

HORA 12



Diagramas de componentes

En las horas anteriores ha visto diagramas que tratan temas conceptuales. Un diagrama de clases representa un concepto, es decir, la abstracción de elementos que confluyen en una categoría; un diagrama de estados también representa un concepto, como son los cambios en el estado de un objeto.

Ahora verá el uso de un diagrama que es algo distinto a los que ha visto, y aprenderá lo correspondiente a un diagrama UML que representa a una entidad real: un componente de software.

En esta hora se tratarán los siguientes temas:

- Qué es un componente
- Componentes e interfaces
- Qué es un diagrama de componentes
- Aplicación de los diagramas de componentes
- Diagramas de componentes en el panorama del UML

Qué es un componente

Un componente de software es una parte física de un sistema, y se encuentra en la computadora, no en la mente del analista. ¿Qué puede tomarse como componente? Una tabla, archivo de datos, ejecutable, biblioteca de vínculos dinámicos, documentos y cosas por el estilo.

¿Cuál es la relación entre un componente y una clase? Imagine a un componente como la personificación en software de una clase. La clase representa una abstracción de un conjunto de atributos y operaciones. Un importante punto por recordar de los componentes y clases es que un componente puede ser la implementación de más de una clase.

Si el componente se encuentra en una computadora y es la parte funcional del sistema, ¿para qué preocuparse por él? Tendrá que modelar componentes y sus relaciones para que:

1. Los clientes puedan ver la estructura del sistema finalizado.
2. Los desarrolladores cuenten con una estructura con la cual trabajar en adelante.
3. Quienes escriban las notas técnicas y la documentación puedan entender de qué escribirán.
4. Usted se aliste para volver a utilizar los componentes.

Exploremos el último punto. Uno de los aspectos más importantes de los componentes es el potencial que tienen de volver a ser utilizados. Con las necesidades actuales de los negocios de soluciones rápidas, entre más rápido presente un sistema para producción, mayor será su competitividad. Si puede crear un componente para un sistema y puede volver a utilizarlo en otro, usted habrá contribuido a esa competitividad. Tómese el tiempo y esfuerzo para modelar un componente que lo ayude a que esta reutilización pueda llevarse a cabo.

Recordaremos la reutilización al final de la siguiente sección.

Componentes e interfaces

Cuando trate con los componentes, tendrá que tratar con sus interfaces; al tratar el tema de las clases y los objetos, hablé de las interfaces. Como recordará en la hora 2, “Orientación a objetos”, un objeto oculta al mundo exterior lo que hace (lo que se conoce como *encapsulación*, *encapsulamiento* u *ocultamiento de información*). El objeto tiene que presentar un “rostro” al mundo exterior, para que los demás objetos (incluso, potencialmente, los humanos) puedan pedirle que ejecute sus operaciones. A este “rostro” se le conoce como *interfaz* del objeto.

TERMINO NUEVO

Di mayores detalles de esta idea en la hora 5, “Agregación, composición, interfaces y relación”. Como lo mencioné en su momento, diversas clases podrían no estar relacionadas con una clase principal (como en la herencia), pero sus acciones podrían incluir algunas de las mismas operaciones con las mismas firmas. Podrá reutilizar

este conjunto de operaciones de clase en clase. La *interfaz* es la construcción UML que le permite hacer esto. Una interfaz es un conjunto de operaciones que especifica algo respecto al comportamiento de una clase. Imagine a una interfaz como una clase que sólo contiene operaciones (no atributos). Para resumir: la interfaz es un conjunto de operaciones que presenta una clase a otras.

Al tratar las interfaces en la hora 5, también mencioné que la relación entre una clase y su interfaz se conoce como *realización*.

¡Un momento! Parecería que modelar una interfaz es la práctica de modelar un concepto. Al principio de esta hora, le dije que cuando modele un componente no será algo conceptual, dado que se encontrará en una computadora. ¿Dónde está la conexión?

En sí, una interfaz puede ser física o conceptual. La interfaz que utiliza una clase es la misma que la que utiliza su implementación de software (un componente). Como modelador, esto significa que la de la misma forma en que represente una interfaz para una clase representará una interfaz para un componente. Aunque la simbología del UML distingue entre una clase y un componente, no hace distinción entre una interfaz conceptual y una física.

TERMINO NUEVO

Hay un punto importante a este respecto: sólo podrá ejecutar las operaciones de un componente a través de su interfaz. De la misma manera que en el caso de una clase y su interfaz, la relación entre un componente y su interfaz se conoce como *realización*.

TERMINO NUEVO

Hay otro punto por destacar: un componente puede hacer disponible su interfaz para que otros componentes puedan utilizar las operaciones que contiene. Es decir, un componente puede acceder a los servicios de otro componente. El componente que proporciona los servicios se dice que provee una *interfaz de exportación*. Al que accede a los servicios se dice que utiliza una *interfaz de importación*.

Sustitución y reutilización

Las interfaces se destacan de forma importante en los conceptos primordiales de sustitución y reutilización de componentes. Puede sustituir un componente con otro si el nuevo contiene las mismas interfaces que el anterior. Podrá reutilizar un componente en otro sistema si éste puede acceder al componente reutilizado mediante sus interfaces. Puede diseñar un componente para ser reutilizado en proyectos de desarrollo a lo largo de su empresa si quiere depurar sus interfaces para que un amplio rango de componentes puedan acceder a ellos.

Es aquí donde son útiles las interfaces en el modelado. Puede simplificarse la vida de un desarrollador que intente sustituir o reutilizar un componente si la información de su interfaz se encuentra disponible como un modelo. Si no, el desarrollador tendrá que pasar por el largo proceso de hacer un seguimiento del código.

Tipos de componentes

Conforme avance en su carrera como modelador, se encontrará con tres tipos de componentes:

1. *Componentes de distribución*, que conforman el fundamento de los sistemas ejecutables (por ejemplo, DLL, ejecutables, controles ActiveX y Java Beans).
2. *Componentes para trabajar en el producto*, a partir de los cuales se han creado los componentes de distribución (como archivos de base de datos y de código).
3. *Componentes de ejecución*, creados como resultado de un sistema en ejecución.

Si es usted un usuario de Windows, encontrará ejemplos de los tres tipos de componentes cuando utilice la ayuda, aunque tal vez no lo sepa. El componente de distribución es el archivo .HLP (Archivo de Ayuda): Cuando haga clic en uno, se abrirá el cuadro de diálogo de los temas de la ayuda y empezará a buscar el tema que elija. La primera vez que haga clic en la ficha Buscar, verá una pequeña animación que le indicará que se está creando un índice. Un archivo .CNT (tema de contenido) describe el esquema del contenido. Dado que este archivo le ayuda a crear un componente de distribución (puede utilizar la página de índice por siempre), es un componente para trabajar en el producto. A su vez, el índice creado se encuentra en un archivo .FTS (búsqueda de texto completo). Finalmente, la primera vez que abra la ayuda, se creará un archivo .GID (índice general), que es resultado de un análisis del sistema de ayuda de Windows que agiliza el acceso a los temas del archivo de ayuda. Con ello, los archivos .FTS y .GID son componentes de ejecución.

Qué es un diagrama de componentes

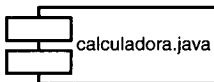
Un diagrama de componentes contiene, obviamente, componentes, interfaces y relaciones. También pueden aparecer otros tipos de símbolos que ya haya visto.

Representación de un componente

El símbolo principal de un diagrama de componentes es un rectángulo que tiene otros dos sobrepuestos en su lado izquierdo. La figura 12.1 le muestra este símbolo. Debe colocar el nombre del componente dentro del símbolo. El nombre es una cadena.

FIGURA 12.1

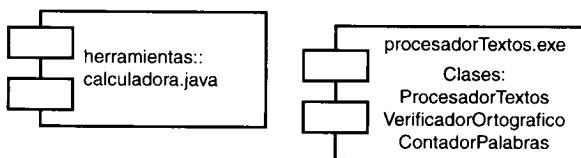
El símbolo que representa a un componente.



La figura 12.2 le muestra que si el componente es miembro de un paquete, puede utilizar el nombre del paquete como prefijo para el nombre del componente. También puede agregar información que muestre algún detalle del componente.

FIGURA 12.2

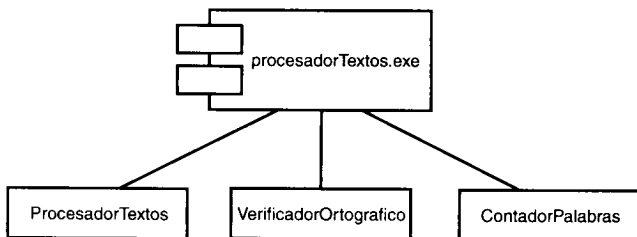
Adición de información al símbolo del componente.



El símbolo a la derecha de la figura 12.2 le muestra las clases que implementa un componente en particular. La figura 12.3 le muestra otra forma de hacer esto, aunque esta técnica por lo general desordena el diagrama. Vea las relaciones de dependencia entre el componente y las clases.

FIGURA 12.3

Símbolos de las relaciones entre un componente y las clases que implementa.

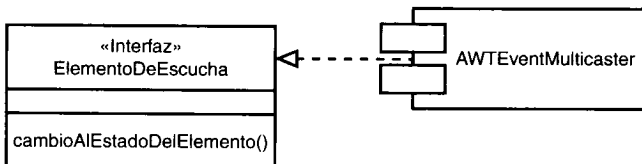


Cómo representar las interfaces

Existen dos formas para representar a un componente y sus interfaces: la primera muestra la interfaz como un rectángulo que contiene la información que se le relaciona, se conecta al componente por la línea discontinua y una punta de flecha representada por un triángulo sin rellenar que visualiza la realización (vea la figura 12.4).

FIGURA 12.4

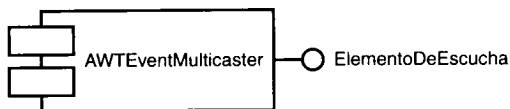
Puede representar a una interfaz como un rectángulo con información, conectado al componente por una flecha de realización.



La figura 12.5 le muestra la segunda forma de representar a un componente y sus interfaces; esta forma es representativa, ya que representará la interfaz como un pequeño círculo que se conecta al componente por una línea continua. En este contexto, la línea representa la relación de realización (compare a las figuras 12.4 y 12.5 con las figuras 5.6 y 5.7).

FIGURA 12.5

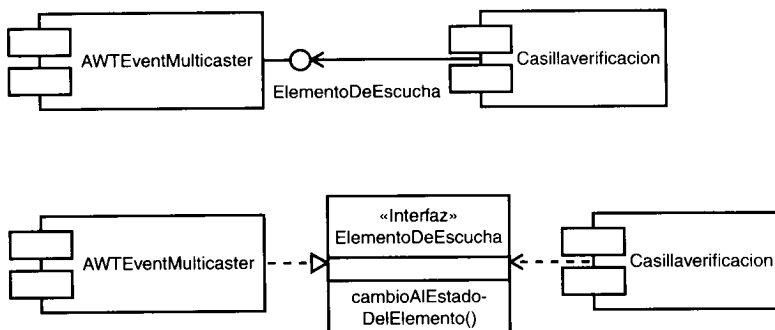
Puede representar a una interfaz como un pequeño círculo, conectado al componente por una línea continua que, en este contexto, se interpreta como realización.



Además de la realización, puede representar a la dependencia, que es la relación entre un componente y una interfaz de importación. Como recordará, la dependencia se vislumbra como una línea discontinua con una punta de flecha. Puede mostrar la realización y la dependencia en el mismo diagrama, como se ve en la figura 12.6.

FIGURA 12.6

Una interfaz que realiza un componente y otra de la que depende.



Aplicación de los diagramas de componentes

Algunos ejemplos le ayudarán a empezar a usar los diagramas de componentes. El primero es un modelo de una página Web que representa a un componente Java. El siguiente también es un modelo de una página Web pero éste utiliza controles ActiveX. Finalizará con un modelo de un paquete de Microsoft conocido como PowerToys. Este paquete, que puede obtener del sitio de Microsoft, le permite modificar aspectos de Win32.

Una página Web con un subprograma Java

Este ejemplo modela un programa tomado del excelente y entretenido libro de Rogers Cadenhead llamado *Aprendiendo programación con Java 1.1 en 24 horas* (Prentice Hall

Hispanoamericana, 1997). El ejemplo aparece en la hora 22, “Escriba un juego para Web”. Rogers muestra cómo generar un applet (o subprograma) que ejecuta el juego de dados “Craps” en una página Web, y utiliza una clase llamada “Die” (para crear los dados) de la hora 21, “A jugar con Java”. Para ver los detalles del código, tendrá que leer el libro. Aquí sólo nos concentraremos en los componentes.



TÉRMINO NUEVO

Un applet es un pequeño programa hecho en Java que funciona en una página Web.

La página Web se llama Craps.html. El código fuente del applet se encuentra en el archivo Craps.java, y el código objeto es el archivo Craps.class. El código fuente de la clase Die se encuentra en Die.java y el código objeto en Die.class. Los cinco archivos se encuentran en el mismo directorio —que llamaremos Tirodedados (que no es el nombre que Roger le dio)—.

Craps.html depende, obviamente, de Craps.class y Die.class. Cada archivo .class es un componente y cada uno es la implementación de una clase. Lo que no es muy obvio (tendría que ver el código fuente para descubrirlo) es que tanto Craps.java como Die.java importan (utilizan las clases de) java.awt, un grupo de clases que muestran y controlan la GUI (el “awt” significa: “Conjunto de herramientas abstractas para manejar ventanas”).



En el contexto de Java, la *importación* permite al desarrollador utilizar sólo el nombre de una operación cuando la escribe en un programa, en lugar de utilizar toda la ruta de la operación (que podría ser muy larga). La importación no “incrusta” una clase en otra. Tan sólo permite escribir algo más corto.

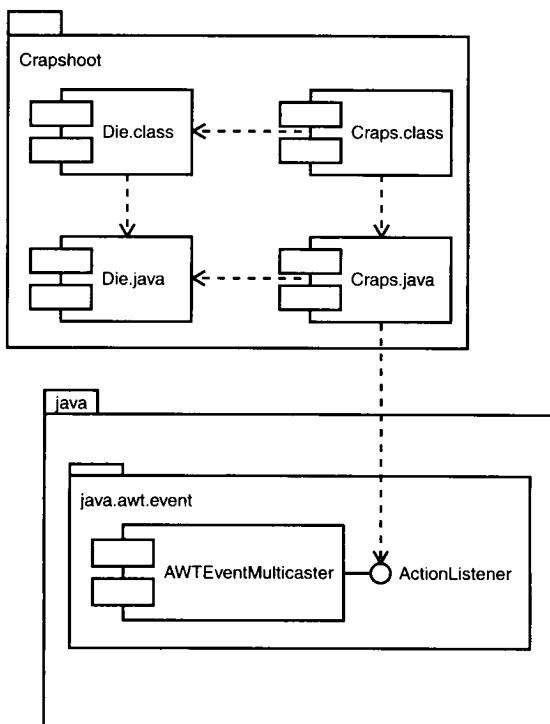
Craps.java es un applet, y por ello se hereda desde la clase java.applet.Applet. Finalmente, Craps.java importa a java.awt.event e implementa una interfaz ActionListener (para responder a los eventos generados por el usuario, como hacer clic con el ratón).

Dentro del código, la interfaz ActionListener proporciona un botón para que el usuario haga clic para que se “tiren los dados”. Al ver la referencia de Java, notará que la clase java.awt.AWTEventMulticaster implementa esta interfaz.

¿Sabe qué? ¡Esto sería más fácil de comprender si viera el modelo UML! La figura 12.7 le muestra el diagrama de componentes. Un paquete corresponde al directorio en donde se encuentran los archivos, el otro al JDK (Conjunto de herramientas para desarrollo en Java).

FIGURA 12.7

El diagrama para el juego de dados basado en la Web de Rogers Cadenhead.



TERMINO NUEVO

Este ejercicio —generar un modelo a partir de un código existente— se conoce como *ingeniería inversa*.

Una página Web con controles ActiveX

ActiveX es el medio de Microsoft para agregar componentes a las aplicaciones. Con tantos tipos de componentes ActiveX (controles) disponibles, podrá encontrar alguno que haga casi todo lo que requiera una aplicación. Una propiedad de un componente ActiveX es su número de identificación hexadecimal único de 32 bits, conocido como CLSID (identificador de la clase).

Si sus requerimientos son especiales, podrá generar su propio componente ActiveX en Visual Basic o Visual C++. Luego podrá reutilizarlo de aplicación en aplicación.

En las páginas Web, los componentes ActiveX se encuentran y trabajan con el código escrito en algún lenguaje para secuencias de comandos como VBScript. En este ejemplo, la página Web cuenta con un control Timer ActiveX, dos cuadros combinados ActiveX y tres botones ActiveX. La página Web permite a un usuario establecer los parámetros para

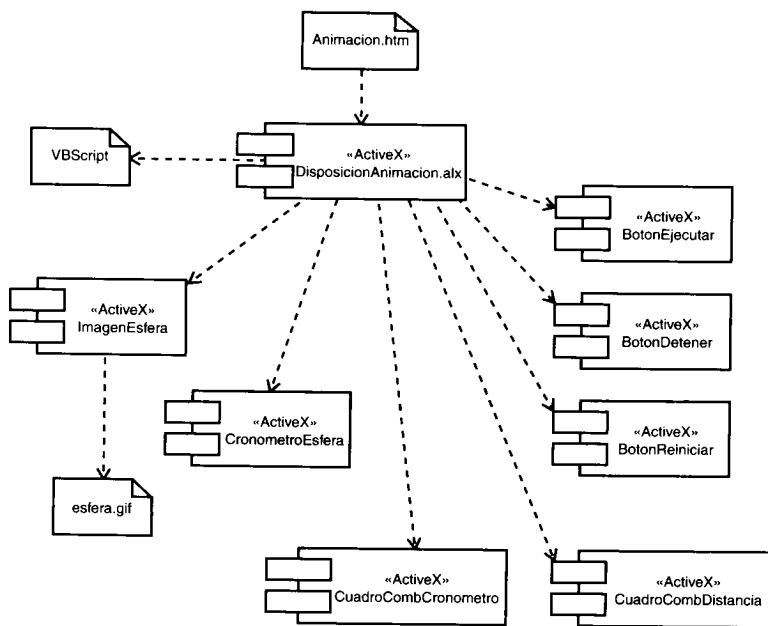
animar el movimiento de una esfera (una imagen .gif) por la pantalla. De un cuadro combinado, el usuario selecciona la cantidad de píxeles por movimiento. Del otro se selecciona la cantidad de milisegundos entre movimientos. Un botón iniciará el movimiento, el otro lo detendrá y el tercero restaurará la esfera a su posición inicial. El cronómetro moverá la esfera cuando pase la cantidad de milisegundos elegida por el usuario.

Los controles ActiveX se encuentran en un componente separado conocido como *Disposición* (Layout). La página HTML y la disposición se encuentran en el mismo directorio.

La figura 12.8 le muestra el diagrama de componentes para esta página. Observe el uso del símbolo de anotación para representar al VBScript. Aunque esto no es absolutamente necesario, destaca una diferencia entre el lenguaje de secuencias de comandos y los componentes compilados ActiveX.

FIGURA 12.8

El diagrama de componentes para una página Web con componentes ActiveX.



PowerToys

Si utiliza cualquier versión de Win32, ya conocerá las horribles flechas pequeñas que se encuentran en la esquina inferior izquierda de cada icono de acceso directo. Microsoft tiene un paquete llamado PowerToys que le permite eliminarlas y hacer varias otras cosas con la GUI, mediante una aplicación llamada TweakUI que es parte del paquete.

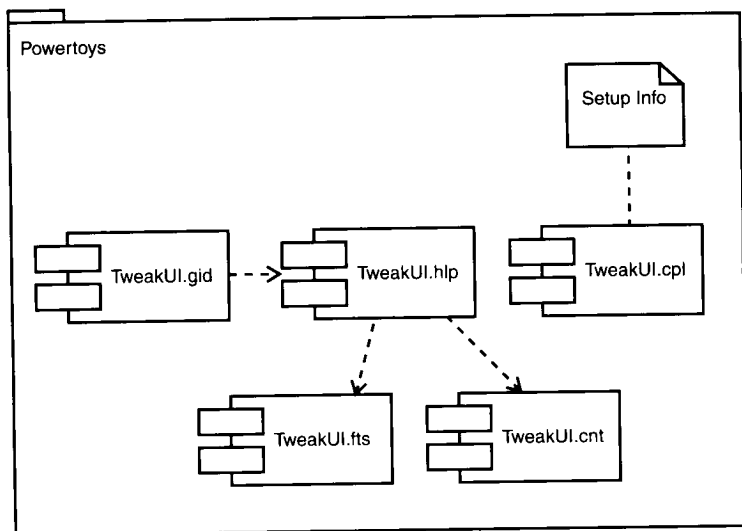
Puede obtener a PowerToys del sitio Web de Microsoft. Es gratis. Cuando lo obtenga y descomprima, verá varios archivos con extensiones .dll. También verá un archivo de

ayuda y otro .CNT. Haga clic en el archivo de ayuda y generará un archivo .GID. Utilice la característica Buscar y creará un archivo .FTS.

La figura 12.9 le muestra un diagrama de componentes que modela a TweakUI en el paquete PowerToys, mismo que muestra las dependencias entre los diversos tipos de componentes.

FIGURA 12.9

*Modelado de TweakUI
en el paquete
PowerToys.*

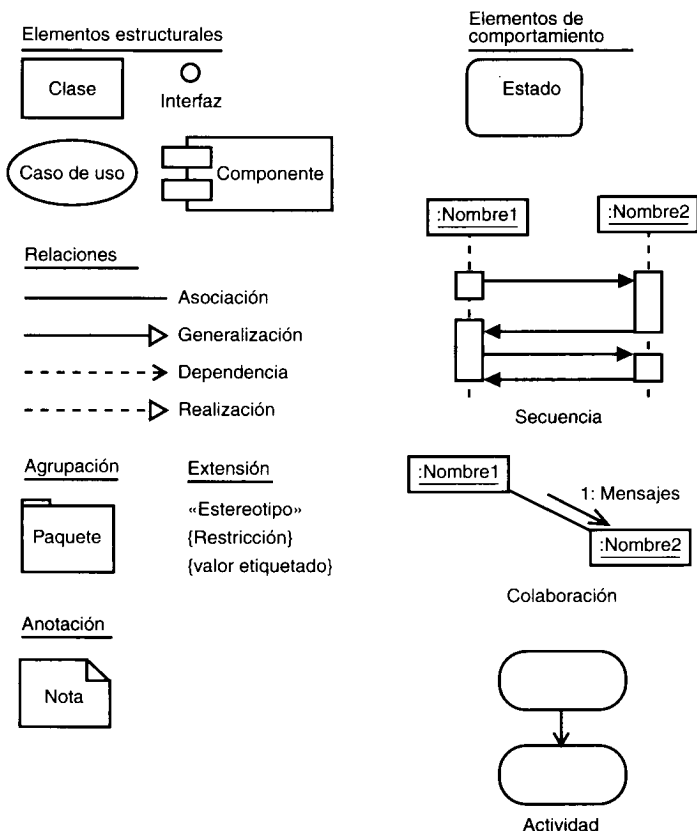


Diagramas de componentes en el panorama

Ya casi tiene todo su panorama. La figura 12.10 incluye el diagrama de componentes, que se enfoca en una arquitectura de software del sistema. En la siguiente hora, verá cómo modelar la arquitectura de hardware.

FIGURA 12.10

Su panorama del UML ahora incluye al diagrama de componentes.



Resumen

El diagrama de componentes UML es un conglomerado de figuras de los diagramas que ya ha visto. En lugar de representar una entidad conceptual como una clase o estado, un diagrama de componentes representa a un elemento real: un componente de software. Estos componentes se encuentran en las computadoras, no en la mente del analista.

Un componente puede accederse a través de su interfaz, una colección de operaciones. La relación entre un componente y su interfaz se llama *realización*. Un componente puede acceder los servicios de otro. Cuando se hace, utiliza una *interfaz de importación*. El componente que realiza la interfaz con tales servicios proporciona una *interfaz de exportación*.

La representación de un componente es un rectángulo con otros dos rectángulos pequeños superpuestos en su lado izquierdo. Puede representar una interfaz de dos formas: la primera es un rectángulo que contiene información de la interfaz y se conecta con el componente mediante una línea discontinua con una punta de flecha representada por triángulo sin relleno. La otra es un pequeño círculo conectado al componente con una línea continua. Ambos tipos de conexión pretenden mostrar una relación de realización.

Preguntas y respuestas

- P** En un diagrama de componentes, ¿cuál será la regla de oro para usar símbolos que no representen a componentes?
- R** Esto lo hará cuando desee indicar algo que sea ciertamente distinto de un componente compilado. No es necesario, pero podría ayudar a tener otro punto de vista. Podría utilizar el símbolo de la anotación para representar archivos de encabezado, dll, o archivos de secuencias de comandos. Otra posibilidad es la de utilizar el símbolo regular del componente con un estereotipo que indique el tipo de archivo.
- P** Ha utilizado a VBScript como un componente de la página Web. El código de VBScript consta de varios procedimientos. ¿No podría modelar cada uno como componente?
- R** Sí, podría. No obstante, podría desordenar su modelo si depurara al VBScript (o JavaScript) hasta tal nivel, así que podría, mejor, agregar una nota que abunde al respecto.

Taller

En este taller reforzará su conocimiento respecto a los componentes y cómo modelarlos. Uno de los ejercicios trata de los conceptos de una página Web, el otro de DLL. El Apéndice A, “Respuestas a los cuestionarios”, será el componente del libro que contenga las respuestas.

Cuestionario

1. ¿Cuáles son los tres tipos de componentes?
2. ¿Cómo llamaría a la relación entre un componente y su interfaz?
3. ¿Cuáles son las dos formas de representar a esta relación?
4. ¿Qué es una interfaz de exportación? ¿Qué es interfaz de importación?

Ejercicios

1. Adentrémonos en la ingeniería inversa para modelar una página Web. Visite <http://www.pearson.com.mx>, la página Web de Pearson Educación (la empresa que amablemente publicó el libro que ahora lee). Utilice el menú Ver de su explorador para seleccionar la opción que le deje a la vista el código fuente. No encontrará ningún componente ActiveX o applet de Java, pero verá diversos archivos .gif y cierto código en JavaScript. No incluya todos los archivos .gif en su modelo, sólo algunos para reafirmar su comprensión. Vaya al principio del código y verá una referencia a una hoja de estilo. La referencia se encuentra en un elemento LINK de HTML. Este es un archivo por separado que contiene la información de estilo para la página Web. Asegúrese de incluirla en su modelo.

2. Si cuenta con Office97 (o superior), explore el directorio que contiene a Excel. Será algo como C:\Archivos de Programa \Microsoft Office\Office (la unidad de disco y la carpeta Archivos de Programas podrían ser diferentes). Vea las DLL que utiliza Excel. Empiezan con “X1” y finalizan con .DLL. Modele tales dependencias con un diagrama de componentes. Excel depende de más DLL que éstas, pero le servirá de práctica.

NOTA PARA HACER ESTE EJERCICIO (esta semana) y para la exposición del lunes

Si no encuentra este programa, entonces pruebe con cualquier otro para describir sus componentes.



HORA 13



Diagramas de distribución

Hasta ahora nos hemos concentrado en el entorno conceptual, aunque en la hora anterior vimos los modelos de la arquitectura de software. Es momento de concentrarnos en el hardware. Como podrá ver, hemos trascendido desde los elementos (como las clases) que se encuentran en los análisis, hasta los componentes en los equipos de cómputo y al hardware existente.

Claro está que el hardware es un tema primordial en un sistema de varios componentes. En el mundo actual de la computación, un sistema podría abarcar diversos tipos de plataformas en ubicaciones dispersas. Un diseño sólido de distribución de hardware es básico para el diseño del sistema. El UML le da los símbolos para crear una imagen clara de la forma en que deberá lucir el hardware final.

En esta hora se tratarán los siguientes temas:

- Qué es un diagrama de distribución
- Aplicación de los diagramas de distribución
- Los diagramas de distribución en el panorama del UML

Qué es un diagrama de distribución

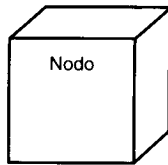
TERMINO NUEVO

El elemento primordial del hardware es un *nodo*, que es un nombre genérico para todo tipo de recurso de cómputo. Es posible usar dos tipos de nodos: un procesador, el cual puede ejecutar un componente, y un dispositivo que no lo ejecuta. Normalmente, un dispositivo (como impresora o monitor) tiene contacto de alguna forma con el mundo exterior.

En el UML, un cubo representa a un nodo. Deberá asignar un nombre para el nodo, y podrá utilizar un estereotipo para indicar el tipo de recurso que sea. La figura 13.1 le muestra un nodo.

FIGURA 13.1

Representación de un nodo en el UML.



El nombre es una cadena de texto. Si el nodo es parte de un paquete, su nombre puede contener también el del paquete. Puede dividir al cubo en compartimientos que agreguen información (como componentes colocados en el nodo), como en la figura 13.2.

FIGURA 13.2

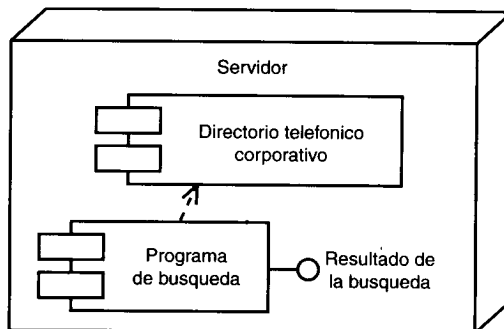
Cómo agregar información a un nodo.



Otra forma de indicar los componentes distribuidos es la de mostrarlos en relaciones de dependencia con un nodo (vea la figura 13.3).

FIGURA 13.3

Puede mostrar los componentes en relaciones de dependencia con un nodo.

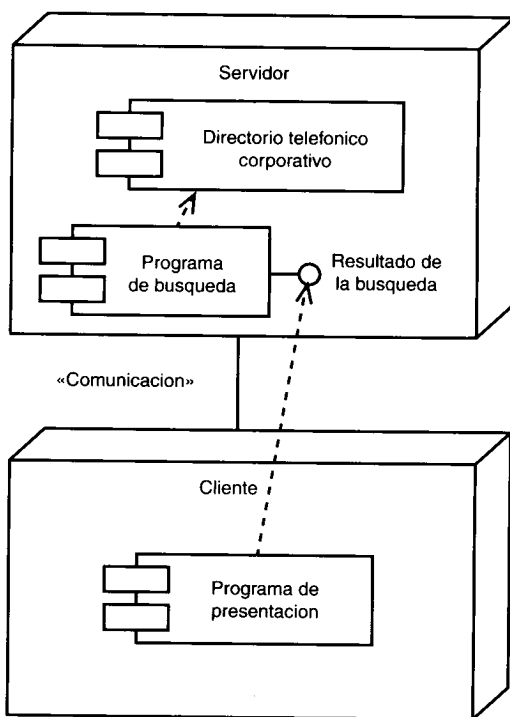


Una línea que asocie a dos cubos representará una conexión entre ellos. Podrá utilizar un estereotipo para dar información respecto a la conexión. La figura 13.4 proporciona ejemplos de conexiones entre nodos.



Tenga en cuenta que una conexión no es necesariamente un cable o alambre. También puede visualizar conexiones inalámbricas como las infrarrojas o satelitales.

FIGURA 13.4
*Representación
de conexiones
entre nodos.*



Aunque la conexión es el tipo común de asociación entre dos nodos, es posible utilizar otros (como la agregación o la dependencia). Podrá representarlas de las formas ya conocidas.

Aplicación de los diagramas de distribución

Un buen lugar para empezar es con un equipo de cómputo doméstico, por lo que el primer ejemplo es un diagrama de distribución del sistema que utilicé para escribir este libro.

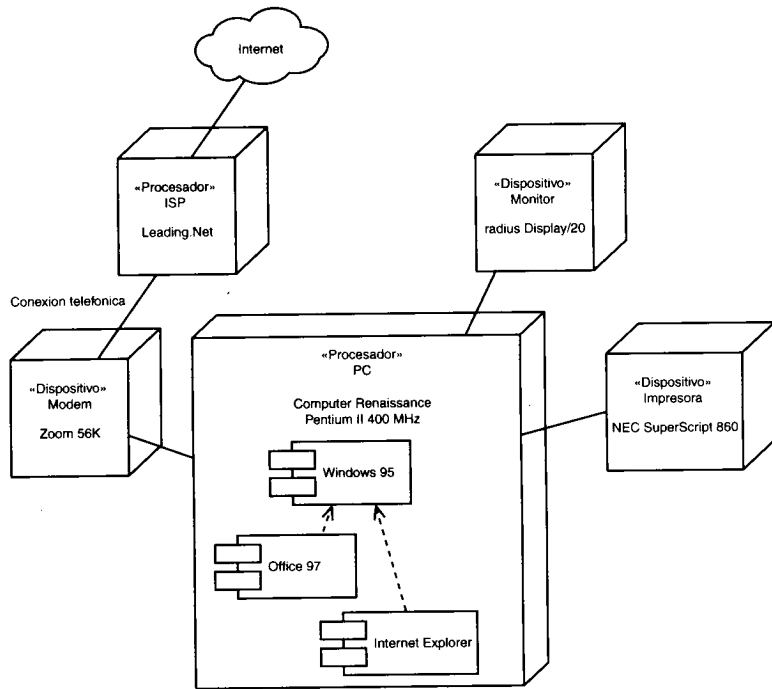
No obstante, como lo dije, los sistemas actuales de varios procesadores conectan nodos que podrían encontrarse lejanos entre sí. Para visualizar completamente este problema, necesitará también ver los ejemplos de los diagramas de distribución aplicados a las redes. Incluiré ejemplos que podrán servirle para adaptarlos a su propio entorno. Cada ejemplo incluye restricciones que reflejan las reglas de la red particular.

Un equipo doméstico

Para modelar mi equipo de cómputo, he incluido al procesador y los dispositivos, a la vez que he modelado mi conexión telefónica con mi proveedor de servicios de Internet y su conexión. La nube que representa la Internet no es parte de la simbología del UML, pero es útil para clarificar el modelo. La figura 13.5 presenta el diagrama de distribución.

FIGURA 13.5

El diagrama de distribución de mi equipo de cómputo.



Una red token-ring

En una red token-ring, las computadoras equipadas con una NIC (tarjeta de interfaz de red) se conectan a una MSAU (unidad central de acceso a multiestaciones). Se conectan varias MSAU en una serie que podría parecer un anillo (por ello la parte "ring" del nombre). El anillo de MSAU se combina para fungir como un policía de tránsito, mediante una señal

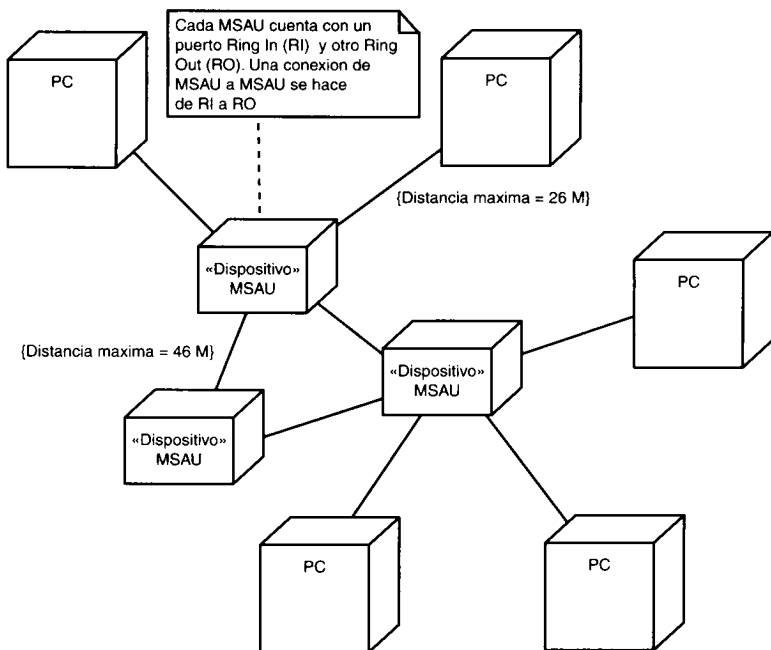
conocida como *token* que permite a cada equipo de cómputo saber cuándo puede transmitir información. Así es, el token va de equipo en equipo hasta que uno de ellos contenga información por enviar. En realidad, el token se mueve por el anillo de MSAU.

Cuando se obtiene el token, sólo esa información del equipo puede ir por la red. Una vez que se envía, la información viaja hasta su destino. Cuando llega, se devuelve un acuse de recibo al equipo que la envió.

En este ejemplo, que se aprecia en la figura 13.6, he modelado una red que consta de tres MSAU y sus respectivos equipos.

FIGURA 13.6

El diagrama de distribución de una red token-ring que consta de tres MSAU.



ARCnet

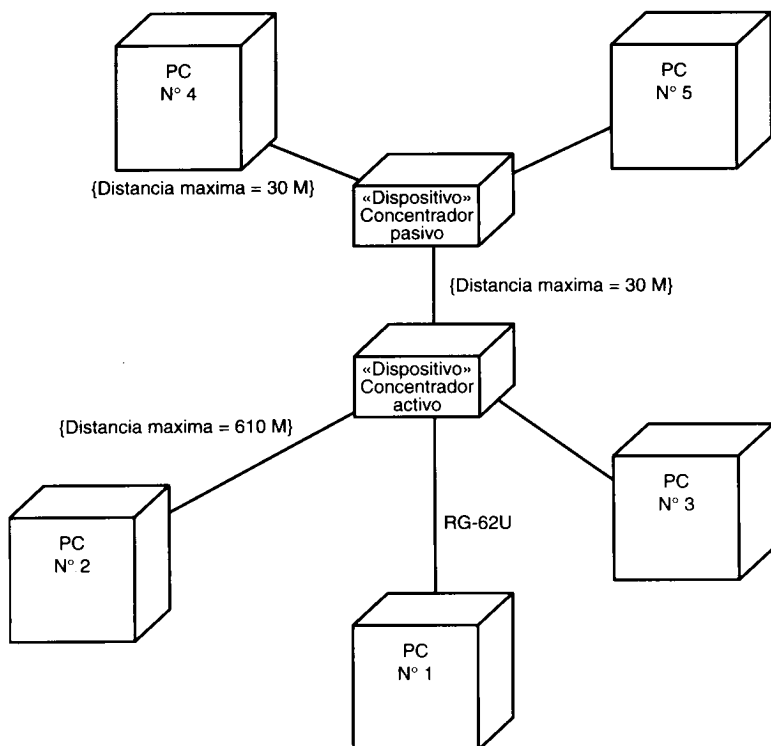
Como en una red token-ring, una red ARCnet (Red de Cómputo de Recursos Adjuntos) implica pasar un token o señal de un equipo a otro. La diferencia es que en ARCnet cada equipo tiene asignado un número. El orden numérico determina cuál equipo obtendrá al token. Cada equipo se conecta a un concentrador o hub que podrá ser activo (amplificará la información que llega antes de transmitirla) o pasivo (transmitirá la información sin amplificarla).

A diferencia de los MSAU en una red token-ring, los concentradores ARCnet no mueven el token en un anillo. Los equipos se lo pasan entre sí.

La figura 13.7 modela una red ARCnet con un concentrador pasivo, uno activo y varios equipos.

FIGURA 13.7

Un diagrama de distribución de una red ARCnet.



Thin ethernet

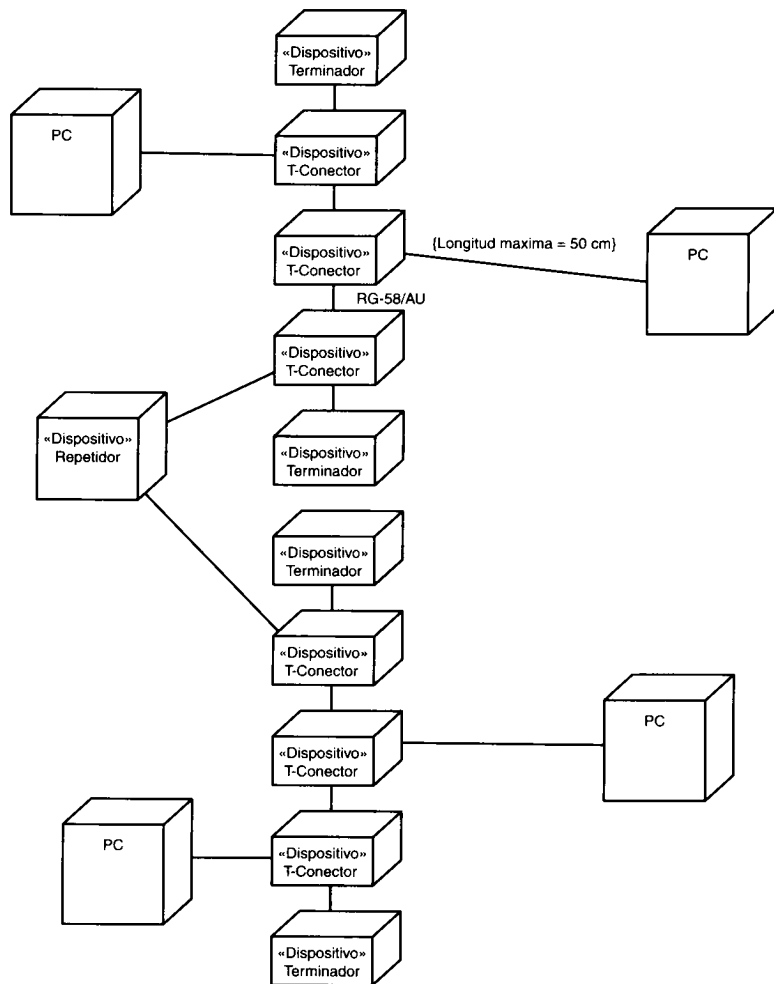
TERMINO NUEVO

La red thin ethernet es un tipo muy popular. Los equipos se conectan a un cable de red mediante dispositivos conocidos como conectores T. Un segmento de red puede unirse a otro mediante un *repetidor*, un dispositivo que amplifica una señal antes de transmitirla. También pueden hacerse conexiones de tipo RJ-45, aunque, en este caso, nos concentraremos tan sólo en la conexión T.

La figura 13.8 modela una red thin ethernet.

FIGURA 13.8

Diagrama de distribución de una red thin ethernet.



Red inalámbrica Ricochet de Metricom

Metricom, Inc, empresa localizada en Los Gatos, CA, cuenta con una solución inalámbrica por módem para obtener acceso móvil a Internet. Su módem inalámbrico se conecta al puerto serial de un equipo de cómputo y se comunica con su red Ricochet.

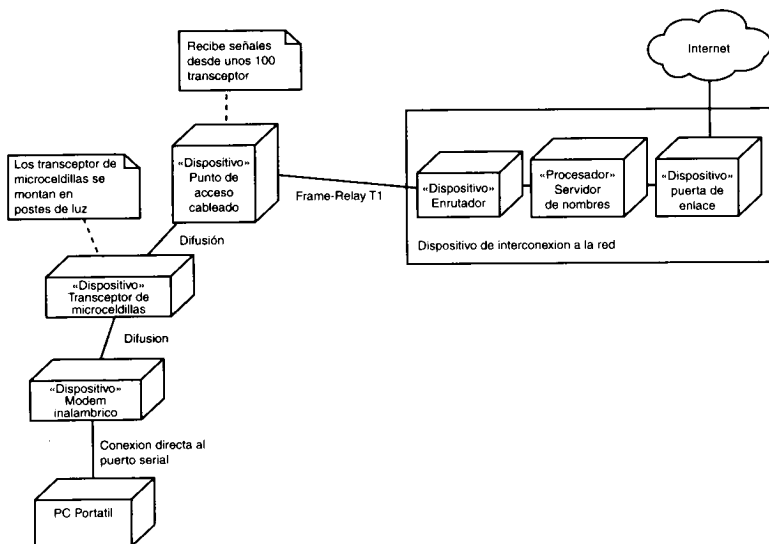
La red Ricochet consta de transmisores y receptores de radio, cuyo tamaño es de una caja de zapatos. Tales radios de microceldilla se montan en la parte superior de los postes de luz a distancias de 400 a 800 metros, en un patrón de tablero de ajedrez. Cada radio de microceldilla obtiene una pequeña cantidad de energía de su poste de luz si se equipa con un adaptador especial.

Los radios de microceldillas difunden señales a Puntos de acceso cableados que llevan la información a un NIF (dispositivo de interconexión a la red). El NIF consta de un servidor de nombres (una base de datos que valida las conexiones), un enrutador (dispositivo que enlaza a las redes entre sí), y una puerta de enlace (un dispositivo que traduce la información de un protocolo de comunicaciones a otro). La información se lleva del NIF a la Internet.

La figura 13.9 muestra un diagrama de distribución para esta red.

FIGURA 13.9

*Red inalámbrica
Ricochet de
Metricom.*

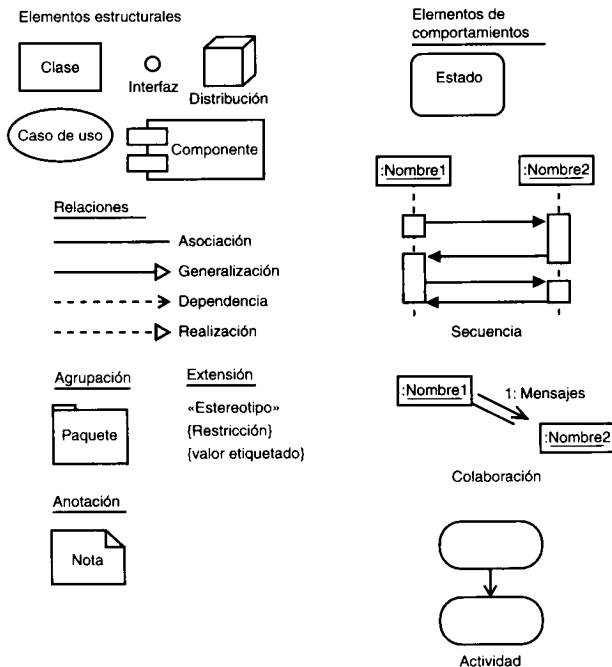


Los diagramas de distribución en el panorama

Ha llegado al final del conjunto de diagramas. El panorama incluye al diagrama de distribución, y ha quedado finalizado.

FIGURA 13.10

Su panorama del UML incluye al diagrama de distribución y está completo.



Resumen

El diagrama de distribución del UML ilustra la forma en que luce un sistema físicamente cuando sea conjugado. Un sistema consta de nodos, donde cada nodo se representa por un cubo. Una línea asocia a dos cubos y simboliza una conexión entre ellos. Los tipos de nodos son procesador (que puede ejecutar un componente) y dispositivo (que no lo puede hacer). Los dispositivos por lo general interactúan con el mundo.

Como puede imaginar, los diagramas de distribución son útiles para modelar redes. Los modelos presentados en esta hora incluyeron a redes token-ring, ARCnet, thin ethernet y la red inalámbrica Ricochet.

Preguntas y respuestas

- P** Usted utilizó una nube para representar a la Internet, y dijo que no era parte de la simbología del UML. ¿Un modelador puede utilizar símbolos que no están en la simbología?
- R** Así es. De hacerlo, no habrá policía UML que lo lleve a prisión. La idea es utilizar el UML para expresar una visión. En ninguna parte esto es tan útil como en los diagramas de distribución. Si tiene una imagen que pueda mostrar claramente los equipos de escritorio, portátiles, servidores y otros procesadores (o dispositivos), podrá utilizarlos en sus diagramas. Claro que estará creando un estereotipo gráfico. (Por cierto que el símbolo de la nube es una interesante nota al margen por aprender en el UML. Uno de los creadores del UML, Grady Booch, solía representar objetos como nubes en la simbología de su esquema de modelado antes de que se convirtiera en parte del equipo del UML.)
- P** Suponga que cuenta con una gran cantidad de figuras para representar a ciertos objetos y no a otros. ¿Se pueden mezclar con los símbolos del UML?
- R** Claro que puede. El objeto es dibujar diagramas para clarificar una visión, no para (perdón por el juego de palabras) nublarla.

Taller

Ahora que ha finalizado con todo el conjunto de diagramas del UML, pruebe su conocimiento respecto a la forma de representar hardware. Las respuestas se destacan en el apéndice A, “Respuestas a los cuestionarios”.

Cuestionario

1. ¿Cómo representa a un nodo en un diagrama de distribución?
2. ¿Qué tipo de información puede aparecer en un nodo?
3. ¿Cuáles son los dos tipos de nodos?
4. ¿De qué forma funciona una red token-ring?

Ejercicios

1. Imagine a su equipo de cómputo doméstico como un conjunto de nodos. Dibuje un diagrama de distribución que incluya a su gabinete, monitor, impresora y cualquier otro periférico. Incluya cualquier estereotipo necesario y compartimientos para clarificar la información (posiblemente resultará en un modelo algo diferente al de mi equipo).
2. Es posible conectar una red a otra. Una forma de hacerlo es conectar a cada red con un enrutador y a cada uno de ellos con un (posiblemente muy grande) circuito de LAN a LAN. Dibuje un diagrama de distribución de una pequeña red token-ring que se conecte a una pequeña red thin ethernet.

HORA 14



Nociones de los fundamentos del UML

Ahora que ha visto los diagramas en el UML, ya está listo para ciertos conceptos fundamentales.

En esta hora se tratarán los siguientes temas:

- Estructura del UML
- Capa del metamodelado
- Extensión del UML
- Estereotipos, restricciones y valores etiquetados

Si éste fuera un texto académico en lugar de uno de autoaprendizaje, esta hora hubiese aparecido al principio de la parte I, en lugar de estar casi al final. Hice esto para darle la oportunidad de conocer las trincheras del UML (qué es y lo que hace). Así, ya estará listo para conocer los fundamentos y trabajar con ellos.

Ahora que ha visto los diagramas y sabe cómo utilizarlos, ¿para qué podría servirle esta hora? Si ya comprendió en qué se basa el UML, podrá extenderlo y adaptarlo cuando empiece a utilizarlo en el mundo real. Como cualquier analista de sistemas le diría: cada proyecto es diferente. No existe ningún texto de referencia, libro o manual que lo pueda preparar para todas las situaciones con las que se encontrará. No obstante, el tener una buena base de los conceptos fundamentales lo preparará para la mayoría de los sistemas que tenga que modelar.

Es como aprender un idioma extranjero. La mejor forma de hacerlo es sumergirse en él, como lo hizo en las horas 1 a la 13 (y como lo hará en la parte II, “Estudio de un caso”). Luego podrá empezar a captar las reglas gramaticales y sintácticas dado que estará preparado para comprenderlas (¡por desgracia, muchos cursos académicos de idiomas proceden en orden opuesto!).

Estructura del UML

Su panorama del UML le muestra las categorías de los diagramas y a éstos en cada categoría. Como lo dije en la hora 1, “Introducción al UML”, necesitará todos los diagramas, ya que le permitirán ver su sistema desde diversos puntos. Debido a que hay varias personas a las que les interesa el sistema por distintas razones, deberá tener la capacidad de comunicar una visión consistente del sistema de diversas formas.

Aunque su panorama es útil como una forma de tener a la mano los elementos del UML, no funciona como una definición de éste. Los tres amigos estructuraron al UML de una manera formal para asegurarse que los elementos que habían creado pudieran mostrar una idea clara de un sistema propuesto, o completamente reestructurado.

El UML cuenta con una arquitectura de cuatro capas. Tales capas se distinguen por la generalidad de los elementos que en ellas residen.



En la actualidad, la palabra *arquitectura* nos brinca en el mundo del desarrollo de sistemas. Imagine una arquitectura como una forma de resumir un conjunto de decisiones respecto a la forma en que se organiza un sistema. Tales decisiones se enfocan muy específicamente en los elementos del sistema: qué son, qué hacen, cómo se comportan, cómo se relacionan y se combinan.

En las horas anteriores estuvo operando en las dos capas más específicas. Cuando siguió un ejemplo o realizó un ejercicio que involucraba instancias específicas de un dominio en particular (como el diagrama de componentes de mi sistema de cómputo personal), se encontraba en la capa más específica. Esta capa se llama *capa de objetos del usuario*, donde “usuario” se refiere a quien utiliza el UML (no al que utiliza el sistema en sí).

TERMINO NUEVO

Estuvo en la siguiente capa cuando vio las clases, como en el ejemplo donde el analista habló con el entrenador de baloncesto para distinguir las clases en el dominio baloncesto. Los primeros estados del análisis tienen que ver con esta capa: trabajará con un experto o un cliente para obtener la información de un dominio, y con los usuarios potenciales para comprender los casos de uso que tendrán que tomarse en cuenta al generar el sistema. A esta capa se le denomina *capa de modelado*.

TERMINO NUEVO

¿En qué momento estuvo en una capa menor? Al principio de cada hora cuando aprendió un concepto como el de una clase o un nodo, estaba en la tercera de cuatro capas. Ésta define al lenguaje para un modelo específico. Luego de que obtenga algo de experiencia, se familiarizará tanto con el UML que esta tercera capa le será algo muy natural. Dado que esta capa define lo que va en un modelo, se conoce como *capa de metamodelado*.



Puesto que en su panorama se muestran los símbolos de las clases, nodos, componentes, casos de uso y cosas así, tal panorama pertenece a la capa de metamodelado.

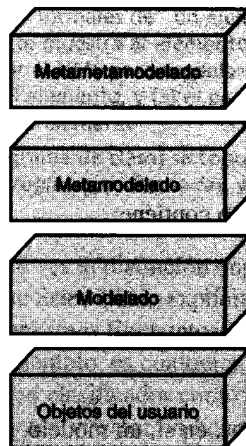
TERMINO NUEVO

¿Y la cuarta capa? Durante su carrera como analista, posiblemente nunca tendrá que tratar con ella. Imagínela como una forma de definir un lenguaje que especifique clases, casos de uso, componentes y todos los demás elementos del UML con los que trabajará. Esto es más de la incumbencia de los teóricos que diseñan y comparan lenguajes que de los analistas que lo usan. Dado que esta capa define lo que va en el metamodelo, se conoce como *capa de metametamodelado*.

La figura 14.1 le muestra las cuatro capas.

FIGURA 14.1

Las cuatro capas del UML.

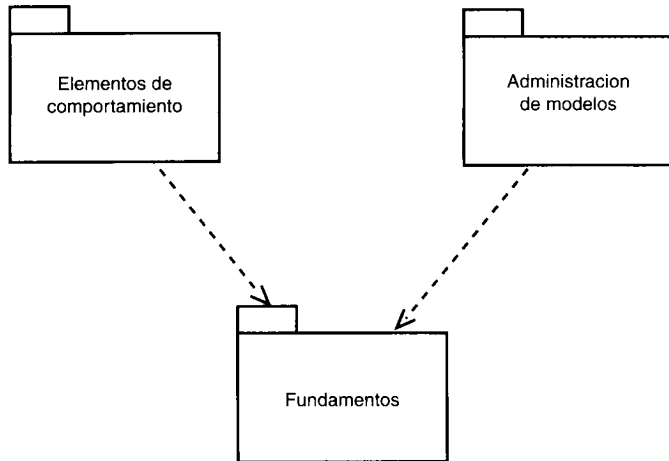


Capa del metamodelado: cercano y personal

Puesto que la capa de metamodelado es la base de su panorama, examinémosla más detalladamente.

Imagine a esta capa como una formación de tres paquetes: *Fundamentos*, *Elementos de comportamiento* y *Administración de modelos*. La figura 14.2 le muestra lo que quiero decir.

FIGURA 14.2
Los paquetes en la capa de metamodelado del UML.



Como en el caso de cualquier paquete, cada uno de estos grupos relacionan elementos entre sí. (¿Utilizamos al UML para modelar al UML? ¡Por supuesto!)

¿Cuáles son estos elementos? El paquete Fundamentos contiene:

- Núcleo
- Elementos auxiliares
- Tipos de datos
- Mecanismos de extensión

El paquete de Elementos de comportamiento contiene:

- Comportamiento en común
- Colaboraciones
- Casos de uso
- Máquinas de estado

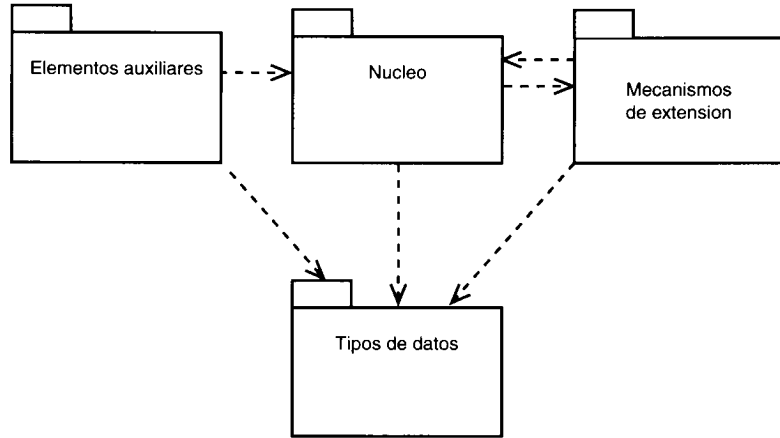
El paquete de Administración de modelos es, en sí, un modelo. Es un diagrama de clases que muestra cómo se relacionan los elementos del UML entre sí, como son los paquetes o subsistemas.

El paquete de Fundamentos

Hagamos una retrospectiva y entremos a otro nivel. Empezaré con el paquete de fundamentos, cuyos componentes aparecen en la figura 14.3.

FIGURA 14.3

Los paquetes del propio paquete de Fundamentos.



El núcleo define lo que necesita para crear un modelo UML. Cada uno de los elementos definidos es *abstracto* (lo que significa que no puede crear instancias de él) o *concreto* (con el que sí podrá). Entre los elementos abstractos se encuentran *ElementoDeModelo*, *ElementoGeneralizable* y *Clasificador*. Entre los concretos se encuentran *Clase*, *Interfaz*, *Asociación* y *Tipo de datos*.



Verá en esta sección que diré que un paquete “define”, “establece” o “da los detalles formales de” un elemento (o concepto). Ello significa tres cosas: (1) el paquete muestra al elemento dentro de un diagrama de clases (otro ejemplo de usar al UML para modelar al UML), (2) el paquete contiene reglas para utilizar el elemento, y (3) el paquete proporciona información respecto al significado del elemento.

El diagrama de clases se conoce como *sintaxis abstracta*, las reglas se conocen como *reglas bien formadas*, y al significado se le conoce como *semántica*.

He aquí otra forma de ver la distinción entre abstracto y concreto. Nunca tendrá nada en su modelo a lo que usted llame explícitamente *ElementoDeModelo* o *Clasificador* (aunque claro, utilizará *tipos* *ElementosDeModelo* y *Clasificadores* todo el tiempo). Un clasificador, por ejemplo, es cualquier elemento que describe estructura y comportamiento. Imagine a un clasificador como una forma resumida de referirse a una *clase*, *componente*, *nodo*, *actor*, *interfaz*, *indicación*, *subsistema*, *caso de uso* o *tipo de datos*. Decir que algo se le aplica a un clasificador es equivalente a decir que se aplica a cualquiera de estos otros denominadores.

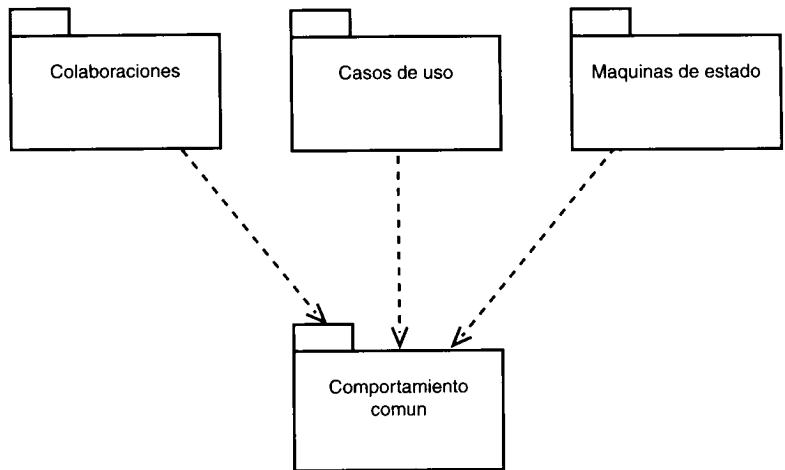
En tal caso, ¿los elementos concretos se derivan de los abstractos? Así es. ¿Ello significará que hablamos de clases y herencia? Por supuesto, pero dado que estamos en la capa de metamodelado, en realidad hablamos de *metaclases*. Por ello, un “clasificador” es una “metaclase”. (¿Qué tanto sentido podría haber tenido esta frase en la hora 1?)

Continuemos con los otros paquetes dentro de los fundamentos. Elementos Auxiliares, un paquete que redeondea al Núcleo, define la Dependencia, Componentes y Nodos, entre otros. El paquete Tipo de Datos, especifica los tipos de datos que el UML utiliza, incluyendo los tipos primitivos (enteros, cadena y tiempo) y enumeraciones. Una enumeración como por ejemplo el Booleano lista los valores posible. El paquete de Mecanismos de Extensión le especifica cómo puede extender el UML e incluye algunas extensiones ya hechas. Analizaremos de manera más detallada este paquete más adelante, en la sección llamada Extensión del UML.

El paquete de los elementos de comportamiento

Este paquete es la parte del UML que se encarga de modelar el procedimiento de un sistema. Los paquetes de que consta aparecen en la figura 14.4.

FIGURA 14.4
Los paquetes que conforman al propio paquete de Elementos de comportamiento.



El paquete de comportamiento común proporciona los conceptos de los elementos dinámicos, y soporta otros paquetes como son: casos de uso, máquinas de estado y colaboraciones. Estos “conceptos” incluyen *Señal*, *Enlace* y *Punto final de asociación*.

El paquete de colaboraciones abarca un ámbito más amplio que tan sólo los diagramas de colaboraciones que utilizó en la hora 10, “Diagramas de colaboraciones”. En este contexto, una “colaboración” describe la forma en que los clasificadores y sus asociaciones trabajan en conjunto para realizar una tarea en particular. Los diagramas de colaboraciones y los de caso de uso son parte de la perspectiva. La idea es que los clasificadores conformen al *contexto* de la colaboración; las asociaciones conforman a la *interacción*.

No es de extrañar que el paquete de casos de uso detalle los conceptos (como a un *Actor* y un *CasoDeUso*) que en él se encuentran. (Ambos conceptos son clasificadores, como lo indiqué en la sección anterior.) El objetivo general es tener la posibilidad de describir el comportamiento de un sistema sin entrar en detalles.

Tampoco debe sorprender que el paquete de Máquinas de estado dé los detalles formales de los conceptos que hay detrás de los diagramas de estados y de actividad que ya ha utilizado.

Administración de modelos

Este paquete define al *Modelo*, *Subsistema* y *Paquete*. La meta de estos elementos es agrupar los *ElementosDeModelo* de todo tipo.

Extensión del UML

Como ya lo ha visto en las horas anteriores, podrá pulir sus diagramas UML mediante la adición de detalles que expliquen mejor su significado. Los estereotipos, restricciones y valores etiquetados son herramientas útiles dentro del UML.

Puede crear una extensión sobre la marcha para agregar a su modelo cuestiones e ideas importantes de su dominio. Esto fue evidente la hora anterior: las restricciones en algunas de las comunicaciones entre nodos revelan las reglas de los diversos tipos de redes.

El UML incluye varios estereotipos, restricciones y valores etiquetados. Como ya lo mencioné, son parte del paquete Extensiones el cual, a su vez, se encuentra en el paquete Fundamentos de la capa de metamodelado. Cada una de estas extensiones incluidas es adecuada para uno (a veces dos) de los elementos del UML. Las siguientes secciones tratarán a tales extensiones.

Estereotipos

El propósito de un estereotipo es extender a un elemento del UML para que sea una instancia de una nueva metaclase, y se escribe entre dos pares de paréntesis angulares. Esto agrega una gran flexibilidad. Lo que significa que usted podrá utilizar un elemento existente del UML como base para crear sus propios elementos: elementos que capturen algún aspecto de su propio sistema o dominio de una forma en que no podrían hacerlo los elementos del UML.

La intención del estereotipo es permitir a la entidad recién creada que embone con los demás dentro de una herramienta de modelado. Las herramientas de modelado (como Rational Rose, SELECT Enterprise o Visual UML) tienen que almacenar y manejar las clases para la generación de código y la de informes. El mecanismo del estereotipo les permite hacerlo con sus creaciones.

El UML incluye un extenso conjunto de estereotipos generados. Podrá agregar de uno en uno o dos elementos. Las siguientes subsecciones organizan a los estereotipos en términos de los elementos con que confluyen.

Dependencia

La relación de dependencia puede tomar la mayor cantidad de estereotipos ya creados. Cada uno extiende una relación de dependencia entre un origen (el elemento del cual parte la flecha punteada) y un destino (el elemento al que apunta la flecha). Veamos rápidamente a cada dependencia estereotipada.

Una dependencia de tipo «se convierte en» muestra que el origen y el destino son el mismo objeto en distintos momentos. El origen se convierte en el destino con (posiblemente) diferentes roles y valores. «llamar» tiene una operación como origen y otra como su destino. En esta dependencia estereotipada, la operación de origen invoca a la de destino. Una dependencia «copiar» indica que el destino es una copia exacta del origen. En una dependencia «derivar», el origen se deriva del destino.

¿Recuerda el concepto de visibilidad de la hora 5, “Agregación, composición, interfaces y realización”? Si tiene una operación que sea privada dentro de una clase en particular, aún podrá hacerla accesible a otra clase. Coloque la otra clase (origen) y la operación (destino) en una dependencia «reunir» o «friend». El origen tendrá acceso al destino sin importar la visibilidad.

Una dependencia entre dos casos de uso también puede tener un estereotipo. Ya ha utilizado dos de ellos, «extender» y «usar», aunque substituyó «incluir» por «usar». «extender» le indica que los comportamientos del caso de uso de destino se agregan al caso de uso de origen. «usar» indica que algunos casos de uso tienen cierto comportamiento en común, y este estereotipo le permite utilizar dicho comportamiento sin tener que repetirlo una y otra vez.

Una dependencia «importar» se establece entre dos paquetes. Este estereotipo agrega el contenido del destino al espacio de nombres del origen (el aspecto del paquete que agrupa los nombres que lo constituyen).

El estereotipo «instancia» indica que el origen es una instancia de su destino, que siempre será un clasificador. En una dependencia de «metadestino», tanto el destino como el origen son clasificadores, y el destino es la metaclase del origen.

En un «enviar», el origen es una operación y el destino es una señal. El estereotipo muestra que el origen envía la señal.

Clasificador

Los estereotipos extienden a los clasificadores de diversas formas. El estereotipo «metaclase» muestra que el clasificador al que está adjunto es una metaclase de otra clase. El «tipodeautoridad» indica que un clasificador tiene objetos que provienen de un antecesor en particular. También puede usar «tipodeautoridad» en una dependencia para mostrar que el destino es un tipo de potestad del origen. (Normalmente utilizará éste cuando modele bases de datos.)

Los estereotipos «proceso» y «subproceso» tienen que ver con el flujo de control. Ambos indican que su clasificador es una clase activa; es decir, sus objetos pueden iniciar la actividad de control. Un proceso puede consistir de varios subprocesos (flujos de control), y puede ejecutarse al mismo tiempo que otros procesos. Un subproceso puede ejecutarse junto con otros subprocesos en el mismo proceso.

Un clasificador con el estereotipo «utileria» es una colección titulada de atributos y operaciones que no son miembros de tal clasificador: un clasificador que no tiene instancias.

Finalmente, un clasificador puede tener al abuelo (¡casi literal!) de todos los estereotipos: «estereotipo». Este indica que el clasificador funciona como un estereotipo, y le permite modelar jerarquías de estereotipos.

Clase

Puede obtener algo más específico que con los clasificadores: también es posible extender a una clase. Un «tipo» es una clase que establece un dominio de objetos junto con atributos, operaciones y asociaciones. El «tipo» no contiene métodos (algoritmos ejecutables para sus operaciones).

Una «claseDeImplementacion» es lo contrario de un «tipo». Representa la implementación de una clase en un lenguaje de programación.

Generalización

Es una relación entre clasificadores, con su propio pequeño conjunto de estereotipos. «heredar» significa que las instancias del subtipo no pueden substituirse por instancias del supertipo. «subclase» hace lo mismo que en las clases: significa que las instancias de la subclase no son sustituibles por instancias de la superclase. «privado» denota una herencia exclusiva: oculta los atributos heredados y operaciones de una clase a sus ancestros.

Paquete

Los estereotipos de los paquetes son directos. Una «fachada» es un paquete que contiene referencias a elementos de otro paquete, y que no contiene elementos propios. Un «sistema» es una colección de modelos de un sistema. Un «cabo» es un paquete que proporciona sólo las partes públicas de otro paquete.

TÉRMINO NUEVO

Además de los elementos que he dicho, un paquete puede incluir patrones.

Un patrón es un tipo de colaboración entre los elementos que ha probado su efectividad en diversas situaciones. Un «marcoDeTrabajo» es un paquete estereotipado que sólo contiene patrones.

Dado que los paquetes pueden encontrarse dentro de paquetes, sirve de mucho tener un estereotipo que indique cuál de los paquetes está en el nivel superior. Tal estereotipo es el «paqueteDeNivelSuperior». (¡Le dije que los estereotipos de los paquetes son directos!)

Componente

Los estereotipos para los componentes son aún más directos. Puede mostrar que un componente es un documento, un ejecutable, un archivo, una tabla de datos o una biblioteca. Los estereotipos respectivos son «documento», «ejecutable», «archivo», «tabla» y «biblioteca».

Algunos otros estereotipos

En esta subsección, he conjugado algunos estereotipos utilizados con otros elementos del UML.

Un comentario que aparece en una nota adjunta puede tener un estereotipo «requerimiento», que denota que el comentario establece un requisito para el elemento adjunto a la nota.

Dentro de una clase, una operación o un método pueden crear una instancia o destruirla. (Quizá haya visto métodos como *constructor* y *destructor* en Java.) Tales características se indican mediante «crear» y «destruir» respectivamente.

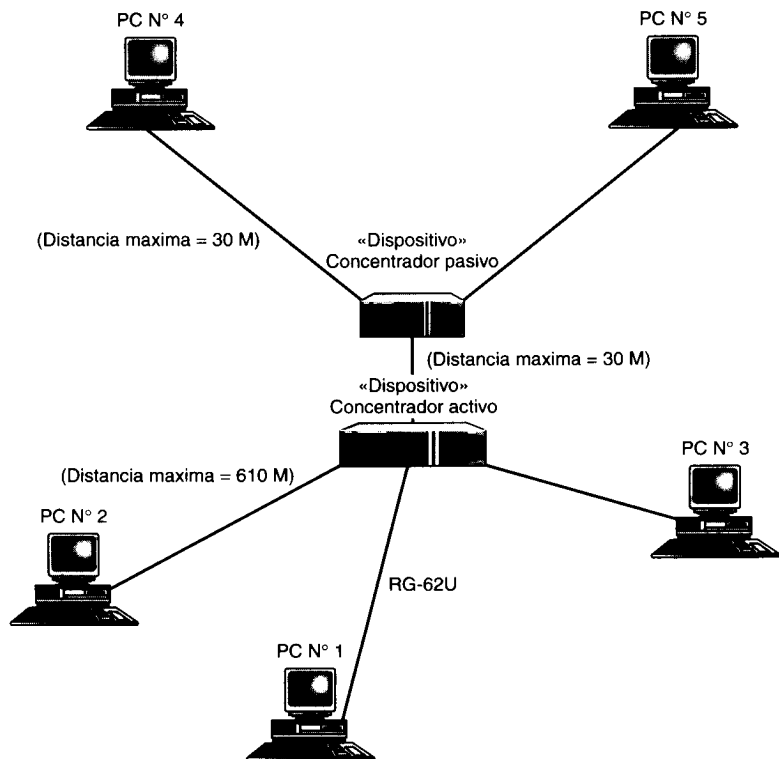
Las restricciones, que son el mecanismo de extensión que ahora trataré, pueden también funcionar con los estereotipos. En ocasiones usará una restricción para mostrar las condiciones previas a una operación; aunque algunas veces mostrará sus condiciones posteriores. Tales restricciones las estereotipará con «condicionPrevia» o «condicionPosterior». En ocasiones adjuntará una restricción a un conjunto de clasificadores o relaciones, y necesitará indicar que las condiciones de la restricción deberán tener todos los clasificadores, relaciones e instancias. Para ello, deberá estereotipar la restricción como «invariable».

Estereotipos gráficos

Algunas veces en su dominio, tal vez tendrá que obtener un nuevo símbolo o dos para transmitir algo a un cliente. Como lo mencioné en la hora anterior, los diagramas de distribución pueden encajar de forma importante en esto. Por lo general, hay disponibles figuras de procesadores y dispositivos, y pueden remplazar a los cubos que vió en la hora 13, “Diagramas de distribución”. La figura 14.5 le muestra un ejemplo. Es una versión estilizada de la figura 13.7, un modelo de una ARCNet.

FIGURA 14.5

*Un modelo estilizado
de una ARCNet.*



Restricciones

Las restricciones se encuentran entre llaves. Proporcionan las condiciones para las asociaciones, extremos de vínculos, generalizaciones y peticiones (transmisiones de señales o llamadas a operaciones).

La restricción {o} se aplica a un conjunto de asociaciones, y muestra que sólo una asociación podrá utilizarse para tal conjunto. Otra restricción basada en asociaciones, {implícito}, indica que una asociación es conceptual.

Los extremos de vínculos, que son puntos finales de vínculos entre objetos, pueden contener cualquier cantidad de restricciones. Cada restricción indica el porqué el objeto en el extremo del vínculo es visible. {parámetro} muestra que el objeto es un valor necesario relativo al vínculo. {propio} le indica que el objeto es el despachador de una petición. {global} y {local} indican el ámbito del objeto respecto al vínculo. {asociación} denota que el objeto es visible por su coalición.

Un conjunto de generalizaciones pueden ser {completo} (todos sus subtipos han sido especificados) o {incompleto} (aún pueden agregarse subtipos). Otro conjunto de generalizaciones pueden ser {traslapado} (más de un subtipo puede funcionar como un tipo de instancia) o {desarticulado} (sólo un subtipo puede ser un tipo de una instancia, lo que es predeterminado en la generalización).

Si una petición se envía a diversas instancias de destino en un orden no especificado, es una {difusión}. Si varias instancias devuelven valores, y la mayoría de tales valores determinan un solo valor, la restricción será un {voto}.

Valores etiquetados

Un valor etiquetado se escribe entre llaves. Consiste en una *etiqueta*, un signo = y un *valor*.

Los valores etiquetados ya elaborados son adecuados para los clasificadores, componentes, atributos, instancias y operaciones. Una etiqueta {documentación = }, se aplica a cualquier elemento. A la derecha del =, debe colocar una descripción, explicación o comentario respecto al elemento al que adjuntó este valor etiquetado.

Puede adjuntar una {ubicación = } a un clasificador o componente. Cuando lo adjunta a un clasificador, debe proporcionar al componente del cual es parte. Cuando lo hace a un componente, debe indicar el nodo donde se encuentra.

El valor etiquetado {persistencia = } puede ir en un atributo, clasificador o instancia. Denota la permanencia del estado del elemento al que lo ha adjuntado. Los valores posibles son *permanente* (el estado persiste cuando la instancia se destruye) y *transitorio* (cuando la instancia se destruye, también lo hace el estado).

{semántica = } especifica el significado de un clasificador o una operación. {responsabilidad} es una obligación de un clasificador.

Resumen

Esta hora trató los conceptos básicos del UML. El objetivo fue el de darle un conocimiento profundo que le permitirán aplicar al UML en situaciones reales que no siempre puedan reflejarse en los ejercicios de un libro de texto. Hemos tratado estos conceptos luego de todos los diagramas porque tenía que comprender los elementos del lenguaje antes de profundizar en sus fundamentos.

El UML consta de cuatro capas: objetos del usuario, modelado, metamodelado y meta-metamodelado (que van desde específicos hasta generales). Cuando analice un sistema, típicamente trabajará en las primeras dos capas. Cuando aprenda los conceptos del UML, por lo general se encontrará en la tercera. La cuarta capa está orientada a los teóricos y diseñadores del lenguaje, en lugar de los usuarios del lenguaje y analistas de sistemas.

El UML incluye varias extensiones propias. Cada uno de estos estereotipos, restricciones y valores etiquetados se orientan a ser usados por uno o dos de los símbolos del UML.

Si le hubiera dado todos estos conceptos fundamentales al iniciar en la hora 1 ¿los habría entendido?

Preguntas y respuestas

P Veo que el UML tiene varias reglas. ¿Quién las implementa?

R Como ya dije, la policía UML no le acecha para verificar que su modelo sea preciso. No obstante, una herramienta de modelado le podrá ayudar a ceñirse a las reglas. Por ejemplo: Visual UML tiene un archivo de estereotipos que usará de un modo sensible al contexto. Cuando intente colocar un estereotipo en un elemento en particular, sólo le permitirá seleccionar de los estereotipos adecuados para tal elemento, todos ellos en inglés. Además, le permitirá agregar estereotipos a su archivo de estereotipos.

P Los valores etiquetados parecen ser esotéricos. ¿Debo utilizarlos?

R Sí, y con frecuencia. Los valores etiquetados integrados, aunque son útiles, palidecen en comparación con los que usted definirá para sí. Puede utilizar un valor etiquetado para mantener información relacionada con la administración de un importante proyecto en su modelo: como los números de versión y los autores de las clases. Es decir, “estado”, “versión” o “autor” podrían ser sus etiquetas, y usted establecerá los valores adecuados.

Taller

Este taller reafirmará su conocimiento de los fundamentos del UML. Utilice sus procesos de reflexión en el cuestionario y localice las respuestas en el Apéndice A, “Cuestionario”. Ésta es una hora teórica, por lo que no he incluido ningún ejercicio.

Cuestionario

1. ¿Cuáles son las cuatro capas del UML?
2. ¿Qué es un clasificador?
3. ¿Porqué es importante el poder extender al UML?
4. ¿Cuáles son los mecanismos de extensión del UML?

