

MODELADO DE OBJETOS

Bibiana ROSSI, Paola BRITOS y Ramón GARCIA MARTINEZ,

CAPIS - Centro de Actualización Permanente en Ingeniería de Software
Escuela de Posgrado. ITBA.

{brossi,pbritos,rgm}@itba.edu.ar

0. INTRODUCCION

Este artículo es el primero de una serie de artículos que sintetizan como aplicar el paradigma de Orientación a Objetos para el Análisis y Diseño de Sistemas de Información, tomando como referencia el método OMT (Object Modelling Techniques).

En este artículo se presenta una revisión de las nociones básicas del paradigma de objetos y su correlato con el método mencionado.

1. PRESENTACION DEL MODELO

La técnica de Modelado de Objetos (Object Modeling Technique OMT) se basa en un conjunto de conceptos que definen que es Orientación a Objetos y una notación gráfica independiente.

La tecnología orientada a objetos propone una forma de pensar de modo abstracto acerca de problemas a resolver empleando conceptos del mundo real y no conceptos de computadoras. La notación gráfica propuesta ayuda al desarrollo de software visualizando el problema sin recurrir en forma prematura a la implementación.

La práctica ha demostrado que con el objeto de mantener la flexibilidad una buena técnica de diseño retrasa los detalles de la implementación hasta las últimas etapas del mismo.

El Modelado y Diseño Orientado a Objetos se funda en pensar acerca de problemas a resolver empleando modelos que se han organizado tomando como base conceptos del mundo real. La unidad básica es el objeto que combina las estructuras de datos con los comportamientos en una entidad única.

La Metodología OMT se extiende desde el análisis hasta la implementación pasando por el diseño. En primer lugar, se construye un modelo de análisis para abstraer los aspectos esenciales del dominio de la aplicación sin tener en cuenta la implementación eventual. En este modelo se toman decisiones importantes que después se completan para optimizar la implementación en segundo lugar. Los objetos del dominio de la aplicación constituyen el marco de trabajo del modelo de diseño, pero se implementan en términos de objetos del dominio de la computadora. Por

último, el modelo de diseño se implementa en algún lenguaje de programación , base de datos o hardware.

Los objetos del dominio de la aplicación y del dominio de la computadora se pueden modelar, diseñar e implementar utilizando los mismos conceptos y la misma notación orientada a objetos. Esta misma notación se usa desde el análisis hasta la implementación pasando por el diseño, de una forma tal que la información añadida en una fase de desarrollo no necesita perderse , ni ser traducida, para la próxima fase.

La esencia del desarrollo orientado a objetos es la identificación y organización de conceptos (objetos) del dominio de la aplicación. La mayor parte del esfuerzo realizado hasta el momento en la comunidad orientada a objetos ha estado centrada en temas de lenguajes de programación. Sin embargo, en cierto sentido, este énfasis supone un retroceso para la ingeniería del software al concentrarse excesivamente en mecanismos de implementación y no en el proceso de pensamiento subyacente al cual sirven de base (Análisis y Diseño). Sólo cuando se han identificado, organizado y comprendido los conceptos inherentes de la aplicación se pueden tratar en forma efectiva los detalles de las estructuras de datos y de las funciones. Es una premisa básica que los errores de las primeras fases del proceso de desarrollo tienen mucha influencia sobre el producto final y también sobre el tiempo requerido para finalizar.

El beneficio principal del desarrollo orientado a objetos no es reducir el tiempo de desarrollo; el desarrollo orientado a objetos puede requerir mas tiempo que el desarrollo convencional porque se pretende que promueva la reutilización futura y la reducción de los posteriores errores y el futuro mantenimiento. El tiempo transcurrido hasta que el código se completa por primera vez es posiblemente el mismo que el transcurrido en una aproximación convencional o, quizás ligeramente mayor. El beneficio del desarrollo orientado a objetos consiste en que las iteraciones subsiguientes son más rápidas y más fáciles que empleando un desarrollo convencional porque las revisiones están más localizadas. La práctica muestra que suelen ser necesarias menos iteraciones porque se descubren y se corrigen más problemas durante el desarrollo.

2. CONCEPTOS BASICOS

Hay varios conceptos que son propios de la orientación a objetos y otros inherentes a la tecnología. Aunque no todos son exclusivos de los sistemas orientados a objetos están bien apoyados por el paradigma.

Orientado a Objetos: significa que el software se organiza como una colección de objetos discretos que contienen tanto estructuras de datos como comportamiento.

Identidad: los datos están cuantificados en entidades discretas y distinguibles denominadas objetos. Cada objeto posee su propia identidad inherente. Pueden ser ejemplos de objetos, *un párrafo de un documento, la reina blanca del juego de ajedrez, una silla*. En otras palabras: dos objetos serán distintos aun cuando los valores de todos sus atributos (tales como el nombre y el tamaño) sean idénticos. Por ejemplo en un conjunto de 6 sillas de un mismo juego los valores de los atributos son los mismos y cada una de las sillas tiene su propia identidad.

Identificación: en el mundo real los objetos se limitan a existir, pero dentro de un entorno de computación cada objeto posee una identificación mediante la cual se puede hacer alusión a él de modo exclusivo. La identificación se puede implementar de distintas maneras que pueden ser como una dirección, un índice de una matriz, o un valor exclusivo de un atributo.

Clasificación: significa que los objetos con la misma estructura de datos (atributos) y comportamiento (operaciones) se aglutinan para formar una clase. Son ejemplos de clases: *párrafo, pieza de ajedrez*.

Clase: es una abstracción que describe propiedades importantes para una aplicación y que ignora el resto. La selección de clases es arbitraria y depende de la aplicación. Una clase contiene el molde (estructura, esquema) a partir del cual se crean los objetos que pertenecen a ella y el código que debe ejecutarse cada vez que un objeto de la clase recibe un mensaje. Una clase contiene la descripción de las características comunes de todos los objetos que pertenecen a ella: la especificación del comportamiento, la definición de la estructura interna y la implementación de los métodos.

Instancia: se dice que cada objeto es una instancia de su clase. Toda clase describe un conjunto posiblemente finito de objetos individuales. Toda instancia de la clase posee su propio valor para cada uno de los atributos pero comparte los nombres de los atributos y las operaciones con las demás instancias de la clase. Todo objeto contiene una referencia implícita a su propia clase; "sabe la clase de cosa que es". Los objetos contienen los valores de los atributos (que lo distinguen de otros objetos) y una identidad

Operación: es una acción o una transformación que se lleva a cabo o que se aplica a un objeto. *Mover, justificar a la derecha* son ejemplos de operaciones que se aplican a un objeto *párrafo*.

Método: es una implementación específica de una operación ejecutable por una cierta clase.

Polimorfismo: significa que una operación puede comportarse de modos distintos en distintas clases teniendo el mismo nombre de método. La operación *mover* se puede comportar

distinto en las clases *párrafo* y *pieza de ajedrez*. En términos prácticos, el polimorfismo permite referirse a objetos de diferentes clases por medio del mismo elemento de programa y realizar la misma operación de formas diferentes, de acuerdo al objeto a que se hace referencia en cada momento. En el mundo real una operación es simplemente, una abstracción de comportamiento análogo entre distintas clases de objetos. Cada objeto sabe llevar a cabo sus propias operaciones. Sin embargo, en un lenguaje orientado a objetos es este el que selecciona automáticamente el método correcto para implementar una operación basándose en el nombre de la operación y en la clase del objeto que está siendo afectado. El usuario de una operación no necesita ser consciente del número de métodos que existen para implementar una cierta operación polimórfica. Se pueden añadir clases sin modificar el código existente, siempre y cuando se proporcione métodos para todas las operaciones aplicables a las nuevas clases.

Herencia es compartir atributos y operaciones entre clases tomando como base una relación jerárquica. En términos generales se puede definir una clase que después se irá refinando sucesivamente para producir subclases. Todas las subclases poseen o heredan todas y cada una de las propiedades de su superclase y añaden, además, sus propiedades exclusivas. No es necesario repetir las propiedades de las superclases en cada subclase. Por ejemplo, *ventana de desplazamiento* y *ventana fija* son subclases de *ventana*. Ambas subclases heredan las propiedades de *ventana* tales como una región visible en la pantalla. La *ventana de desplazamiento* añade una *barra de desplazamiento* y un *ascensor*. La capacidad de sacar factor común a las propiedades de varias clases en una superclase común y de heredar las propiedades de la superclase puede reducir muchísimo la repetición en el diseño y en los programas siendo una de las ventajas principales de un sistema orientado a objetos.

Abstracción: consiste en centrarse en los aspectos esenciales de una entidad e ignorar sus propiedades accidentales. En el desarrollo de sistemas esto significa centrarse en lo que es y lo que hace un objeto antes de decidir cómo debería ser implementado. La capacidad de utilizar herencia y polimorfismo proporciona una potencia adicional. El uso de la abstracción durante el análisis significa tratar solamente conceptos del dominio de la aplicación y no tomar decisiones de diseño o de implementación antes de haber comprendido el problema. Un uso adecuado de la abstracción permite utilizar el mismo modelo para el análisis, diseño de alto nivel, estructura del programa, estructura de una base de datos y documentación. Un estilo de diseño independiente del lenguaje pospone los detalles de programación hasta la fase final, relativamente mecánica del desarrollo.

Encapsulamiento: denominado también *ocultamiento de información* consiste en separar los aspectos externos del objeto, a los cuales pueden acceder otros objetos, de los

detalles internos de implementación del mismo, que quedan ocultos para los demás. El encapsulamiento evita que el programa sea tan interdependiente que un pequeño cambio tenga efectos secundarios masivos. La implementación de un objeto se puede modificar sin afectar a las aplicaciones que la utilizan. Quizás sea necesario modificar la implementación de un objeto para mejorar el rendimiento, corregir un error, consolidar el código o para hacer un transporte a otra plataforma. El encapsulamiento no es exclusivo de los lenguajes orientados a objetos pero la capacidad de combinar la estructura de datos y el comportamiento de una única entidad hace que el encapsulamiento sea mas potente y claro que en los lenguajes convencionales que separan las estructuras de datos y el comportamiento.

Combinación de datos y comportamientos: el enfoque orientado a objetos tiene una sola jerarquía, jerarquía de clases. Unifica la jerarquía de estructuras de datos y jerarquía de procedimientos que presentan los enfoques convencionales. Cuando un objeto invoca una operación no necesita considerar cuántas implementaciones existen de una operación dada. El polimorfismo de operadores traslada la carga de decidir que implementación hay que utilizar llevándola del código que hace la llamada a la jerarquía de clases. El mantenimiento es mas sencillo porque el código que hace la llamada no necesita ser modificado cuando se añade una clase nueva. En el contexto de un sistema orientado a objetos la jerarquía de estructuras de datos es idéntica a la jerarquía de herencia de operaciones.

Reutilización: la herencia tanto de estructuras de datos como de comportamientos, permite compartir una estructura común entre varias subclases similares sin redundancia. Una de las principales ventajas de los lenguajes orientados a objetos es compartir código empleando la herencia. Todavía mas importante que el ahorro de código es la claridad conceptual que surge al reconocer que distintas operaciones son todas ellas, realmente, una misma cosa. Esto reduce el número de clases distintas que es preciso comprender y analizar.

El desarrollo orientado a objetos no sólo permite compartir información dentro de una aplicación sino que, además, ofrece la perspectiva de volver a utilizar diseños y códigos en futuros proyectos. Aún cuando esta posibilidad se ha hecho resaltar excesivamente como justificación de la tecnología orientada a objetos, el desarrollo orientado a objetos proporciona herramientas tales como la abstracción, encapsulado y herencia para construir bibliotecas de componentes reutilizables. Se debe tener presente que la reutilización no sucede, debe ser planeada pensando más allá de la aplicación inmediata y se debe invertir un esfuerzo adicional en lograr un diseño mas general.

Énfasis en la estructura de objetos: el desarrollo orientado a objetos pone un mayor énfasis en la estructura de objetos y hace menos hincapié en la estructura de procedimientos que las metodologías tradicionales de

descomposición funcional. En este aspecto, el desarrollo orientado a objetos es parecido a las técnicas de modelado de información que se utilizan en el diseño de bases de datos, si bien el desarrollo orientado a objetos añade el concepto de comportamiento dependiente de clase.

3. CICLO DE VIDA

OMT (Object Modelling Technique) es una metodología (y una notación gráfica) para el desarrollo orientado a objetos que consiste en construir un modelo de un dominio de aplicación añadiéndosele detalles de implementación durante el diseño del sistemas.

Esta metodología consta de las siguientes fases:

Análisis: comenzando en la descripción del problema el analista construye un modelo de la situación del mundo real que muestra sus propiedades importantes. Dicho analista debe trabajar con quien hace la solicitud para comprender el problema porque las definiciones del mismo no suelen ser completas ni correctas. El modelo de análisis es una abstracción resumida y precisa de lo que debe hacer el sistema deseado y no de la forma en que se hará. Los objetos del modelo deberán ser conceptos del dominio de la aplicación y no conceptos de implementación de la computadora tales como estructuras de datos. Un buen modelo podrá ser comprendido y criticado por expertos de la aplicación que no sean programadores. El modelo de análisis no deberá contener ninguna decisión de implementación, los objetos se describirán en términos de atributos y operaciones que son visibles para el usuario.

Diseño del sistema: se toman decisiones de alto nivel acerca de la arquitectura global. Durante el diseño, el sistema de destino se organiza en subsistemas basados tanto en la estructura del análisis como en la arquitectura propuesta. El diseñador de sistemas deberá decidir qué características de rendimiento hay que optimizar. Seleccionando una estrategia para atacar el problema y efectuando las reservas de recursos tentativas.

Diseño de objetos: se construye un modelo de diseño basándose en el modelo de análisis que lleven incorporados detalles de implementación. El diseñador añade detalles al modelo de acuerdo con la estrategia establecida durante el diseño del sistema. El foco de atención del diseño de objetos son las estructuras de datos y los algoritmos necesarios para implementar cada una de las clases. Las clases de objetos procedentes del análisis siguen siendo significativas pero se aumentan con estructuras de datos y algoritmos del dominio de la computadora seleccionados para optimizar medidas importantes de rendimiento. Tanto los objetos del dominio de la aplicación como los objetos del dominio de la computadora se describen utilizando unos mismos conceptos y una misma notación orientados a objetos aún cuando existan en planos conceptuales diferentes.

Implementación: las clases de objetos y las relaciones desarrolladas durante su diseño se traducen finalmente a un lenguaje de programación concreto, a una base de datos o a una implementación en hardware. La programación debería ser una parte relativamente pequeña del ciclo de desarrollo y fundamentalmente mecánica porque todas las decisiones importantes deberán hacerse durante el diseño. El lenguaje de destino influye en cierta medida sobre las decisiones de diseño pero éste no debería depender de la estructura final de un lenguaje de programación. Durante la implementación es importante respetar las ideas de la ingeniería del software, de tal manera que el seguimiento hasta el diseño sea sencillo y de tal forma que el sistema implementado siga siendo flexible y extensible.

Es posible aplicar conceptos orientados a objetos a lo largo del todo el ciclo de vida de desarrollo del sistema, desde el análisis hasta la implementación pasando por el diseño. Se pueden traspasar las mismas clases de una etapa a otra sin modificar la notación aunque ganarán detalles adicionales de implementación en las etapas posteriores. Siendo el punto de vista de análisis y el de implementación de "Ventana" correctos tienen propósitos distintos y representan grados de abstracción. Los mismos conceptos orientados a objetos de identidad, clasificación, polimorfismo y herencia son aplicables a todo el ciclo de desarrollo completo.

Algunas clases no forman parte del análisis sino que se presentan como parte del diseño o de la implementación. Por ejemplo, las estructuras de datos tales como árboles, tablas de dispersión y listas enlazadas no suelen estar presentes en el mundo real. Se presentan para que presten su apoyo a algoritmos concretos durante el diseño. Estos objetos de estructuras de datos se utilizan para implementar objetos del mundo real dentro de la computadora y no derivan directamente sus propiedades del mundo real.

4. MODELOS

Un modelo es una abstracción de algo, cuyo objetivo es comprenderlo antes de construirlo. Dado que los modelos omiten los detalles no esenciales es más sencillo manipularlos que manipular la entidad original. La abstracción permite enfrentarse a la complejidad. Los ingenieros, artistas y artesanos han estado construyendo modelos durante miles de años para probar los diseños antes de ejecutarlos. El desarrollo de sistemas hardware y software no es una excepción. Para construir sistemas complejos, el desarrollador debe abstraer distintas vistas del sistema, construir modelos utilizando notaciones precisas, verificar que los modelos satisfacen los requisitos del sistema y añadir, gradualmente, detalles para transformar los modelos en una implementación.

Los modelos tienen varios objetivos:

- probar una entidad física antes de construirla
- comunicación con el cliente
- visualización del conjunto
- reducción de la complejidad

La abstracción es el examen selectivo de ciertos aspectos de un problema. Su finalidad es aislar aquellos aspectos que sean importantes para algún objetivo y suprimir los aspectos que no lo sean. La abstracción siempre debe de hacerse con algún objetivo prefijado, porque el propósito determina lo que es y no es importante. Es posible efectuar muchas abstracciones diferentes de la misma cosa, dependiendo del propósito para el cual se hagan esas abstracciones.

Todas las abstracciones son incompletas e imprecisas. La realidad es una red sin costuras. Todo lo que digamos acerca de ella, cualquier descripción, será una versión reducida. Todas las palabras y lenguajes humanos son abstracciones, descripciones incompletas del mundo real. Esto no elimina su utilidad. El propósito de una abstracción es limitar al universo para que podamos hacer cosas. Al construir modelos, por tanto, no debe uno buscar la verdad absoluta, sino su adecuación para algún propósito. No existe un único modelo "correcto" de una situación, solo existen modelos adecuados o inadecuados. Un buen modelo captura los aspectos cruciales del problema y omite los demás.

La metodología OMT emplea tres clases de modelos para describir el sistema: el Modelo de Objetos que describe los objetos del sistema y sus relaciones; el Modelo Dinámico que describe las interacciones existentes entre objetos del sistema; y el Modelo Funcional que describe las transformaciones de datos del sistema. Todos los modelos son aplicables en la totalidad de las fases del desarrollo y van adquiriendo detalles de implementación a medida que progresa el desarrollo.

Una descripción completa del sistema requiere los tres modelos. Un procedimiento típico de software contiene estos tres aspectos:

- utiliza estructuras de datos (modelo de objetos),
- secuencia las operaciones en el tiempo (modelo dinámico) y
- transforma valores (modelo funcional).

Cada modelo referencia a entidades de los otros modelos, los tres modelos están relacionados entre si. Las interconexiones entre los distintos modelos son limitadas y explícitas. Los buenos diseños aíslan los distintos aspectos del sistema y limitan el acoplamiento entre ellos.

El más importante es el modelo de objetos porque es necesario para describir "qué" está cambiando o transformándose, antes de describir "cuándo" y "cómo"

cambia. El enfoque orientado a objetos se centra primordialmente en identificar objetos procedentes del dominio de la aplicación ajustándoles después los procedimientos. Soporta mejor las evoluciones de los requisitos porque está basado en el entorno subyacente del dominio de la aplicación en si, más que en los requisitos funcionales ad-hoc de un único problema.

4.1. Modelo de Objetos

Describe la estructura estática (de datos), de los objetos del sistema (identidad, atributos y operaciones) y también sus relaciones. El modelo de objetos contiene diagramas de objetos. Un diagrama de objetos es un grafo cuyos nodos son clases de objetos y cuyos arcos son relaciones entre las clases. El diagrama contiene clases de objetos organizados en jerarquías que comparten una estructura y comportamiento comunes y que están asociadas a otras clases. Estas clases definen los atributos que lleva cada instancia de objeto y las operaciones que efectúa o sufre cada uno. En cada instancia de la clase se guardan los valores de esos atributos.

4.2. Modelo Dinámico

Describe los aspectos de comportamiento (de control) de un sistema que cambian con el tiempo. El modelo dinámico se utiliza para especificar e implementar los aspectos del control del sistema. Los modelos dinámicos contienen diagramas de estados. Un diagrama de estados es un grafo cuyos nodos son estados y cuyos arcos son transiciones entre estados causadas por sucesos o eventos.

Se especifican en este modelo la temporización y secuencia de operaciones (sucesos que marcan los cambios, secuencias de sucesos, estados que definen el contexto para los sucesos), y la organización de sucesos y de estados. El modelo dinámico captura el control, aquel aspecto de un sistema que describe las secuencias de operaciones que se producen sin tener en cuenta lo que hagan las operaciones, aquello a lo que afecten o la forma en la que estén implementadas. Las acciones de los diagramas de estado se corresponden con funciones procedentes del modelo funcional; los sucesos de un diagrama de estado pasan a ser operaciones que se aplican a objetos dentro del modelo de objetos.

4.3. Modelo Funcional

Describe las transformaciones (de función), de valores de datos que ocurren dentro del sistema, captura lo que hace el sistema, independientemente de cuando se haga o de la forma en que se haga. El modelo funcional contiene diagramas de flujo de datos. Un diagrama de flujo de datos es un grafo cuyos nodos son procesos y cuyos arcos son flujos de datos, se muestra las dependencias entre los valores y el cálculo de valores de salida a partir de los de entrada y de funciones, sin considerar cuando se

ejecutan las funciones, ni siquiera si llegan a ejecutarse. Las funciones se invocan como acciones en el modelo dinámico y se muestran como operaciones que afectan a objetos en el modelo de objetos.

5. CONCLUSION

La característica fundamental de OMT es tener en cuenta todos los principios del paradigma de orientación a objetos y refuerza particularmente la necesidad de modelos consistentes de análisis y diseño del negocio como paso previo y fundamental al desarrollo de la programación y la definición de las bases de datos.

6. BIBLIOGRAFIA

Booch, G. *Análisis y Diseño Orientado a Objetos con Aplicaciones*. Addison-Wesley. 1996.

Jacobson, I. *Object Oriented Software Engineering*. Addison-Wesley. 1992.

Martin, J. y Odell, J. *Análisis y Diseño Orientado a Objetos*. Prentice Hall. 1994.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. y Lorensen, W. *Modelado y Diseño Orientado a Objetos*. Prentice Hall. 1996.