EMA 56

ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

ÍNDICE

1. INTRODUCCIÓN

Con el paso del tiempo se han ido decantando las impurezas y clarificando el panorama de la orientación a objetos.

Algunas de las ideas que aparecen con más nitidez son:
Se ha perdido demasiado tiempo en vanas discusiones sobre lo que es programación, diseño y lo que es análisis. La separación estricta entre estas fases sobre todo entre las de análisis y diseño, era más bien una argucia propia de los métodos clásicos que no tienen por qué extenderse al mundo de los objetos.
Las ideas principales del análisis O.O presentes en la mayoría de los métodos son:
- Las de clase y asociación entre clase.
- La partición en subsistemas utilizando los conceptos de generalización, especialización y agregación.
- La importancia de recoger los requisitos del sistema a partir del estudio de la interacción entre usuarios y sistema.

La orientación a objetos supone una evolución ene el nivel de abstracción con el que trabaja la programación funcional. Se pretende el condicionamiento de la estructura del sistema informático sobre la representación de la realidad, acercando los métodos de representación al mismo objeto representado y no al revés.
De este modo propone un método de la descomposición, no basado únicamente en lo que hace el sistema, sino que tienen muy en cuenta también cómo es el sistema.

La definición del sistema se hace ya empleando metáforas obtenidas del mismo universo real que se pretende representar, con lo que se independiza de su implementación informática final. Par ello se facilita la migración del producto fianl, que ya no depende de la plataforma donde se implementa.

2. ELEMENTOS DEL MODELO DE OBJETOS

Diremos que el desarrollo orientado a objetos considera a un programa como una colección de agentes ampliamente autónomos, llamados **objetos .**Cada objeto es responsable de tareas específicas. Mediante la interacción de los objetos avanza la ejecución del programa. Por tanto, en cierto sentido, el desarrollo O.O es ni más ni menos que la simulación de un universo modelo.

2.2. ELEMENTOS DEL MODELO DE OBJETOS

Diremos que el desarrollo orientado a objetos considera a un programa como una colección de agentes ampliamente autónomos, llamados **objetos.** Cada objeto es responsable de tareas específicas.
Mediante la interacción de los objetos avanza la ejecución del programa. Por lo tanto, en cierto sentido, el desarrollo O.O es ni más ni menos que ls simulación de un universo modelo.

2.2.1 Definición de objeto.

Un objeto es una encapsulación del estado (valores de datos, propiedades) y del comportamiento (métodos) correspondientes a la representación informática de un problema de mayor nivel.

El término **encapsulación** se refiere al aoslamiento de los componentes del objeto respecto al exterior del mismo.

2.2.2 Clasificación

Cada objeto es un ejemplar de alguna clase, se suele decir que el objeto es una **instancia de la clase**. Es decir la clase es una generalización del concepto de objeto. El comportamiento de los objetos queda de este modo determinado por la clase a la que pertenecen.

Todos los objetos de la misma clase invocan el mismo método como respuesta a una solicitud similar.

2.2.3 Comunicación entre objetos : los mensajes

Denominaremos **mensaje** a la llamada recibida por un objeto desde fuera o dentro de sí mismo. Así pues un objeto se activará mediante la invocación a un método como respuesta a un mensaje recibido.
De aquí podemos redefinir.el concepto de método como la implementación en una clase de un protocolo de respuesta a los mensajes dirigidos a los objetos de la misma.

2.2.4 Herencia.

Las clases pueden organizarse en un cuasi-árbol de herencia jerárquico. Las clases que se encuentran en niveles más bajos en el árbol, tienen acceso y pueden usar datos y comportamientos asociados de sus clases ascendientes. A esta posibilidad se le denomina **herencia** y supone una mejora sustancial sobre la programación modular convencional, ya que facilita sobremanera la reutilización del código junto con otros benenficios, tales como la **consistencia de la interfaz, el modelado rápido de prototipos, la ocultación de la información.**

Cuando se deriva una clase de otra, la clase hija puede heredar todas las propiedades y métodos de su antecesora.

2.2.5 Polimorfismo.

El polimorfismo de los lenguajes de programación permite al programador generar componentes reutilizables de alto nivel que pueden adaptarse para que se ajusten a diferentes aplicaciones mediante el cambio de sus componenetes de bajo nivel. Un ejemplo de polimorfismo es la definición de mensaje homónimos aplicables a distintas clase en los cuales el mensaje sabe a qué clase dirigirse en funcion de los valores recibidos para transportar.

3. REPRESENTACIÓN DE LOS ELEMENTOS BÁSICOS.

El objeto es un elemento simple compuesto por tres elementos característicos: estado , comportamiento e identidad.
El objeto encapsula su contenido protegiéndolo del exterior con lo que se consiguen cumplir dos principios de la programación robusta: maximizar la cohesión dentro del objeto y minimizar el acoplamiento con el exterior.
Los objetos cumplen a cabo su papel de sacar adelante el funcionamiento de la aplicación mediante el intercambio de mansajes con otros objetos. Es entonces cuando se pone de manifiesto su comportamiento.

En UML un objeto se representa mediante un rectángulo con una etiqueta, el nombre del objeto, inscrita y subrayada.
Los enlaces (líneas continuas) simbolizan la relación (los objetos "se conocen").
El estado de un objeto está constituido por los valores de sus atributos en un momento dado.

Los objetos no son estáticos sino que manifiestan un comportamiento.
Los objetos interactúan con el fin de sacar adelante el funcionamiento del programa . Según el modo en que se comportan los objetos se suelen clasificar en actores, servidors y agentes.

ACTORES, están siempre en el origen de una interacción. Generalmente son objetos activos que poseen un hilo de ejecución y pasan el testigo a otros ojetos.
SERVIDORES, no generan mensajes, sino que los reciben. Su misión suele ser de proveer de servicios a los actores.
AGENTES, son capaces de enviar y recibir mensajes. Actúan como intermediarios entre actores y servidores.

Los mensajes actúan como soporte de la comunicación entre objetos,que de otra forma permanecerían incomunicados entre sí.
Integran flujos de control y flujos de datos.
Las principales categorías de los mensajes son:

- constructores
- destructores
- selectores
- modificados

Los mensajes determinan también el modo en que se sincroniza el comportamiento de los objetos. Es posible distinguir varias formas de sincronización (secuencial, síncrona, asíncrona esperada etc).

CLASES

Una clase describe el ámbito de definición de un conjunto de objetos con estructura, conducta , relaciones y semántica comunes.
La clase es una abstracción que representa a los objetos, cada uno de los cuales sólo adquirirá significado al crearse (instanciarse) a partir de la definición de clase.

Las clases se relacionan entre sí mediante asociaciones.

4. NOTACIÓN UML

Un caso de uso es una interacción entre un usuario y un sistema informático.
Los objetivos de los casos de uso son los siguientes:

capturar los requisitos funcionales del sistema y expresarlos desde el punto de vista del usuario
guiar todo el proceso de desarrollo del sistema de información.

Un caso de uso recoge en un primer momento, una descripción general. Esta descripción reflejará posiblemente uno o varios requisitos funcionales del sistema o formará parte de algún requisito.

Se puede completar la descripción definiendo cuáles son las precondiciones y postcondicones del sistema.

Un escenario es cada uno de los distintos caminos por los que puede evolucionar un cado de uso, dependiendo de las condiciones que se van dando en su realización.

Diagramas de casos de uso:

Estos diagramas presentan dos tipos de elementos fundamentales:

Un actor es algo o alguuien que se encuentra fuera del sistema y que interactúa con él.
Un caso de uso representa el comportamiento que ofrece el sistema de información desde el punto de vista del usuario. Un diagrama de casos de uso presenta relaciones entre actores y casos de uso o entre casos de uso.

Las relaciones entre casos de uso es una relación unidireccional. Esta relación puede presentar uno de los siguientes estereotipos : “usa” y “extiende”;

La relación “usa” se utiliza cuando se quiere reflejar un comportamiento común en varios casos de uso.
La relación “extiende” se utiliza cuando se quiere reflejar un comportamiento opcional de un caso de uso.

El diagrama de casos de uso es un grafo que presenta actores, casos de uso y relaciones entre estos elementos.

El diagrama de clases recoge las clases propiamente dichas y sus asociaciones, pero sin especificar datos sobre la evolución temporal, misión que queda reservada a los diagramas de transición de estados.

Se pueden incluir paquetes como un elemento del diagrama.Cada paquete representará la agrupación de un conjunto de clases a las que sustituirá en el gráfico.

Los elementos básicos del diagrama son las clases y las asociaciones o relaciones. También conviene destacar el concepto de interfaz.

CLASES

Para la determinación de cuales son las clases del modelo en elaboración, no existe ningún método inmediato aunque sí algunas recomendaciones. La búsqueda de sustantivos que hagan referencia al ámbito del sistema de información y que se encuentren en los documentos de las especificaciones de requisitos y los casos de uso.

Se llama asociación a un vínculo existente entre clases.

Los tipos de asociación son:
La generalización, relación de herencia entre clases.
La relación de dependencia, denota una relación semántica entre dos elementos.
La agregación, las clases que se agregan son parte de otra clase de un nivel de complejidad superior,

INTERFAZ

Un interfaz es una especificación de la semántica de un conjunto de operaciones de una clase o paquete que son visibles desde otras clases o paquetes.

5. CONSIDERACIONES FINALES

5.1. Beneficios que se obtienen del desarrollo con OOP

Día a día los costos del Hardware decrecen. Así surgen nuevas áreas de aplicación cotidianamente: procesamiento de imágenes y sonido, bases de datos multimedia, automatización de oficinas, ambientes de ingeniería de software, etc. Aún en las aplicaciones tradicionales encontramos que definir interfaces hombre-máquina tipo Windows suele ser bastante conveniente.

Lamentablemente, los costos de producción de software siguen aumentando; el mantenimiento y la modificación de sistemas complejos suele ser una tarea trabajosa; cada aplicación, (aunque tenga aspectos similares a otra) suele encararse como un proyecto nuevo, etc.

Todos estos problemas aún no han sido solucionados en forma completa. Pero como los objetos son portables (teóricamente) mientras que la herencia permite la reusabilidad del código orientado a objetos, es más sencillo modificar código existente porque los objetos no interaccionan excepto a través de mensajes; en consecuencia un cambio en la codificación de un objeto no afectará la operación con otro objeto siempre que los métodos respectivos permanezcan intactos. La introducción de tecnología de objetos como una herramienta conceptual para analizar, diseñar e implementar aplicaciones permite obtener aplicaciones más modificables, fácilmente extensibles y a partir de componentes reusables. Esta reusabilidad del código disminuye el tiempo que se utiliza en el desarrollo de software y hace que dicho desarrollo sea más intuitivo porque la gente piensa naturalmente en términos de objetos más que en términos de algoritmos de software.

5.2. Problemas derivados de la utilización de OOP en la actualidad

Un sistema orientado a objetos, por lo visto, puede parecer un “paraíso virtual”. El problema, sin embargo, surge en la implementación de tal sistema. Muchas compañías perciben los beneficios de un sistema orientado a objetos e invierten gran cantidad de recursos en él, luego comienzan a darse cuenta que han impuesto una nueva cultura que es ajena a los programadores actuales. Específicamente los siguientes temas suelen aparecer repetidamente:

Curvas de aprendizaje largas. Un sistema orientado a objetos ve al mundo en una forma única. Involucra la conceptualización de todos los elementos de un programa, desde subsistemas a los datos, en la forma de objetos. Toda la comunicación entre los objetos debe realizarse en la forma de mensajes. Esta no es la forma en que están escritos los programas orientados a objetos actualmente; al hacer la transición a un sistema orientado a objetos la mayoría de los programadores deben capacitarse nuevamente antes de poder usarlo.

Dependencia del lenguaje. A pesar de la portabilidad conceptual de los objetos en un sistema orientado a objetos, en la práctica existen muchas dependencias. Muchos lenguajes orientados a objetos están compitiendo actualmente para dominar el mercado. Cambiar el lenguaje de implementación de un sistema orientado a objetos no es una tarea sencilla; por ejemplo C++ soporta el concepto de herencia múltiple mientras que Smalltalk no lo soporta; en consecuencia la elección de un lenguaje tiene ramificaciones de diseño muy importantes.

Determinación de las clases. Una clase es un molde que se utiliza para crear nuevos objetos. En consecuencia es importante crear el conjunto de clases adecuado para un proyecto. Desgraciadamente la definición de las clases es más un arte que una ciencia. Si bien hay muchas jerarquías de clase predefinidas usualmente se deben crear clases específicas para la aplicación que se este desarrollando. Luego, en 6 meses ó 1 año nos damos cuenta de que las clases que se establecieron no son posibles; en ese caso será necesario reestructurar la jerarquía de clases devastando totalmente la planificación original.

Rendimiento. En un sistema donde todo es un objeto y toda interacción es a través de mensajes, el tráfico de mensajes afecta al rendimiento. A medida que la tecnología avanza y la velocidad de microprocesamiento, potencia y tamaño de la memoria aumentan, la situación mejorará; pero en la situación actual, un diseño de una aplicación orientada a objetos que no tiene en cuenta el rendimiento no será viable comercialmente.

Idealmente, habría una forma de atacar estos problemas eficientemente al mismo tiempo que se obtienen los beneficios del desarrollo de una estrategia orientada a objetos. Debería existir una metodología fácil de aprender e independiente del lenguaje, y fácil de reestructurar que no drene el rendimiento del sistema

6. BIBLIOGRAFÍA

Alfonseca, M.
Programación orientada a objetos
Anaya Multimedia, 1992

O'Brien, Stephen
Turbo Pascal 6
Mc Graw-Hill, 1991

Atkinson, Lee
Programación en Borland C++
Anaya Multimedia, 1992