Modelado de Sistemas com UML

Popkin Software and Systems

Modelado de Sistemas com UML
por Popkin Software and Systems

Tabla de contenidos

Capítulo 1. Introducción

Esta guía introduce el Lenguaje Unificado de Modelado (UML), versión 1.1. Analiza los diagramas que componen UML y ofrece acercamientos a casos de uso guiados sobre cómo estos diagramas se usan para modelar sistemas. La guía también trata los mecanismos de extensibilidad de UML, los cuales permiten ampliar su notación y su semántica. También sugiere la extensión de UML mediante dos técnicas no incorporadas: tarjetas CRC para análisis guiados por la responsabilidad, y diagramas de Entidad de Relación (ER) para modelar bases de datos relacionales.

Capítulo 2. ¿Qué es UML?

El Lenguaje Unificado de Modelado preescribe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan. Mientras que ha habido muchas notaciones y métodos usados para el diseño orientado a objetos, ahora los modeladores sólo tienen que aprender una única notación.

UML se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware, y organizaciones del mundo real. UML ofrece nueve diagramas en los cuales modelar sistemas.

- Diagramas de Casos de Uso para modelar los procesos 'business'.
- Diagramas de Secuencia para modelar el paso de mensajes entre objetos.
- Diagramas de Colaboración para modelar interacciones entre objetos.
- Diagramas de Estado para modelar el comportamiento de los objetos en el sistema.
- Diagramas de Actividad para modelar el comportamiento de los Casos de Uso, objetos u operaciones.
- Diagramas de Clases para modelar la estructura estática de las clases en el sistema.
- Diagramas de Objetos para modelar la estructura estática de los objetos en el sistema.
- Diagramas de Componentes para modelar componentes.
- Diagramas de Implementación para modelar la distribución del sistema.

UML es una consolidación de muchas de las notaciones y conceptos más usadas orientados a objetos. Empezó como una consolidación del trabajo de Grade Booch, James Rumbaugh, e Ivar Jacobson, creadores de tres de las metodologías orientadas a objetos más populares.

En 1996, el Object Management Group (OMG), un pilar estándar para la comunidad del diseño orientado a objetos, publicó una petición con propósito de un metamodelo orientado a objetos de semántica y notación estándares. UML, en su versión 1.0, fue propuesto como una respuesta a esta petición en enero de 1997. Hubo otras cinco propuestas rivales. Durante el transcurso de 1997, los seis promotores de las propuestas, unieron su trabajo y presentaron al OMG un documento revisado de UML, llamado UML versión 1.1. Este documento fue aprobado por el OMG en Noviembre de 1997. El OMG llama a este documento OMG UML versión 1.1. El OMG está actualmente en proceso de mejorar una edición técnica de esta especificación, prevista su finalización para el 1 de abril de 1999.

2.1. UML ofrece notación y semántica estándar

UML preescribe una notación estándar y semánticas esenciales para el modelado de un sistema orientado a objetos. Previamente, un diseño orientado a objetos podría haber sido modelado con cualquiera de la docena de metodologías populares, causando a los revisores tener que aprender las semáticas y notaciones de la metodología empleada antes que intentar entender el diseño en sí. Ahora con UML,

diseñadores diferentes modelando sistemas diferentes pueden sobradamente entender cada uno los diseños de los otros.

2.2. UML no es un Método

Aun así, UML no preescribe un proceso o método estándar para desarrollar un sistema. Hay varias metodologías existentes; entre las más populares se incluyen las siguientes:

- *Catalysis*: Un método orientado a objetos que fusiona mucho del trabajo reciente en métodos orientados a objetos, y además ofrece técnicas específicas para modelar componentes distribuidos.
- *Objetory*: Un método de Caso de Uso guiado para el desarrollo, creado por Ivar Jacobson.
- *Shlaer/Mellor*: El método para diseñar sistemas de tiempo real, puesto en marcha por Sally Shlaer y Steven Mellor en dos libros de 1991, Ciclos de vida de Objetos, modelando el Mundo en Estados y Ciclos de vida de Objetos, Modelando el mundo en Datos (Prentice Hall). Shlaer/Mellor countinúan actualizando su método continuamente (la actualización más reciente es el OOA96 report), y recientemente publicaron una guía sobre cómo usar la notación UML con Shlaer/Mellor.
- *Fusion*: Desarrollado en Hewlett Packard a mediados de los noventa como primer intento de un método de diseño orientado a objetos estándar. Combina OMT y Booch con tarjetas CRC y métodos formales. (www.hpl.hp.com/fusion/file/teameps.pdf)
- *OMT*: La Técnica de Modelado de Objetos fue desarrollada por James Rumbaugh y otros, y publicada en el libro de gran influencia "Diseño y Modelado Orientado a Objetos" (Prentice Hall, 1991). Un método que propone análisis y diseño ’iterative’, más centrado en el lado del análisis.
- *Booch*: Parecido al OMT, y también muy popular, la primera y segunda edición de "Diseño Orientado a Objetos, con Aplicaciones" (Benjamin Cummings, 1991 y 1994), (Object-Oriented Design, With Applications), detallan un método ofreciendo también diseño y análisis ’iterative’, centrándoso en el lado del diseño.

Además, muchas organizaciones han desarrollado sus propias metodologías internas, usando diferentes diagramas y técnicas con orígenes varios. Ejemplos son el método Catalyst por Computer Sciences Corporation (CSC) o el Worlwide Solution Design and Delivery Method (WSDDM) por IBM. Estas metodologías difieren, pero generalmente combinan análisis de flujo de trabajo, captura de los requisitos, y modelado de negocio con modelado de datos, con modelado de objetos usando varias notaciones (OMT, Booch, etc), y algunas veces incluyendo técnicas adicionales de modelado de objetos como Casos de Uso y tarjetas CRC. La mayoría de estas organizaciones están adoptando e incorporando el UML como la notación orientada a objetos de sus metodologías.

Algunos modeladores usarán un subconjunto de UML para modelar ’what they’re after’, por ejemplo simplemente el diagrama de clases, o solo los diagramas de clases y de secuencia con Casos de Uso. Otros usarán una suite más completa, incluyendo los diagramas de estado y actividad para modelar sistemas de tiempo real, y el diagrama de implementación para modelar sistemas distribuidos. Aun así,

otros no estarán satisfechos con los diagramas ofrecidos por UML, y necesitarán extender UML con otros diagramas como modelos relacionales de datos y ’CRC cards’.

2.3. Extensiones UML 1.1

Los mecanismos de de extensibilidad incorporados permiten a UML ser una especie de especificación abierta que puede cubrir aspectos de modelado no especificados en el documento 1.1. Estos mecanismos permiten extender la notación y semática de UML.

2.3.1. Esteroetipos

Los estereotipos son el mecanismo de extensibilidad incorporado más utilizado dentro de UML. Un estereotipo respresenta una distinción de uso. Puede ser aplicado a cualquier elemento de modelado, incluyendo clases, paquetes, relaciones de herencia, etc. Por ejemplo, una clase con estereotipo ’actor’ es una clase usada como un agente externo en el modelado de negocio. Una clase patrón es modelada como una clase con estereotipo parametrizado, lo que significa que puede contener parámetros.

2.3.2. Extensiones de Modelado de Negocio

Un documento separado dentro de la especificación UML define clases y estereotipos de asociación específicos que extienden UML hasta cubrir conceptos de modelado de negocio. Esto incluye ’stereotyping’ una clase como un actor, un trabajador (’both internal and case’), o una entidad, y ’stereotyping’ una asociación como una comunicación simple, o una subcripción entre un origen y un objetivo.

2.3.3. Lenguaje restrictivo (constraint) de objetos (OCL)

Una imagen puede describir muchas palabras. De igual modo, un modelo gráfico puede describir una cierta parte del comportamiento, después de la cual es necesario rellenar detalles adicionales con palabras. Describiendo algo con palabras, sin embargo, casi siempre desemboca en ambiguedades; por ejemplo, "¿que quería decir cuando escribió eso?". El Lenguaje Restrictivo (constraint) de Objetos (OCL) está incorporado en UML como un estándar para especificar detalles adicionales, o precisar detalles en la estrucutura de los modelos.

Desarrollado dentro de la IBM Insurace Division como un lenguaje de modelado de negocio, el OCL es un lenguaje formal diseñado para ser fácil de leer y de escribir. OCL es más funcional que el lenguaje natural, pero no tan preciso como un lenguaje de programación - no puede ser usado para escribir lógicas de lógica de programación o control de flujo. Puesto que OCL es un lenguaje para la expresión pura, sus declaraciones están garantizadas de no tener efectos laterales - simplemente transportan un valor y nunca pueden cambiar el estado del sistema.

2.4. Más Extensiones

Dos áreas específicas que UML no cubre actualmente, ni con sus extensiones, son análisis guiados por la responsabilidad y modelado de bases de datos relacionales. Esta guía introduce estas técnicas como extensiones actuales del mundo real para UML que se deberían tener en cuenta.

2.4.1. Análisis guiados por la responsabilidad con tarjetas CRC

Una técnica muy usada para hacerse a la idea de cómo hay que pensar trantando con orientación a objetos son los análisis guiados por la responsabilidad con las tarjetas CRC (CRC - Colaborador y Responsabilidad de Clase). Con esta técnica, las clases descubiertas durante el análisis pueden ser filtradas para determinar qué clases son realmente necesarias para el sistema.

2.4.2. Modelo Relacional de datos

Aunque las bases de datos orientadas a objetos se están volviendo más populares, en el entorno de desarrollo actual, la base de datos relacional sigue siendo el método predominante para almacenar datos. Los diagramas de clases de UML se pueden usar para modelar la base de datos relacional en la que el sistema está basado, sin embargo, los diagramas tradicionales de modelado de datos capturan más información sobre la base de datos relacional y son más adecuados para modelarla. Esta guía trata el uso de Diagramas de Relaciones de Entidad (ER) como una extensión importante de UML para el modelado de bases de datos relacionales.

Capítulo 3. Una perspectiva general de UML

3.1. Una vuelta por un caso de uso

Una vez más, UML es una notación, no un método. No preescribe un proceso para modelar un sistema. No obstante, como UML incluye los diagramas de casos de uso, se le considera estar dotado de una aproximación al diseño centrada en el problema con los casos de uso. El Diagrama de Caso de Uso nos da el punto de entrada para analizar los requisitos del sistema, y el problema que necesitamos solucionar. La Figura 1 muestra un flujo general de cómo los diagramas de UML, con extensiones, interactuan en una aproximación al diseño con los casos de uso.

3.2. Casos de Uso y Diagramas de Interacción

Un caso de uso se modela para todos los procesos que el sistema debe llevar a cabo. Los procesos se describen dentro de el caso de uso por una descripción textual o una secuencia de pasos ejecutados. Los Diagramas de Actividad se pueden usar también para modelar escenarios gráficamente. Una vez que el comportamiento del sistema está captado de esta manera, los casos de uso se examinan y amplian para mostrar qué objetos se interrelacionan para que ocurra este comportamiento. Los Diagramas de Colaboración y de Secuencia se usan para mostrar las relaciones entre los objetos.

3.3. Clases y Diagramas de Implementación

Conforme se van encontrando los objetos, pueden ser agrupados por tipo y clasificados en un Diagrama de Clase. Es el diagrama de clase el que se combierte en el diagrama central del análisis del diseño orientado a objetos, y el que muestra la estructura estática del sistema. El diagrama de clase puede ser dividido en capas: aplicación, y datos, las cuales muestran las clases que intervienen con la interfaz de usuario, la lógica del software de la aplicación, y el almacentamiento de datos respectivamente. Los Diagramas de Componentes se usan para agrupar clases en componentes o módulos. La distribución general del hardware del sistema se modela usando el Diagrama de Implementación.

3.4. Tarjetas CRC (CRC cards) - Una extensión informal de UML

Como una extensión informal a UML, la técnica de las tarjetas CRC se puede usar para guiar el sistema a través de análisis guiados por la responsabilidad. Las clases se examinan, se filtran y se refinan en base a sus responsabilidades con respecto al sistema, y las clases con las que necesitan colaborar para completar sus responsabilidades.

3.5. Diagramas de Estado

El comportamiento en tiempo real de cada clase que tiene comportamiento dinámico y significativo, se modela usando un Diagrama de Estado. El diagrama de actividad puede ser usado también aquí, esta vez como una extensión del diagrama de estado, para mostrar los detalles de las acciones llevadas a cabo por los objetos en respuesta a eventos internos. El diagrama de actividad se puede usar también para representar gráficamente las acciones de métodos de clases.

Figura 1

Figura 1: Aproximación a un caso de uso guiado para el desarrollo orientado a objetos con UML, incluyendo las extensiones de tarjetas CRC y extensiones de modelado de datos.

3.6. Implementando el diseño

La implementación del sistema trata de traducir información desde múltiples modelos UML en código y estructura de bases de datos. Cuando se modela un sistema grande, es útil fragmentar el sistema en su capa ’business’ (incluyendo los objetos de la interfaz de usuario), su capa de aplicación (incluyendo los objetos de implementación), y su capa de datos (incluyendo la estrucutra de la base de datos y el acceso a objetos).

3.7. Implementando la aplicación

El Diagrama de Clase se usa para generar una estructura base del código en el lenguaje escogido. Información de los diagramas de interacción, estado, y actividad, puede ofrecer detalles de la parte procedimental del código de implementación.

3.8. Implementando el diseño de Bases de Datos

La capa de datos del diagrama de clase se puede usar para implementar direcatmente un diseño orientado a objetos de una base de datos, o, como extensión de UML, puede ser referenciado en un diagrama de relación de entidad para más análisis de relaciones de entidad. Está en el diagrama de relación de entidad (ER diagram, entity relationship) el cual relaciona entre entidades que pueden ser modeladas basadas en atributos clave. El diagrama de relación de entidad lógico ofrece una base desde la cual construir un diagrama físico representando las tablas y relaciones actuales de la base de datos relacional.

3.9. Probar teniendo en cuenta los requisitos

Los casos de uso se utilizan también para probar el sistema y ver si satisface los requisitos iniciales. Los pasos de los casos de uso van llevando a cabo para determinar si el sistema está satisfacciendo los requisitos del usuario.

Capítulo 4. Un estudio a fondo de UML

Las siguientes secciones presentan una vista más detallada al modelado con UML. Un sistema de reserva de aerolíneas simple se va a usar para mostrar los diagramas y técnicas de modelado con UML. Se cubren los siguientes puntos:

- Organizando tu sistema con paquetes
- Modelando con Casos de Uso, y usándolos para averiguar los requisitos del sistema
- Modelando con Diagramas de Secuencia y Colaboración
- Analizando y diseñando con el Diagrama de Clase, y extendiendo UML con la técnica de las tarjetas CRC
- Modelando comportamiento con Diagramas de Actividad y de Estado
- Modelando componentes de software, distribución e implementación
- Extendiendo UML con el diseño de Bases de Datos relacionales

Una de las tareas clave para modelar un sistema de sofware de grandes dimensiones es dividirlo primero en áreas manejables. Aunque estas áreas se llaman dominios, categorías o subsistemas, la idea es la misma: dividir el sistema en áreas que tengan competencias parecidas.

UML introduce la noción de un paquete como el ítem universal para agrupar elementos, permitiendo a los modeladores subdividir y categorizar sistemas. Los paquetes pueden ser usados en cualquier nivel, desde el nivel más alto, donde son usados para subdividir el sistema en dominios, hasta el nivel más bajo, donde son usados para agrupar casos de uso individuales, clases, o componentes.

Figura 2

Figura 2: Organizando el sistema mediante el uso de paquetes

4.1. Modelado de Casos de Uso

El modelado de Casos de Uso es la técnica más efectiva y a la vez la más simple para modelar los requisitos del sistema desde la perspectiva del usuario. Los Casos de Uso se utilizan para modelar cómo un sistema o negocio funciona actualmente, o cómo los usuarios desean que funcione. No es realmente una aproximación a la orientación a objetos; es realmente una forma de modelar procesos. Es, sin embargo, una manera muy buena de dirigirse hacia el análisis de sistemas orientado a objetos. Los casos de uso son generalmente el punto de partida del análisis orientado a objetos con UML.

El modelo de casos de uso consiste en actores y casos de uso. Los actores representan usuarios y otros sistemas que interaccionan con el sistema. Se dibujan como "muñecos" de palo. Actualmente representan el tipo de usuario, no una instancia de usuario. Los casos de uso representan el comportamiento del sistema, los escenarios que el sistema atraviesa en respuesta a un estímulo desde un actor. Se dibujan como elipses.

Figura 3

Figura 3: Modelado de Casos de Uso.

Cada caso de uso se documenta por una descripción del escenario. La descripción puede ser escrita en modo de texto o en un formato paso a paso. Cada caso de uso puede ser también definido por otras propiedades, como las condiciones pre- y post- del escenario --- condiciones que existen antes de que el escenario comience, y condiciones que existen después de que el escenario se completa. Los Diagramas de Actividad ofrecen una herramienta gráfica para modelar el proceso de un Caso de Uso. éstos son descritos en una sección posterior de este documento.

4.1.1. Estudiar y descubrir los requisitos

El objetivo final en cualquier diseño de software es satisfacer los requisitos del usuario para el sistema. Estos requisitos pueden ser requisitos de software, requisitos de productos, o requisitos de pruebas. La meta de capturar y comprobar los requisitos del usuario es asegurar que todos los requisitos son completados por el diseño, y que el diseño es acorde con los requisitos especificados.

Muchas veces los requisitos del sistema ya existen en forma de documentos de requisitos. Los casos de uso se utilizan para correlacionar cada escenario con los requisitos que completa. Si los requisitos no existen, modelar el sistema a través de los Casos de Uso, permite el descubrimiento de estos requisitos.

4.1.2. Organización de Diagramas de Casos de Uso

Durante el análisis de negocio (business) del sistema, puedes desarrollar un modelo de caso de uso para este sistema, y construir paquetes para representar los varios dominios de negocio (business) del sistema. Puedes descomponer cada paquete con un Diagrama de Caso de Uso que contenga los Casos de Uso de un dominio, con interacciones de actor.

4.1.3. Un Caso de Uso para cada escenario

El objetivo es construir un Diagrama de Caso de Uso para cada tipo de escenario diferente en el sistema. Cada escenario muestra una secuencia diferente de interacciones entre actores y el sistema, sin condiciones ’or’.

4.1.4. Modelar secuencias alternas a través de la relación "Extiende" (extends)

Típicamente, uno modela cada Caso de Uso con una secuencia normal de acciones. El usuario entonces considera condiciones "que si" para cada paso, y desarrolla Casos de Uso basados en estas secuencias alternas de eventos. Las secuencias alternas se modelan en casos de uso separados, los cuales están relacionados con el caso de uso original mediante una relación "Extiende" (extends). Las relacciones Extiende (extends) pueden ser pensadas como un caso de uso equivalente a herencia, en el cual el caso de uso extendido, hereda y modifica el comportamiento del caso de uso original.

4.1.5. Eliminar el modelado redundante a través de la relación "Usa" (uses)

Para eliminar el modelado redundante de buena parte del comportamiento que aparezca en varios casos de uso, la parte del comportamiento puede ser modelada en un caso de uso separado que está relacionado con los otros casos de uso mediante la relación "Usa" (uses). La relación Usa (uses) se puede pensar como un caso de uso equivalente ’of aggregation’.

Figura 4

Figura 4:: Relación caso de uso Extiende (extends) frente a relación de caso Usa (uses).

4.1.6. Ayuda en casos de uso probando el sistema frente a los requisitos

Los Casos de Uso también se utilizan para constriur scripts de prueba que se usan a su vez para comprobar que la aplicación satisface los requisitos de negocio y de sistema.

Cuando has llegado al caso de uso del nivel más bajo, podrías crear un Diagrama de Secuencia para el Caso de Uso. Con los Diagramas de Secuencia y de Colaboración puedes modelar la implementación del escenario.

4.2. Diagramas de Secuencia

El Diagrama de Secuencia es uno de los diagramas más efectivos para modelar interacción entre objetos en un sistema. Un diagrama de secuencia se modela para cada caso de uso. Mientras que el diagrama de caso de uso permite el modelado de una vista ’business’ del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes pasados entre los objetos.

Típicamente uno examina la descripción de un caso de uso para determinar qué objetos son necesarios para la implementación del escenario. Si tienes modelada la descripción de cada caso de uso como una

secuencia de varios pasos, entonces puedes "caminar sobre" esos pasos para descubrir qué objetos son necesarios para que se puedan seguir los pasos.

Un diagrama de secuencia muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes pasados entre los objetos como vectores horizontales. Los mensajes se dibujan cronológicamente desde la parte superior del diagrama a la parte inferior; la distribución horizontal de los objetos es arbitraria.

Figura 5

Figura 5:: Diagrama de Secuencia para un escenario

Durante el análisis inicial, el modelador típicamente coloca el nombre ’business’ de un mensaje en la línea del mensaje. Más tarde, durante el diseño, el nombre ’business’ es reemplazado con el nombre del método que está siendo llamado por un objeto en el otro. El método llamado, o invocado, pertenece a la definición de la case instanciada por el objeto en la recepción final del mensaje.

4.3. Diagramas de Colaboración

El Diagrama de Colaboración presenta una alternativa al diagrama de secuencia para modelar interacciones entre objetos en el sistema. Mientras que el diagrama de secuencia se centra en la secuencia cronológica del escenario que estamos modelando, el diagrama de colaboración se centra en estudiar todos los efectos de un objeto dado durante un escenario. Los objetos se conectan por medio de enlaces, cada enlace representa una instancia de una asociación entre las clases implicadas. El enlace muestra los mensajes enviados entre los objetos, el tipo de mensaje (sincrónico, asincrónico, simple, blanking, y ’time-out’), y la visibilidad de un objeto con respecto a los otros.

Figura 6

*Figura 6:*Diagrama de Colaboración para un grupo de Objetos

4.4. Análisis y Diseño con el Diagrama de Clase

El Diagrama de Clase es el el diagrama principal de diseño y análisis para un sistema. En él, la estrucutra de clases del sistema se especifica, con relaciones entre clases y estructuras de herencia. Durante el análisis del sistema, el diagrama se desarrolla buscando una solución ideal. Durante el diseño, se usa el mismo diagrama, y se modifica para satisfacer los detalles de las implementaciones.

4.4.1. Desarrollo de Diagramas de Clase durante el análisis

4.4.1.1. Aproximación a un Caso de Uso guiado

En una aproximación a un Caso de Uso guiado hacia el análisis orientado a objetos, el diagrama de clases se desarrolla a través de información obtenida en los Casos de Uso, Diagramas de Secuencia y Diagramas de Colaboración. Los objetos encontrados durante el análisis son modelados en términos de la clase a la que instancian, y las interacciones entre objetos son referenciados a relaciones entre las clases instanciadas.

Figura 7

Figura 7: Diagrama de Clase durante la fase de análisis

4.4.1.2. Extensión guiada por la responsabilidad

La técnica de la tarjeta CRC se usa a veces como una extensión a UML para análisis guiados por la responsabilidad. Las definiciones de clase son refinadas basándose en las responsabilidades de clase y en otras clases con las que colabora para completar sus responsabilidades.

Cada clase se representa en una tarjeta índice (index card), y los diseñadores establecen los papeles (roles) de las clases en el sistema para definir su trabajo, y con qué otras necesitan colaborar para completar sus responsabilidades. Esta información se pasa directamente a un diagrama de clase; las responsabilidades coinciden con los métodos de clase, las colaboraciones se traducen en asociaciones entre clases.

Figura 8

Figura 8: Extensión informal de UML -- Tarjetas CRC para análisis guiados por la responsabilidad.

4.4.2. Diseño del sistema con Diagramas de Clase

Durante el diseño, el Diagrama de Clase se elabora para tener en cuenta los detalles concretos de la implementación del sistema.

4.4.2.1. Arquitecturas Multicapas

Una vez concienciados del diseño, estableceremos la arquitectura del sistema. Esto incluye establecer si será un sistema simple diseñado para correr en una sola máquina, un sistema ’two-tiered’ consistente en un cliente y un servidor, o un sistema ’multi-tiered’ con objetos interfaz de usuario separados de los objetos ’business’, separado de la base de datos, cada uno corriendo en plataformas distintas.

Una aproximación a dirigir el diagrama de clase para un sistema complejo es separar el diagrama en seccionesque muestren la lógica de la aplicación, el diseño de la interfaz de usuario, y las clases implicadas con el almacenamiento de los datos. Esto se puede hacer físicamente segmentando el diagrama de clase, usando diagramas separados para cada sección, o simplemente añadiendo una propiedad a cada clase que ’tracks’ cada ’tier’ al cual pertenece.

4.4.2.2. Diseño de Componentes

Un componente es un grupo de objetos o componentes más pequeños que interaccionan entre ellos y se combinan para dar un servicio. Un componente es similar a una caja negra, en la cual los servicios del componente se especifican por su interface o interfaces, sin ofrecer concimiento del diseño e implementación internas del componente. El desarrollo basado en componentes es el proceso de ensamblar la combinación correcta de componentes en la configuración correcta para llevar acabo la funcionalidad deseada para un sistema. Los componenetes se representan en el diagrama de clases de UML especificando la interfaz de una clase o paquete. Hay dos notaciones para mostrar una interfaz - una es mostrar la interfaz como una ’regular class symbol’ con el estereotipo "interfz", con una lista de operaciones soportadas por esta interfaz, detalladas en el ’operation department’ (departamento de operación). ’The alternate, shortcut notation’ es mostrar la interfaz como un circulo pequeño junto con la clase con una línea sólida, con el nombre de la interfaz en el círculo.

El ejemplo de la Figura 9 muestra que la clase ’Pasajero’ ofrece la operación move(x coord, y coord) para su apariencia en un GUI, a través de su interfaz ’Displayable’. Ambas notaciones UML de interfaz, se muestran en la figura. Además, la clase Pasajero también ofrece una opción save(store at) a través de su interfaz Persistente. Una clase de o componente para conectar con una base de datos podría usar esta interfaz.

Figura 9

Figura 9: Diseño de Componentes: Especificando la Interfaz de la Clase

4.4.2.3. Análisis y diseño ’Iterative’

El diagrama de clase se puede desarrollar en una ’iterative fashion’, a través de un ciclo repetido de análisis, diseño e implementación, y después vuelta al análisis, para empezar el ciclo de nuevo. Este proceso se suele llamar ’round-trip engineering’. El modelado de herramientas como System Architect 2001 puede facilitar este proceso permitiéndote implementar el diseño en un lenguaje como C++ o Java, y después traer de vuelta al código a al diagrama de clase, automáticamente actualizando la información contenida en el diagrama y en el ’underlying repository’.

Figura 10

Figura 10: Análisis, diseño y documentación ’Iterative’ con el Diagrama de Clase

4.5. Modelando el comportamiento de las Clases con Diagramas de Estado

Mientras los diagramas de interacción y colaboración modelan secuencias dinámicas de acción entre grupos de objetos en un sistema, el diagrama de estado se usa para modelar el comportamiento dinámico de un objeto en particular, o de una clase de objetos.

Un diagrama de estado se modela para todas las clases que se consideran con un comportamiento dinámico. En él, modelas la secuencia de estado que un objeto de la clase atraviesa durante su vida en respuesta a los estímulos recibidos, junto con sus propias respuestas y acciones.

Por ejemplo, un comportamiento de un objeto se modela en términos de en qué estado está inicialmente, y a qué estado cambia cuando recibe un evento en particular. También modelas qué acciones realiza un objeto en un estado en concreto.

Los estados representan las condiciones de objetos en ciertos puntos en el tiempo. Los eventos representan icendentes que hacen que los objetos pasen de un estado a otro. Las líneas de transición describen el movimiento desde un estado hasta otro. Cada línea de transición se nombre con el evento que causa esta transición. Las acciones ocurren cuando un objeto llega a un estado.
Figura 11

Figura 11: Modelando Comportamiento Dinámico de un objeto ’Vuelo’ con un diagrama de estado

4.6. Diagramas de Actividad

El Diagrama de Actividad es un diagrama de flujo del proceso multi-propósito que se usa para modelar el comportamiento del sistema. Los diagramas de actividad se pueden usar para modelar un Caso de Uso, o una clase, o un método complicado.

Un diagrama de actividad es parecido a un diagrama de flujo; la diferencia clave es que los diagramas de actividad pueden mostrar procesado paralelo (parallel processing). Esto es importante cuando se usan diagramas de actividad para modelar procesos ’bussiness’ algunos de los cuales pueden actuar en paralelo, y para modelar varios hilos en los programas concurrentes.

4.6.1. Usando Diagramas de Actividad para modelar Casos de Uso

Los Diagramas de Actividad ofrecen una herramienta gráfica para modelar el proceso de un Caso de Uso. Se pueden usar como un añadido a una descripción textual del caso de uso, o para listar los pasos del caso de uso. Una descripción textual, código, u otros diagramas de actividad pueden detallar más la actividad.

4.6.2. Usando Diagramas de Actividad para modelar Clases

Cuando se modela el comportamiento de una clase, un diagrama de estado de UML se suel usar normalmente para modelar situaciones donde ocurren eventos asincrónicos. El diagrama de actividad se usa conado todos o la mayoría de los elementos representan el desarrollo de los pasos dados por las acciones generadas internamente. Deberías asignar actividades a las clases antes de terminar con el diagrama de actividad.

Figura 12

Figura 12: Diagrama de Actividad

4.7. Modelando Componentes de Software

El Diagrama de Componentes se usa para modelar la estructura del software, incluyendo las dependencias entre los componentes de software, los componentes de código binario, y los componentes ejecutables. En el Diagrama de Componentes modelas componentes del sistema, a veces agrupados por paquetes, y las dependencias que existen entre componentes (y paquetes de componentes).

Figura 13

Figura 13: Modelando componentes con el Diagrama de Componentes

4.8. Modelando la Distribución y la Implementación

Los Diagramas de Implementación se usan para modelar la configuración de los elementos de procesado en tiempo de ejecución (run-time processing elements) y de los componentes, procesos y objetos de software que viven en ellos. En el diagrama 'deployment', empiezas modelando nodos físicos y las asociaciones de comunicación que existen entre ellos. Para cada nodo, puedes indicar qué instancias de componentes viven o corren (se ejecutan) en el nodo. También puedes modelar los objetos que contiene el componente.

Los Diagramas de Implementación se usan para modelar sólo componentes que existen como entidades en tiempo de ejecución; no se usan para modelar componentes solo de tiempo de compilación o de tiempo de enlazado. Puedes también modelar componentes que migran de nodo a nodo u objetos que migran de componente a componente usando una relación de dependencia con el estereotipo 'becomes' (se transforma)

Figura 14

Figura 14: Modelando la Distribución del Sistema con el Diagrama de Implementación

4.9. Diseño de Bases de Datos Relacionales -- Una extensión informal de UML

El Diagrama de Clase presenta un mecanismo de implementación neutral para modelar los aspectos de almacenado de datos del sistema. Las clases persistentes, sus atributos, y sus relaciones pueden ser implementadas directamente en una base de datos orientada a objetos. Aun así, en el entorno de desarrollo actual, la base de datos relacional es el método más usado para el almacenamiento de datos. Es en el modelado de este área donde UML se queda corto. El diagrama de clase de UML se puede usar para modelar algunos aspectos del diseño de bases de datos relacionales, pero no cubre toda la semántica involucrada en en el modelado relacional, mayoritariamente la noción de atributos clave que relacionan entre sí las tablas unas con otras. Para capturar esta información, un Diagrama de Relación de Entidad (ER diagram) se recomienda como extensión a UML.

El Diagrama de Clase se puede usar para modelar el estructura lógica de la base de datos, independientemente de si es orientada a objetos o relacional, con clases representando tablas, y atributos de clase representando columnas. Si una base de datos relacional es el método de implementación escogido, entonces el diagrama de clase puede ser referenciados a un diagrama de relación de entidad lógico. Las clases persistentes y sus atributos hacen referencia directamente a las entidades lógicas y a sus atributos; el modelador dispone de varias opciones sobre cómo inferir asociaciones en relaciones entre entidades. Las relaciones de herencia son referenciadas directamente a super-sub relaciones entre entidades en un diagrama de relación de entidad (ER diagram).

Figura 15

Figura 15: Extensión de UML -- Diseño de Bases de Datos Relacionales con el Diagrama de Relación de Entidad (ER Diagram)

Ya en el Diagrama de Relación de Entidad, el modelador puede empezar el proceso de determinar cómo el modelo relacional encaja; y qué atributos son claves primarias, claves secundarias, y claves externas basadas en relaciones con otras entidades. La idea es constriur un modelo lógico que sea conforme a las reglas de normalización de datos.

Al implementar el diseño relacional, es una estrategia encaminada a hacer referencia al diagrama de relación de entidad lógico a un diagrama físico que represente el objetivo, el RDBMS. El diagrama físico puede ser denormalizado para lograr un diseño de base de datos que tiene tiempos eficientes de acceso a los datos. Las relaciones super-sub entre entidades se resuelven por las estructuras de tablas actuales. Además, el diagrama físico se usa para modelar propiedades específicas de cada fabricante para el RDBMS. Se crean varios diagramas físicos si hay varios RDBMSs siendo ’deployed’; cada diagrama físco representa uno de los RDBMS que son nuestro objetivo.

Figura 16

Figura 16: Relaciones clave entre entidades en un Diagrama de Relación de Entidad

Capítulo 5. Uso de una Herramienta de Modelado

El intercambio de información de diseño e ideas usando la notación UML sería hecho en los medios que siempre han sido populares: pizarras, cuadernos y trozos de papel por nombrar algunos. Pero UML se sirve mejor por una herramienta de modelado, la cual puede ser usada para capturar, guardar, rechazar, integrar automáticamente información, y diseño de documentación.

Una característica que beneficia a los modeladores, UML también hace más fácil escoger una herramienta de modelado. Hace tiempo, el modelador primero tenía que selecionar una notación de metodología, y después estaba limitado a seleccionar una herramienta que la soportara. Ahora con UML como estándar, la elección de notación ya se ha hecho para el modelador. Y con todas las herramientas de modelado soportando UML, el modelador puede seleccionar la herramienta basada en las áreas claves de funcionalidad soportadas que permiten resolver los problemas y documentar las soluciones.

Como una buena caja de herramientas, una buena herramienta de modelado ofrece todas las herramientas necesarias para conseguir hacer eficientemente varios trabajos, sin dejarte nunca sin la herramienta correcta. Dentro de la estructura de diseño de sistemas descrito en esta guía, esto incluye lo siguiente:

- Soporte para toda la notación y semántica de UML
- Soporte para una cantidad considerable de técnicas de modelado y diagramas para complementar UML - incluyendo tarjetas CRC, modelado de datos, diagramas de flujo, y diseño de pantallas de usuario. Posibilidad de reutilizar información obtenida por otras ténicas todavía usadas, como modelado tradicional de procesos.
- Facilitar la captura de información en un repositorio subyacente - permitiendo la reutilización entre diagramas.
- Posibilidad de personalizar las propiedades de definición de elementos subyacentes de modelos UML.
- Permitir a varios equipos de analistas trabajar en los mismos datos a la vez.
- Posibilidad de capturar los requisitos, asociarlos con elementos de modelado que los satisfagan y localizar cómo han sido satisfechos los requisitos en cada uno de los pasos del desarrollo.
- Posibilitar la creación de informes y documentación personalizados en tus diseños, y la salida de estos informes en varios formatos, incluyenod HTML para la distribución en la Internet o Intranet local.
- Posibilidad para generar y ’reverse’ código (por ejemplo C++, Java, etc) para facilitar el análisis y diseño ’iterative’, para volver a usar código o librerías de clase existentes, y para documentar el código.

5.1. System Architect 2001

Popkin software ofrece soporte para modelar sistemas con UML en System Architect 2001. Ofrece todas las características descritar arriba para permitir el modelado eficiente de sistemas. Para más información en los distintos productos de Popkin Software, visite www.popkin.com

Capítulo 6. Sumario

UML 1.1 es un buen comienzo - ofrece a los arquitectos de sistemas una notación estándar para modelar sistemas. Ahora, igual que los arquitectos leen los planos, un modelador de objetos puede coger cualquier diseño y entender qué se está capturando. UML también es bueno para la comunidad del modelado - en vez de gastar tiempo acordando cómo expresar la información que se captura, los modeladores puden resolver el problema a mano, lo cual es diseñar el sistema.

Sin embargo, aunque UML se presta a una aproximación a los Casos de Uso guiados, UML no responde a la pregunta de cómo construir un sistema. Esta elección de qué metodología, o proceso usar tdoavía queda abierta para que el modelador lo decida o dé con él.

Desde un punto de vista notacional, UML 1.1 no es todavía la solución completa; pero se espera que UML continúe evolucionando con el tiempo. Mientras tanto, los modeladores usarán UML cono una base, extendiéndolo mediante una combinación de otras técnicas, como los análisis guiados por la responsabilidad y el modelado relacional de datos, para modelar el mundo real.

Capítulo 7. Referencias

- Entendiendo UML: La guía del desarrollador, con una aplicación java basada en web, por Paul Harmon y Mark Watson; Morgan Kauffman Publishers, Inc., 1998 (www.mkp.com/books_catalog/1-55860-465-0.asp).
- ¿Qué le falta a UML? un artículo por Scott Ambler, 'Objet Magacine', Octubre de 1997, SIGS Publications (www.sigs.com/omo/articles/ambler.html)
- Especificación UML 1.1 (Centro de Recursos de UML en www.popkin.com)
- Objetos, componentes y Estructuras con UML, The Catalysis Aproach, por Desmond F. D'Souza y Alan C. Wills, Addison Wesley Longman, 1998.