Luis Antonio Salazar Caraballo

Prolegómenos[1] Sobre el Lenguaje de Modelado Unificado (UML)

Prolegómeno.

(Del gr. προλεγόμενα, preámbulos).

1. m. Tratado que se pone al principio de una obra o escrito, para establecer los fundamentos generales de la materia que se ha de tratar después. U. m. en pl.

Resumen

El Lenguaje de Modelado Unificado o UML es una notación estándar con carácter universal utilizado para escribir modelos de sistemas, sobre todo, de sistemas de software, que utiliza una serie de diagramas y una semántica bien definida con el propósito de elaborar los artefactos de un sistema a través de las distintas etapas de su ciclo de vida, específicamente durante el análisis y el diseño del mismo.

Este es un artículo con formato de "generalidades" sobre UML en las que les hablo de algunas características mayores del lenguaje, lo mismo que de los varios de los modelos que utiliza (Casos de Uso, Secuencia, Colaboración, Clases entre otros) y de algunos mitos que se han extendido junto con el uso del lenguaje.

Palabras Clave

UML
Lenguaje
Modelo de Sistemas
Análisis y Diseño
Artefacto de Software
Caso de Uso
Actor
Diagrama de Secuencia
Colaboración
Clase
Paquete
Máquina de Estados
Componentes
Subsistema
Asociación
Agregación
Generalización
UML 2.0

Generalidades

UML (*Unified Modeling Language*) es un lenguaje para expresar distintos modelos de software durante las fases de Análisis y Diseño de un sistema.

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

Luis Antonio Salazar Caraballo

Este es el primero, los prolegómenos, de una serie de artículos, a manera de tutorial o de curso introductorio, sobre UML, y sobre cómo modelar sistemas en la forma orientada a objetos.

Textualmente:

> El Lenguaje de Modelado Unificado (UML) es un lenguaje para especificar, visualizar, construir y documentar los artefactos de sistemas de software, así como también para hacer modelamiento de negocio y de otros sistemas que no son software[2].

En este contexto, Especificar significa enumerar y construir modelos que son precisos, no ambiguos y completos. UML dirige la especificación de todas las decisiones importantes de análisis, diseño e implementación. Por ejemplo, como Arquitecto del software en desarrollo yo puedo decidir que la conexión desde una página Web a una base de datos la haré a través de ODBC o de JDBC o de ADO.Net y que además usaré *recordsets* (cursores) desconectados de la base de datos. Al hacerlo, puedo usar Diagramas de Actividad de UML, o Diagramas de Estado para documentar esas decisiones al programador de la aplicación.

Mientras tanto, Construir quiere decir que los modelos que se construyen con UML están relacionados con los lenguajes de programación orientada a objetos. UML se usa para hacer Ingeniería Hacía Adelante, esto es, un mapeo directo de un modelo UML con código fuente. UML también se usa para hacer Ingeniería en Reversa, o sea, una reconstrucción de un modelo UML desde una implementación específica, normalmente en un lenguaje OO. Por supuesto estas actividades requieren de herramientas y de intervención humana, del desarrollador, para evitar pérdida de información. En mi última revisión[4] encontré varias docenas de estas herramientas, desde las que sólo permiten “dibujar” y son gratis, hasta las que permiten “ejecutar” los modelos, transformar un diagrama en otro, aplicar patrones de diseño de software y generar código en los más populares lenguajes de programación del momento.

UML además se usa para Documentar la arquitectura de los sistemas, los requerimientos, las pruebas y actividades como planeación del proyecto y manejo de la entrega del producto.

Aquí me siento con el deber de anunciarles que Modelar no es lo mismo que Documentar. Sin embargo, aunque los conceptos de “modelo” y “documento” son ortogonales, esto es, es posible tener modelos que no son documentos y documentos que no son modelos, también es posible que si el esfuerzo dedicado a modelar es el suficiente, y si de casualidad se da la fortuna de tener una buena herramienta de modelado, una parte de los modelos se puede convertir en documentos valiosos para todo el equipo de desarrollo de un sistema.

El Lenguaje

Como cualquier lenguaje, UML tiene una sintaxis y una semántica. La sintaxis de UML está compuesta por un conjunto de construcciones gráficas útiles para especificar diagramas y modelos de sistemas. Algunos de estos símbolos son, de

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

izquierda a derecha en la figura 1: Actor, Caso de Uso, Clase, Paquete y el inferior es una Relación de Generalización.

Cada uno de estos símbolos tiene un significado de manera independiente, como lo tienen las palabras reservadas *Begin, End y While* en Pascal o los términos *private, void y string* en C# (léase C Sharp).

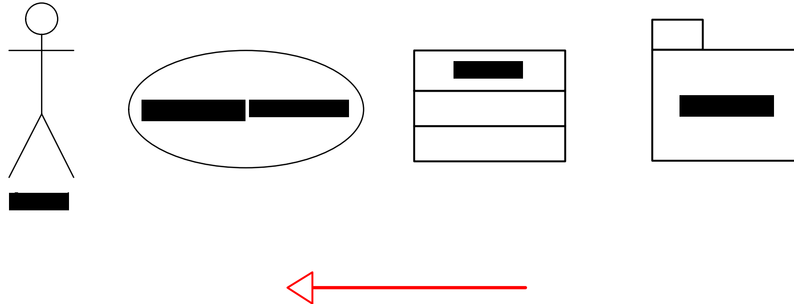

Figura 1: Símbolos comunes en UML

Pero los símbolos por sí solos no dicen mucho, aunque algunos tienen identidad propia como el de Actor que representa un grupo de usuarios o el de Caso de Uso que representa un proceso del sistema, normalmente interactivo. (De la misma forma en que Begin, End y While no dicen gran cosa, ni separadas, ni dispuestas secuencialmente en cualquier orden posible.)

Es la semántica de UML, como la semántica de cualquier otro lenguaje, la que nos dice por ejemplo que un Actor solo se puede "comunicar" con el sistema a través de los casos de uso mediante una relación de Asociación, o que un Paquete puede estar compuesto de Clases, Casos de Uso, Componentes u otros Paquetes y que, finalmente, existen distintos tipos de diagramas, a manera de modelos, que pueden ser construidos a partir de uno o más símbolos de UML.

Siguiendo con nuestra parodia de los lenguajes de programación, la semántica de C# nos dice que una instrucción válida tiene la sintaxis:

while <Expresión-Lógica> *{* <Secuencia> *}*

Donde *while* es una palabra clave, *{* y *}* son símbolos gramaticales obligatorios, y <Expresión-Lógica> y <Secuencia> son elementos gramaticales compuestos del lenguaje cuyo significado está fuera del alcance de estos prolegómenos y no tiene sentido detallar aquí. De esta forma,

```
while (i <= 10)
{
        i = i + 1;
}
```

es una secuencia aceptada por el compilador de C#.

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

Desde este punto de vista, una "instrucción válida" en UML es la que muestro en la figura 2.

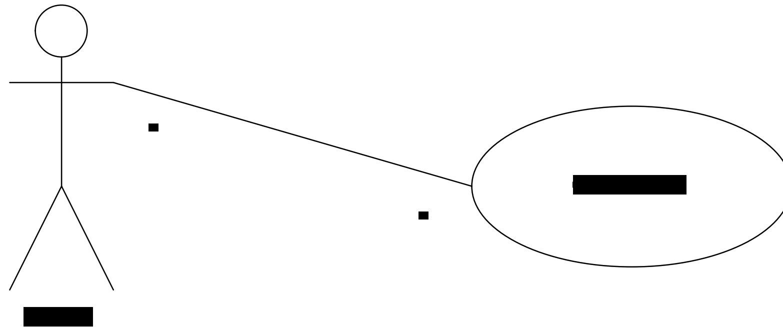

Figura 2: Una "instrucción" válida UML

En esta figura, la línea que comunica al Actor con el Caso de Uso es un símbolo UML y significa una relación de Asociación entre el uno y el otro. Esta línea tiene un valor semántico en esta "instrucción" y es el de señalar que el Actor inicia e interactúa con el proceso determinado por el Caso de Uso. Los puntos (.) debajo de cada lado de la línea representan los extremos de la misma y cada uno de ellos tiene un conjunto de atributos que los hacen más complejos de lo que aparentemente se ve en el modelo. En un próximo artículo hablaré de estos y otros detalles importantes.

Pero, a diferencia de un lenguaje de programación convencional, UML no produce instrucciones ejecutables (realmente ya existen compiladores de modelos UML ejecutables tal como existen compiladores de Pascal o C# que son capaces de traducir los modelos UML en instrucciones ejecutables[3], pero ese no es el objeto de este tratado). UML se usa más bien para expresar, para dibujar modelos, para plasmar o traducir las ideas alrededor de una solución de software específica para que ésta sea entendible por todo el equipo de desarrollo y, en parte, por los usuarios finales del sistema en desarrollo. De hecho UML sirve para documentar la solución, incluyendo las decisiones que se vayan tomando a lo largo del ciclo de vida del proyecto. En particular, UML se usa para elaborar modelos que nos permitan entender mejor el sistema en construcción.

No entraré en el detalle de los elementos sintácticos y semánticos de UML sino hasta que les presente los demás artículos de esta serie. Mejor haré una breve revisión de los modelos que podemos construir con esos elementos sintácticos y semánticos.

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

Los Modelos

Con UML podemos trazar nueve modelos clásicos. Creo que es tiempo de recordar aquí que un Modelo es una abstracción de la realidad, para lo que me ocupa, un esbozo de un sistema que, a su vez, es un conjunto de elementos interrelacionados entre sí que buscan un fin común. Los modelos que podemos dibujar con UML son precisamente abstracciones de los elementos reales que componen un sistema de software: clases, atributos, subsistemas, interfaz humana y procesos, entre muchos otros.

Los modelos, llamados comúnmente Diagramas, son: Casos de Uso, Secuencia, Colaboración, Clases, Estados, Actividad, Componentes, Paquetes y Despliegue. Permítanme mostrarles, a grandes rasgos, para que sirve cada uno de ellos.

Casos de Uso

Este modelo representa un proceso, o un conjunto de procesos, del sistema, incluyendo quien o que lo ejecuta o lo inicia.

El modelo de casos de uso está compuesto por Actores y Casos de Uso y las relaciones entre ellos. Un Actor representa un grupo de usuarios (que tienen el mismo papel para el sistema), un sistema de software externo, con el que el sistema en desarrollo se comunica (en una o en ambas vías) o un sistema de hardware con el que también existe esa comunicación. Un Caso de Uso, por su parte, es un conjunto de secuencias de tareas o actividades, a manera de proceso, que el sistema ejecuta en un momento dado del tiempo.

Existen relaciones entre un Actor y un Caso de Uso y entre un Caso de Uso y otro. En el primer caso, el modelo señala que un Actor inicia un caso de uso determinado por la relación de Asociación como señalé en la figura 2. Esta es la única relación existente entre un Actor y el Sistema.

En el segundo caso, existen tres tipos de relaciones entre los casos de uso: Uso, Extensión y Generalización.

En la parte superior de la figura, CasoUso1 <<usa>> CasoUso2, es decir, en algún punto de la secuencia de actividades especificada en CasoUso1, se activa la secuencia de tareas, el proceso, establecido en CasoUso2; una vez terminada esta segunda secuencia, el flujo del proceso regresa a CasoUso1. En particular, esta relación especifica cómo el comportamiento definido por el caso de uso incluido (CasoUso1) es explícitamente insertado en el comportamiento definido por el caso de uso base (CasoUso2).

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

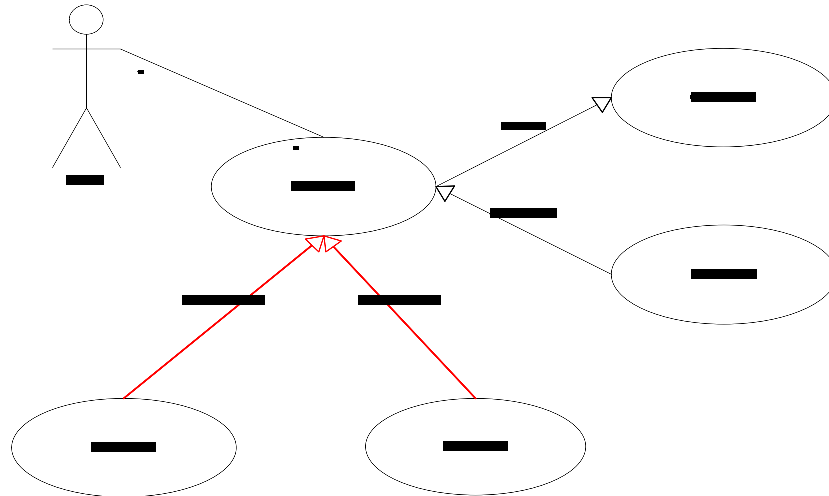

Figura 3: Relaciones entre casos de uso

En la parte central del diagrama, CasoUso3 <<extiende>> CasoUso1, o sea, especifica cómo el comportamiento definido por el caso de uso extendido (CasoUso3) puede ser insertado en el comportamiento definido por el caso de uso base (CasoUso1). Está implícitamente insertado en el sentido de que la extensión no es mostrada en el caso de uso base.

En la parte inferior del diagrama, CasoUso1 <<generaliza>> a CasoUso4 y CasoUso5, lo cual significa que los casos de uso hijos (CasoUso4 y CasoUso5) especializan todo el comportamiento y las características descritas por el padre (CasoUso1). Les recuerdo que una generalización es una relación taxonómica entre un elemento más general y un elemento más específico.

Secuencia

El diagrama de Secuencia es uno de los que conforman el grupo de Diagramas de Interacción porque muestra precisamente la interactividad entre distintos componentes del sistema, normalmente un Actor, a través de una Interfaz, y un conjunto de objetos y/o de clases.

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

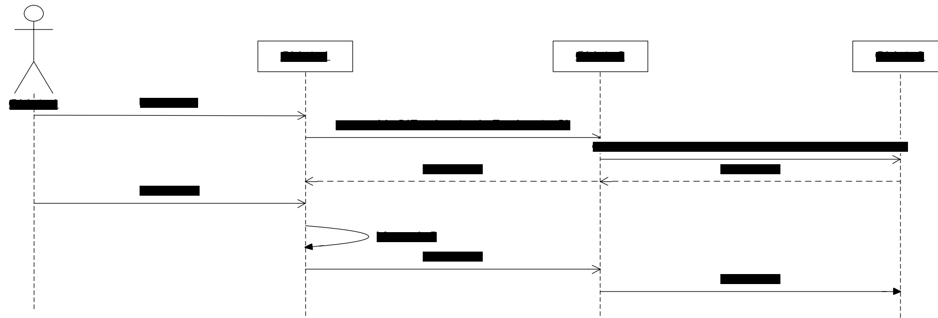

Figura 4: Diagrama de Secuencia típico

Este modelo muestra una secuencia de mensajes, con sus operaciones, que pasan de un objeto a otro y que parten desde un Actor específico. En este diagrama, el objeto con el que se comunica Actor1, Objeto1, es normalmente una interfaz, en este caso, una interfaz gráfica de usuario, mediante la cual el usuario interactúa con el software.

Los mensajes que pasan entre uno y otro objeto se componen de una o más operaciones que a su vez pueden o no llevar parámetros. También es posible establecer mensajes de retorno (los de línea punteada) y mensajes que se ejecutan en el mismo objeto (Mensaje5). Las líneas verticales que bajan desde cada objeto se llaman Líneas de Vida de los objetos e indican el tiempo y espacio de vida de cada instancia en el proceso que se describe mediante el diagrama de secuencia.

En resumen, un diagrama de secuencia es una vista externa de un proceso, los elementos que intervienen en ese proceso y las relaciones entre estos elementos (Actores, Objetos, Clases).

Colaboración

Semánticamente hablando el diagrama de Colaboración tiene el mismo significado que el de Secuencia. Ambos muestran la interacción entre los objetos del sistema. Sin embargo, la relación en este modelo es espacial, mientras que en el diagrama de Secuencia es temporal.

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

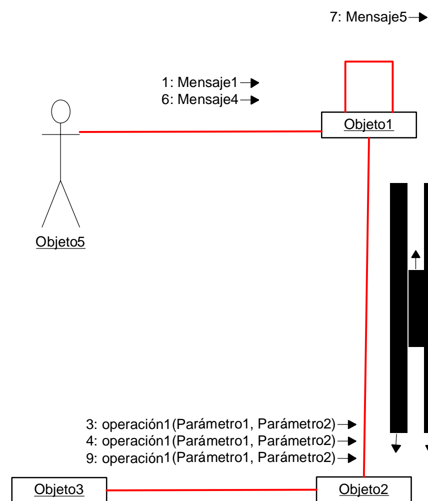

Figura 5: Diagrama de Colaboración clásico

Este diagrama muestra la interacción que existe entre los objetos, o sea, muestra el Acoplamiento, esta es la medida de afinidad con la que dos componentes se relacionan.

Clases

Un diagrama de clases muestra las clases del sistema y las relaciones entre ellas. Un diagrama así muestra la estructura estática del modelo, pero no revela ninguna información relacionada con lo que sucede a través del tiempo cuando el modelo se ejecuta.

En resumen, una clase es una descripción de un conjunto de objetos que comparten las mismas especificaciones de atributos, operaciones, relaciones, restricciones y semántica.

En particular, una clase posee atributos, características o propiedades (datos) y expone un determinado comportamiento o métodos (programas). Pero esta no es una lección de Programación Orientada a Objetos, ese será el tema de otro artículo.

Lo que nos interesa aquí es que en UML una clase se representa mediante un rectángulo dividido en tres partes, como lo ilustra la figura 6. La primera sección de la clase contiene el nombre de la clase (también puede contener su clasificador o su estereotipo, conceptos de los que les hablaré más adelante en

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

este mismo tratado). La sección siguiente contiene los atributos de la clase, junto con su tipo de dato, longitud, valor inicial y su visibilidad (esta puede ser Privada, Protegida o Pública). Y la sección inferior detalla los métodos de la clase con sus argumentos y su visibilidad. Recordemos además que el conjunto de métodos públicos de una clase conforman el protocolo de la clase, el mecanismo mediante el cual otras clases o elementos del modelo se comunican con ella.

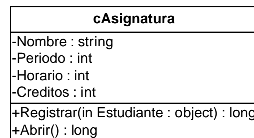

Figura 6: Una Clase Típica

Otro aspecto importante lo constituyen las relaciones entre las clases. Estas relaciones pueden ser: Asociación, Agregación y Generalización.

La asociación modela una conexión bi-direccional semántica entre instancias. En este contexto, semántica quiere decir que esa comunicación tiene un significado único que está dado por la cardinalidad y por el papel que se le asigne a dicha relación. Realmente una asociación representa una relación estructural entre instancias de dos clases distintas; ella constituye una conexión entre objetos de dos o más clases que existen durante algún tiempo. Por ejemplo, en la expresión "John Alexander Enseña una o más Asignaturas", "John Alexander" es una instancia de la clase Profesor y "Asignatura" es una instancia de la clase del mismo nombre; entre tanto, "Enseña" es el Rol o Papel que "John Alexander" juega con el objeto Curso (todas las instancias de Profesor tienen ese papel con los objetos de la clase Asignatura); "uno o más" se refiere a la multiplicidad o cardinalidad de ese papel "Enseña" entre los objetos de las clases citadas.

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

Figura 7: Una Asociación Entre Clases

La agregación es una forma especial de asociación que modela una relación todo-parte entre un agregado (el todo) y sus partes. Es usada para modelar una relación de composición (o composicional) entre los elementos de un modelo (por lo que también se conoce como relación de composición, aunque una composición, como tal, tiene un significado distinto como reseñaré más adelante). Una Institución Educativa, por ejemplo, está compuesta de Profesores, Empleados y Auxiliares o Supernumerarios. Para modelar esto, el agregado (la Institución Educativa) tiene una asociación de agregación con sus partes constituyentes (Profesores, Empleados y Auxiliares). Ver figura 8.

Por su parte, una composición es una forma de agregación con un sentido de propiedad fuerte y un ciclo de vida coincidente de la parte con el agregado. En este caso, la multiplicidad del extremo agregado no puede exceder de uno, esto es, no puede ser compartida. Por ejemplo, un Automóvil está compuesto de una carrocería, cuatro llantas y un motor, y no es un Automóvil completo si falta alguna de las partes (un auto no arranca si el motor está dañado o no tiene motor del todo, o no tiene carrocería o le falta por lo menos una llanta; recíprocamente, una carrocería no constituye un auto por sí sola, ni las cuatro llantas, ni el motor).

La generalización, de otro lado, es una relación taxonómica entre un elemento más general y un elemento más específico. Este último es completamente consistente con el primero, el más general y puede contener información adicional. En programación (orientada a objetos), esta relación se conoce comúnmente como herencia. Esta relación también es usada para Paquetes, Casos de Uso y otros elementos.

Para no entrar en detalles, les dibujo en la figura 9un caso común de generalización con la clase Empleado en donde anoto que un Empleado de una Institución Educativa puede ser un Profesor o un Auxiliar o un Empleado de índole Administrativo.

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

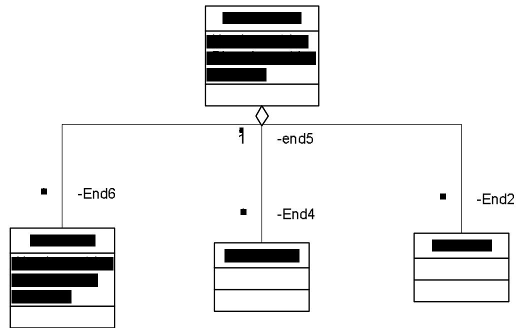

Figura 8: Un Ejemplo de Composición

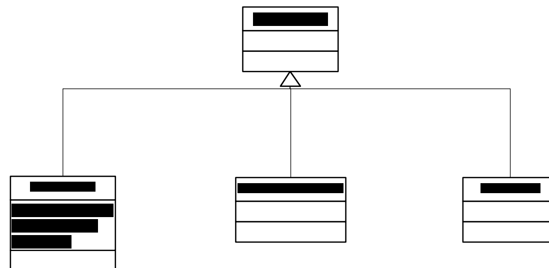

Figura 9: Un Caso de Generalización o Herencia

Estados

Un diagrama de estados se usa para describir el comportamiento de las instancias de un elemento del modelo tal como un objeto o una interacción. Específicamente, este diagrama describe las secuencias posibles de estados y acciones a través de las cuales las instancias de los elementos pueden proceder durante su ciclo de vida como resultado de reaccionar a eventos discretos (por ejemplo, señales o invocaciones a operaciones).

Así como un diagrama de clases muestra una fotografía estática de la estructura del modelo, un diagrama de estados muestra el comportamiento de una clase específica a través del tiempo. En particular, se crea un diagrama de estados

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

para cada clase (instancia) en el modelo que durante su vida útil cambie de un estado a otro como respuesta a una acción o evento en el sistema. No todas las clases requieren de un diagrama de estados.

Sintácticamente, un diagrama de estados es un gráfico que representa una máquina de estados (finitos). El número de estados de una clase debe ser el mínimo posible, con una tendencia a cero. En un diagrama de estados, puede haber Súper-Estados que contienen otros estados, como el de la figura 10.

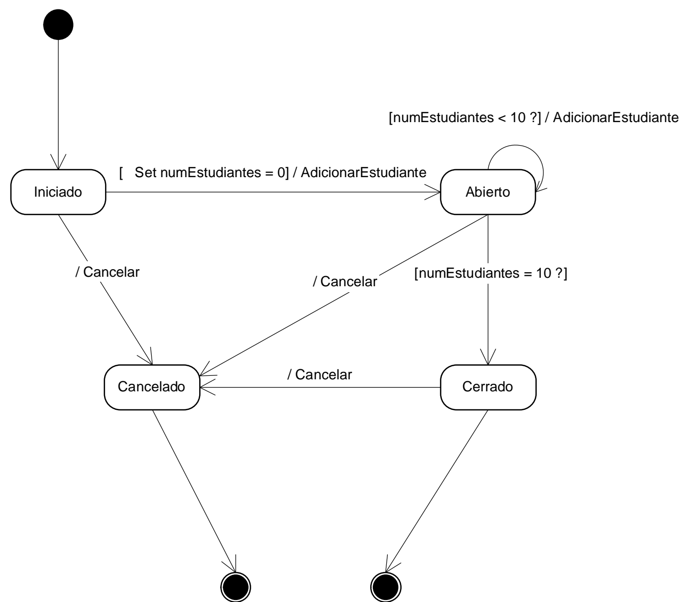

Figura 10: Un Diagrama de Estados

Actividad

Un diagrama de actividad es una variación de una máquina de estados en la que el estado representa la ejecución de acciones o subactividades y las transiciones son disparadas al completar acciones o subactividades.

En otras palabras, un diagrama de actividad es una vista interna de un proceso, establece cuando inicia, como se ejecuta (las acciones) y cuando termina. También es posible determinar mediante este diagrama que acciones ocurren en paralelo o que secuencia de actividades (sub) seguir luego de determinar el valor de una condición (lógica).

Un modelo requiere de tantos diagramas de actividad (y de estados) como sea posible para tomar todas las decisiones relevantes del sistema y para entender éste en su totalidad.

Un diagrama de actividad se asocia (en el modelo) a un clasificador como un caso de uso o un paquete o a la implementación de una operación.

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

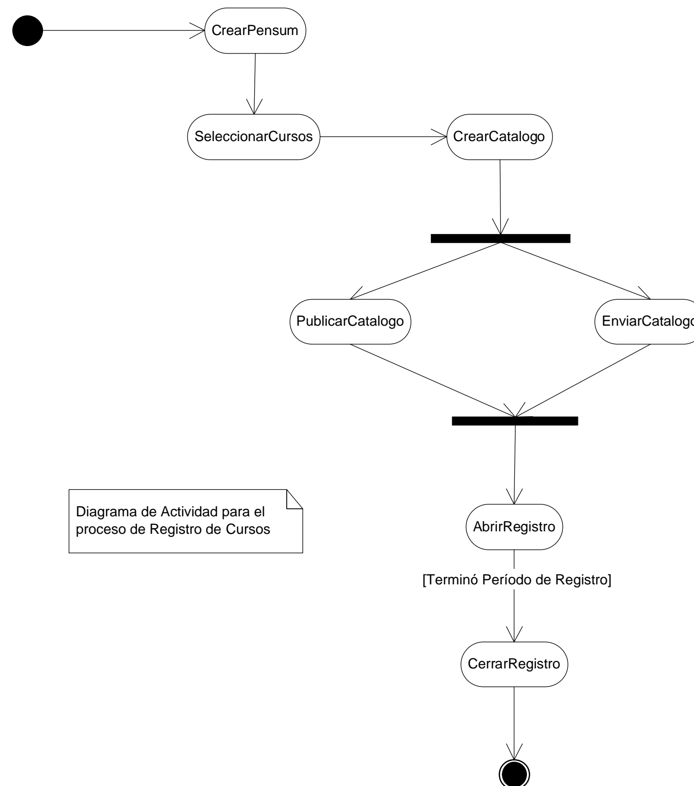

Figura 11: Un Diagrama de Actividad

Componentes

Un diagrama de componentes muestra las dependencias entre los componentes de software, incluyendo los clasificadores que los especifican (por ejemplo, clases de implementación) y los artefactos que los implementan, tales como, archivos de código fuente, archivos de código binario, archivos ejecutables, *scripts*.

En materia notacional, un diagrama de componentes es un grafo de componentes conectados por relaciones de dependencia, aunque es posible también que dos componentes se conecten mediante una relación de composición (cuando un componente contiene a otro).

Aunque un componente no tiene sus propias características (por ejemplo, atributos u operaciones), actúa como un contenedor de otros clasificadores (normalmente clases) que ya están definidos con sus características. Los componentes típicamente exponen un conjunto de interfaces que representan servicios proporcionados por los elementos que habitan el componente.

En el siguiente ejemplo, Seguridad.dll y Registro.dll son componentes que dependen de AccesoDatos.dll.

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

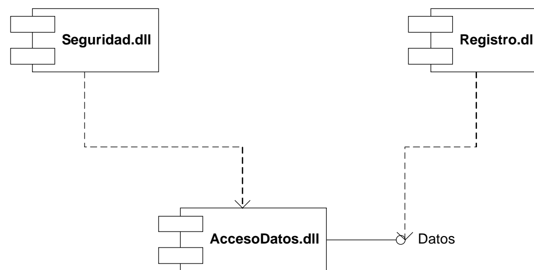

Figura 12: Diagrama de Componentes de un Sistema

Despliegue

Un diagrama de despliegue muestra la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes de software y hardware, procesos y objetos que los ejecutan. Este diagrama es útil para ilustrar la arquitectura física de un sistema.

La sintaxis de un diagrama de despliegue es un grafo de nodos conectados por relaciones de asociación. Los nodos pueden contener instancias de los componentes. Esto señala que los componentes corren o se ejecutan en el nodo.

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

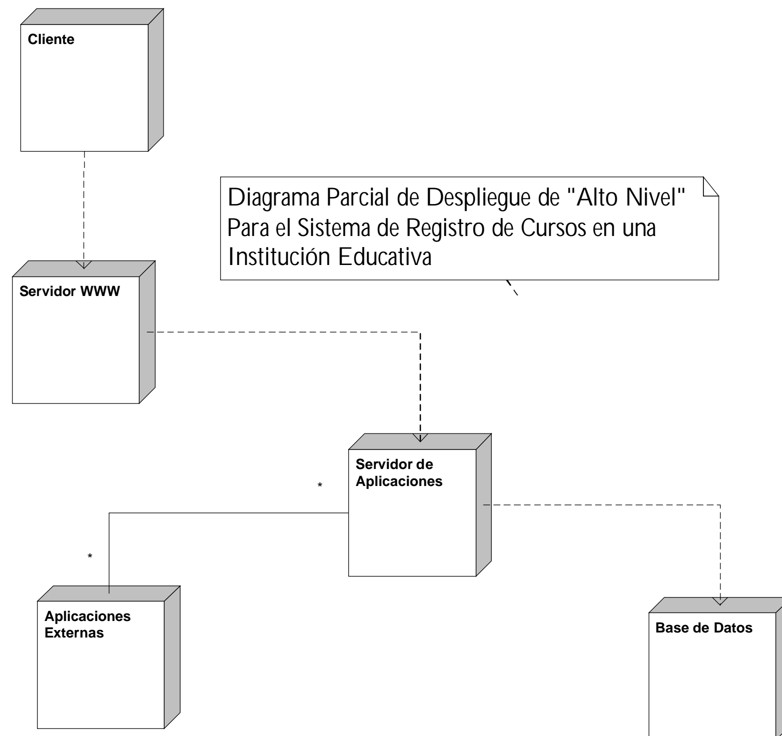

Figura 13: Diagrama de Despliegue de un Sistema

Paquetes, Subsistemas y Modelos

Un paquete es una agrupación de elementos del modelo. Los paquetes mismos pueden ser anidados dentro de otros paquetes. Un paquete puede contener paquetes subordinados así como otras clases de elementos del modelo (componentes, clases, casos de uso). En general, todos los tipos de elementos de un modelo UML pueden ser organizados en paquetes.

Un paquete se representa en UML mediante un rectángulo grande con un rectángulo pequeño (un "tabulador") adjunto al lado izquierdo superior del rectángulo grande. Es el mismo símbolo con el que se representan las carpetas en Windows.

Ahora bien, mientras que un paquete es un mecanismo genérico para organizar elementos de un modelo, un subsistema, por su parte, representa una unidad de comportamiento en el sistema físico y, por consiguiente, en el modelo. Un subsistema proporciona interfaces y tiene operaciones. Un subsistema se representa de la misma forma que un paquete, con un símbolo en el tabulador, como se ilustra en la figura.

Mientras tanto, un modelo captura una vista de un sistema físico. Por supuesto, un modelo es una abstracción del sistema físico con un propósito definido. Por

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

ejemplo, para describir aspectos de comportamiento del sistema físico a una categoría específica de interesados o en o involucrados con el sistema.

Diferentes modelos de un mismo sistema muestran distintos aspectos del sistema, como lo he ilustrado a lo largo de este artículo con los nueve modelos básicos de UML.

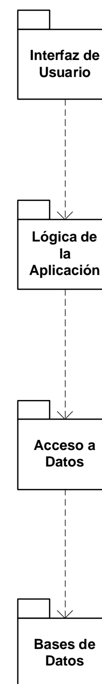

Figura 14: Diagrama de Paquetes de Alto Nivel de un Sistema

Otros Conceptos Básicos

En el ámbito de UML se manejan muchos otros conceptos que están fuera del alcance de estos prolegómenos, motivo por el cual, apenas les voy a mencionar algunos temas que son importantes para el estudio del modelado con UML.

UML es un lenguaje extensible a través de Mecanismos de Extensibilidad que permiten a los modeladores personalizar UML para dominios específicos, por ejemplo, para modelar sistemas que van a ser implementados en una plataforma específica como .NET o J2EE, o para crear un perfil de UML para un proceso de desarrollo de software como RUP (*Rational Unified Process*), por ejemplo o, más específicamente, para Modelar el Negocio.

Uno de esos mecanismos que juegan un papel de relevancia en UML cuando se trata de extenderlo son los Estereotipos. Por ejemplo, se pueden crear

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

estereotipos para Clases Límite, Clases Persistentes y Clases de Control (que son tipos de Clases –clases de Clases-), un tema que abordaré en el capítulo correspondiente a los Diagramas de Clases. También se pueden crear estereotipos para eventualmente cualquier tipificación no existente en el lenguaje de manera nativa (para tablas de una base de datos, para procedimientos almacenados, para *triggers*, para páginas ASP.NET, para procesos de carga y transformación de datos, entre muchos otros).

El Lenguaje de Restricción de Objetos (OCL –*Object Constraint Language*) es un lenguaje íntimamente vinculado con UML. Se usa precisamente para declarar o establecer reglas para los elementos de un modelo (para las clases, por ejemplo). OCL llena el vacío que deja el uso del lenguaje natural cuando lo usamos para definir esas condiciones o semántica adicional requerida en todo modelo, uso que casi siempre causa ambigüedades o faltantes en el modelo. Por tales motivos, OCL es otro aspecto de importancia a la hora de estudiar UML y de modelar, por supuesto.

Lo Que NO ES UML

En la primera parte de estos prolegómenos hice una analogía de UML con los lenguajes de programación tradicionales como Visual C# o Java; sin embargo, insisto, UML no es un lenguaje de programación visual, en el sentido de que no tiene todo el soporte visual y semántico necesario para reemplazar cualquiera de esos lenguajes. Sí, les hablé de un UML ejecutable, pero ese será tema de otro tratado.

Adicionalmente, UML no tiene ninguna relación con las herramientas de modelado existentes que lo soportan. Si bien es cierto que los documentos de UML incluyen algunos *tips* para los vendedores de herramientas en materia de alternativas de implementación, esa documentación no apunta a ninguna característica ni necesidad específica del lenguaje. Por supuesto esta relación tampoco la tiene con los lenguajes de programación en que estas herramientas generan código fuente.

También, en los mitos que expuse en la pasada entrega, afirmaba que UML no es un proceso. Eso es correcto. Intencionalmente, UML se diseñó independiente de cualquier proceso y definir un proceso estándar no fue un objetivo de OMG al diseñar el lenguaje. No obstante, y puesto que la adopción de un proceso es una clave discriminatoria entre proyectos hiper-productivos y aquellos que no son exitosos, usar UML en el marco de un proceso de desarrollo como RUP, MSF o cualquier proceso Ágil (*Agile Modeling*, *eXtreme Programming, SCRUM, Crystal*), es una garantía definitiva para asegurar la calidad de los modelos y de todos los artefactos producidos con la notación.

UML 2.0

La versión actual de UML es la 1.5. Desde su adopción en 1997 UML ha sufrido varias revisiones y modificaciones menores hasta esta versión 1.5 de Marzo de 2003. Sin embargo, esa primera versión del lenguaje está dando señas de cansancio en cuanto que en los últimos años las necesidades de desarrollo de software han evolucionado de sistemas cliente/servidor a sistemas Web distribuidos, de unos pocos usuarios a cientos o miles de ellos, de técnicas de desarrollo orientadas a objetos a técnicas basadas en componentes (y últimamente a técnicas orientadas a aspectos) y, aunque en sus revisiones, se ha

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

tratado de incorporar algunas de estas características, finalmente UML 1.x no resistió los requisitos cambiantes y novedosos en materia de modelado de software que trajo consigo el nuevo milenio.

Afortunadamente, durante los últimos tres años OMG ha venido trabajando en el diseño de la versión 2 de UML. Este trabajo tuvo tres fases: En la primera fase, OMG solicitó propuestas para UML 2.0 durante el año 2000. En la segunda fase, varios equipos respondieron con propuestas iniciales durante el año 2001. En 2002, estos equipos combinaron sus agendas y trabajos para finalizar sus propuestas para la especificación de UML 2.0. El pasado 6 de junio, OMG votó completo la versión de UML 2.0 (30 votos a favor, 0 en contra)

En un próximo artículo les hablaré de las novedades que trae esta nueva versión, lo mismo que referencias a documentación de interés sobre el asunto.

Mitos Sobre UML

No puedo terminar esta primera disertación sobre el Lenguaje de Modelado Unificado sin antes tratar, amigo lector, de desmitificar algunas de las creencias, a manera de fe, que tenemos quienes a diario caminamos por los vericuetos espinosos de las tecnologías informáticas.

He aquí algunos de ellos.

1. UML es un proceso. Falso. UML es una notación estándar, adoptada por OMG para expresar modelos de sistemas. Como notación, puede ser usada en cualquier proceso de desarrollo de software, como RUP (*Rational Unified Process*), AM (*Agile Modeling*), XP (*eXtreme Programming*), EUP (*Enterprise Unified Process*) y cualquier derivación de estos. Cada uno de estos procesos dice cómo y cuándo usar y cuáles modelos son útiles.

2. UML es para generar código. Falso. UML es un lenguaje para visualizar abstracciones de los componentes de un sistema de software. Sirve para entender el sistema que estoy construyendo, no para generar código fuente. Existen herramientas que, a partir de uno o más de los diagramas construidos, son capaces de generar cierta cantidad de código (normalmente "esqueletos" o plantillas) para una aplicación.

3. La decisión de usar UML para modelar depende del lenguaje a usar para programar. Falso. Recientemente me preguntaban en un foro si tenía experiencia usando UML con Cobol. Mi respuesta, además de negativa, fue categórica: no es necesario preocuparse del lenguaje de programación cuando estamos modelando con UML (ni con cualquier otra notación), a no ser que queramos aprovechar las características de generación de código de algunas de las herramientas de modelado que soportan UML. En todo caso, la mayoría de estas herramientas sólo producen plantillas (esqueletos) de código para llenar (algunas, como Rational® XDE, permiten del diseñador la aplicación de patrones de diseño y la generación del código fuente respectivo, más rico que simples plantillas, pero el uso de una herramienta como esta requiere de un amplio conocimiento no solo en Análisis, Diseño y Programación Orientada a Objetos, sino también en el peliagudo tema de los Patrones de Diseño y otros conceptos "avanzados"). Lo que sí debemos

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

asegurar es que la versión de Cobol, o de cualquier otro lenguaje que usemos para programar, permita la orientación a objetos (ya existen versiones de Cobol OO, inclusive hace un par de años existe un Cobol.Net)

4. UML se usa sólo con Rational Unified Process. Falso. Puesto que UML fue creado, como reza el comercial, por los mismos creadores de RUP, cientos de desarrolladores que han adoptado o usan otros procesos de desarrollo como MSF, SCRUM, XP, o cualquiera de los que he mencionado antes, piensan que no es posible usar UML como notación. He insistido en estos prolegómenos que UML es independiente del proceso (al igual que del lenguaje de programación) que se use para llevar a cabo e implantar una solución de software. Es más, algunos casos muestran que UML ha sido usado también para modelar sistemas que no son software con procesos que no tienen nada que ver con el desarrollo de software.

Lo que Sigue

En el siguiente artículo relacionaré en detalle el modelo de Casos de Uso, incluyendo Actores y el Manejo de Requisitos.

Más adelante, los pormenores del resto de modelos. Y dada su proximidad, trataré estos temas a la luz de la versión 2.0 de UML, aprobada hace poco por OMG.

Conclusiones y Recomendaciones

Aunque este artículo es apenas una presentación de los aspectos básicos de UML, los invito a que vayan más allá, las referencias que siguen son un buen punto para continuar.

Espero además que mis palabras sean un estímulo para que empiecen a aplicar los conocimientos que han adquirido a través de estas lecturas en sus propios proyectos. Por supuesto, una vez experimentados en la materia, compartan y multipliquen las mejores prácticas, los usos más comunes, los errores más frecuentes, los documentos que encuentren y se puedan distribuir.

Referencias

1. prolegómeno. m. Tratado que se pone al principio de una obra o escrito, para establecer los fundamentos generales de la materia que se ha de tratar después. U. m. en pl. http://www.gazafatonario.com.
2. OMG01: Object Management Group, 2001. *OMG Unified Modeling Language Specification*. http://www.omg.org.
3. http://www.informit.com/content/images/0201748045/samplechapter/mellorch01.pdf
4. En http://www.objectsbydesign.com/tools/umltools_byCompany.html se enumeran algunas de estas herramientas y hace una pequeña revisión de cada una, incluyendo proveedor, versión, plataforma y precio.
5. http://www.rational.com/uml/resources/documentation/index.jp, aquí se ofrecen versiones de la actual especificación de UML.

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.

6. http://www.aaii.org.co/html/modules.php?name=Forums. Foro de la Asociación Antioqueña de Ingeniería Informática, donde hay una categoría sobre Ingeniería del Software en la que se tratan temas como el presente.

Todos los Derechos Reservados ® 2003 por Luis Antonio Salazar Caraballo.