INGENIERÍA DE SOFTWARE AVANZADA

(SESIÓN 5)

2.2 UML

2.3 RUP

2.4 MDA

Objetivo: Conocer las principales características de cada uno de los acrónimos de los subtemas arriba citados.

Comencemos por describir en un breve párrafo a cada uno de los acrónimos y más adelante iremos desarrollando el tema de manera individual:

2.2 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.

2.3 RUP: El **Proceso Unificado Racional** (*Rational Unified Process* en inglés, habitualmente resumido como **RUP**) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

2.4 MDA: Es el acrónimo de Model Driven Architecture, en español Arquitectura dirigida por modelos.

Podemos definir a MDA como un marco de trabajo que ha definido el Object Management Group (OMG).

El OMG es una organización de compañías de sistemas de información creada en 1990 con el fin de potenciar el desarrollo de aplicaciones orientadas a objetos distribuidas. Esta organización ha definido estándares importantes como UML, CORBA, MOF, entre otros. Su sitio Web es http://www.omg.org.

Escrito lo anterior entremos en detalle de cada uno de los acrónimos descritos brevemente.

2.2 UML

Se recomienda leer http://www.uv.mx/personal/maymendez/files/2011/05/umlTotal.pdf

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 faces del proceso. El método del UML recomienda utilizar los procesos que otras metodologías tienen definidos.

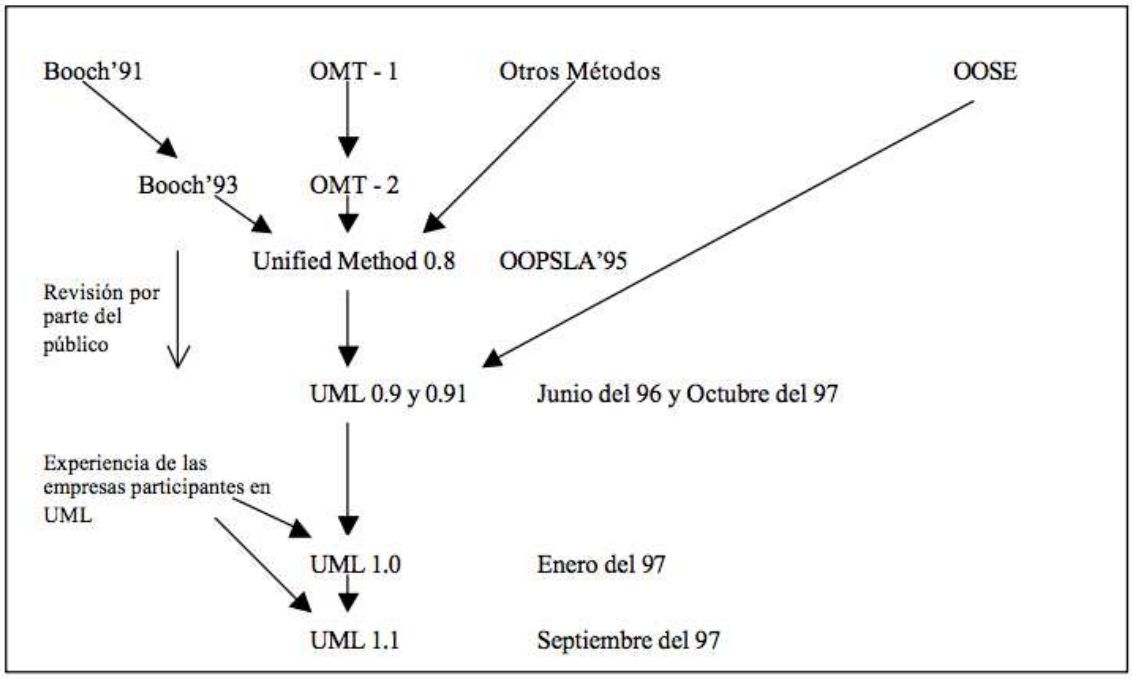


Figura 2 Historia de UML

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, Ivan Jacobson, creador de OOSE (Object Oriented Software Engineer) se añade al grupo.

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 descripto 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 carboncillos 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 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

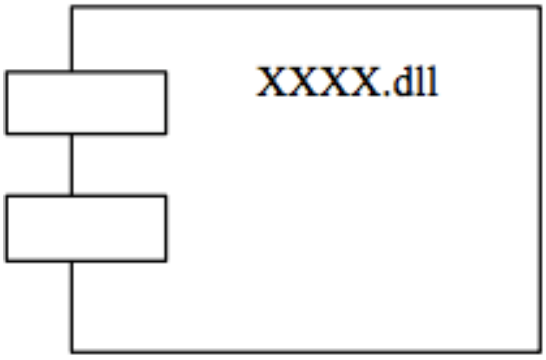
software habitual donde se utilizan estructuras de datos (modelo de objetos), las operaciones que se realizan con ellos tienen una secuencia en el tiempo (modelo dinámico) y se realiza una transformación sobre sus valores (modelo funcional).
UML utiliza parte de este planteamiento obteniendo distintos 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 dibuja dé forma que se resaltan 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 Implementación.
•□Diagramas de Comportamiento o Interacción.
•□Diagramas de Casos de uso.
•□Diagramas de Clases.
Ejemplo de algunos de los diagramas que utiliza UML.
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

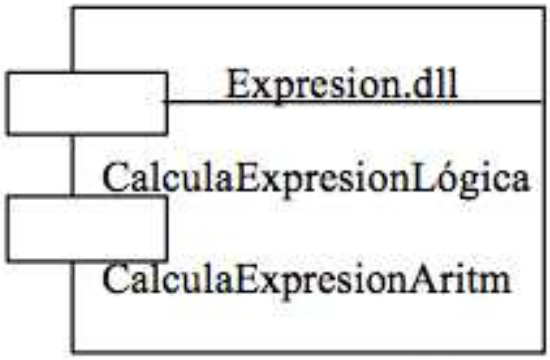
•Diagrama de plataformas despliegue

4. 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.



Representación de un componente



Representación extendida de un componente

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.

Diagramas de Interacción o Comportamiento

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

Diagrama de secuencia. Diagrama de colaboración. Diagrama de estado. Diagrama de actividad.

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 diagramas 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

INGENIERÍA DE SOFTWARE AVANZADA MIS 410

Diagramas de Casos de Uso

Unos 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.

DIFERENCIAS	
Las Clases	Los Componentes
Representan abstracciones lógicas	Representan elementos físicos
Es decir los componentes pueden estar en nodos y las clases no	
Pueden tener atributos y operacionesdirectamente accesibles.	Sólo tienen operaciones y estas son alcanzables a través de la interfaz del componente.

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 clases 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 puede 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 él 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.

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

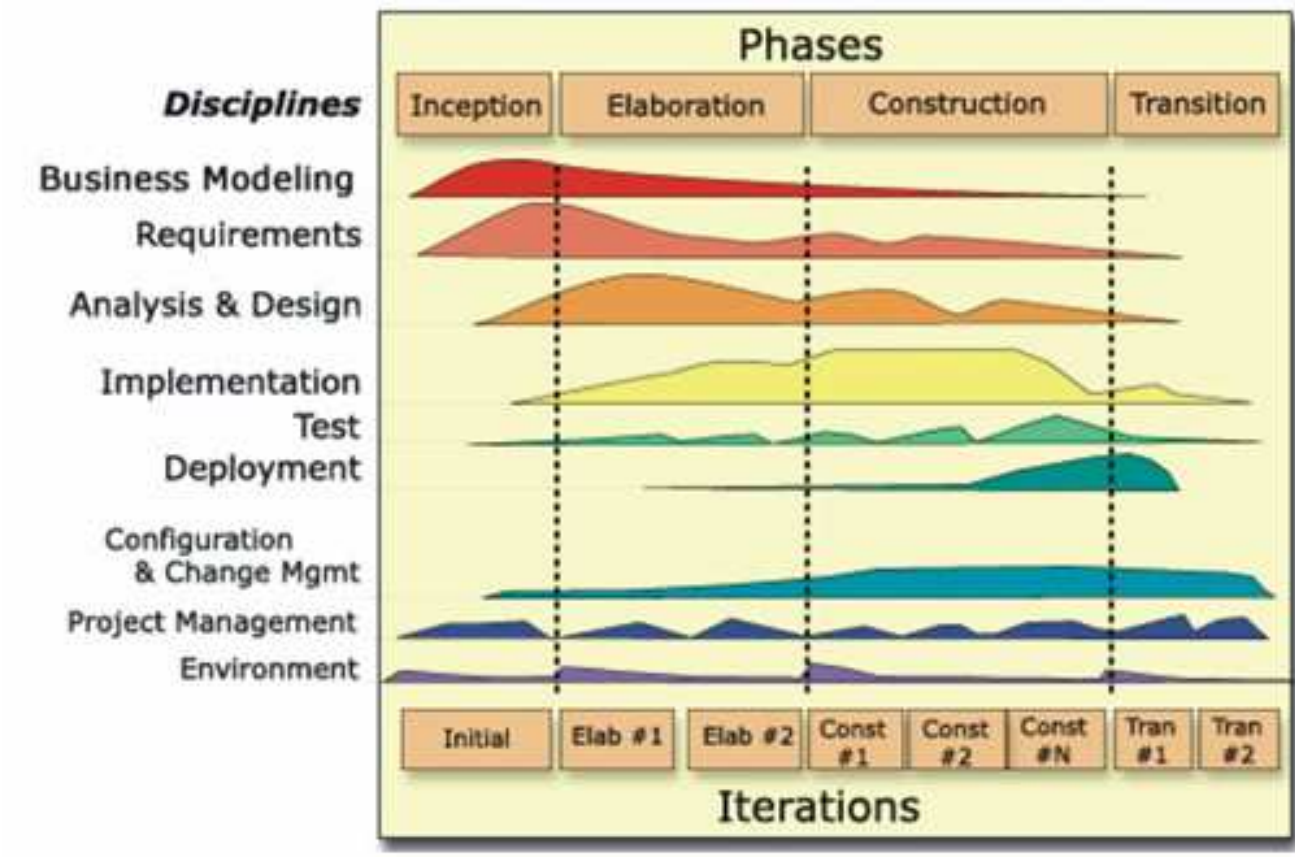
Rational Rose proporciona mecanismos para realizar la denominada Ingeniería Inversa, es decir, a partir del código de un programa, se puede obtener información sobre su diseño.

2.3 RUP

Fuente: http://www.utvm.edu.mx/OrganoInformativo/orgJul07/RUP.htm

> *Durante varios años se ha utilizado el modelo tradicional en cascada, demostrando en la practica que no refleja en la realidad la complejidad inherente al proceso de desarrollo de software. Este problema es derivado de la naturaleza implícita de la estructura de este modelo, definido por una secuencia de grandes etapas que requieren alcanzar hitos que deben ser concluidos antes de continuar con la siguiente fase.*
>
> *Como una alternativa de solución a este problema, se definieron posteriormente los modelos iterativos e incrementales que trabajan adecuadamente con niveles altos de riesgo, y permiten entregar liberaciones de software en etapas tempranas; tal es el caso del Proceso Unificado propuesto por IBM, que incluye prácticas claves y aspectos relacionados a la planeación estratégica y administración de riesgos; y actualmente guían de forma natural el proceso de desarrollo de software complejo por lo que ha sido considerado como un estandar el desarrollo de software en las empresas.*
>
> *El proceso unificado conocido como RUP, es un modelo de software que permite el desarrollo de software a gran escala, mediante un proceso continuo de pruebas y retroalimentación, garantizando el cumplimiento de ciertos estándares de calidad. Aunque con el inconveniente de generar mayor complejidad en los controles de administración del mismo. Sin embargo, los beneficios obtenidos recompensan el esfuerzo invertido en este aspecto.*
>
> *El proceso de desarrollo constituye un marco metodológico que define en términos de metas estratégicas, objetivos, actividades y artefactos (documentación) requerido en cada fase de desarrollo. Esto permite enfocar esfuerzo de los recursos humanos en términos de habilidades, competencias y capacidades a asumir roles específicos con responsabilidades bien definidas.*

Estructura del ciclo de vida del proceso de desarrollo unificado



Fase de concepción
Esta fase tiene como propósito definir y acordar el alcance del proyecto con los patrocinadores, identificar los riesgos potenciales asociados al proyecto, proponer una visión muy general de la arquitectura de software y producir el plan de las fases y el de iteraciones.

Fase de elaboración
En la fase de elaboración se seleccionan los casos de uso que permiten definir la arquitectura base del sistema y se desarrollaran en esta fase, se realiza la especificación de los casos de uso seleccionados y el primer análisis del dominio del problema, se diseña la solución preliminar.

Fase de construcción
El propósito de esta fase es completar la funcionalidad del sistema, para ello se deben clarificar los requerimientos pendientes, administrar los cambios de acuerdo a las evaluaciones realizados por los usuarios y se realizan las mejoras para el proyecto.

Fase de transición
El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados en las pruebas de aceptación, capacitar a

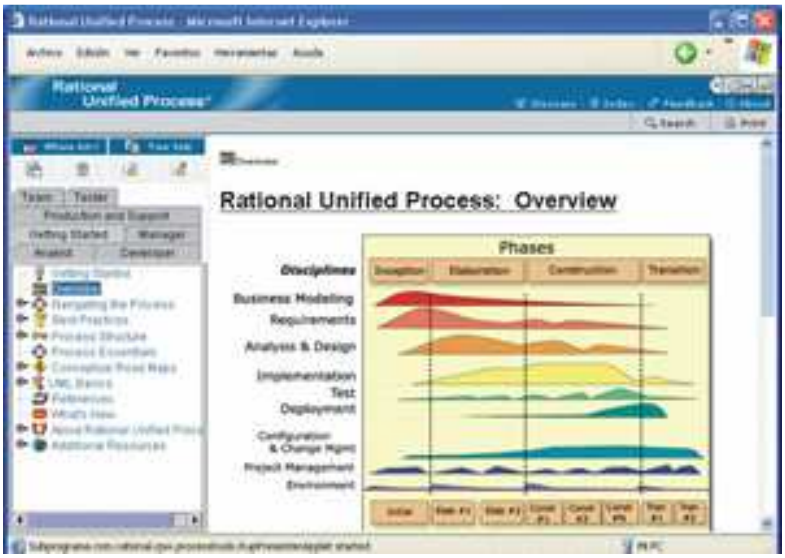
los usuarios y proveer el soporte técnico necesario. Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto.

Este tipo de metodología no ha sido aplicada probablemente por su complejidad de administración o desconocimiento de la misma, desaprovechando sus considerables ventajas respecto a los métodos tradicionales. Por esto, es necesario entonces desarrollar mecanismos de apropiación tecnológica más eficaces, que permitan mantener actualizadas las prácticas organizacionales y los marcos de referencia aquí mencionados. Es aquí, donde es necesario considerar que el conocimiento de la metodología y desarrollo de habilidades de los analistas, programadores, administradores de bases de bases de datos y demás miembros del equipo de desarrollo, comienzan desde su preparación universitaria donde es necesario conocer este enfoque y aplicarlo en proyectos en donde utilicen las guías de trabajo definidas en el RUP y desarrollen los artefactos asociados;

esperando que con la practica alcancen un nivel de madurez en la asimilación del proceso unificado (RUP).

De esta manera en la asignatura de análisis y diseño de sistemas de información II, que se imparte a los estudiantes del programa educativo de Tecnologías de la información y comunicación, se desarrolla un proyecto de simulación tipo basado en la metodología de trabajo del Proceso Unificado Rational, utilizando la herramienta CASE "Rational Unified Process" que es un sitio WEB en línea que los alumnos consultan para entender los

términos en que debe ser realizada la documentación y diseño de los programas informáticos que construyen.



Con el objetivo de promover el conocimiento de la estructura metodología del RUP y en complemento al uso de herramienta CASE descrita anteriormente, se diseño una Tecnología Educativa de Alto Impacto (TEAI) en su modalidad de material didáctico que permite que los alumnos conozcan, comprendan y analicen la naturaleza iterativa de desarrollo de proyectos con el proceso unificado en relación al avance ó estatus del proyecto y la evolución de los artefactos generados.

Con esta (TEAI) se busca que los estudiantes manipulen directamente la estructura del ciclo de vida del RUP, mediante el manejo de piezas adheribles al pizarrón le permiten identificar roles o artefactos por fases o por disciplinas disminuyendo la complejidad que resulta el trabajar por primera vez con procesos evolutivos promoviendo de la misma manera la participación activa del alumno en la asimilación de su propio conocimiento.

Bibliografía: Rational Unified Process. http://www-306.ibm.com/software/awdtools/rup/
Material didáctico elaborado por: Lic. Mónica Flores López y Mtra. María de Lourdes Santiago.

[Fuente: http://cursos.aiu.edu/Ingenieria%20de%20Software%20Avanzada/pdf/Tema%202.pdf]

El **Proceso Unificado Racional** (*Rational Unified Process* en inglés, habitualmente resumido como **RUP**) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

También se conoce por este nombre al software desarrollado por Rational, hoy propiedad de IBM, el cual incluye información entrelazada de diversos artefactos y descripciones de las diversas actividades. Está incluido en el **Rational Method Composer** (RMC), que permite la personalización de acuerdo a necesidades.
Originalmente se diseñó un proceso genérico y de dominio público, el Proceso Unificado, y una especificación más detallada, el ***Rational Unified Process***, que se vendiera como producto independiente.

Principios de desarrollo

El RUP está basado en 6 principios clave que son:

Adaptar el proceso

El proceso deberá adaptarse a las características propias del proyecto u organización. El tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

Equilibrar prioridades

Los requerimientos de los diversos participantes pueden ser diferentes, contradictorios o disputarse recursos limitados. *Debe encontrarse un equilibrio que satisfaga los deseos de todos*. Gracias a este equilibrio se podrán corregir desacuerdos que surjan en el futuro.

Demostrar valor iterativamente

Los proyectos se entregan, aunque sea de un modo interno, en **etapas iteradas**. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados

Colaboración entre equipos

El desarrollo de software no lo hace una única persona sino múltiples equipos. Debe haber una comunicación fluida para coordinar requerimientos, desarrollo, evaluaciones, planes, resultados, etc. también esta metodología esta basada en 2 punto 1,2

Elevar el nivel de abstracción

Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes 4GL o marcos de referencia (frameworks) por nombrar algunos. Esto evita que los ingenieros de software vayan directamente de los requisitos a la codificación de software a la medida del cliente, sin saber con certeza qué codificar para satisfacer de la mejor manera los requerimientos y sin comenzar desde un principio pensando en la reutilización del código. Un alto nivel de abstracción también permite discusiones sobre

diversos niveles y soluciones arquitectónicas. Éstas se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con el lenguaje UML.focarse en la calidad=== El control de calidad no debe realizarse al final de cada iteración, sino en **todos** los aspectos de la producción. El aseguramiento de la calidad forma parte del proceso de desarrollo y no de un grupo independiente.

Ciclo de vida

Esfuerzo en actividades según fase del proyecto

El ciclo de vida RUP es una implementación del Desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semiordenadas. El ciclo de vida organiza las tareas en fases e iteraciones.

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades. En la Figura muestra cómo varía el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto RUP.

Las primeras iteraciones (en las fases de Inicio y Elaboración) se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos, y al establecimiento de una baseline (Línea Base) de la arquitectura.

Durante la fase de inicio las iteraciones hacen mayor énfasis en actividades de modelado del negocio y de requerimientos.

En la fase de elaboración, las iteraciones se orientan al desarrollo de la baseline de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la baseline de la arquitectura.

En la fase de construcción, se lleva a cabo la construcción del producto por medio de una serie de iteraciones.

Para cada iteración se selecciona algunos Casos de Uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realiza una pequeña cascada para cada ciclo. Se realizan tantas iteraciones hasta que se termine la implementación de la nueva versión del producto.

En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios.

Como se puede observar en cada fase participan todas las disciplinas, pero que dependiendo de la fase el esfuerzo dedicado a una disciplina varía.

Principales características

- Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo)
- Pretende implementar las mejores prácticas en Ingeniería de Software
- Desarrollo iterativo
- Administración de requisitos
- Uso de arquitectura basada en componentes
- Control de cambios
- Modelado visual del software
- Verificación de la calidad del software

El RUP es un producto de Rational (IBM). Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos (que son los productos tangibles del proceso como por ejemplo, el modelo de casos de uso, el código fuente, etc.) y roles (papel que desempeña una persona en un determinado momento, una persona puede desempeñar distintos roles a lo largo del proceso)....

Fases

- Establece oportunidad y alcance
- Identifica las entidades externas o actores con las que se trata
- Identifica los casos de uso

RUP comprende 2 aspectos importantes por los cuales se establecen las disciplinas:

Proceso: Las etapas de esta sección son: (Revise nuevamente la gráfica)

- Modelado de negocio
- Requisitos
- Análisis y Diseño
- Implementación
- Pruebas
- Despliegue

Soporte: En esta parte nos encontramos con las siguientes etapas:

- Gestión del cambio y configuraciones
- Gestión del proyecto

•Entorno

La estructura dinámica de RUP es la que permite que éste sea un proceso de desarrollo fundamentalmente iterativo, y en esta parte se ven inmersas las 4 fases descritas anteriormente:

•Inicio(También llamado Incepción)

•Elaboración

•Desarrollo(También llamado Implementación, Construcción)

•Cierre (También llamado Transición)

Artefactos

RUP en cada una de sus fases (pertenecientes a la estructura estática) realiza una serie de artefactos que sirven para comprender mejor tanto el análisis como el diseño del sistema (entre otros). Estos artefactos (entre otros) son los siguientes:

Inicio:

•Documento Visión

•Especificación de Requerimientos **Elaboración**:

•Diagramas de caso de uso **Construcción**:

•Documento Arquitectura que trabaja con las siguientes vistas: 47

Vista Lógica:

•Diagrama de clases

•Modelo E-R (Si el sistema así lo requiere) Vista de Implementación:

•Diagrama de Secuencia

•Diagrama de estados

•Diagrama de Colaboración

Vista Conceptual:

•Modelo de dominio

Vista física:

•Mapa de comportamiento a nivel de hardware.

Un poco de historia

Los orígenes de RUP se remontan al modelo espiral original de Barry Boehm. Ken Hartman, uno de los contribuidores claves de RUP colaboró con Boehm en la investigación. En 1995 Rational Software compró una compañía sueca llamada Objectory AB, fundada por Ivar Jacobson, famoso por haber incorporado los casos de uso a los métodos de desarrollo orientados a objetos.

El Rational Unified Process fue el resultado de una convergencia de Rational Approach y

Objectory (el proceso de la empresa Objectory AB). El primer resultado de esta fusión fue el Rational Objectory Process, la primera versión de RUP, fue puesta en el mercado en 1998, siendo el arquitecto en jefe Philippe Kruchten.

Comentarios sobre Alcance del RUP

La metodología RUP es más apropiada para proyectos grandes (Aunque también pequeños), dado que requiere un equipo de trabajo capaz de administrar un proceso complejo en varias etapas. En proyectos pequeños, es posible que no se puedan cubrir los costos de dedicación del equipo de profesionales necesarios.

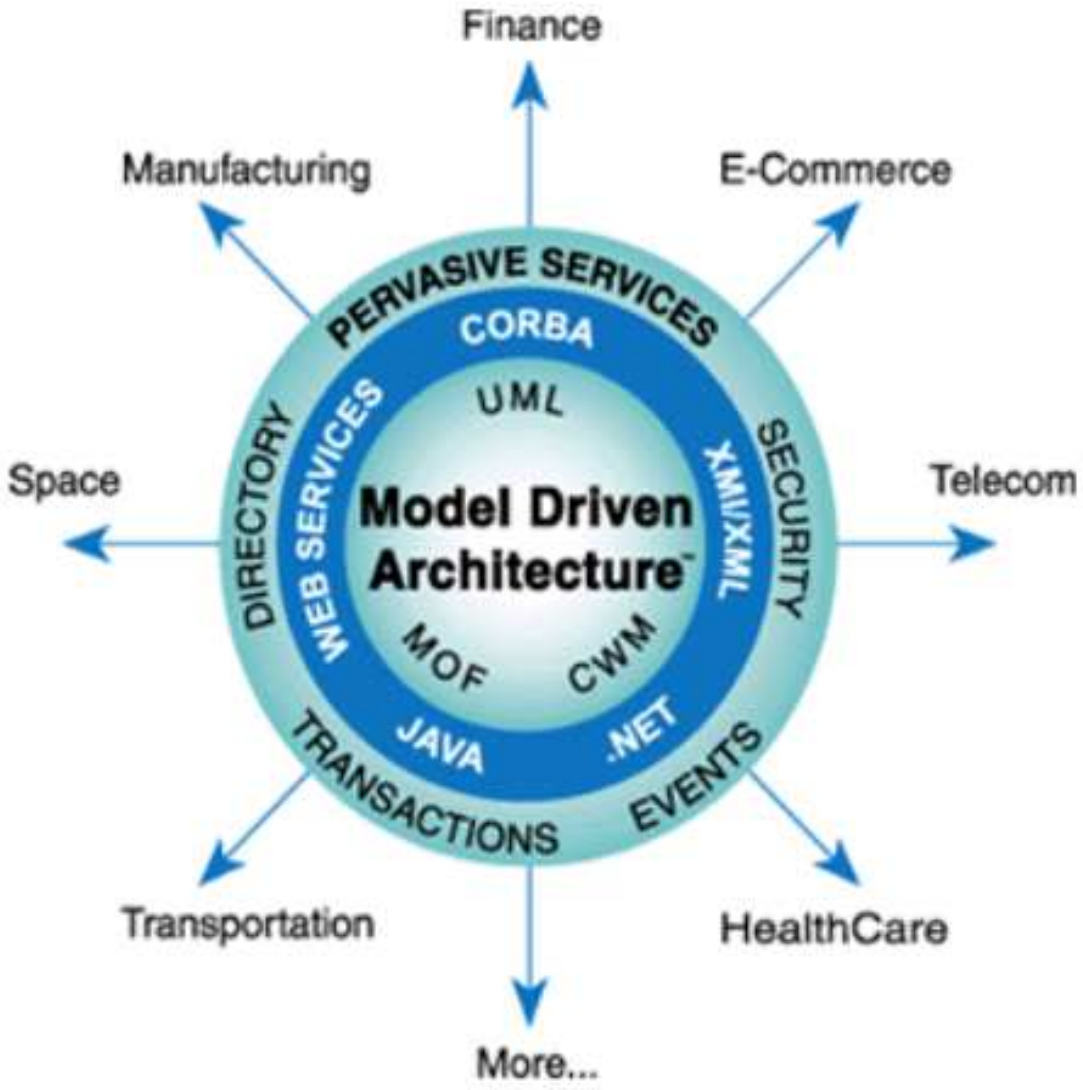
Comentarios sobre Metodología

Por otro lado, en lo que se refiere a la metodología esta comprende tres frases claves: Dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.

En lo referente a dirigido por los casos de uso, está enfocado hacia el cliente y se utilizan con algunas modificaciones tal vez, hasta la disciplina de pruebas, en la cual, un caso de uso puede a su vez tener uno o más casos de prueba.

2.4 MDA

Fuente: http://astreo.ii.uam.es/~jlara/doctorado.2006/tema5.pdf



MDA es el acrónimo de Model Driven Architecture, en español Arquitectura dirigida por modelos.

Podemos definir a MDA como un marco de trabajo que ha definido el Object Management Group (OMG).

El OMG es una organización de compañías de sistemas de información creada en 1990 con el fin de potenciar el desarrollo de aplicaciones orientadas a objetos distribuidas. Esta organización ha definido estándares importantes como UML, CORBA, MOF, entre otros. Su sitio Web es http://www.omg.org.

En los últimos años se ha tomado mayor interés e importancia al modelado en el desarrollo de cualquier tipo de software, debido a la facilidad que ofrece un buen diseño

tanto a la hora de desarrollar como al hacer la integración y mantenimiento de sistemas de software.

Es así que en el año 2001, el OMG definió un marco de trabajo nuevo llamado MDA. La clave del MDA es la importancia de los modelos en el proceso de desarrollo de software. MDA propone la definición y uso de modelos a diferente nivel de abstracción, así como la posibilidad de la generación automática de código a partir de los modelos definidos y de las reglas de transformación entre dichos modelos.

Ciclo de vida del desarrollo con MDA:

1. Modelo independiente de la plataforma (PIM)
2. Modelo específico a la plataforma (PSM)
3. Código

Usando la metodología MDA, la funcionalidad del sistema será definida en primer lugar como un modelo independiente de la plataforma (Platform-Independent Model o PIM) a través de un lenguaje específico para el dominio del que se trate.

Dado un modelo de definición de la plataforma (Platform Definition Model o PDM) correspondiente a CORBA, .NET, web, etc., el modelo PIM puede traducirse entonces a uno o más modelos específicos de la plataforma (Platform-specific models o PSMs) para la implementación correspondiente, usando diferentes lenguajes específicos del dominio, o lenguajes de propósito general como Java, C#, Python, etc. La traducción entre el PIM y los PSMs se realizan normalmente utilizando herramientas automatizadas, como herramientas de transformación de modelos (por ejemplo aquellas herramientas que cumplen con el nuevo estándar de OMG denominado QVT).

El proceso completo se encuentra documentado en un documento que actualiza y mantiene OMG denominado la Guía MDA.Martin Sullon

Los principios de MDA pueden aplicarse a otras áreas como el modelado de procesos de negocios donde el PIM, independiente de la tecnología y de la arquitectura es adaptado tanto a los sistemas como a los procesos manuales.

El modelo MDA está relacionado con múltiples normas, incluyendo el lenguaje de modelado unificado (Unified Modeling Language o UML), Meta-Object Facility (MOF), XML Metadata Interchange (XMI), Enterprise Distributed Object Computing (EDOC), el Software Process Engineering Metamodel (SPEM) y el Common Warehouse Metamodel (CWM).

Fíjese que el término "arquitectura" en los metamodelos no se refiere a la arquitectura del sistema modelizado sino más bien a la arquitectura de los distintos estándares y formas

del modelo que sirven de base tecnológica al MDA.

El Object Management Group mantiene la marca registrada sobre MDA, así como sobre varios términos similares incluyendo MDD (Model Driven Development), Model Driven Application Development, Model Based Application Developtment, Model Based Programming y otros similares.

El acrónimo principal que aun no ha sido depositado por OMG hasta ahora es MDE. A consecuencia de esto, el acrónimo MDE es usado actualmente por la comunidad investigadora internacional cuando se refieren a ideas relacionadas con la ingeniería de modelos sin centrarse exclusivamente en los estándares OMG.

Acercamiento MDA

Uno de los principales objetivos de MDA es separar el diseño de la arquitectura y de las tecnologías de construcción, facilitando que el diseño y la arquitectura puedan ser alterados independientemente. El diseño alberga los requerimientos funcionales (casos de uso) mientras que la arquitectura proporciona la infraestructura a través de la cual se hacen efectivos requerimientos no funcionales como la escalabilidad, fiabilidad o rendimiento.

MDA se asegura de que el modelo independiente de la plataforma (PIM), el cual representa un diseño conceptual que concreta los requerimientos funcionales, sobrevive a los cambios que se produzcan en las tecnologías de fabricación y en las arquitecturas software. De particular importancia en MDA es la noción de transformación de modelos. Uno de los estándares definidos para la transformación de modelos se denomina QVT.

Herramientas MDA

Un amplio conjunto de herramientas con soporte para MDA se están desarrollando por los principales fabricantes y proyectos Open Source. Estas herramientas permiten comúnmente la especificación rudimentaria de arquitecturas. Algunos ejemplos simples de estas especificaciones de arquitecturas incluyen:

- seleccionar una de las arquitecturas de referencia tales como Java EE o Microsoft .NET.
- Especificar la arquitectura a un nivel de mayor detalle incluyendo tecnologías de la capa de presentación, de la capa de negocio, de persistencia o tecnología de mapeo de persistencia (como p.e. mapeo relacional a objeto).

Conferencias

Entre las distintas conferencias sobre el tema podemos citar la ECMDA, Conferencia Europea sobre MDA, o incluso MoDELS, anteriormente llamadas conferencias <<UML>>. También hay distintas conferencias y seminarios de trabajo (en OOPSLA, ECOOP y demás) orientadas en aspectos más específicos de MDA como la transformación de modelos, la composición o su generación.

V Bibliografía

[Booch99] **El Lenguaje Unificado de Modelado**. G. Booch, J. Rumbaugh, I. Jacobson. Addison Wesley Iberoamericana, 1999.

[Booch94] **Object-Oriented Analysis and Design**. G. Booch. Benjamin/Cummings, 1994.

[BJR97] **The UML Specification Document**. G. Booch, I. Jacobson and J. Rumbaugh. Rational Software Corp., 1997.

[Jacobson92] **Object-Oriented Software Engineering: A Use Case Driven Approach**. I. Jacobson. Addison-Wesley, 1992.

[Larman99] **UML y Patrones**. C. Larman. Prentice Hall, 1999.

[Rumbaugh91] **Object-Oriented Modeling and Design**. J. Rumbaugh et al.Prentice-Hall,1991.

[UML00] **UML Resource Center**. Rational Software. http://www.rational.com/uml/