

UML

Unified Modeling Language

UML [UML] es un *lenguaje para especificar, construir, visualizar y documentar los artefactos de un sistema de software orientado a objetos (OO)*. Un artefacto es una información que es utilizada o producida mediante un proceso de desarrollo de software.

UML se quiere convertir en un *lenguaje estándar* con el que sea posible modelar todos los componentes del proceso de desarrollo de aplicaciones. Sin embargo, hay que tener en cuenta un aspecto importante del modelo: no pretende definir un modelo estándar de desarrollo, sino únicamente un lenguaje de modelado. Otros métodos de modelaje como OMT (*Object Modeling Technique*) o Booch sí definen procesos concretos. En UML los procesos de desarrollo son diferentes según los distintos dominios de trabajo; no puede ser el mismo el proceso para crear una aplicación en tiempo real, que el proceso de desarrollo de una aplicación orientada a gestión, por poner un ejemplo. Las diferencias son muy marcadas y afectan a todas las facetas del proceso. El método del UML recomienda utilizar los procesos que otras metodologías tienen definidos.

Los Inicios

A partir del año 1994, Grady Booch [Booch96](precursor de Booch '93) y Jim Rumbaugh (creador de OMT) se unen en una empresa común, *Rational Software Corporation*, y comienzan a unificar sus dos métodos. Un año más tarde, en octubre de 1995, aparece UML (*Unified Modeling Language*) 0.8, la que se considera como la primera versión del UML. A finales de ese mismo año, Ivar Jacobson, creador de OOSE (*Object Oriented Software Engineer*) se añade al grupo.

GRADY BOOCH JIM RUMBAUGH IVAR JACOBSON



Como objetivos principales de la consecución de un nuevo método que aunara los mejores aspectos de sus predecesores, sus protagonistas se propusieron lo siguiente:

- El método debía ser capaz de modelar no sólo sistemas de software sino otro tipo de sistemas reales de la empresa, siempre utilizando los conceptos de la orientación a objetos (OO).
- Crear un lenguaje para modelado utilizable a la vez por máquinas y por personas.
- Establecer un acoplamiento explícito de los conceptos y los *artefactos* ejecutables.
- Manejar los problemas típicos de los sistemas complejos de misión crítica.

Lo que se intenta es lograr con esto que los lenguajes que se aplican siguiendo los métodos más utilizados sigan evolucionando en conjunto y no por separado. Y además, unificar las perspectivas entre diferentes tipos de sistemas (no sólo software, sino también en el ámbito de los negocios), al aclarar las fases de desarrollo, los requerimientos de análisis, el diseño, la implementación y los conceptos internos de la OO.

Modelado de objetos

En la especificación del UML podemos comprobar que una de las partes que lo componen es un metamodelo formal. Un *metamodelo* es un modelo que define el lenguaje para expresar otros modelos. Un modelo en OO es una *abstracción cerrada semánticamente de un sistema y un sistema es una colección de unidades conectadas que son organizadas para realizar un propósito específico*. Un sistema puede ser descrito por uno o más modelos, posiblemente desde distintos puntos de vista.

Una parte del UML define, entonces, una abstracción con significado de un lenguaje para expresar otros modelos (es decir, otras abstracciones de un sistema, o conjunto de unidades conectadas que se organizan para conseguir un propósito). Lo que en principio puede parecer complicado no lo es tanto si pensamos que uno de los objetivos del UML es llegar a convertirse en una manera de definir modelos, no sólo establecer una forma de modelo, de esta forma simplemente estamos diciendo que UML, además, define un lenguaje con el que podemos abstraer cualquier tipo de modelo.

El UML es una técnica de modelado de objetos y como tal supone una abstracción de un sistema para llegar a construirlo en términos concretos. El modelado no es más que la construcción de un modelo a partir de una especificación. Un modelo es una abstracción de algo, que se elabora para comprender ese algo antes de construirlo. El modelo omite detalles que no resultan esenciales para la comprensión del original y por lo tanto facilita dicha comprensión.

Los modelos se utilizan en muchas actividades de la vida humana: antes de construir una casa el arquitecto utiliza un plano, los músicos representan la música en forma de notas musicales, los artistas pintan sobre el lienzo con caboncillos antes de empezar a utilizar los óleos, etc. Unos y otros abstraen una realidad compleja sobre unos bocetos, modelos al fin y al cabo. La OMT, por ejemplo, intenta abstraer la realidad utilizando tres clases de modelos OO: el *modelo de objetos*, que describe la estructura estática; el *modelo dinámico*, con el que describe las relaciones temporales entre objetos; y el *modelo funcional* que describe las relaciones funcionales entre valores. Mediante estas tres fases de construcción de un modelos, se consigue una abstracción de la realidad que tiene en sí misma información sobre las principales características de ésta.

Los modelos además, al no ser una representación que incluya todos los detalles de los originales, permiten probar más fácilmente los sistemas que modelan y determinar los errores. Según se indica en la *Metodología OMT* (Rumbaugh), los modelos permiten una mejor comunicación con el cliente por distintas razones:

- Es posible enseñar al cliente una posible aproximación de lo que será el producto final.
- Proporcionan una primera aproximación al problema que permite visualizar cómo quedará el resultado.
- Reducen la complejidad del original en subconjuntos que son fácilmente tratables por separado.

Se consigue un modelo completo de la realidad cuando el modelo captura los aspectos importantes del problema y omite el resto. Los lenguajes de programación que estamos acostumbrados a utilizar no son adecuados para realizar modelos completos de sistemas reales porque necesitan una especificación total con detalles que no son importantes para el algoritmo que están implementando. En OMT se modela un sistema desde tres puntos de vista diferentes donde cada uno representa una parte del sistema y una unión lo describe de forma completa. En esta técnica de modelado se utilizó una aproximación al proceso de implementación de software habitual donde se utilizan estructuras de datos (modelo de objetos), las operaciones que se realizan con ellos tienen una secuencia en el tiempo (modelo dinámico) y se realiza una transformación sobre sus valores (modelo funcional).

UML utiliza parte de este planteamiento obteniendo distintos puntos de vista de la realidad que modela mediante los distintos tipos de diagramas que posee. Con la creación del UML se persigue obtener un lenguaje que sea capaz de abstraer cualquier tipo de sistema, sea informático o no, mediante los diagramas, es decir,

mediante representaciones gráficas que contienen toda la información relevante del sistema. Un **diagrama** es *una representación gráfica de una colección de elementos del modelo, que habitualmente toma forma de grafo donde los arcos que conectan sus vértices son las relaciones entre los objetos y los vértices se corresponden con los elementos del modelo*. Los distintos puntos de vista de un sistema real que se quieren representar para obtener el modelo se dibujan de forma que se resalten los detalles necesarios para entender el sistema.

Artefactos para el Desarrollo de Proyectos

Un artefacto es una información que es utilizada o producida mediante un proceso de desarrollo de software. Pueden ser artefactos un modelo, una descripción o un software. Los artefactos de UML se especifican en forma de diagramas, éstos, junto con la documentación sobre el sistema constituyen los artefactos principales que el modelador puede observar.

Se necesita más de un punto de vista para llegar a representar un sistema. UML utiliza los diagramas gráficos para obtener estos distintos puntos de vista de un sistema:

- Diagramas de Casos de Uso.

Un **casos de uso** es *una* secuencia de transacciones que son desarrolladas por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema. Los diagramas de casos de uso sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y/o otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Una relación es una conexión entre los elementos del modelo, por ejemplo la relación y la generalización son relaciones.

Los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema al mostrar como reacciona una respuesta a eventos que se producen en el mismo. En este tipo de diagrama intervienen algunos conceptos nuevos: un *actor* es una entidad externa al sistema que se modela y que puede interactuar con él; un ejemplo de actor podría ser un usuario o cualquier otro sistema. Las relaciones entre casos de uso y actores pueden ser las siguientes:

- Un actor se comunica con un caso de uso.
- Un caso de uso extiende otro caso de uso.
- Un caso de uso usa otro caso de uso.

- Diagramas de Clases.

Los **diagramas de clases** representan un conjunto de elementos del modelo que son estáticos, como las clases y los tipos, sus contenidos y las relaciones que se establecen entre ellos.

Algunos de los elementos que se pueden clasificar como estáticos son los siguientes:

Paquete: Es el mecanismo de que dispone UML para organizar sus elementos en grupos, se representa un grupo de elementos del modelo. Un sistema es un único paquete que contiene el resto del sistema, por lo tanto, un paquete debe poder anidarse, permitiéndose que un paquete contenga otro paquete.

Clases: Una clase representa un conjunto de objetos que tienen una estructura, un comportamiento y unas relaciones con propiedades parecidas. Describe un conjunto de objetos que comparte los mismos atributos, operaciones, métodos, relaciones y significado. En UML una clase es una implementación de un tipo. Los componentes de una clase son:

Atributo. Se corresponde con las propiedades de una clase o un tipo. Se identifica mediante un nombre. Existen atributos simples y complejos.

Operación. También conocido como método, es un servicio proporcionado por la clase que puede ser solicitado por otras clases y que produce un comportamiento en ellas cuando se realiza.

Las clases pueden tener varios parámetros formales, son las clases denominadas plantillas. Sus atributos y operaciones vendrán definidas según sus parámetros formales. Las plantillas pueden tener especificados los valores reales para los parámetros formales, entonces reciben el nombre de *clase parametrizada instanciada*. Se puede usar en cualquier lugar en el que se podría aparecer su plantilla.

Relacionando con las clase nos encontramos con el término utilidad, que se corresponde con una agrupación de variables y procedimientos globales en forma de declaración de clase, también puede definirse como un estereotipo (o nueva clase generada a partir de otra ya existente) de un tipo que agrupa variables globales y procedimientos en una declaración de clase. Los atributos y operaciones que se agrupan en una utilidad se convierten en variables y operaciones globales. Una utilidad no es fundamental para el modelado, pero puede ser conveniente durante la programación.

Metaclase: Es una clase cuyas instancias son clases. Sirven como depósito para mantener las variables de clase y proporcionan operaciones (método de clase) para inicializar estas variables. Se utilizan para construir metamodelos (modelos que se utilizan para definir otros modelos).

Tipos: Es un descriptor de objetos que tiene un estado abstracto y especificaciones de operaciones pero no su implementación. Un tipo establece una especificación de comportamiento para las clases.

Interfaz: Representa el uso de un tipo para describir el comportamiento visible externamente de cualquier elemento del modelo.

Relación entre clases: Las clases se relacionan entre sí de distintas formas, que marcan los tipos de relaciones existentes:

Asociación:

Es una relación que describe un conjunto de vínculos entre clases. Pueden ser binarias o n-arias, según se implican a dos clases o más. Las relaciones de asociación vienen identificadas por los *roles*, que son los nombres que indican el comportamiento que tienen los tipos o las clases, en el caso del rol de asociación (existen otros tipos de roles según la relación a la que identifiquen). Indican la información más importante de las asociaciones. Es posible indicar el número de instancias de una clase que participan en una relación mediante la llamada *multiplicidad*. Cuando la multiplicidad de un rol es mayor que 1, el conjunto de elementos que se relacionan pueden estar ordenado. Las relaciones de asociación permiten especificar qué objetos van a estar asociados con otro objeto mediante un calificador. El calificador es un atributo o conjunto de atributos de una asociación que determina los valores que indican cuales son los valores que se asociarán.

Una asociación se dirige desde una clase a otra (o un objeto a otro), el concepto de *navegabilidad* se refiere al sentido en el que se recorre la asociación.

Existe una forma especial de asociación, la agregación, que especifica una relación entre las clases donde el llamado "agregado" indica el todo y el "componente" es una parte del mismo.

Composición:

Es un tipo de agregación donde la relación de posesión es tan fuerte como para marcar otro tipo de relación.

Las clases en UML tienen un tiempo de vida determinado, en las relaciones de composición, el tiempo de vida de la clase que es parte del todo (o agregado) viene determinado por el tiempo de vida de la clase que representa el todo, por tanto es equivalente a un atributo, aunque no lo es porque es una clase y puede funcionar como tal en otros casos.

Generalización:

Cuando se establece una relación de este tipo entre dos clases, una es una Superclase y la otra es una Subclase. La subclase comparte la estructura y el comportamiento de la superclase. Puede haber más de una clase que se comporte como subclase.

Dependencia:

Una relación de dependencia se establece entre clases (u objetos) cuando un cambio en el elemento independiente del modelo puede requerir un cambio en el elemento dependiente.

Relación de Refinamiento:

Es una relación entre dos elementos donde uno de ellos especifica de forma completa al otro que ya ha sido especificado con cierto detalle.

- Diagramas de Comportamiento o Interacción.

Muestran las interacciones entre objetos ocurridas en un escenario (parte) del sistema. Hay varios tipos:

Diagrama de secuencia

Muestran las interacciones entre un conjunto de objetos, ordenadas según el tiempo en que tienen lugar. En los diagramas de este tipo intervienen objetos, que tienen un significado parecido al de los objetos representados en los diagramas de colaboración, es decir son instancias concretas de una clase que participa en la interacción. El objeto puede existir sólo durante la ejecución de la interacción, se puede crear o puede ser destruido durante la ejecución de la interacción. Un diagrama de secuencia representa una forma de indicar el período durante el que un objeto está desarrollando una acción directamente o a través de un procedimiento.

En este tipo de diagramas también intervienen los mensajes, que son la forma en que se comunican los objetos: el objeto origen solicita (llama a) una operación del objeto destino. Existen distintos tipos de mensajes según cómo se producen en el tiempo: simples, síncronos, y asíncronos.

Los diagrama de secuencia permiten indicar cuál es el momento en el que se envía o se completa un mensaje mediante el *tiempo de transición*, que se especifica en el diagrama.

Diagrama de colaboración

Muestra la interacción entre varios objetos y los enlaces que existen entre ellos. Representa las interacciones entre objetos organizadas alrededor de los objetos y sus vinculaciones. A diferencia de un diagrama de secuencias, un diagrama de colaboraciones muestra las relaciones entre los objetos, no la secuencia en el tiempo en que se producen los mensajes. Los diagramas de secuencias y los diagramas de colaboraciones expresan información similar, pero en una forma diferente.

Formando parte de los diagramas de colaboración nos encontramos con objetos, enlaces y mensajes. Un *objeto* es una instancia de una clase que participa como una interacción, existen objetos simples y complejos. Un objeto es activo si posee un *thread* o hilo de control y es capaz de iniciar la actividad de control, mientras

que un objeto es pasivo si mantiene datos pero no inicia la actividad.

Un *enlace* es una instancia de una asociación que conecta dos objetos de un diagrama de colaboración. El enlace puede ser reflexivo si conecta a un elemento consigo mismo. La existencia de un enlace entre dos objetos indica que puede existir un intercambio de mensajes entre los objetos conectados.

Los diagramas de interacción indican el *flujo de mensajes* entre elementos del modelo, el flujo de mensajes representa el envío de un mensaje desde un objeto a otro si entre ellos existe un enlace. Los mensajes que se envían entre objetos pueden ser de distintos tipos, también según como se producen en el tiempo; existen mensajes simples, sincrónicos, *balking*, *timeout* y asíncronos.

Durante la ejecución de un diagrama de colaboración se crean y destruyen objetos y enlaces.

Diagramas de actividad

Son similares a los diagramas de flujo de otras metodologías OO. En realidad se corresponden con un caso especial de los diagramas de estado donde los estados son *estados de acción* (estados con una acción interna y una o más transiciones que suceden al finalizar esta acción, o lo que es lo mismo, un paso en la ejecución de lo que será un procedimiento) y las transiciones vienen provocadas por la finalización de las acciones que tienen lugar en los estados de origen. Siempre van unidos a una clase o a la implementación de un caso de uso o de un método (que tiene el mismo significado que en cualquier otra metodología OO). Los diagramas de actividad se utilizan para mostrar el flujo de operaciones que se desencadenan en un procedimiento interno del sistema.

Diagramas de estado

Representan la secuencia de estados por los que un objeto o una interacción entre objetos pasa durante su tiempo de vida en respuesta a estímulos (eventos) recibidos. Representa lo que podemos denominar en conjunto una máquina de estados. Un *estado* en UML es cuando un objeto o una interacción satisface una condición, desarrolla alguna acción o se encuentra esperando un evento.

Cuando un objeto o una interacción pasa de un estado a otro por la ocurrencia de un evento se dice que ha sufrido una *transición*, existen varios tipos de transiciones entre objetos: simples (normales y reflexivas) y complejas. Además una transición puede ser interna si el estado del que parte el objeto o interacción es el mismo que al que llega, no se provoca un cambio de estado y se representan dentro del estado, no de la transición. Como en todas las metodologías OO se envían mensajes, en este caso es la acción de la que puede enviar mensajes a uno o varios objetos destino.

- Diagramas de Implementación.

Se derivan de los diagramas de proceso y módulos de la metodología de Booch, aunque presentan algunas modificaciones. Los diagramas de implementación muestran los aspectos físicos del sistema. Incluyen la estructura del código fuente y la implementación, en tiempo de implementación. Existen dos tipos:

Diagramas de componentes

Muestra la dependencia entre los distintos componentes de software, incluyendo componentes de código fuente, binario y ejecutable. Un componente es un fragmento de código software (un fuente, binario o ejecutable) que se utiliza para mostrar dependencias en tiempo de compilación.

Diagrama de plataformas o despliegue

Muestra la configuración de los componentes hardware, los procesos, los elementos de procesamiento en tiempo de ejecución y los objetos que existen en tiempo de ejecución. En este tipo de diagramas intervienen nodos, asociaciones de comunicación, componentes dentro de los nodos y objetos que se encuentran a su vez dentro de los componentes. Un *nodo* es un objeto físico en tiempo de ejecución, es decir una máquina que se compone habitualmente de, por lo menos, memoria y capacidad de procesamiento, a su vez puede estar formado por otros componentes.

Nuevas características del UML

Además de los conceptos extraídos de métodos anteriores, se han añadido otros nuevos que vienen a suplir carencias antiguas de la metodología de modelado. Estos nuevos conceptos son los siguientes:

- Definición de estereotipos: un estereotipo es una nueva clase de elemento de modelado que debe basarse en ciertas clases ya existentes en el metamodelo y constituye un mecanismo de extensión del modelo.
- Responsabilidades.
- Mecanismos de extensibilidad: estereotipos, valores etiquetados y restricciones.
- Tareas y procesos.
- Distribución y concurrencia (para modelar por ejemplo ActiveX/DCOM y CORBA).
- Patrones/Colaboraciones.
- Diagramas de actividad (para reingeniería de proceso de negocios)
- Clara separación de tipo, clase e instancia.
- Refinamiento (para manejar relaciones entre niveles de abstracción).
- Interfaces y componentes.

El Proceso de Desarrollo

UML no define un proceso concreto que determine las fases de desarrollo de un sistema, las empresas pueden utilizar UML como el lenguaje para definir sus propios procesos y lo único que tendrán en común con otras organizaciones que utilicen UML serán los tipos de diagramas.

UML es un método independiente del proceso. Los procesos de desarrollo deben ser definidos dentro del contexto donde se van a implementar los sistemas.

Herramientas CASE

Rational Rose es la herramienta CASE que comercializan los desarrolladores de UML y que soporta de forma completa la especificación del UML 1.1.



Esta herramienta propone la utilización de cuatro tipos de modelo para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, uno lógico y otro físico. Permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software.

Desarrollo Iterativo

Rational Rose utiliza un proceso de desarrollo iterativo controlado (*controlled iterative process development*), donde el desarrollo se lleva a cabo en una secuencia de iteraciones. Cada iteración comienza con una primera aproximación del análisis, diseño e implementación para identificar los riesgos del diseño, los cuales se utilizan para conducir la iteración, primero se identifican los riesgos y después se prueba la aplicación para que éstos se hagan mínimos.

Cuando la implementación pasa todas las pruebas que se determinan en el proceso, ésta se revisa y se añaden los elementos modificados al modelo de análisis y diseño. Una vez que la actualización del modelo se ha modificado, se realiza la siguiente iteración.

Trabajo en Grupo

Rose permite que haya varias personas trabajando a la vez en el proceso iterativo controlado, para ello posibilita que cada desarrollador opere en un espacio de trabajo privado que contiene el modelo completo y tenga un control exclusivo sobre la propagación de los cambios en ese espacio de trabajo.

También es posible descomponer el modelo en unidades controladas e integrarlas con un sistema para realizar el control de proyectos que permite mantener la integridad de dichas unidades.

Generador de Código

Se puede generar código en distintos lenguajes de programación a partir de un diseño en UML.

Ingeniería Inversa

Rational Rose proporciona mecanismos para realizar la denominada Ingeniería Inversa, es decir, a partir del código de un programa, se puede obtener información sobre su diseño.

Bibliografía

Booch96	Booch, Grady. 1996. <i>Análisis y Diseño Orientado a Objetos</i> . 2da edición. Ed. Addison-Wesley / Díaz de Santos.
Pressman98	Pressman, Robert. 1998. <i>Ingeniería de Software</i> .
Pfigueroa	http://agamenon.uniandes.edu.co/~pfigueroa/soo/uml
UML	http://www.rational.com/uml/