

# EL LENGUAJE UNIFICADO DE MODELADO. MANUAL DE REFERENCIA

**JAMES RUMBAUGH**  
**IVAR JACOBSON**  
**GRADY BOOCH**



*La referencia  
definitiva de UML  
escrita por sus  
creadores*



Addison  
Wesley

Incluye CD-ROM  
con la versión 1.3 de UML 

**J. Rumbaugh, I. Jacobson, G. Booch**

**EL LENGUAJE UNIFICADO DE MODELADO.**

**MANUAL DE REFERENCIA**

PEARSON EDUCACIÓN, S. A., Madrid, 2000

ISBN: 84-7829-037-0

Materia: Informática 681.3

Formato 195 × 250

Páginas: 552

**J. Rumbaugh, I. Jacobson, G. Booch**

**EL LENGUAJE UNIFICADO DE MODELADO. MANUAL DE REFERENCIA**

No está permitida la reproducción total o parcial de esta obra  
ni su tratamiento o transmisión por cualquier medio o método  
sin autorización escrita de la Editorial.

**DERECHOS RESERVADOS**

© 2000 respecto a la primera edición en español por:

PEARSON EDUCACIÓN, S. A.

Núñez de Balboa, 120

28006 Madrid

**ISBN: 84-7829-037-0**

Depósito Legal: TO-1.310-2000

ADDISON WESLEY es un sello editorial autorizado de PEARSON EDUCACIÓN, S. A.

*Traducido de:*

THE UNIFIED MODELING LANGUAGE. REFERENCE MANUAL

Addison Wesley Longman Inc.

© 1999

ISBN: 0-201-30998-X

*Edición en español:*

Editor: Andrés Otero

Asistente editorial: Ana Isabel García

Diseño de cubierta: DIGRAF, S. A.

Composición: COPIBOOK, S. L.

Impreso por: GRAFILLES, S.L.

IMPRESO EN ESPAÑA - PRINTED IN SPAIN



<b>Prefacio</b> .....	XI
Objetivos .....	XI
Esquema general del libro .....	XII
Convenciones en el formato de los artículos de la enciclopedia .....	XII
Convenciones de sintaxis .....	XIV
CD .....	XIV
Información adicional .....	XV
Reconocimientos .....	XV
<b>Prólogo a la edición en español</b> .....	XVII
<b>Nota sobre la traducción</b> .....	XXI

## **Parte 1: Antecedentes**

<b>Capítulo 1: Perspectiva general de UML</b> .....	3
Breve resumen de UML .....	3
Historia de UML .....	4
Objetivos de UML .....	7
Áreas conceptuales de UML .....	8
Sintaxis de los diagramas y las expresiones .....	10
<b>Capítulo 2: La naturaleza y propósito de los modelos</b> .....	11
¿Qué es un modelo? .....	11
¿Para qué sirven los modelos? .....	11
Niveles de los modelos .....	13
¿Qué hay en un modelo? .....	15
¿Cuál es el significado de un modelo? .....	16

## **Parte 2: Conceptos de UML**

<b>Capítulo 3: Un paseo por UML</b> .....	21
Vistas de UML .....	21

Vista estática .....	22
Vista de los casos de uso .....	24
Vista de interacción .....	25
Vista de la máquina de estados .....	27
Vista de actividades .....	29
Vistas físicas .....	29
Vista de gestión del modelo .....	32
Construcciones de extensión .....	33
Conexiones entre vistas .....	34
<b>Capítulo 4: La vista estática .....</b>	<b>37</b>
Descripción .....	37
Clasificadores .....	38
Relaciones .....	41
Asociaciones .....	42
Generalización .....	45
Realización .....	48
Dependencias .....	50
Restricción .....	52
Instancias .....	53
<b>Capítulo 5: La vista de casos de usos .....</b>	<b>55</b>
Descripción .....	55
Actor .....	56
Caso de uso .....	56
<b>Capítulo 6: La vista de la máquina de estados .....</b>	<b>59</b>
Descripción .....	59
Máquina de estados .....	59
Evento .....	60
Estado .....	62
Transición .....	62
Estados compuestos .....	66
<b>Capítulo 7: La vista de actividades .....</b>	<b>71</b>
Descripción .....	71
Diagrama de actividades .....	71
Actividades y otras vistas .....	74
<b>Capítulo 8: La vista de interacción .....</b>	<b>75</b>
Descripción .....	75
Colaboración .....	75
Interacción .....	76
Diagrama de secuencia .....	76

Activación .....	77
Diagrama de colaboración .....	78
Patrones .....	80

## **Capítulo 9: Vistas físicas** ..... 83

Descripción .....	83
Componente .....	83
Nodo .....	84

## **Capítulo 10: La vista de gestión del modelo** ..... 87

Descripción .....	87
Paquete .....	87
Dependencias en los paquetes .....	88
Dependencia de acceso e importación .....	89
Modelo y subsistema .....	89

## **Capítulo 11: Mecanismos de extensión** ..... 91

Descripción .....	91
Restricción .....	91
Valor etiquetado .....	92
Estereotipos .....	93
Adaptación de UML .....	94

## **Capítulo 12: El entorno de UML** ..... 95

Descripción .....	95
Responsabilidades semánticas .....	95
Responsabilidades de notación .....	96
Responsabilidades del lenguaje de programación .....	97
Modelado con herramientas .....	98

## **Parte 3: Referencia**

### **Capítulo 13: Enciclopedia de términos** ..... 103

### **Capítulo 14: Elementos estándar** ..... 479

## **Parte 4: Apéndices**

### **Apéndice A: Metamodelo de UML** ..... 495

Documentos de definición de UML .....	495
Estructura del metamodelo .....	495

Paquete de fundamentos .....	496
Paquete de elementos de comportamiento .....	497
Paquete de administración de modelos .....	497
 <b>Apéndice B: Resumen de la notación</b> .....	 499
 <b>Apéndice C: Extensiones de proceso</b> .....	 511
Cómo personalizar UML .....	511
Extensiones del proceso de desarrollo del software .....	511
Extensiones de modelado de negocios .....	513
 <b>Bibliografía</b> .....	 517
 <b>Índice</b> .....	 519

## Descripción

La vista estática es la base de UML. Los elementos de la vista estática de un modelo son los conceptos significativos en una aplicación, incluyendo conceptos del mundo real, conceptos abstractos, conceptos de implementación, conceptos de computación y todo tipo de conceptos encontrados en los sistemas. Por ejemplo, un sistema de entradas para un teatro tiene conceptos tales como entradas, reservas, planes de suscripción, algoritmos de la asignación de asientos, páginas Web interactivas para hacer pedidos, y datos de archivo para redundancia.

La vista estática captura la estructura del objeto. Un sistema orientado a objetos unifica la estructura de datos y características del comportamiento en una sola estructura de objeto. La vista estática incluye todo lo concerniente a las estructuras de datos tradicionales, así como la organización de las operaciones sobre los datos. Los datos y las operaciones son cuantificadas en clases. En la perspectiva orientada a objetos, los datos y el comportamiento se relacionan estrechamente. Por ejemplo, un objeto **de entrada** lleva datos, como su precio, fecha de la representación, y número de asiento, así como operaciones sobre él, como reservar o calcular su precio con un descuento especial.

La vista estática describe entidades de comportamiento como elementos de modelado discretos, pero no contiene los detalles de su comportamiento dinámico. Los trata como elementos para ser nombrados, poseídos por las clases, e invocados. Su ejecución dinámica es descrita por otras vistas que muestran los detalles internos de su faceta dinámica. Estas otras vistas incluyen la vista de interacción y la vista de máquina de estados. Las vistas dinámicas requieren la vista estática para describir las cosas que interactúan dinámicamente: no se puede decir *cómo* interactúa algo sin decir primero *qué* está interactuando. La vista estática es la base sobre la que se construyen las otras vistas.

Los elementos clave en la vista estática son los clasificadores y sus relaciones. Un clasificador es un elemento de modelado que describe cosas. Hay varias clases de clasificadores, como las clases, interfaces, y tipos de datos. Los aspectos de comportamiento son materializados por otros clasificadores, como los casos de uso y las señales. Los propósitos de implementación están detrás de varias clases de clasificadores, tales como subsistemas, componentes, y nodos.

Los modelos grandes se deben organizar en unidades más pequeñas para la comprensión humana y la reutilización. Un paquete es una unidad de organización de uso general para poseer y manejar el contenido de un modelo. Cada elemento está contenido en algún paquete. Un modelo es un paquete que describe una vista completa de un sistema y se puede utilizar más o menos independientemente de otros modelos; es la raíz poseedora de los paquetes más detallados que describen el sistema.



Un objeto es una unidad discreta, fuera de la cual el modelador entiende y construye un sistema. Es una instancia de una clase, es decir, un individuo con identidad cuya estructura y comportamiento son descritos por la clase. Un objeto es una pieza de estado identificable con un comportamiento bien definido que puede ser invocado.

Las relaciones entre clasificadores son asociación, generalización, y varias clases de dependencia, como la realización y el uso.

## Clasificadores

Un clasificador es un concepto discreto en el modelo, que tiene identidad, estado, comportamiento, y relaciones. Las clases de clasificadores incluyen la clase, la interfaz, y los tipos de datos. Otras clases de clasificadores son materializaciones de conceptos de comportamiento, de cosas del entorno, o de estructuras de implementación. Estos clasificadores incluyen caso de uso, actor, componente, nodo, y subsistema. La Tabla 4.1 enumera las distintas clases de clasificadores y sus funciones. El término clasificador del metamodelo incluye todos estos conceptos, pero como la clase es el término más familiar, lo discutiremos primero y definiremos los otros conceptos por diferencia con ella.

**Clase.** Una clase representa un concepto discreto dentro de la aplicación que se está modelando: una cosa física (tal como un aeroplano), una cosa de negocios (tal como un pedido), una cosa lógica (tal como un horario de difusión), una cosa de una aplicación (tal como un botón de cancelar), una cosa del computador (tal como una tabla hash), o una cosa del comportamiento (tal como una tarea). Una clase es el descriptor de un conjunto de objetos con una estructura, comportamiento, y relaciones similares. Todos los atributos y operaciones están unidos a clases o a otros clasificadores. Las clases son los focos alrededor de los cuales se organizan los sistemas orientados a objetos.

Un objeto es una entidad discreta con identidad, estado y un comportamiento invocable. Los objetos son piezas individuales, fuera de las cuales se construye un sistema ejecutable; las clases son los conceptos individuales, por los cuales se entiende y describe la multiplicidad de objetos individuales.


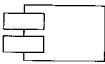


Una clase define un conjunto de objetos que tienen estado y comportamiento. El estado es descrito por atributos y asociaciones. Los atributos se utilizan generalmente para los valores puros de los datos sin identidad, tales como números y cadenas, y las asociaciones se utilizan para

Suscripción	nombre de clase
serie: Cadena categoriaDePrecio: Categoría número: Entero	atributos
conocerCoste (): Dinero reservar (serie: Cadena, nivel: NivelDe Asiento) cancelar ()	operaciones

Figura 4.1 Notación de clases



**Tabla 4.1** Tipos de Clasificadores

<i>Clasificador</i>	<i>Función</i>	<i>Notación</i>
actor	Un usuario externo al sistema	
clase	Un concepto del sistema modelado	<div style="border: 1px solid black; padding: 2px; display: inline-block;"><b>Nombre</b></div>
clase en un estado	Una clase restringida a estar en el estado dado	<div style="border: 1px solid black; padding: 2px; display: inline-block;"><b>Nombre[S]</b></div>
rol	Clasificador restringido a un uso particular en una colaboración	<div style="border: 1px solid black; padding: 2px; display: inline-block;"><b>rol:Nombre</b></div>
componente	Una pieza física de un sistema	
tipo de dato	Un descriptor de un conjunto de valores primitivos que carecen de identidad	<b>Nombre</b>
interfaz	Un conjunto de operaciones con nombre que caracteriza un comportamiento	<div style="text-align: center;">○ <b>lnombre</b></div>
nodo	Un recurso computacional	
señal	Una comunicación asíncrona entre objetos	<div style="border: 1px solid black; padding: 2px; display: inline-block;">«signal»</div>
subsistema	Un paquete que es tratado como una unidad con una especificación, implementación, e identidad	<div style="border: 1px solid black; padding: 2px; display: inline-block;">«subsystem»</div>
caso de uso	Una especificación del comportamiento de una identidad y su interacción con los agentes externos	

las conexiones entre objetos con identidad. Las piezas individuales de comportamiento invocable se describen mediante operaciones; un método es la implementación de una operación. La historia del curso de vida de un objeto es descrita por una máquina de estados, unida a una clase. La notación para una clase es un rectángulo con compartimentos para el nombre de la clase, los atributos, y las operaciones, según se muestra en la Figura 4.1.

Un conjunto de clases puede utilizar la relación de generalización y el mecanismo de herencia construidos en ella para compartir piezas comunes de estado y descripción del comportamiento. La generalización relaciona clases más específicas (subclases) con clases más ge-

nerales (superclases) que contienen propiedades comunes a varias subclases. Una clase puede tener cero o más padres (superclases) y cero o más hijos (subclases). Una clase hereda estado y descripción del comportamiento de sus padres y de otros antecesores, y define el estado y descripción del comportamiento que sus hijos y otros descendientes heredan.

Una clase tiene un nombre único dentro de su contenedor, que es generalmente un paquete, pero algunas veces es otra clase. La clase tiene visibilidad con respecto a su contenedor; la visibilidad especifica cómo puede ser utilizada por otras clases externas al contenedor. Una clase tiene una multiplicidad que especifica cuantas instancias de ella pueden existir. La mayoría de las veces, es muchos (cero o más, sin límite explícito), pero existen clases unitarias de las que existe una sola instancia durante la ejecución.

**Interfaz.** Una interfaz es la descripción del comportamiento de objetos sin dar su implementación o estado; una interfaz contiene operaciones pero no atributos, y no tiene asociaciones salientes que muestren la visibilidad desde la propia interfaz.

Una o más clases o componentes pueden realizar una interfaz, y cada clase implementa las operaciones de la interfaz.

**Tipo de datos.** Un tipo de datos es la descripción de los valores primitivos, que carecen de identidad (existencia independiente y posibilidad de efectos secundarios). Los tipos de datos incluyen números, cadenas, y valores enumerados. Los tipos de datos son pasados por valor y son entidades inmutables. Un tipo de dato no tiene atributos pero puede tener operaciones. Las operaciones no modifican valores de los datos, sino que pueden devolver valores de datos como resultados.

**Niveles de significado.** Las clases pueden existir en varios niveles de significación en un modelo, incluyendo los niveles de análisis, diseño, e implementación. Al representar conceptos del mundo real, es importante capturar el estado, las relaciones, y el comportamiento del mundo real. Pero los conceptos de implementación, tales como ocultación de información, eficacia, visibilidad, y métodos, no son conceptos relevantes del mundo real (son conceptos relevantes de diseño). Muchas propiedades potenciales de una clase son simplemente inaplicables en este nivel. Una clase en el análisis representa un concepto lógico en el dominio de la aplicación o en la aplicación misma. El modelo de análisis debe ser una representación mínima del sistema que se está modelando, suficiente para capturar la lógica esencial del sistema, sin entrar en temas de rendimiento o construcción.

Al representar un diseño de alto nivel, conceptos tales como hallar los estados de una clase particular, la eficacia de la navegación entre objetos, la separación del comportamiento externo y de la implementación interna, y la especificación de las operaciones exactas son relevantes a una clase. Una clase en el diseño representa la decisión de empaquetar la información de estado y las operaciones en una unidad discreta. Captura las decisiones clave del diseño, la localización de la información y la funcionalidad de los objetos. Las clases en el diseño contienen aspectos del mundo real y aspectos del sistema informático.

Finalmente, al representar código del lenguaje de programación, la forma de una clase se asemeja mucho a la del lenguaje de programación elegido, y se puede renunciar a algunas capacidades de una clase general, si no tienen ninguna implementación directa en el lenguaje. Una clase para implementación se corresponde directamente con el código del lenguaje programación.

El mismo sistema puede contener más de un nivel de clase; las clases orientadas a la implementación pueden realizar clases más lógicas en el modelo. Una clase de implementación representa la declaración de una clase como la encontramos en un lenguaje de programación particular. Captura la forma exacta de una clase, según lo requerido por el lenguaje. En muchos casos, sin embargo, el análisis, el diseño, y la información de la implementación se pueden anidar en una sola clase.

## Relaciones

Las relaciones entre clasificadores son asociación, generalización, flujo, y varias clases de dependencia, que incluyen la realización y el uso (véase la Tabla 4.2).

La relación de asociación describe conexiones semánticas entre los objetos individuales de clases dadas. Las asociaciones proporcionan las conexiones, con las cuales los objetos de diversas clases pueden interactuar. Las relaciones restantes relacionan las descripciones de clasificadores con ellos mismos, y no con sus instancias.

La relación de generalización relaciona descripciones generales de los clasificadores padre (superclases) con clasificadores hijos especializados (subclases). La generalización facilita la descripción de clasificadores, sin piezas de declaración incremental, cada uno de los cuales se agrega a la descripción heredada de sus antecesores. El mecanismo de herencia construye descripciones completas de clasificadores a partir de descripciones incrementales que utilizan relaciones de generalización. La generalización y la herencia permiten a diferentes clasificadores, compartir atributos, operaciones, y relaciones que tienen en común, sin repetirlas.

La relación de realización relaciona una especificación con una implementación. Una interfaz es una especificación del comportamiento sin la implementación; una clase incluye la estructura de implementación. Una o más clases pueden realizar una interfaz, y cada clase implementa las operaciones de interfaz.

**Tabla 4.2** Tipos de Relaciones

<i>Relación</i>	<i>Función</i>	<i>Notación</i>
asociación	Una descripción de una conexión entre instancias de clases	— — —
dependencia	Una relación entre dos elementos del modelo	— — — ➤
flujo	Una relación entre dos versiones de un objeto en sucesivas veces	— — — ➤
generalización	Una relación entre una descripción más general y una variedad más específica de la general, usada para herencia	— ➤
realización	Relación entre una especificación y su implementación	— — ➤
uso	Una situación en la que un elemento requiere otro para su correcto funcionamiento	— — — ➤

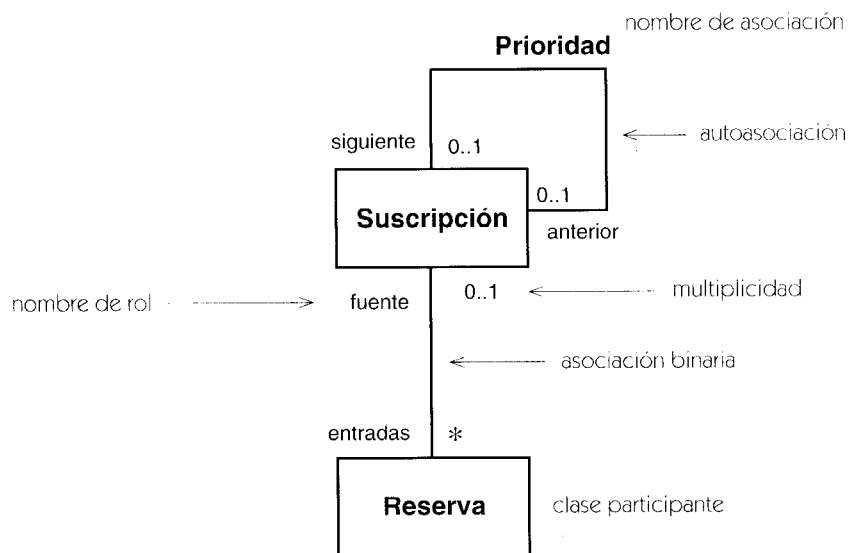
La relación de flujo relaciona dos versiones de un objeto en momentos sucesivos. Representa una transformación del valor, estado, o localización de un objeto. La relación de flujo puede conectar roles en una interacción. Las variedades de flujo son conversión (dos versiones del mismo objeto) y copia (un nuevo objeto creado de un objeto existente).

La relación de dependencia relaciona las clases cuyo comportamiento o implementación afecta a otras clases. Hay varias clases de dependencia además de la realización, incluyendo la traza (una conexión leve entre elementos de diversos modelos), refinamiento (la correspondencia entre dos niveles de significación), uso (un requisito para la presencia de otro elemento dentro de un solo modelo), y ligadura (la asignación de valores a los parámetros de una plantilla). La dependencia de uso se utiliza con frecuencia para representar relaciones de implementación, tales como relaciones al nivel de código. La dependencia es particularmente útil cuando está resumida en unidades de organización del modelo, tales como paquetes, en los cuales se muestra la estructura arquitectónica de un sistema. Por ejemplo, las restricciones para la compilación se pueden mostrar con dependencias.

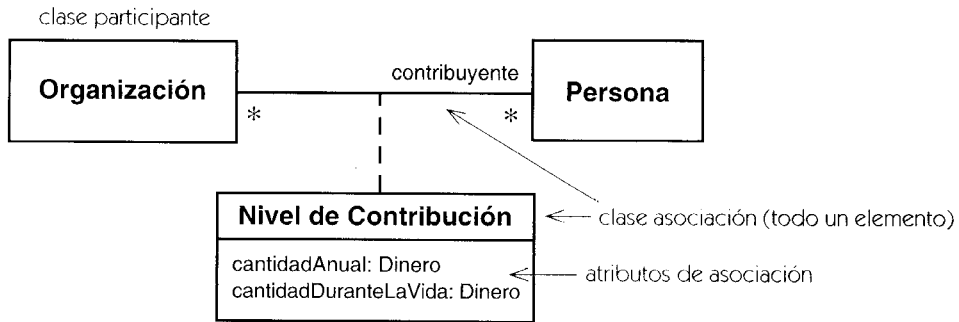
## Asociaciones

Una asociación describe conexiones discretas entre objetos u otras instancias de un sistema. Una asociación relaciona una lista ordenada (tupla) de dos o más clasificadores, con las repeticiones permitidas. El tipo más común de asociación es una asociación binaria entre un par de clasificadores. Una instancia de una asociación es un enlace. Un enlace abarca una tupla (una lista ordenada) de objetos, cada uno dibujado a partir de su clase correspondiente. Un enlace binario abarca un par de objetos.

Las asociaciones llevan la información sobre relaciones entre objetos en un sistema. Cuando se ejecuta un sistema, los enlaces entre objetos se crean y se destruyen. Las asociaciones son el “pegamento” que mantiene unido un sistema. Sin asociaciones, no hay nada más que clases aisladas que no trabajan juntas.



**Figura 4.2** Notación de asociación



**Figura 4.3** Clase de asociación

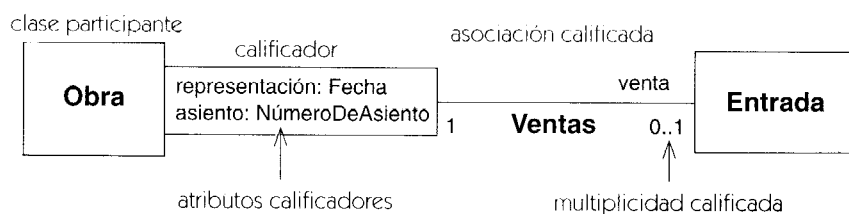
Un solo objeto se puede asociar a sí mismo si la misma clase aparece más de una vez en una asociación. Si la misma clase aparece dos veces en una asociación, las dos instancias no tienen que ser el mismo objeto, y no lo son generalmente.

Cada conexión de una asociación a una clase se llama extremo de la asociación. La mayoría de la información sobre una asociación se une a uno de sus extremos. Los extremos de la asociación pueden tener nombres (nombres de rol) y visibilidad. La propiedad más importante que tienen es la multiplicidad: cuantas instancias de una clase se pueden relacionar con una instancia de otra clase. La multiplicidad es más útil para las asociaciones binarias porque su definición para las asociaciones de aridad- $n$  es complicada.

La notación para una asociación binaria es una línea o una trayectoria que conecta las clases que participan. El nombre de asociación se pone a lo largo de la línea, con el nombre de rol y la multiplicidad en cada extremo, según lo mostrado en la Figura 4.2.

Una asociación puede también tener atributos por sí misma, en cuyo caso es una asociación y una clase, es decir, una clase asociación (véase la Figura 4.3). Si un atributo de la asociación es único dentro de un conjunto de objetos relacionados, entonces es un calificador (véase la Figura 4.4). Un calificador es un valor que selecciona un objeto único del conjunto de objetos relacionados a través de una asociación. Las tablas de valores y los vectores se pueden modelar como asociaciones calificadas. Los calificadores son importantes para modelar nombres y códigos de identificación. Los calificadores también modelan índices en un modelo de diseño.

Durante el análisis, las asociaciones representan relaciones lógicas entre objetos. No hay gran necesidad de imponer la dirección, o de tratar sobre cómo implementarla. Las asociaciones redundantes deben ser evitadas porque no añaden ninguna información lógica. Durante el diseño, las asociaciones capturan las decisiones de diseño sobre la estructura de datos, así como la se-



**Figura 4.4** Asociación calificada

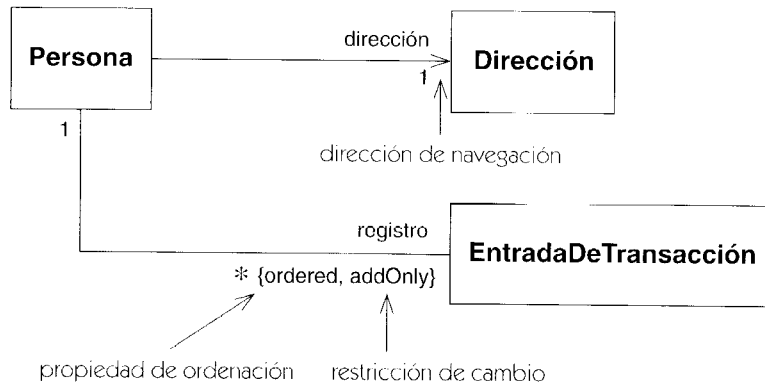


Figura 4.5 Propiedades del diseño de asociaciones

paración de responsabilidades entre clases. La direccionalidad de las asociaciones es importante, y las asociaciones redundantes pueden ser incluidas por eficacia en el acceso al objeto, así como para localizar la información en una clase particular. Sin embargo, en esta etapa de modelado, las asociaciones no se deben comparar con los punteros de C++. Una asociación navegable en la etapa de diseño representa información de estado disponible para una clase, pero puede implementarse en código del lenguaje de programación de varias maneras. La implementación puede ser un puntero, una clase contenedora embebida en una clase, o incluso una tabla de objetos totalmente separada. Otras clases de propiedades del diseño incluyen visibilidad y posibilidad de cambio de enlaces. La Figura 4.5 muestra algunas propiedades del diseño de asociaciones.

**Agregación y composición.** Una agregación es una asociación que representa una relación todo-parte. Se muestra adornando con un diamante hueco el extremo de la trayectoria unida a la clase agregada. Una composición es una forma más fuerte de asociación, en la cual el compuesto es el responsable único de gestionar sus partes, por ejemplo su asignación y desasignación. Se muestra con un diamante relleno adornando el extremo compuesto. Hay una asociación separada entre cada clase que representa una parte y la clase que representa el todo, pero por conveniencia, las trayectorias unidas al todo pueden ensamblarse juntas para dibujar el sistema entero de asociaciones como un árbol. La Figura 4.6 muestra un agregado y un compuesto.

**Enlaces.** Una instancia de una asociación es un enlace. Un enlace es una lista ordenada de referencias a objetos, cada uno de los cuales debe ser una instancia de la clase correspondiente en la asociación o una instancia de un descendiente de la clase. Los enlaces en un sistema constituyen parte del estado del sistema. Los enlaces no existen independientemente de los objetos; to-

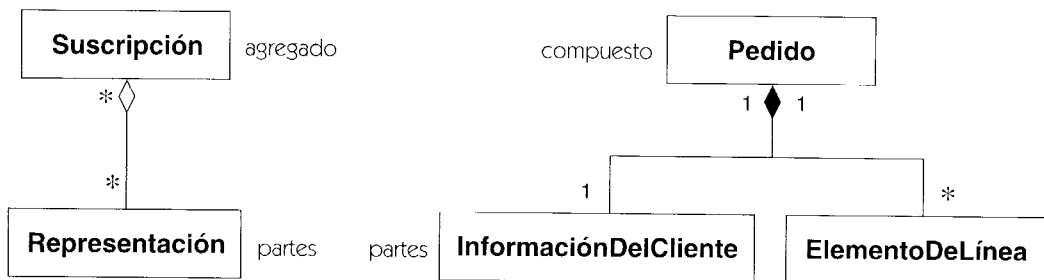


Figura 4.6 Agregación y Composición

man su identidad de los objetos que se relacionan (en términos de bases de datos, la lista de objetos es la clave para el enlace). Conceptualmente, una asociación es distinta de las clases que relaciona. En la práctica, las asociaciones se implementan a menudo usando punteros en las clases participantes, pero se pueden implementar como objetos contenedores, separados de las clases que conectan.

***Bidireccionalidad.*** Los diferentes extremos de una asociación son distinguibles, incluso si dos de ellos implican la misma clase. Esto significa simplemente que diversos objetos de la misma clase pueden ser relacionados. Debido a que los extremos son distinguibles, una asociación no es simétrica (excepto en casos especiales); los extremos no pueden ser intercambiados. Esto es sólo sentido común: el sujeto y el predicado de un verbo no son permutables. A veces, se dice que una asociación es bidireccional. Esto significa que las relaciones lógicas trabajan en ambos sentidos. Esta declaración se entiende mal con frecuencia, incluso por algunos metodólogos. No significa que cada clase “conozca” a la otra clase, o que, en una implementación, sea posible tener acceso a cada clase desde la otra. Significa simplemente que cualquier relación lógica tiene un contrario, tanto si lo contrario es fácil de computar como si no. Las asociaciones se pueden marcar con direcciones de navegación, para asignar la posibilidad de atravesar una asociación en una dirección pero no en la otra, como una decisión del diseño.

¿Por qué es relacional el modelo básico, en vez de con punteros, que son un modelo más frecuente en lenguajes de programación? La razón es que un modelo procura capturar el objetivo subyacente a una implementación. Por consiguiente, si una relación entre dos clases se modela como un par de punteros, los punteros deben estar relacionados. El concepto de asociación reconoce que las relaciones son significativas en ambas direcciones, sin importar cómo se implementan. Es sencillo convertir una asociación en un par de punteros para su implementación, pero es muy difícil reconocer dos punteros que son contrarios entre sí, a menos que este hecho sea parte del modelo.

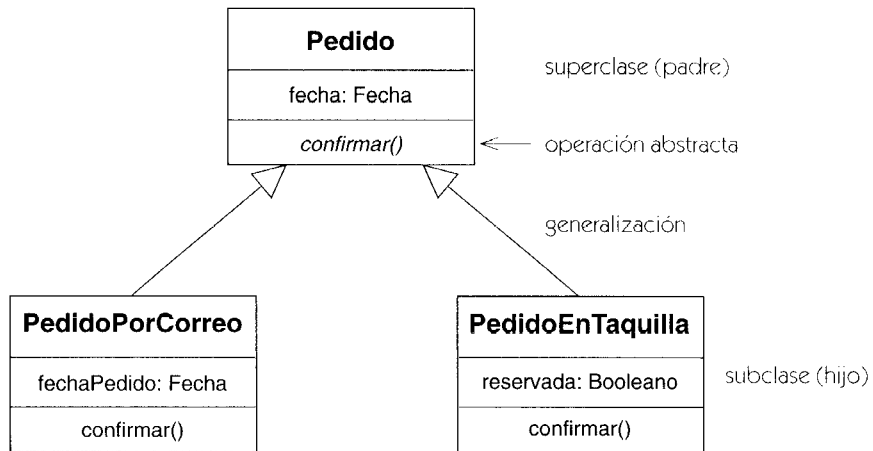
## Generalización

La relación de generalización es una relación taxonómica entre una descripción más general y una descripción más específica, que se construye sobre ella y la extiende. La descripción más específica es completamente consistente con la más general (tiene todas sus propiedades, miembros y relaciones), y puede contener información adicional. Por ejemplo, una hipoteca es una clase más específica de préstamo. Una hipoteca guarda las propiedades básicas de un préstamo pero agrega propiedades adicionales, tales como una casa o como seguridad para el préstamo. La descripción más general se llama padre; un elemento en la clausura transitiva es un antecesor. La descripción más específica se llama hijo; un elemento en la clausura transitiva es un descendiente. En el ejemplo, el **préstamo** es la clase del padre y la **hipoteca** es la clase hija. La generalización se utiliza para los clasificadores (clases, interfaces, tipos de los datos, casos de uso, actores, señales, etcétera), paquetes, máquinas de estado, y otros elementos. Para las clases, los términos superclase y subclase se utilizan para el padre y el hijo.

Una generalización se dibuja como una flecha desde hijo al padre, con un triángulo hueco en el extremo conectado con el padre (Figura 4.7). Varias relaciones de generalización se pueden dibujar como un árbol con una punta de flecha que se ramifica en varias líneas a los hijos.

***Propósito de la generalización.*** La generalización tiene dos propósitos. El primero debe definir las condiciones bajo las cuales una instancia de una clase (u otro elemento) puede ser





**Figura 4.7** Notación de la generalización

utilizado cuando se declara una variable (tal como un parámetro o variable del procedimiento), conteniendo valores de una clase dada. Esto se llama principio de sustitución (de Bárbara Liskov). La regla es que una instancia de un descendiente se puede utilizar dondequiera que esté declarado el antecesor. Por ejemplo, si una variable se declara para contener préstamos, un objeto hipoteca es un valor legal para esa variable.

La generalización permite operaciones polimórficas, es decir, operaciones cuya implementación (método) es determinada por la clase de objeto a la que se aplican, en vez de ser indicada explícitamente por el llamador. Esto funciona porque una clase padre puede tener muchos hijos posibles, cada uno de los cuales implementa su propia variación de una operación, que se define a través de todo el conjunto de clases. Por ejemplo, el cálculo de intereses funcionaría de forma diferente en una hipoteca y en un préstamo para un automóvil, pero cada uno de ellos es una variación del cálculo de interés de la clase padre **Préstamo**. Cuando se declara una variable de la clase padre, cualquier objeto de clases hijas puede ser asociado a ella, lógicamente cada uno de ellos con sus propias operaciones según la clase a la que pertenezca. Esto es particularmente útil porque se pueden agregar nuevas clases más adelante, sin necesidad de modificar las llamadas polimórficas existentes. Por ejemplo, se podría agregar más adelante una nueva clase préstamo, y el código existente que utiliza la operación del **cálculo de interés** todavía funcionaría. Una operación polimórfica se puede declarar sin una implementación en una clase padre con objeto de que se proporcione una implementación para cada clase descendiente. Tal operación incompleta es abstracta y se representa poniendo su nombre en cursiva.

El otro propósito de la generalización es permitir la descripción incremental de un elemento que comparte las descripciones de sus antecesores. Esto se llama herencia. La herencia es el mecanismo por el cual la descripción de los objetos de una clase se ensambla a partir de los fragmentos de declaración de la clase y de sus antecesores. La herencia permite que las partes compartidas de la descripción sean declaradas una vez y compartidas por muchas clases, en lugar de que se repitan en cada clase que las utiliza. Al compartir se reduce el tamaño del modelo. Más importante aún, reduce el número de los cambios que se deben realizar en una actualización del modelo y reduce la posibilidad de inconsistencia accidental. La herencia trabaja de una manera similar con otras clases de elementos, tales como estados, señales, y casos de uso.

## Herencia

Cada clase de elemento generalizable tiene un conjunto de propiedades que se puede heredar. Para cualquier elemento del modelo, éstas incluyen restricciones. Para los clasificadores, también incluyen las propiedades (atributos, operaciones, y recepción de señales) y la participación en asociaciones. Un hijo hereda todas las propiedades heredables de todos sus antecesores. Su conjunto completo de propiedades es el conjunto de propiedades heredadas junto con las propiedades que declara directamente.

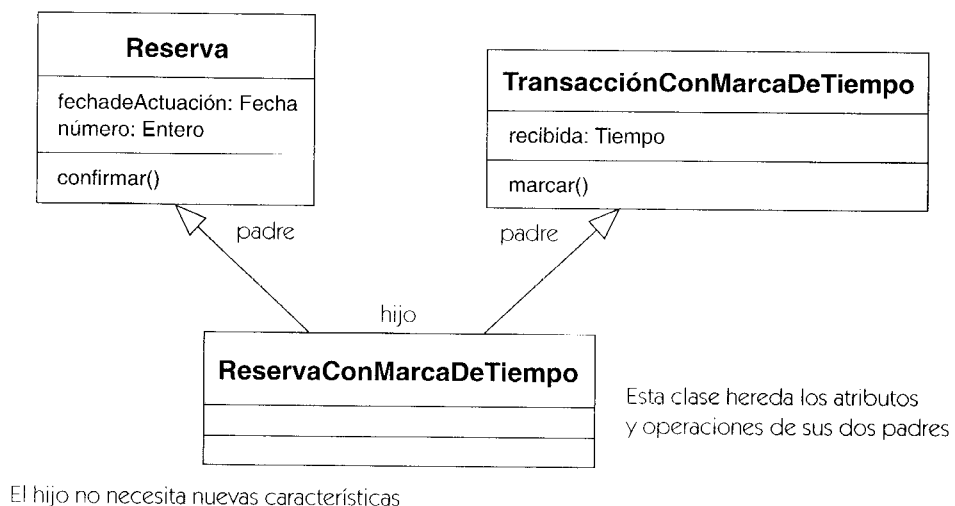
En un clasificador, ningún atributo con la misma signature se puede declarar más de una vez (directamente o heredado). De lo contrario, hay un conflicto y el modelo está mal formado. Es decir, un atributo declarado en un antecesor no puede estar redeclarado en un descendiente. Una operación se puede declarar en más de un clasificador, con tal que las especificaciones sean consistentes (los mismos parámetros, restricciones, y significados).

Las declaraciones adicionales son simplemente redundantes. Un método puede ser declarado por múltiples clases en una jerarquía. Un método unido a un descendiente reemplaza y sustituye (elimina) a un método con la misma signature declarado en cualquier antecesor. Sin embargo, si dos o más copias distintas de un método son heredadas por una clase (vía herencia múltiple de diversas clases), entonces están en conflicto y el modelo está mal formado. (Algunos lenguajes de programación permiten elegir uno de los métodos explícitamente. Encontramos más simple y más seguro tener que redefinir el método en la clase hija.) Las restricciones en un elemento son la unión de las restricciones en el propio elemento y todos sus antecesores; si cualesquiera de ellas son contrarias, entonces el modelo está mal formado.

En una clase concreta, cada operación heredada o declarada debe tener un método definido, directamente o por herencia de un antecesor.

## Herencia múltiple

Si un clasificador tiene más de un padre, hereda de ambos (Figura 4-8). Sus propiedades (atributos, operaciones, y señales) son la unión de los de sus padres. Aunque la misma clase



**Figura 4.8** Herencia múltiple

aparezca como antecesor por más de una ruta, sólo contribuye con una copia de cada uno de sus miembros. Si una propiedad, con la misma signatura, es declarada por dos clases que no la heredan de un antecesor común (declaraciones independientes), entonces las declaraciones están en conflicto y el modelo está mal formado. UML no proporciona una regla de resolución del conflicto, para esta situación, porque la experiencia ha demostrado que el diseñador debe resolverla explícitamente. Algunos lenguajes, como Eiffel, permiten que los conflictos sean resueltos explícitamente por el programador, que es mucho más seguro que las reglas implícitas de resolución del conflicto, que conducen con frecuencia a sorpresas para el desarrollador.

## **Clasificación simple y múltiple**

En la formulación más simple, un objeto tiene una clase directa. Muchos lenguajes orientados a objetos tienen esa restricción. No hay necesidad lógica de que un objeto tenga una sola clase; nosotros observamos casi siempre los objetos del mundo real desde muchos ángulos simultáneamente. En la formulación más general de UML, un objeto puede tener una o más clases directas. El objeto se comporta como si perteneciera a una clase implícita que fuera hija de cada una de las clases directas; en este sentido, con herencia múltiple no hay necesidad realmente de declarar la nueva clase.

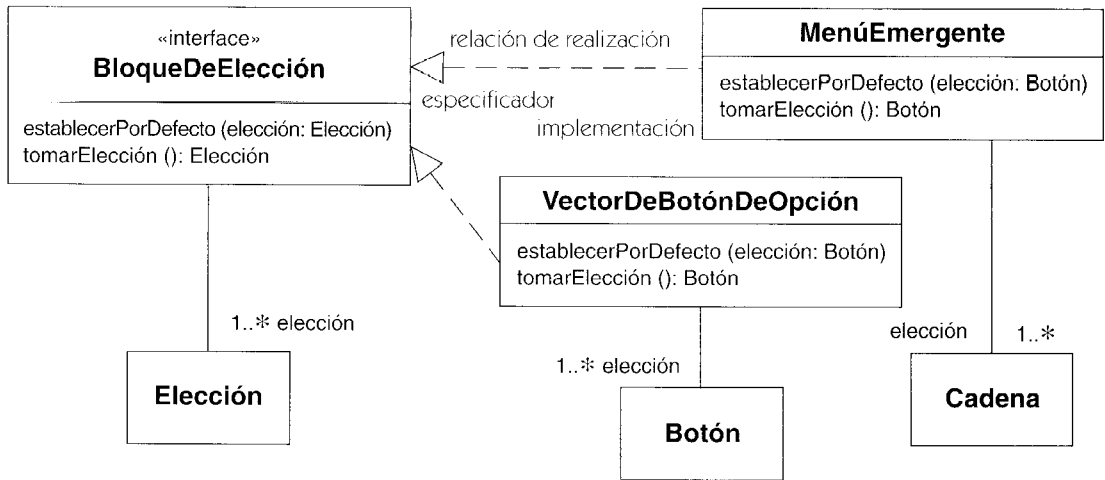
## **Clasificación estática y dinámica**

En la formulación más simple, un objeto no puede cambiar su clase después de haber sido creado. Una vez más, no hay necesidad lógica para esta restricción. Tiene como objeto sobre todo, hacer la implementación de lenguajes de programación orientados a objetos más sencilla. En la formulación más general, un objeto puede cambiar su clase directa dinámicamente. Haciendo eso, puede perder o ganar atributos o asociaciones. Si los pierde, la información en ellas se pierde y no se puede recuperar más adelante, incluso si cambia de nuevo a la clase original. Si gana atributos o asociaciones, entonces deben ser inicializados en el momento del cambio, de manera similar a la inicialización de un nuevo objeto.

Cuando la clasificación múltiple se combina con la clasificación dinámica, un objeto puede ganar y perder clases durante toda su vida. Las clases dinámicas a veces se llaman roles o tipos. Un patrón de modelado común es requerir que cada objeto tenga una sola clase inherente estática (una que no pueda cambiar durante toda la vida de objeto) y cero o más clases rol que se puedan agregar o quitar durante el curso de vida del objeto. La clase inherente describe sus propiedades fundamentales, y las clases de rol describen propiedades transitorias. Aunque muchos lenguajes de programación no apoyan la clasificación dinámica múltiple en la jerarquía de declaración de clases es, sin embargo, un concepto de modelado valioso, que se puede representar con asociaciones.

## **Realización**

La relación de realización conecta un elemento del modelo, tal como una clase, con otro elemento, tal como una interfaz, que especifica su comportamiento pero no su estructura o implementación. El cliente debe tener (por herencia o por declaración directa), al menos todas las



**Figura 4.9** Relación de realización

operaciones que tiene el proveedor. Aunque la realización pretende ser utilizada con elementos de especificación, tales como interfaces, también puede ser utilizada con un elemento concreto de la implementación para indicar que su especificación (pero no su implementación) debe estar presente. Por ejemplo, esto se puede utilizar para mostrar la relación de una versión optimizada de una clase a una versión más simple pero ineficaz.

La generalización y la realización relacionan una descripción más general, con versiones más detalladas de la misma. La generalización relaciona dos elementos en el mismo nivel semántico (en el mismo nivel de abstracción, por ejemplo), generalmente dentro del mismo modelo; la realización relaciona dos elementos en diversos niveles semánticos (una clase del análisis y una clase de diseño, por ejemplo, o una interfaz y una clase), a menudo en diversos modelos. Puede haber dos o más jerarquías completas de clases en distintas etapas de desarrollo, cuyos elementos están relacionados por realización. Las dos jerarquías no necesitan tener la misma forma, porque las clases que la realizan pueden tener dependencias de implementación que no sean relevantes en las clases que actúan de especificación.

La realización se indica con una flecha de línea discontinua con una punta de flecha hueca cerrada (Figura 4.9). Es similar al símbolo de la generalización con una línea discontinua, para indicar que es similar a un tipo de herencia.

Hay una notación reducida especial para mostrar interfaces (sin su contenido) y las clases o los componentes que los realizan. Se muestra la interfaz como un círculo pequeño, unido al rectángulo del clasificador por una línea sólida (Figura 4.10).



**Figura 4.10** Interfaze iconos de realización

## Dependencias

Una dependencia indica una relación semántica entre dos o más elementos del modelo. Relaciona los elementos del modelo entre ellos, y no requiere un conjunto de instancias para su significado. Indica una situación, en la cual un cambio al elemento proveedor puede requerir un cambio o indicar un cambio en el significado del elemento cliente en la dependencia.

Las relaciones de asociación y generalización son dependencias según esta definición, pero tienen semántica específica con consecuencias importantes. Por lo tanto, tienen sus propios nombres y semántica detallada. Utilizamos normalmente la palabra dependencia para el resto de relaciones, que no encajan en categorías más definidas. La Tabla 4.3 enumera las clases de dependencia encontradas en el modelo base de UML.

Una traza es una conexión conceptual entre elementos de diversos modelos, a menudo modelados en diferentes etapas del desarrollo. Carece de semántica detallada. Se utiliza típicamente para seguir la pista de requisitos del sistema, a través de los modelos, y para no perder de vista los cambios realizados en un modelo, que puedan afectar otros modelos.

Un refinamiento es una relación entre dos versiones de un concepto en diversas etapas del desarrollo o en diversos niveles de abstracción. Los dos conceptos no pretenden coexistir en el modelo final detallado. Uno de ellos es generalmente una versión menos acabada del otro. En principio, existe una correspondencia del concepto menos acabado al concepto acabado. Esto no significa que la traducción sea automática. Generalmente, el concepto más detallado contiene las decisiones tomadas por el diseñador, las decisiones del diseño que podrían haberse tomado de muchas maneras. En principio, los cambios a un modelo se podrían validar contra el otro, señalando las desviaciones. En la práctica, las herramientas no pueden hacer todo esto hoy en día, aunque se pueden implementar algunas correspondencias más sencillas. Por lo tanto un refinamiento es sobre todo un recordatorio al modelador de la presencia de múltiples modelos relacionados de una manera previsible.

Una dependencia de derivación indica que un elemento se puede computar a partir de otro elemento (se puede haber incluido explícitamente el elemento derivado en el sistema para evitar una recomputación costosa). La derivación, la realización, el refinamiento, y la traza son dependencias de abstracción; relacionan dos versiones de la misma cosa subyacente.

Una dependencia de uso es una declaración de que el comportamiento o la implementación de un elemento, afectan al comportamiento o a la implementación de otro elemento. Con frecuencia, esto se debe a temas de implementación, tales como requisitos del compilador que necesita la definición de una clase para compilar otra clase. La mayoría de las dependencias de uso se pueden derivar del código y no necesitan ser declaradas explícitamente, a menos que sean parte de un estilo de diseño descendente, que limita la organización del sistema (por ejemplo, usando componentes y bibliotecas predefinidos). El tipo específico de dependencia de uso puede ser especificado, pero esto se omite a menudo porque el propósito de la relación es destacar la dependencia. Los detalles exactos se pueden obtener a menudo del código de implementación. Los estereotipos de uso incluyen la llamada y la instanciación. La dependencia de llamada indica que un método, en una clase, llama a una operación en otra clase; la instanciación indica que un método, en una clase, crea una instancia de otra clase.

Algunas variedades de dependencia de uso conceden permiso para que los elementos tengan acceso a otros elementos. La dependencia de acceso permite que un paquete vea el contenido de otro paquete. La dependencia de importación va más lejos, y agrega los nombres de los contenidos del paquete destino al espacio de nombres del paquete desde el que se importa. La de-

Tabla 4.3 Tipos de Dependencias

<i>Dependencia</i>	<i>Función</i>	<i>Palabra clave</i> <sup>1</sup>
acceso	Permiso para un paquete para acceder a los contenidos de otro paquete	<b>access/accede</b>
ligadura	Asignación de valores a los parámetros de una plantilla, para generar un nuevo elemento de modelo	<b>bind*/ligado</b>
llamada	Establece que un método de una clase llama a una operación de otra	<b>call/llama</b>
derivación	Establece que una instancia puede ser calculada a partir de otra	<b>derive/deriva</b>
amigo	Permiso para un elemento para acceder a otro elemento referente a la visibilidad	<b>friend/amiga</b>
importación	Permiso para un paquete para acceder a los contenidos de otro paquete y añadir alias de sus nombres en el espacio de nombres del importador	<b>import/importa</b>
instanciación	Establece que un método de una clase crea instancias de otra clase	<b>instantiate/usa instancias</b>
parámetro	Relación entre una operación y sus parámetros	<b>parameter/parámetro</b>
realización	Correspondencia entre una especificación y la implementación de la misma	<b>realize/realiza</b>
refinamiento	Establece que existe una correspondencia entre elementos a dos niveles semánticos diferentes	<b>refine*/refina</b>
envío	Relación entre el emisor de una señal y el receptor de la misma	<b>send/envía</b>
traza	Establece que existe alguna conexión entre elementos en diferentes modelos, pero menos preciso que una correspondencia	<b>trace*/traza</b>
uso	Establece que un elemento requiere la presencia de otro elemento, para su correcto funcionamiento (incluye llamada, instanciación, parámetro, envío, pero está abierto a otros tipos)	<b>use*/usa</b>

pendencia de amistad es una dependencia de acceso, que permite que el cliente vea incluso el contenido privado del proveedor.

Una ligadura es la asignación de valores a los parámetros de una plantilla. Es una relación altamente estructurada, con la semántica exacta obtenida sustituyendo los argumentos por los parámetros, en una copia de la plantilla.

<sup>1</sup> Estas palabras clave son estereotipos (véase capítulo 11) y por tanto pueden estar en cualquier idioma. La forma inglesa puede ser la empleada por algunas herramientas (las marcadas con \*) y la que aparecerá en modelos UML en lengua inglesa. Pero las formas traducidas pueden utilizarse para que un modelo no tenga que hacer uso de términos ingleses, que podrían oscurecer la comprensión (*N. del Revisor*.)

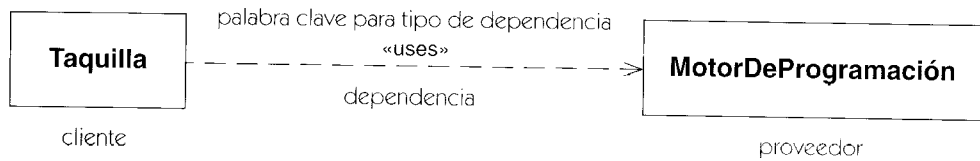


Figura 4.11 Dependencias

Las dependencias de uso y de ligadura implican una semántica fuerte entre elementos al mismo nivel semántico. Deben conectar elementos al mismo nivel del modelo (ambos en análisis o en diseño, y al mismo nivel de abstracción). Las dependencias de traza y de refinamiento son más vagas, y pueden conectar elementos de distintos modelos o niveles de abstracción.

La instancia de la relación (una metarrelación, no estrictamente una dependencia) indica que un elemento (tal como un objeto) es una instancia de otro elemento (tal como una clase).

Una dependencia se dibuja como una flecha de línea discontinua, desde el cliente al proveedor, con una palabra clave de estereotipo para distinguirla, según lo mostrado en la Figura 4.11.

## Restricción

UML provee un sistema de conceptos y de relaciones para modelar sistemas como grafos de elementos de modelado. Sin embargo, algunas cosas se expresan mejor lingüísticamente, es decir, usando la potencia de un lenguaje textual. Una restricción es una expresión booleana representada como una cadena interpretable en un determinado lenguaje. Para expresar restricciones se puede utilizar el lenguaje natural, notación de teoría de conjuntos, lenguajes de restricciones o varios lenguajes de programación.

UML incluye la definición de un lenguaje de restricción, llamado OCL, adecuado para expresar restricciones de UML. Esperamos que reciba un amplio apoyo y soporte. Véase la referencia de OCL y el libro [Warmer-99] para más información sobre el mismo.

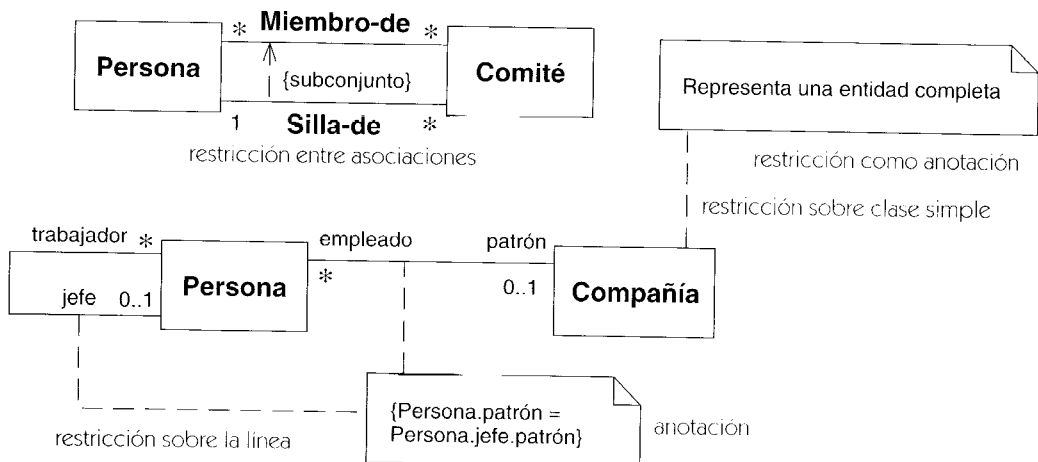


Figura 4.12 Restricciones



Se pueden utilizar las restricciones para indicar varias relaciones no locales, tales como restricciones en las asociaciones. En detalle, las restricciones se pueden utilizar para indicar propiedades de existencia (*existe un X tal que la condición C es verdadera*) y propiedades universales (*para toda y en Y, la condición D debe ser verdadera*).

Algunas restricciones estándar han sido predefinidas como elementos estándar de UML, por ejemplo las asociaciones en una relación de “o exclusivo” y varias restricciones en las relaciones entre subclases en la generalización.

Véase el capítulo 14, elementos estándar, para más información.

Una restricción se muestra como expresión del texto entre llaves. Puede ser escrito en un lenguaje formal o natural. La cadena de texto se puede colocar en una nota o unir a una flecha de dependencia. La Figura 4.12 muestra algunas restricciones.

## Instancias

Una instancia es una entidad de ejecución con identidad, es decir, algo que puede ser distinguido de otras entidades de ejecución. Tiene un valor en todo momento. A lo largo del tiempo, el valor puede cambiar en respuesta a operaciones sobre él.

Un propósito de un modelo es describir los posibles estados de un sistema y de su comportamiento. Un modelo es una declaración de potencial, de las posibles colecciones de objetos que pueden existir, y de la posible historia de comportamiento que los objetos pueden experimentar. La vista estática define y restringe las posibles configuraciones de los valores que un sistema en ejecución puede asumir.

La vista dinámica define las maneras por las cuales un sistema en ejecución puede pasar de una configuración a otra. La vista estática, junto a las distintas vistas dinámicas basadas en ella, definen la estructura y el comportamiento de un sistema.

Una configuración estática particular de un sistema en un instante se llama una instantánea. Una instantánea abarca objetos y otras instancias, valores, y enlaces. Un objeto es una instancia de una clase. Cada objeto es una instancia directa de la clase que lo describe totalmente y una instancia indirecta de los antecesores de esa clase. (Si se permite la clasificación múltiple, entonces un objeto puede ser la instancia directa de más de una clase.) Del mismo modo, cada enlace es una instancia de una asociación, y cada valor es una instancia de un tipo de datos.

Un objeto tiene un valor para cada atributo de su clase. El valor de cada atributo debe ser consistente con el tipo de dato del atributo. Si el atributo tiene multiplicidad opcional o múltiple, entonces el atributo puede contener cero o múltiples valores. Un enlace abarca una tupla de valores, cada uno de los cuales es una referencia a un objeto de una clase dada (o uno de sus descendientes). Los objetos y los enlaces deben obedecer cualquier restricción en las clases o las asociaciones de las cuales son instancias (incluyendo tanto restricciones explícitas como restricciones incorporadas, tales como la multiplicidad).

El estado de un sistema es una *instancia válida del sistema* si cada instancia en él es instancia de algún elemento de un mismo modelo de sistema, este modelo está bien formado, y todas las restricciones impuestas por el modelo son satisfechas por los casos.

La vista estática define el conjunto de objetos, valores, y enlaces que pueden existir en una sola instantánea. En principio, cualquier combinación de objetos y de enlaces que sea consistente

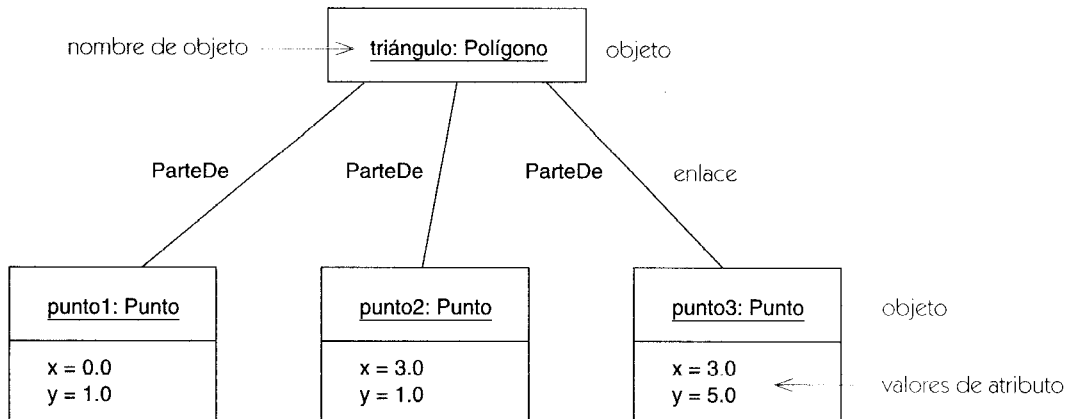
con una vista estática es una configuración posible del modelo. Esto no significa que toda instantánea posible pueda ocurrir u ocurra. Algunas instantáneas pueden ser legales estáticamente, pero pueden no ser dinámicamente accesibles bajo las vistas dinámicas del sistema.

Los elementos y diagramas para comportamiento de UML describen las secuencias válidas de las instantáneas que pueden ocurrir como resultado de efectos del comportamiento externo e interno. Las vistas dinámicas definen cómo se mueve el sistema de una instantánea a otra.

## Diagrama de objetos

Un diagrama de una instantánea es una imagen de un sistema, en un instante en el tiempo. Debido a que contiene imágenes de objetos, se llama diagrama de objetos. Puede ser útil como ejemplo del sistema, por ejemplo, ilustrar las estructuras de datos complicadas o mostrar el comportamiento con una secuencia de instantáneas en un cierto plazo (Figura 4.13). Recuerde que todas las instantáneas son ejemplos de sistemas, no definiciones de sistemas. La definición de la estructura y del comportamiento del sistema se encuentra en las vistas de definición, y construir las vistas de definición es el objetivo del modelado y el diseño.

La vista estática describe los casos posibles que pueden ocurrir. Los casos reales generalmente no aparecen directamente en modelos, excepto como ejemplos.



**Figura 4.13** Diagrama de objetos