Capítulo VIII

Unified Modeling Language

Unified modeling language

Tabla de contenido

1.- El Paradigma de la OO 115
2.- Objetos y Clases 116
3.- Atributos 116
4.- Comportamientos, Operaciones o Métodos 117
4.1.- Encapsulación 117
4.2.- Mensajes 117
4.3.- Herencia 117
4.4.- Abstracción 119
4.5.- Importancia de la Abstracción 120
5.- ¿Qué es UML? 121
6.- Objetivos del UML 122
7.- El Lenguaje UML 123
8.- Vistas UML 127
9.- Modelos y metamodelos 128
10.- Extensiones 130
11.- Model driven architecture 131

Unified modeling language

1.- El Paradigma de la OO

Antes de revisar formalmente las herramientas UML, queremos conversar sobre algunas cosas que nos son familiares y que nos permitirán entender mejor el conjunto de conceptos en los que se basan dichas herramientas.

De una u otra forma, todos estamos familiarizados con un automóvil, sabemos que un automóvil tiene, entre otras cosas, un sistema de encendido y un motor. Cada uno de estos objetos tiene características y estructuras que les son propias. El motor funciona con sus propios elementos, cumpliendo las operaciones para las que fue diseñado. Igualmente, el sistema de encendido tiene sus propios elementos y su forma de funcionar. Son, por así decirlo, objetos independientes.

Cuando el operador del automóvil hace girar la llave, el sistema de encendido realiza ciertas funciones y "hace" que el motor comience marchar.

Esto mismo podríamos expresarlo diciendo que, el operador, al girar la llave, envía un mensaje al sistema de encendido y éste, como respuesta, cumple sus funciones. Una de esas funciones es "enviar un mensaje" al motor, para que este comience a realizar sus operaciones, haciendo uso de sus elementos.

Dentro del paradigma o marco conceptual de la "orientación a objetos", esta visión, aunque pueril, es la forma como visualizamos los sistemas: un sistema está compuesto de objetos, cada uno de ellos con su propia estructura y forma de operar, que se comunican entre si, intercambiando mensajes. Cada mensaje, al ser recibido por el objeto destinatario, desencadena la ejecución de las operaciones que son propias de ese objeto.

2.- Objetos y Clases

Denominamos objeto a cualquier “cosa” que existe en el mundo real, tanto cosas tangibles, como cosas intangibles o inmateriales, como son los conceptos, las ideas, etc.

Una clase es un conjunto de cosas similares, algo acerca de lo cual guardamos información. Ese algo puede ser un conglomerado de objetos tangibles - como es el caso de la clase EMPLEADO - o de objetos de naturaleza totalmente conceptual - como es el caso de la clase CARGO (conjunto de responsabilidades o tareas a las que un empleado puede ser asignado).

Una clase puede estar "poblada" por uno o más objetos. Por ejemplo, la clase EMPLEADO está integrada, en una determinada empresa, por todas las personas que trabajan en ella, y el empleado Juan Pérez vendría a ser un objeto o instancia de esa clase (en UML se utilizan indistintamente los términos objeto e instancia).

Así pues, una clase no es más que un tipo de objetos: personas, libros, automóviles, empleados. En cierta forma, el concepto de clase equivales al de entidad en el modelo de entidad-relación, sólo que una clase, además de atributos, incluye los métodos, como veremos en los puntos siguientes.

3.- Atributos

Los objetos de una clase tienen una serie de características o propiedades que denominamos atributos. Por ejemplo, cada objeto de la clase EMPLEADO tiene nombre, dirección, sexo, edad, fecha de ingreso a la empresa, etc. Todas esas características constituyen los atributos de la clase empleado. Debe mantenerse siempre presente que todos los atributos de una clase se aplican o tienen el mismo significado para todos sus objetos.

Los datos son símbolos matemáticos o de lenguaje o de cualquier otra naturaleza que se utilizan para describir los atributos de las clases. La descripción de una característica particular de un objeto específico dentro de una clase es el valor del dato. Para cada objeto de una clase, cada atributo tiene un valor específico: Juan Lagartón (objeto de la clase empleado) tiene una estatura (atributo) de 1,75 metros.

4.- Comportamientos, Operaciones o Métodos

Con los objetos que componen una clase se realizan operaciones o transacciones que son típicas o características de la clase; por ejemplo, con una Cuenta Corriente Bancaria, se realizan operaciones de depósito o de retiro de dinero.

Con esta idea intuitiva en mente, diremos que los métodos u operaciones de una clase son los actos, sucesos o acciones que, de alguna manera, afectan a uno o más objetos de esa clase, o que constituyen la forma como interactúa el objeto con su entorno.

4.1.- Encapsulación

Dentro del marco conceptual de las metodologías orientadas a objetos, se expresan los conceptos que hemos descrito en los párrafos precedentes, diciendo que un "objeto se define a la vez por sus atributos y sus métodos" o que "un objeto encapsula atributos y métodos".

4.2.- Mensajes

El uso de un objeto o la ejecución de alguna de sus operaciones "se dispara" mediante la recepción de un mensaje.

Dentro de la orientación a objetos, decimos que se envía un mensaje a un objeto y éste, como respuesta al mensaje, puede "reaccionar" en dos formas diferentes:

- Si el mensaje está asociado a alguno de los métodos definidos para la clase del objeto, se ejecutarán las funciones correspondientes a ese método. Vale decir, el objeto responderá al mensaje que se le ha enviado, ejecutando el método correspondiente.
- Si el mensaje no corresponde a ninguno de los métodos definidos para la clase del objeto, el objeto rechazará el mensaje, señalando el error.

Dentro del paradigma de orientación a objetos, el comportamiento de un objeto es responsabilidad del propio objeto, pues cada objeto es responsable del control y la reacción a los mensajes que se le envían.

4.3.- Herencia

La noción de clase, como arriba observamos, puede asimilarse a la noción de tipo de objeto:

- Una clase define las propiedades de todos los objetos que la integran.

- Un objeto es una instancia, ocurrencia o caso particular o, si se quiere, una materialización particular de su clase.
- Los objetos de una misma clase tienen los mismos atributos y estos atributos adquieren valores particulares para cada objeto.
- Todas las instancias u objetos de una clase se comportan de la misma forma, esto es, comparten las mismas operaciones o métodos.

Nuestro poder de abstracción nos permite que, en muchas oportunidades, para poder explicar las cosas en forma más general y sencilla, agrupemos “en un mismos saco” (en una misma clase) cosas que, si bien son diferentes, guardan alguna relación o característica común. Por ejemplo, hablamos de vehículos, agrupando en ese concepto los camiones, los automóviles, las bicicletas, etc.

Cuando hacemos este tipo de abstracción, estamos reconociendo que, si bien las bicicletas, los camiones y los automóviles son cosas totalmente diferentes, estas familias de objetos (clases) tienen algunas características comunes que nos permiten “ponerlas en un mismo saco”, para tratarlas como un único conglomerado, como si se tratara de una única clase. Ello, a su vez, nos permite que, de una sola vez, podamos decir cosas que aplican a los camiones, los automóviles y las bicicletas, sin tener que estar repitiendo la lista.

Es obvio que al agrupar cosas, estamos aceptando que el conglomerado (vehículos) tiene una serie de características (atributos) y opera en ciertas formas (métodos) que son comunes a todas las subclases o clases subordinadas (camiones, automóviles y bicicletas).

En la terminología OO esta noción intuitiva se expresa diciendo que las clases subordinadas o especializadas heredan los atributos y métodos de la superclase. También se dice que entre las clases subordinadas (secundaria) y la superclase (primaria) existe una relación de Generalización o de Herencia.

Así pues, la existencia de una relación de herencia entre dos clases, indica que la subclase hereda los métodos y atributos de la superclase, además de poseer también métodos y atributos que le son propios.

En nuestro ejemplo, Automóvil, Camión y Bicicleta heredan, de la superclase Vehículo, las características: métodos (transportar) y atributos (precio, peso, etc.). Adicionalmente, la clase Automóvil posee atributos particulares, como cantidad de puertas, cantidad de puestos, etc. y, a su vez, Camión también hereda los atributos de Vehículo, pero posee particularidades propias: Tipo de Acoplado y Capacidad de Carga.

4.4.- Abstracción

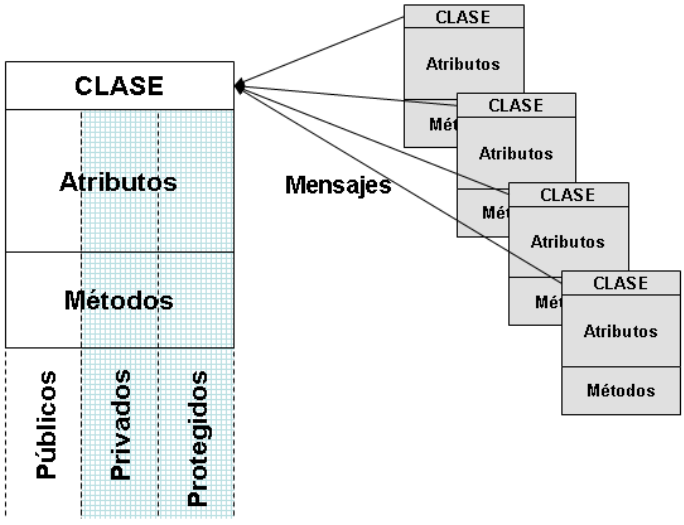
Decíamos que una clase se caracteriza por sus atributos y sus operaciones. Añadiremos ahora otra consideración, su abstracción. La abstracción de un objeto es la separación de su parte pública, su parte privada y su parte de implementación.

La parte pública de un objeto es accesible directamente por otros objetos. Por el contrario, la parte privada de un objeto no es accesible desde otros objetos.

La parte de implementación es la forma como se construye el objeto, para producir los resultados o proporcionar los servicios que los demás objetos esperan de él.

Así pues, de acuerdo con su "visibilidad" o grado de comunicación con el entorno, los atributos o características de una clase pueden ser de tres tipos:

- Públicas: un atributo público es visible tanto dentro, como fuera de la clase, es decir, puede ser utilizado y modificado por cualquier método de otra clase.
- Privadas: un atributo privado sólo será visible dentro de la clase, es decir, sólo sus métodos lo podrán utilizar. Normalmente, los atributos de una clase son privados, ya que, por ejemplo, si los atributos nombre, apellido y domicilio, de la clase empleado fuesen atributos públicos, un usuario podría modificarlos sin ningún control.
- Protegidas: un atributo protegido, al igual que los atributos privados, no será accesible desde fuera de la clase, con la excepción de las subclases derivadas.



Al igual que los atributos, las operaciones o métodos de una clase, pueden ser de tres tipos:

- Públicos: un método público será accesible tanto dentro, como fuera de la clase.
- Privados: un método privado sólo será accesible desde dentro de la clase, es decir, sólo podrá ser utilizado por otros métodos de la clase.
- Protegidos: un método protegido, al igual que los métodos privados, no será accesible desde fuera de la clase, con la excepción de las subclases derivadas.

Veamos un ejemplo:

- Consideremos el objeto Factura, el cual posee los atributos Número de Factura, Fecha de Facturación, Monto Total de la Factura y Fecha de Pago. Para este objeto se ha definido el método Asignar Fecha de Pago (determinar la fecha del día y asignarla como fecha de pago).
- Declarar como privado el atributo Fecha de Pago, permitirá asegurar que la asignación de valores a este atributo sea realizada únicamente con el método Asignar Fecha de Pago. Esto es, la asignación de valores al atributo Fecha de Pago será hecha de una sola forma, lo cual asegurará la consistencia de la información.
- El método Asignar Fecha de Pago, sin embargo, deberemos declararlo como método público, de tal forma que cualquier objeto que requiera la actualización de la Fecha de Pago, pueda hacerlo “invocando este método”.

4.5.- Importancia de la Abstracción

La abstracción es indudablemente uno de los aportes importantes de la orientación a objeto al desarrollo de software, pues establece una disciplina para el manejo de los objetos:

- El creador de la clase o persona que tiene la responsabilidad de su construcción e implementación, tiene acceso a todos los detalles de la clase y es el único que puede utilizar libremente los atributos y los métodos privados.
- El usuario de la clase, que instancia y utiliza objetos de la clase no necesita conocer la estructura interna de los objetos y

tampoco debe estar en capacidad de modificar directamente las variables del objeto instanciado, sin que el objeto controle dichas modificaciones.

- Los detalles de la implementación de una clase permanecen ocultos a los programadores que no han creado la clase. Estos programadores, de requerirlo, sólo la utilizarán, sin tener que preocuparse de la complejidad de la arquitectura del objeto.
- La calidad de los sistemas se fortalece, por cuanto los programadores que no han creado una clase no podrán realizar operaciones que puedan dañar la integridad del objeto.
- La implementación de un objeto puede evolucionar, sin necesidad de modificar la interfaz (su utilización) y sin que otros objetos se vean afectados.

5.- ¿Qué es UML?

Tal como señalábamos en el capítulo de introducción, en el tiempo han ido apareciendo diferentes metodologías OO, que pueden ser catalogadas como orientadas a objetos porque se basan en los conceptos de la orientación a objetos, como:

- Object-Oriented Design (OOD), Booch.
- Object Modeling Technique (OMT), Rumbaugh.
- Object Oriented Analysis (OOA), Coad/Yourdon.
- Hierarchical Object Oriented Design (HOOD), ESA.
- Object Oriented Structured Design (OOSD), Wasserman.
- Object Oriented Systems Analysis (OOSA), Shaler y Mellor.

En la actualidad, las metodologías más importantes para el análisis y diseño de sistemas han confluido en lo que se ha ido convirtiendo en un importante lenguaje estándar para el modelaje, denominado UML (Unified Modeling Lenguaje - Lenguaje Unificado de Modelaje), bajo el respaldo de la organización OMG (Object Management Group o Grupo de Administración de Objetos).

UML es, fundamentalmente, la creación de Grady Booch, James Rumbaugh e Ivar Jacobson, conocidos en el medio como “los tres amigos”, quienes habiendo desarrollado metodologías para desarrollo de software con la orientación a objetos, tuvieron la oportunidad de reunir esfuerzos y, como consecuencia de la aceptación general que encontró su propuesta, se pudo crear el Object Management Group, como una organización sin fines de lucro que reúne a la gran mayoría de las

empresas líderes en el mercado de la informática: IBM, HP, Sun Microsystems, Microsoft, Intellicorp, Oracle, Texas Instruments, Rational, etc.

UML, como arriba señaláramos, es un lenguaje de modelaje que permite especificar y documentar un sistema y sus componentes. Las herramientas de modelaje que componen UML permiten poner en "blanco y negro" la información sobre la estructura (elementos estáticos) y el comportamiento (elementos dinámicos) de un sistema.

UML no es ni una metodología, ni un lenguaje de programación; existen, sin embargo, diferentes productos que ofrecen generadores de código a partir de una especificación en UML, para una gran variedad de lenguajes de programación, como Java y C++.

UML es un lenguaje de modelaje visual, de propósito general, para el desarrollo de software orientado a objetos, que permite representar un sistema y sus componentes con la suficiente precisión como para desarrollar esos componentes (generar el código) y, a la vez, con la suficiente "independencia tecnológica" para permitir que el desarrollador utilice las herramientas de construcción que sean de su preferencia.

Como ya indicamos, desde su aparición, alrededor del desarrollo orientado a objetos se propusieron diversos métodos y técnicas, con muchos aspectos en común, pero con distintas notaciones, que dificultaban su aplicación o limitaban el espectro de herramientas y lenguajes de programación que los desarrolladores podían utilizar. El gran valor que tiene UML radica en la estandarización que facilita su uso universal, así como el desarrollo de herramientas con la capacidad de generar código a partir de una especificación UML, para las más diversas plataformas.

6.- Objetivos del UML

UML es un lenguaje de modelaje de propósito general, que pueden usar los ingenieros de sistemas en el cumplimiento de diversas actividades, que trata de ser lo más simple posible, dentro de la complejidad que representa la necesidad de modelar una gran gama de sistemas.

UML no es una metodología, no implica un "proceso de desarrollo paso a paso"; sin embargo, incluye los conceptos y técnicas necesarios para cumplir un proceso iterativo, orientado a definir una arquitectura integral, sobre la cual puedan desarrollarse componentes de software, esto es:

- Visualizar cómo es un sistema o cómo queremos que sea.
- Especificar el comportamiento o funcionamiento de un sistema.
- Especificar la estructura de un sistema.
- Desarrollar especificaciones que guíen la construcción de los sistemas.
- Documentar el diseño de un sistema.

En UML se incluyen:

1. Diferentes tipos de diagramas: de clases, de casos de uso, de interacción, de componentes, de distribución, de paquetes, de transición de estados, etc. Cada diagrama está concebido para "explicar" un aspecto particular de un sistema. Por ejemplo, un diagrama de clases permite ilustrar la estructura estática de un sistema (conceptos del negocio, relaciones, datos y operaciones).
2. Una sintaxis general o simbología que se aplica a cualquiera de los diagramas UML.

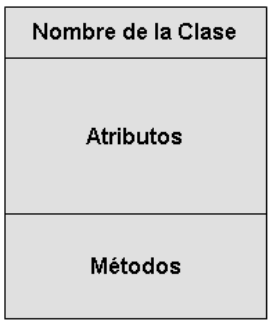
7.- El Lenguaje UML

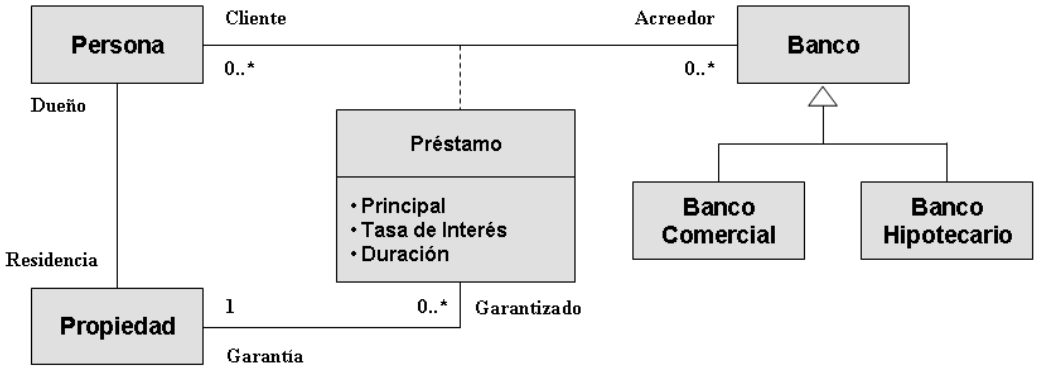
El lenguaje UML contiene todo un conjunto de elementos y reglas de sintaxis que, como ya señaláramos, permiten modelar sistemas desde su perspectiva conceptual, de funcionamiento o de estructura; o como lo expresa la OMG:

> "UML 2.0 define trece tipos de diagramas, divididos en tres categorías: seis tipos del diagrama representan la estructura estática, tres representan tipos generales de comportamiento y cuatro representan diversos aspectos de las interacciones. Los diagramas que permiten modelar la estructura incluyen los diagramas de clase, los diagramas de objeto, los diagramas de componente, los diagramas compuestos de la estructura, los diagramas de paquete y los diagramas del despliegue. Los diagramas que modelan el comportamiento incluyen: los diagramas de use case -utilizado por algunas metodologías durante el levantamiento de requerimientos- los diagramas de actividad y los diagramas de estado.
>
> Los diagramas de interacción incluyen los diagramas de secuencia, los diagramas de comunicación, los diagramas de tiempo y los diagramas de descripción general de interacción."

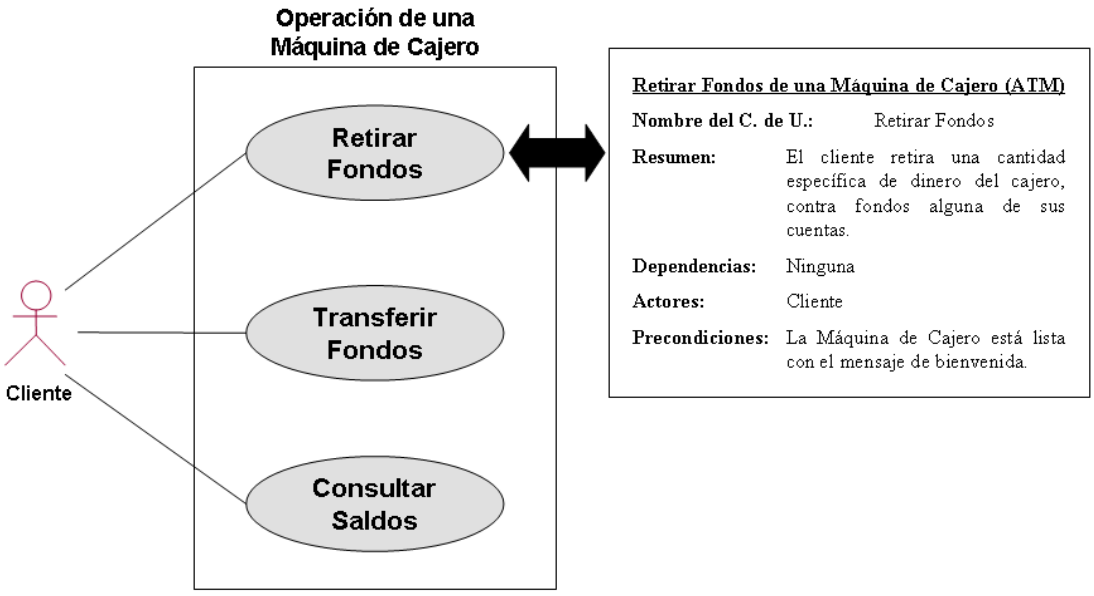
A continuación presentamos en forma general las herramientas de modelaje que conforman UML, que serán discutidas en detalle en los capítulos subsiguientes.

1. Los **diagramas de clase** permiten describir la estructura estática de un sistema.

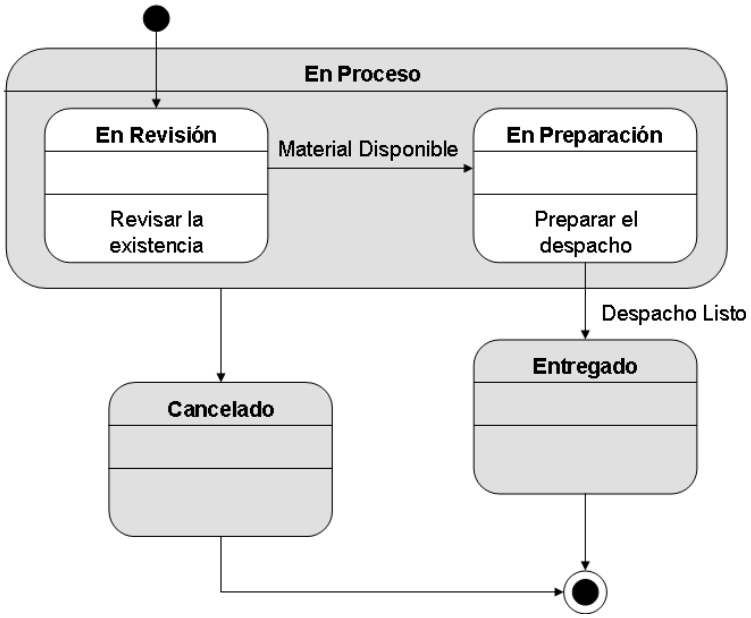




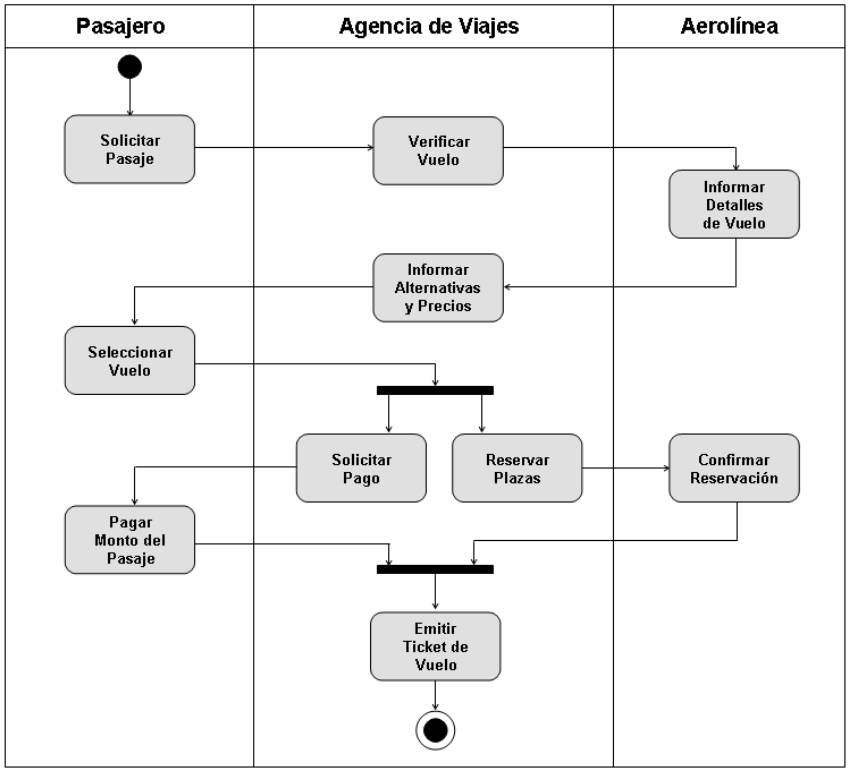
2. Los **diagramas de use case** representan el funcionamiento de un sistema, en términos de actividades y actores.



3. Los **diagramas de estado** describen el comportamiento dinámico de un sistema en respuesta a estímulos externos. Los diagramas de estado son especialmente útiles para modelar las transiciones de estado que sufren los objetos, como consecuencia de acontecimientos específicos.



4. Los **diagramas de actividad** son un caso particular de diagramas de estado y se utilizan para especificar o detallar la secuencia de pasos que se ejecutan en un método, o un use case o un flujo del negocio.



5. Los **diagramas de interacción** pueden ser de dos tipos: de colaboración y de secuencia.

   5.1. Los **diagramas de secuencia** describen interacciones entre clases, en términos de los mensajes que intercambian.

   5.2. Los **diagramas de colaboración** representan el intercambio de mensajes entre objetos.

Diagrama de secuencia:

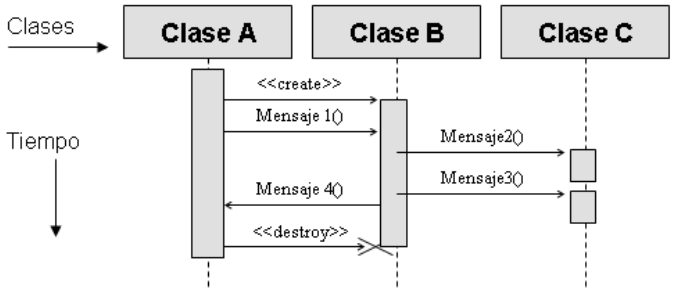
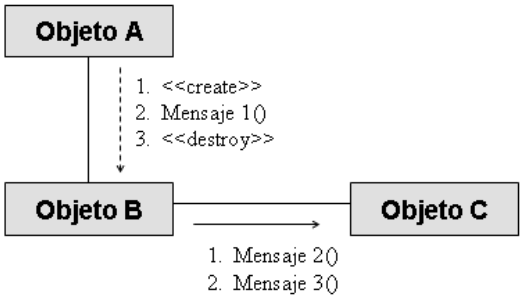
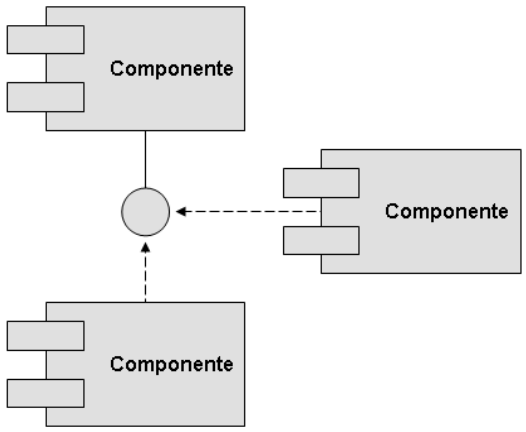


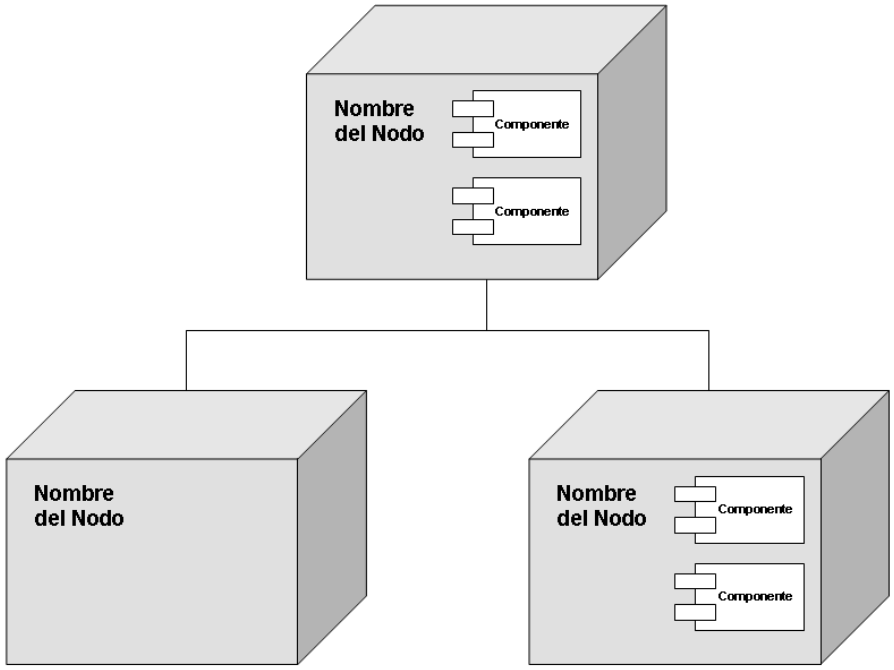
Diagrama de colaboración:



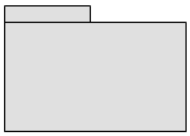
6. Los **diagramas componentes** describen la organización de los componentes de software.



7. Los **diagramas de despliegue** representan los recursos físicos en un sistema, incluyendo nodos, componentes, y conexiones.



8. Los **diagramas de paquete**, más que un tipo de diagramas, constituyen un mecanismo de agrupación con el cual se pueden mostrar las interrelaciones entre cualquier tipo de elemento: "un mecanismo de propósito general, para organizar elementos en grupos semánticamente relacionados".



8.- Vistas UML

En el capítulo II -Modelos de datos y procesos- señalábamos que un modelo es una descripción de una parte de un sistema y que dichas descripciones se realizan mediante un lenguaje especial -los diferentes diagramas- que utilizan una sintaxis específica. En ese capítulo definimos tres puntos de vista para modelar un sistema: el modelo conceptual –el qué-, el modelo de funcionamiento o de utilización –el cómo- y el modelo físico -los componentes-.

En este capítulo hemos revisado en forma muy general el lenguaje UML y hemos observado cómo este lenguaje se ha venido convirtiendo en un estándar de amplia utilización para especificar y documentar sistemas.

Como podrá observarse, en este libro hemos adoptado varias herramientas de UML para trazar lo que denominamos modelos conceptuales, modelos de funcionamiento y modelos físicos. Sin embargo, es importante que señalemos que esa no constituye la denominación estándar de UML, y hemos considerado oportuno aclarar que en la terminología de la OMG -object management group-, sólo se consideran dos grupos de modelos, por lo que los diferentes elementos o construcciones que integran el lenguaje UML se utilizan para representar un sistema desde los siguientes puntos de vista:

- La vista estructural o estática, compuesta por los diagramas de clase, los diagramas de componentes y los diagramas de despliegue.
- La vista de comportamiento o dinámica, integrada por los diagramas de use case, los diagramas de actividad, los diagramas de estado y los diagramas de interacción.

9.- Modelos y metamodelos

Adicionalmente, la organización OMG, creadora de UML, hace una distinción entre los modelos que representan los sistemas y el lenguaje utilizado para crear esos modelos, al cual denomina metamodelo. Así pues, dentro de la terminología de la OMG, UML es un metamodelo.

Para hacer las cosas un poco más generales -o complicadas- la OMG ha definido una estructura de lenguajes compuesta de cuatro niveles, que denomina M0, M1, M2 y M3:

- El nivel M0: las ocurrencias o instancias.

  El nivel M0 modela el sistema real y sus elementos son las instancias que componen dicho sistema. Ejemplos de dichos elementos son el empleado *Juan* que tiene una edad de *35 años*, pesa *80 Kg* y ocupa el cargo de *analista de sistemas* en la empresa *Informática, S.A.*

- El nivel M1: el modelo del sistema.

  Los elementos del nivel M1 son los modelos que hacemos para representar un sistema. En el nivel M1 se definen, por ejemplo, los conceptos de *Empleado*, *Cargo* y *Empresa*, cada uno con sus correspondientes atributos: nombre, descripción del cargo, nombre de la empresa, etc.

  Obviamente, existe una relación muy estrecha entre los niveles M0 y M1, pues los conceptos del nivel M1 clasifican los

elementos del nivel M0 y los elementos del nivel M0 son las ocurrencias o instancias de los elementos del nivel M1.

- El nivel M2: el modelo del modelo o metamodelo.

  Los elementos del nivel M2 son los lenguajes de modelaje. El nivel M2 define los elementos que se utilizarán para crear un modelo del nivel M1. En el caso de un modelo UML de un sistema, los conceptos propios del nivel M2 son *Clase*, *Atributo*, *Asociación*, etc. Al igual que ocurre con los niveles M0 y M1, existe una interrelación entre los conceptos de los niveles M1 y M2, pues los elementos del nivel superior definen las clases de elementos válidos que pueden utilizarse para crear un determinado modelo de nivel M1 y, a su vez, los elementos del nivel M1 pueden ser considerados como ocurrencias o instancias de los elementos del nivel M2.

- El nivel M3: el modelo de M2 (el meta-metamodelo).

  Finalmente, el nivel M3 define los elementos que se utilizan para crear lenguajes de modelaje. De esta forma, el concepto de *clase* definido en UML -que forma parte de los elementos del nivel M2- puede verse como una instancia del elemento de nivel M3, en el cual se define qué es una clase, junto con sus características y las relaciones con otros elementos -i.e., una clase es un clasificador que tiene asociado un comportamiento y dispone de un conjunto de atributos y de operaciones-.

La OMG, junto con estas definiciones, ha creado un lenguaje para describir los elementos del nivel M3, que se denomina MOF -Meta-Object Facility-. MOF puede considerarse como un lenguaje para describir lenguajes de modelado, como son el UML y el CWM -Common Warehouse Metamodel-, que pueden ser considerados instancias del MOF, puesto que MOF establece un lenguaje para describir meta-modelos de lenguajes de modelado; esto es, establece la sintaxis, los constructores y los mecanismos necesarios para describir dichos meta-modelos; es decir, establece los elementos que pueden ser utilizados para constituir un lenguaje particular, y las relaciones entre tales elementos.

Dicho con pocas palabras, la sintaxis de UML viene descrita por su meta-modelo, que ha sido creada con el lenguaje MOF y, si quisiéramos definir un lenguaje diferente a UML, también podríamos expresarlo en MOF.

Más allá de lo rebuscado que nos parezca esta manera de concebir los lenguajes de modelaje, debemos reconocer que constituye un esfuerzo

importante para estandarizar estos aspectos, lo cual permite establecer una base común sobre la que puedan integrarse los diferentes productores de herramientas para el desarrollo de software.

10.- Extensiones

Muchos ingenieros de sistemas encuentran que UML es un lenguaje de carácter general y en muchos casos requiere ajustes o adiciones para poder modelar aspectos o conceptos de algún dominio o área particular de problemas, como: actividades bancarias, telecomunicaciones, transporte aéreo, análisis y diseño de sistemas en tiempo real, prueba, etc.

Como vimos en el punto anterior, utilizando la sintaxis de MOF podríamos desarrollar un nuevo metamodelo –lenguaje- que incluya o incorpore facilidades adaptadas a las necesidades de un área de problemas particular; sin embargo, para no tener que llegar a estos extremos, UML provee la facilidad de incluir extensiones, de tal forma que puedan especializarse algunos de sus conceptos y restringir otros, dentro de la semántica original de los elementos de UML (clases, asociaciones, atributos, operaciones, transiciones, etc.).

La primera opción es la que siguió la misma OMG para desarrollar lenguajes como CWM –common warehouse metamodel-, puesto que la semántica de algunos de sus constructores no coincide con la de UML. Por otro lado, hay situaciones en las que es suficiente con extender el lenguaje UML utilizando una serie de mecanismos recogidos en lo que se denominan Perfiles de UML -UML Profiles-.

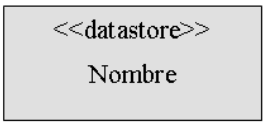
En caso de que no sea necesario crear un nuevo metamodelo –otro lenguaje similar o equivalente a UML- podemos particularizar algunos de sus conceptos utilizando lo que en UML se denomina mecanismos de extensión. La versión 2.0 de UML incluye varios mecanismos para extender y adaptar las metaclases de UML a las necesidades concretas de una plataforma -como puede ser .NET o J2EE- o de un dominio de aplicación -tiempo real, modelado de procesos de negocio, etc.-.

Hoy día, existe una gran cantidad de perfiles UML, algunos de los cuales han sido estandarizados por la OMG, como son: los perfiles UML para CORBA y para CCM -CORBA Component Model-, el perfil UML para EDOC -Enterprise Distributed Object Computing-, el perfil UML para EAI -Enterprise Application Integration- y el perfil UML para programación, desempeño y tiempo -Scheduling, Performance, and Time-, perfil de UML para modelaje del negocio.

Muchas empresas fabricantes de lenguajes de programación y herramientas han creado sus propios perfiles UML, que están disponibles de forma pública y son de amplia utilización, como son: el UML/EJB Mapping Specification, que ha sido definido por JCP -Java Community Process- y los perfiles UML para determinados lenguajes de programación, como Java o C#.

Los mecanismos de extensión definidos dentro de UML son: las restricciones, los estereotipos y los valores etiquetados.

Un estereotipo es una tipificación que puede hacerse sobre algún elemento UML, para extender sus propiedades, como se muestra en la figura:



Un valor etiquetado es una propiedad adicional que se añade a un elemento UML, con el fin de poder mostrar información adicional.

Una restricción permite establecer reglas de composición adicionales a las que establece UML

11.- Model driven architecture

La Arquitectura Centrada en Modelos (MDA - Model Driven Architecture) es el siguiente paso de la organización OMG en la resolución de problemas de integración a través de especificaciones abiertas e independientes de los proveedores.

MDA busca lograr la integración y la interoperatividad de todos los componentes, a través del ciclo de vida de los sistemas, desde el modelaje del negocio y las tareas de diseño, hasta la construcción, integración, prueba, implantación y mantenimiento.

Dentro de su marco conceptual, la MDA incluye dos modelos perfectamente diferenciados: Modelos Independientes de la Plataforma (Platform Independent Model - PIM) y Modelos Para Plataforma Específica (Platform Specific Model - PSM), de tal forma que la funcionalidad que se especifique en un PIM, sin tomar en cuenta las particularidades de la plataforma tecnológica en la que se desarrollará el sistema, pueda ser transformada o traducida a una PSM, adecuada a esa plataforma, con el fin de cumplir las tareas de desarrollo en forma simple.

Tanto los PIM, como los PSM, se modelan haciendo uso de Herramientas UML, ampliando el lenguaje básico con extensiones que

permitan añadir la semántica que pueda ser necesaria en cada caso o tipo de plataforma. Las transformaciones de modelos (PIM a PSM) pueden ser totalmente manuales, pero también pueden automatizarse, utilizando herramientas que se basen en estos estándares.

Entre los diferentes modelos construidos en MDA existe una estrecha relación. Los modelos más abstractos son la base para la construcción de los modelos específicos así como los modelos específicos son los que soportan los modelos de un nivel de abstracción mayor. Esta relación es representada en esta arquitectura por las Transformaciones entre Modelos, las cuales constituyen una de las características fundamentales de esta propuesta. Los procesos de transformación se denominan "mappings" y están conformados por una serie de reglas de transformación que son operativas en un determinado dominio y permiten pasar de un modelo a otro.

Uno de los factores claves para la acogida de MDA por parte de la comunidad informática es su énfasis en los modelos. La especificación MDA sugiere la utilización de UML (Unified Modelling Language) para la creación de modelos, MOF (Meta-Object Facility) para la creación de metamodelos y QVT (Query/View/Transformation) para la especificación de las transformaciones; sin embargo no es obligatorio el uso de estos lenguajes para caso.

En el Website de la OMG (http://www.omg.org/) puede encontrarse abundante información sobre facilidades de software que se ofrecen en el mercado, para apoyar las tareas de modelaje con UML. En cierta forma, la MDA es un nuevo intento, como fue el IBM AD/Cycle en los años 80, de establecer estándares para la creación de herramientas tipo CASE, que permitan automatizar la generación de programas; con la diferencia que, hoy día, se dispone de infraestructuras mucho más versátiles y, sobre la base de las experiencias pasadas, se plantean objetivos más realistas.

Como resumen, podemos decir que la MDA es un enfoque para el desarrollo de sistemas, basado en modelos UML, que ha sido concebido para elevar el nivel de abstracción en el proceso de desarrollo de sistemas al permitir que las actividades se concentren más en la elaboración de modelos y menos en la codificación de los programas. La propuesta de MDA también viene a resolver los problemas comunes de integración e interoperabilidad de sistemas en plataformas disímiles, mediante el uso de los dos niveles de modelos, en lugar de resolverlos a nivel de plataforma con el uso de componentes de "middleware".