clase se encuentra aislada. En vez de ello, cada una trabaja en colaboración con otras para llevar a cabo alguna semántica mayor que la asociada a cada clase individual. Por tanto, aparte de capturar el vocabulario del sistema, también hay que prestar atención a la visualización, especificación, construcción y documentación de la forma en que estos elementos del vocabulario colaboran entre sí. Estas colaboraciones se representan con los diagramas de clases. Cuando se crea un diagrama de clases, se está modelando una parte de los elementos y relaciones que configuran la vista de diseño del sistema. Por esta razón, cada diagrama de clases debe centrarse en una colaboración cada vez. Para modelar una colaboración:

Hay que identificar los mecanismos que se quiere modelar. Un mecanismo representa una función o comportamiento de parte del sistema que se está modelando que resulta de la interacción de una sociedad de clases, interfaces y otros elementos.

Para cada mecanismo, hay que identificar las clases, interfaces y otras colaboraciones que participan en esta colaboración. Asimismo, hay que identificar las relaciones entre estos elementos.

Hay que usar escenarios para recorrer la interacción entre estos elementos. Durante el recorrido, se descubrirán partes del modelo que faltaban y partes que eran semánticamente incorrectas.

Hay que asegurarse de rellenar estos elementos con su contenido. Para las clases hay que comenzar obteniendo un reparto equilibrado de responsabilidades. Después, a lo largo del tiempo, hay que convertir las responsabilidades en atributos y operaciones concretos.

Diagramas de Objetos

En UML los diagramas de clases se utilizan para visualizar los aspectos estáticos de los bloques de construcción del sistema. Los diagramas de interacción se utilizan para ver los aspectos dinámicos del sistema y constan de instancias de estos bloques y mensajes enviados entre ellos. Un diagrama de objetos contiene un conjunto de instancias de los elementos encontrados en un diagrama de clases. Por lo tanto, un diagrama de objetos expresa la parte estática de una interacción, consistiendo en los objetos que colaboran pero sin ninguno de los mensajes enviados entre ellos. En ambos casos, un diagrama de objetos congela un instante en el tiempo, como se muestra en la figura 40. Un diagrama de objetos es simplemente un diagrama que representa un conjunto de objetos y sus relaciones en un momento concreto.

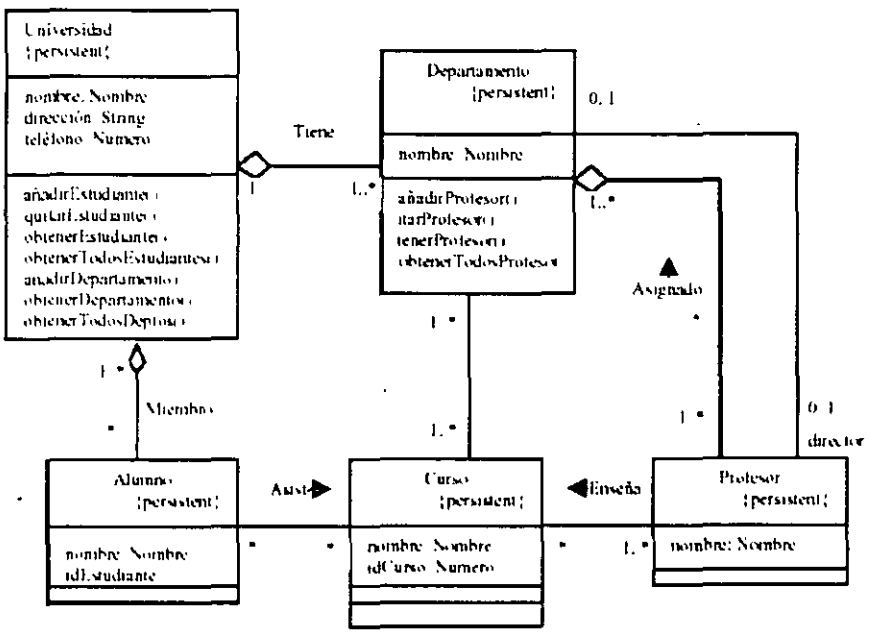


Figura 39 Modelado de un esquema

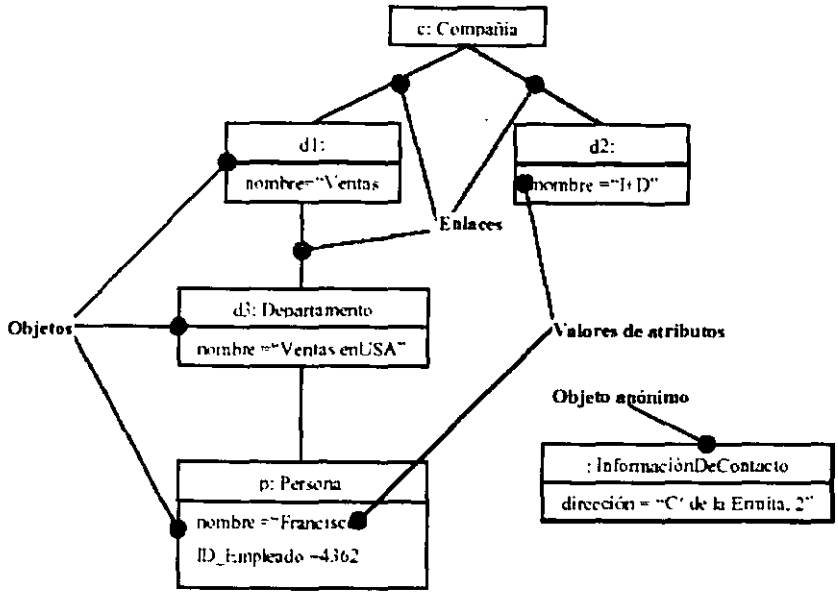


Figura 40 Un diagrama de Objetos

Modelado del comportamiento

Interacciones

En cualquier sistema, los objetos interactúan entre sí pasándose mensajes. Una interacción es un comportamiento que incluye un conjunto de mensajes intercambiados por un conjunto de objetos dentro de un contexto para lograr un propósito.

Las interacciones se utilizan para modelar los aspectos dinámicos de las colaboraciones, que representan sociedades de objetos que juegan roles específicos, y colaboran entre sí para llevar a cabo un comportamiento mayor que la suma de los comportamientos de sus elementos. Estos roles representan instancias prototipicas de clases, interfaces, componentes, nodos y casos de uso. Los aspectos dinámicos se visualizan, se especifican, se construyen y se documentan como flujos más complejos que impliquen bifurcaciones, iteraciones, recursión y concurrencia. Cada iteración puede modelarse de dos formas: bien destacando la ordenación temporal de los mensajes, bien destacando la secuencia de mensajes en el contexto de una organización estructural de objetos. Las interacciones bien estructuradas son como los algoritmos bien estructurados: eficientes, sencillos, adaptables y comprensibles.

En UML, los aspectos estáticos de un sistema se modelan mediante elementos tales como los diagramas de clases y los diagramas de objetos. Estos diagramas permiten especificar, construir y documentar los elementos del sistema, incluyendo clases, interfaces, componentes, nodos y casos de uso e instancias entre ellos, así como la forma en que estos elementos se relacionan entre sí.

UML proporciona una representación gráfica para los mensajes, esta notación permite visualizar un mensaje de forma que podamos destacar todas sus partes más importantes: nombre, parámetros (si los tiene) y secuencia. Gráficamente un mensaje se representa como una línea dirigida y casi siempre incluye el nombre de su operación.

Una **interacción** es un comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos dentro de un contexto para lograr un propósito.

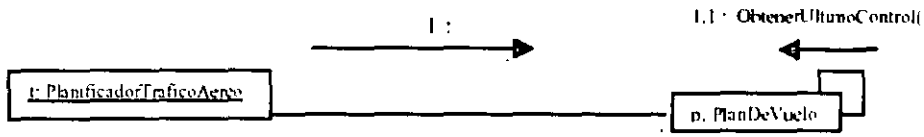


Figura 42 Mensajes, enlaces y secuenciamiento

Un **mensaje** es la especificación de una comunicación entre objetos que transmite información, con la expectativa de que se desencadenará una actividad.