

El estándar UML 2.0:
aspectos básicos para la documentación

Mayo de 2012

Índice general

1. Introducción a las vistas	5
2. La vista estática	7
2.1. Elementos básicos	7
2.2. Clasificadores	7
2.3. Relaciones	7
2.3.1. Asociación	9
2.3.1.1. Agregación y composición	9
2.3.1.2. Bidireccionalidad de las asociaciones	9
2.3.2. Generalización y herencia	9
2.3.3. Realización	9
2.3.4. Dependencia	10
2.3.5. Instancia	10
2.4. Diagramas de objetos	10
3. La vista de interacción	11
3.1. Interacciones	11
3.2. Diagramas de secuencia	11
3.2.1. Especificación de la ejecución	11
4. Vista de despliegue	13
4.1. Nodos	13
4.2. Artefactos	13

Capítulo 1

Introducción a las vistas

Una **vista** es un subconjunto de construcciones de modelado UML que representan una parte del sistema. Las vistas *no* forman parte del estándar UML, y sus conceptos se representan visualmente por medio de diagramas.

En este manual, nos centraremos en tres tipos de vista:

- la **vista estática**, que es la base de UML y describe los componentes significativos de la aplicación.
- la **vista de interacción**, en la que se proporciona una visión del comportamiento de un conjunto de objetos.
- la **vista de despliegue**, en la que se muestra la distribución física de los nodos de procesamiento.

Existen más tipos, como la vista de casos de uso o la vista de máquina de estados. No obstante, los omitiré, ya que no me parecen relevantes para lo que tenemos que hacer. Si necesitáis saber algo de alguna, avisadme para que os la transcriba.

Capítulo 2

La vista estática

La vista estática captura la estructura de los objetos, es decir, tanto las estructuras de datos que albergan como la organización de las operaciones sobre esos datos. Es importante notar que en esta vista *no* se incluyen detalles de comportamiento dinámico, que se describen mediante otras vistas.

2.1. Elementos básicos

Los elementos fundamentales de la vista estática son los siguientes:


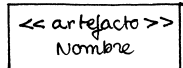

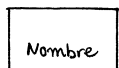
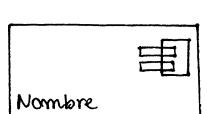
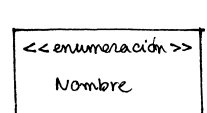
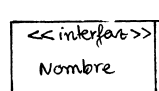
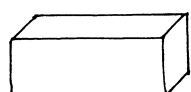
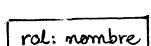
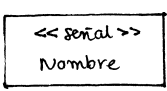

- los **clasificadores**, que sirven para describir elementos que albergan valores. Existen varios tipos: clases, interfaces, actores,...
- las **relaciones entre clasificadores**.
- los **paquetes**. Un paquete es una unidad organizativa de propósito general que alberga y gestiona el contenido de un modelo. *Todo* elemento estará contenido en un paquete.
- un **modelo**, que es el paquete raíz que contiene, de forma indirecta, todos los paquetes que describen el sistema.
- los **objetos**, que son piezas de estado identificables con un comportamiento bien definido que puede ser invocado.

2.2. Clasificadores

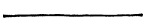
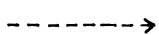
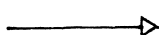
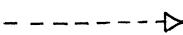
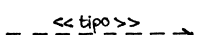
Aparecen en el cuadro 2.1.

2.3. Relaciones

Se recogen en el cuadro 2.2.

Clasificador	Función	Notación UML 2.0
actor	usuario externo al sistema	
artefacto	elemento físico del sistema (ejecutable, fichero,...)	
colaboración	relación contextual entre objetos que representan roles	
clase	concepto del sistema modelado	
componente	módulo del sistema con interfaces bien definidas	
enumeración	tipo de datos con valores literales predefinidos	
interfaz	conjunto de operaciones con nombre que caracterizan un comportamiento	
nodo	recurso computacional	
rol	papel en el contexto de una colaboración	
señal	comunicación <i>asíncrona</i> entre objetos	
tipo primitivo	descriptor de un conjunto de valores primitivos que carecen de identidad	

Cuadro 2.1: Clasificadores que utilizaremos

Relación	Qué describe	Notación UML 2.0
asociación	conexión entre instancias de clases	
dependencia	relación entre dos elementos del modelo	
generalización	relación entre una descripción más específica y otra más general	
realización	relación entre una especificación y su implementación	
uso	un elemento depende de otro para su correcto funcionamiento	

Cuadro 2.2: Relaciones definidas en UML

2.3.1. Asociación

Una **asociación** representa una relación entre los objetos de un sistema. Aunque una misma clase puede participar varias veces en cierta asociación, los objetos de esa clase no tienen por qué ser el mismo.

Cada una de las conexiones de una asociación con una clase es un **extremo**. Los extremos pueden tener roles o incluso una visibilidad, aunque su propiedad más importante es la multiplicidad.

Es importante notar que:

- una asociación puede implementarse de varias formas.
- hay que evitar las asociaciones redundantes, ya que no aportan información lógica.
- conceptualmente, una asociación es distinta de las clases que relaciona.

2.3.1.1. Agregación y composición

Una **agregación** representa una relación todo-parte. Por su parte, una **composición** es una forma más fuerte de asociación, en la que

- las partes *no* pueden existir con independencia del objeto compuesto, y
- el objeto compuesto es el único encargado de gestionar sus partes.

2.3.1.2. Bidireccionalidad de las asociaciones

Como los extremos de una asociación son distinguibles, en general una asociación *no* es simétrica, por lo que sus extremos *no* se pueden intercambiar.

Una asociación es bidireccional cuando la relación lógica funciona en ambos sentidos. En cualquier caso, esto *no* significa que cada clase conozca a la otra, ni tampoco que en una implementación sea posible acceder a una clase a través de la otra.

2.3.2. Generalización y herencia

La **generalización** es una relación taxonómica entre una descripción más general y otra más específica (llamada **hija**), que amplía la descripción general y se construye sobre ella.

Es importante notar que la descripción específica es totalmente consistente con la más general, aunque puede contener información adicional. Un ejemplo típico es la relación existente entre hipoteca y préstamo.

La generalización se utiliza fundamentalmente con **tres propósitos**:

- **principio de sustitución**: una instancia de un descendiente puede utilizarse en cualquier sitio en el que esté declarado el antecesor.
- **operaciones polimórficas**, cuya implementación se determina por la clase del objeto sobre el que se aplican.
- **herencia**. La herencia es el mecanismo mediante el cual la descripción de un objeto de una clase se construye a partir de los fragmentos de declaraciones de la clase y de los de sus antecesores. Gracias a la herencia, las partes comunes se declaran una única vez, siendo compartidas por muchas clases.

La diferencia entre la generalización y la herencia está en el uso de los atributos: la generalización afecta sólo a métodos; la herencia, a métodos y atributos.

2.3.3. Realización

Esta relación conecta un elemento del modelo con otro, que le proporciona la especificación de su comportamiento pero no su estructura ni su implementación.

Es importante notar que

- la realización no tiene por qué utilizarse únicamente con interfaces. Por ejemplo, la relación entre la versión optimizada de una clase y otra más sencilla e ineficiente es también una realización.
- las jerarquías de herencia y realización no tienen por qué guardar ningún parecido.

En algunos casos, se habla de **interfaces proporcionadas** e **interfaces obligatorias**.

- la **interfaz proporcionada** es la interfaz realizada por la clase.
- la **interfaz obligatoria** es la interfaz que rige la implementación del comportamiento interno de la clase.

2.3.4. Dependencia

Una **dependencia** indica una relación *semántica* entre dos o más elementos del modelo. Representa una situación en la que un cambio en el elemento proveedor puede requerir un cambio en el significado del elemento cliente de la dependencia.

Es importante notar que las realizaciones y generalizaciones son dependencias, pero tienen semánticas más específicas y con consecuencias más importantes. Las dependencias se usan para el resto de relaciones que no encajan en categorías más definidas. Las más importantes se recogen en el cuadro 2.3.

Relación	Función	Palabra clave
acceso	importación privada del contenido de otro paquete	access / accede
creación	una clase crea instancias de otra	create / crea
derivación	una instancia se obtiene a partir de otra	derive / deriva
envío	relación entre el emisor y el receptor de una señal	send / envía
instanciación	un método de una clase crea instancias de otra	instantiate / usa instancias
llamada	un método de una clase llama a un método de otra	call / llama
realización	correspondencia entre una especificación y su realización	realize / realiza
sustitución	la clase origen puede sustituirse por la clase destino	substitute / sustituye
uso	un elemento requiere la presencia de otro para su correcto funcionamiento. Esta relación incluye la llamada, creación, instanciación y envío, y también está abierta a otros tipos de dependencia.	use/usa

Cuadro 2.3: Principales relaciones de dependencia

2.3.5. Instancia

Una **instancia** es una entidad con identidad propia que existe en tiempo de ejecución. Las instancias *siempre* tienen un valor, que puede cambiar a lo largo del tiempo en respuesta a las operaciones sobre él.

Una **instantánea** representa la configuración específica del sistema en un instante determinado. Consta de objetos, instancias, valores y enlaces.

2.4. Diagramas de objetos

Estos diagramas representan una instantánea del sistema en un instante de tiempo determinado, y pueden especificar los valores de los objetos, tanto de forma precisa como de forma incompleta (por ejemplo, usando rangos).

Es importante notar que estos diagramas son ejemplos de sistemas, pero *no* definiciones de los mismos.

Capítulo 3

La vista de interacción

Esta vista proporciona una visión general del comportamiento de un conjunto de objetos.

3.1. Interacciones

Una **interacción** es un conjunto de mensajes que se intercambian a través de roles. En nuestro caso, las interacciones siempre tendrán lugar dentro de una colaboración.

Un **mensaje** es un flujo de control, que puede tener argumentos que transportan información del emisor al receptor. Existen dos clases de mensajes:

- las **señales**, que son comunicaciones explícitas y asíncronas.
- las **llamadas**, que constan de
 - la invocación síncrona o asíncrona de una operación, y
 - un mecanismo que devuelve el control al emisor (sólo en el caso de llamadas síncronas).

Es importante notar que sólo los eventos asociados al mismo rol se ordenan cronológicamente. El resto de eventos serán concurrentes, a no ser que exista una cadena de mensajes intermedios o se ordenen explícitamente.

3.2. Diagramas de secuencia

Sus características son las siguientes:

- el transcurso del tiempo se representa en vertical.
- cada rol se representa mediante una **línea de vida**, que será
 - **discontinua** cuando el objeto existe y no se está ejecutando su código
 - **doble** cuando el objeto existe y se está ejecutando su código.
- no se muestran intervalos exactos de tiempo
- los mensajes se representan como flechas que unen líneas de vida. El trazo de estas flechas es continuo cuando los mensajes son síncronos, y discontinuo cuando son asíncronos.

3.2.1. Especificación de la ejecución

La notación que se utiliza es la siguiente:

- la flecha de la llamada apunta al comienzo de la activación

- la recursión o llamadas a otros procedimientos apilan líneas de ejecución
- las flechas de retorno se pueden omitir, aunque suelen mostrarse por claridad.

Cuando un objeto tiene su propio hilo de control, se dice que es un **objeto activo**. Para distinguir los objetos activos, aparece una doble línea en cada lado de su símbolo en la cabecera de la línea de vida. El resto de objetos se llaman **objetos pasivos**: sólo reciben el control cuando se invoca a una de sus operaciones, y lo devuelven cuando estas terminan.

Capítulo 4

Vista de despliegue

La **vista de despliegue** muestra la dispersión física de los nodos.

4.1. Nodos

Un nodo modela un recurso computacional de tiempo de ejecución. Los nodos

- pueden tener estereotipos. Estos nos permiten distinguir los distintos tipos de recursos.
- pueden albergar artefactos.

La notación que se utiliza para representar nodos aparece en la figura 4.1.



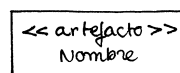
Figura 4.1: Representación de los nodos en UML 2.0

Las asociaciones entre nodos representan canales de comunicación. Para distinguir los distintos tipos de comunicación, es posible utilizar distintos estereotipos.

Finalmente, los nodos pueden tener relaciones de generalización para relacionar una descripción más general de un nodo con una variante más específica.

4.2. Artefactos

Los artefactos modelan entidades físicas: archivos, ejecutables, bases de datos, ... La notación que se utiliza para representarlos es la siguiente:



Los símbolos de los artefactos siempre se anidan en el símbolo del nodo en el que están.