El Lenguaje Unificado de Modelado. Manual de Referencia

JAMES RUMBAUGH
IVAR JACOBSON
GRADY BOOCH

La referencia definitiva de UML escrita por sus creadores

Incluye CD-ROM con la versión 1.3 de UML

Addison Wesley

EL LENGUAJE UNIFICADO DE MODELADO MANUAL DE REFERENCIA

EL LENGUAJE UNIFICADO DE MODELADO MANUAL DE REFERENCIA

James RUMBAUGH
Ivar JACOBSON
Grady BOOCH
RATIONAL SOFTWARE CORPORATION

Traducción:

Salvador Sánchez
Universidad Pontificia de Salamanca en Madrid
Oscar San Juan
Universidad Pontificia de Salamanca en Madrid
Rafael García-Bermejo
Universidad de Salamanca

Revisión Técnica:

Luis Joyanes
Director del Departamento de Lenguajes y Sistemas Informáticos
Universidad Pontificia de Salamanca en Madrid
Carmelo R. Fernández
Director de Sistemas de Información
Cámara de Comercio, Industria y Navegación de Gipuzkoa
Universidad del País Vasco / Euskal Herriko Unibersitatea

Colaboradores en la revisión:

Juan Manuel Cueva
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Oviedo
Miren Bermejo
Departamento de Lenguajes y Sistemas Informáticos
Universidad del País Vasco / Euskal Herriko Unibersitatea

ADDISON WESLEY

Madrid • México • Santafé de Bogotá • Buenos Aires • Caracas • Lima • Montevideo • San Juan
San José • Santiago • São Paulo • Reading, Massachusetts • Harlow, England

Datos de catalogación bibliográfica

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

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

Este libro ha sido impreso con papel y tintas ecológicos

Para Madeline, Nick y Alex

—Jim

Contenido

Prefacio

Objetivos

Este libro está destinado a ser una referencia útil y completa sobre el Lenguaje Unificado de Modelado para el desarrollador, arquitecto, gestor de proyecto, ingeniero de sistemas, programador, analista, contratista, cliente, y cualquier otro que necesite especificar, diseñar, construir o entender sistemas de software complejos. Proporciona una referencia completa sobre los conceptos y construcciones de UML, incluyendo su semántica, sintaxis, notación y propósito. Está organizado para ser una útil, pero minuciosa, referencia para el desarrollador profesional. También intenta ofrecer detalles adicionales sobre temas que no están claros en la documentación estándar y ofrece una razón para muchas decisiones que se tomaron en UML.

Este libro no pretende ser una guía sobre los documentos estándar de UML o sobre la estructura interna del metamodelo contenido en ellos. Los detalles del metamodelo son del interés de los metodólogos y los constructores de herramientas UML, pero la mayoría de los desarrolladores no necesitan de los misteriosos detalles de los documentos del Object Management Group (OMG). Este libro proporciona todos los detalles que necesitan la mayoría de los desarrolladores; en muchos casos, hace explícita información que, de lo contrario, habría que buscar entre líneas en los documentos originales. Para aquellos que quieran consultarlos, se han incluido los documentos originales, en inglés, en el CDROM adjunto.

Este libro está pensado como una referencia para aquellos que ya tienen algún conocimiento de la tecnología de orientación a objetos. Para los principiantes, en la bibliografía hay un listado de libros escritos por nosotros y por otros autores; aunque parte de la notación ha cambiado, libros como [Rumbaugh-91], [Booch-94], [Jacobson-92], y [Meyer-98 (edición en español)] proporcionan una introducción a los conceptos de orientación a objetos que todavía es válida y que por tanto no es necesario duplicar. Para una Introducción a UML que muestra como modelar varios problemas comunes, véase *Guía de Usuario del Lenguaje Unificado de Modelado* [Booch-99]. Aquellos que ya conocen un método orientado a objetos, como OMT, Booch, Objectory, Coad-Yourdon, o Fusion, deberían ser capaces de Leer el *Manual de Referencia* y usarlo para entender la notación y la semántica de UML; para aprender UML rápidamente sin embargo, podrían encontrar útil leer la *Guía del Usuario*.

UML no requiere un proceso de desarrollo en particular, y este libro no describe ninguno. Aunque UML puede usarse con varios procesos de desarrollo, fue diseñado para usarse con un proceso iterativo, incremental, guiado por casos de uso y centrado en la arquitectura, el tipo de proceso que consideramos más apropiado para el desarrollo de sistema complejos modernos. *El Proceso Unificado de Desarrollo de Software* [Jacobson-99] describe el tipo de proceso que creemos que complementa UML y que ayuda mejor al desarrollo de software.

Esquema general del libro

EL Manual de Referencia de UML está organizado en tres partes: una descripción de la historia de UML y del modelado, una visión de conjunto de los conceptos de UML, y una enciclopedia alfabética de los términos y conceptos de UML.

La primera parte es un examen del conjunto de UML —su historia, propósitos y usos— para ayudar a entender el origen de UML y las necesidades que intenta cubrir.

La segunda parte es una revisión de las vistas de UML para poner todos los conceptos en perspectiva. La visión de conjunto proporciona una breve descripción de las vistas disponibles en UML y muestra cómo interactúan las diferentes construcciones. Esta parte empieza con un ejemplo que abarca varias vistas de UML y también contiene un capítulo por cada clase de vista. Esta visión no pretende ser un curso completo ni una extensa descripción de conceptos. Sirve principalmente para resumir y relacionar los diferentes conceptos de UML y proporcionar puntos de partida para lecturas de la enciclopedia en detalle.

La tercera parte contiene el material de referencia organizado para acceder fácilmente a cada tema. La mayor parte del libro es una enciclopedia alfabética de todos los conceptos y construcciones de UML. Cada término de importancia en UML tiene su propia referencia en la enciclopedia. La enciclopedia debe ser completa, por lo que todos los conceptos descritos en la parte dos de este libro se repiten de forma más detallada. Algunas veces se ha incluido la misma o similar información en varios artículos de la enciclopedia, para que el lector pueda encontrarlas oportunamente.

La parte de referencia también contiene una lista alfabética de los elementos estándar de UML. Un elemento estándar es una característica predefinida que utiliza los mecanismos de extensión de UML. Los elementos estándar son extensiones pensadas para ser muy útiles.

Los apéndices muestran el metamodelo de UML, un resumen de la notación y algunos conjuntos estándar de extensiones para dominios particulares. Hay una breve bibliografía de los libros de orientación a objetos más importantes, pero no intenta ser una completa guía sobre las fuentes de ideas para UML y otros métodos. Muchos de los libros de la bibliografía contienen excelentes listas de referencias a libros y artículos para aquellos interesados en seguir el desarrollo de las ideas.

Convenciones en el formato de los artículos de la enciclopedia

La enciclopedia está organizada como una lista alfabética de entradas, cada una de las cuales describe un concepto en mayor o menor detalle. Los artículos representan una lista "plana" de los conceptos de UML a diferentes niveles conceptuales. Un concepto de alto nivel típicamente contiene un resumen de sus conceptos subordinados, cada uno de los cuales se describe completamente en un artículo separado. Los artículos tienen muchas referencias cruzadas. Esta organización "plana" de la enciclopedia permite presentar la descripción de cada concepto a un nivel bastante uniforme de detalle, sin los constantes desplazamientos de nivel que serían necesarios para una presentación secuencial de las descripciones anidadas. El formato de hiper-

texto también debería hacerlo útil como referencia. No debería ser necesario usar el índice; en su lugar vamos directamente al artículo principal de la enciclopedia para cualquier término de interés y seguimos las referencias cruzadas. Este formato no es necesariamente el ideal para aprender el lenguaje; se aconseja a los principiantes leer la descripción de UML que se encuentra en la Parte 2 o leer los libros introductorios sobre UML, tales como *El lenguaje Unificado de Modelado [Booch-99].*

Los artículos de la enciclopedia tienen las siguientes divisiones (aunque no todas las divisiones aparecen en todos los artículos):

Breve definición

El nombre del concepto aparece en negrita, colocado a la izquierda del texto del artículo. A continuación una breve descripción en letra normal. Esta definición tiene la intención de capturar la idea principal del concepto, pero puede simplificar el concepto para su presentación concisa. Debemos remitirnos al artículo principal para significados precisos.

Semántica

Esta sección contiene una descripción detallada del significado del concepto, incluyendo las restricciones en su uso y las consecuencias de su ejecución. Esta sección no abarca la notación, aunque los ejemplos utilizan la notación apropiada: primero se da la semántica general. Para conceptos con propiedades estructurales subordinadas, a la semántica general sigue una lista de las propiedades, a menudo bajo el subtítulo de estructura. En la mayoría de los casos, las propiedades se presentan como una tabla ordenada alfabéticamente por nombre, con la descripción de cada propiedad a la derecha. Si una propiedad tiene una breve lista de opciones, aparecerán como una sublista sangrada. En casos más complicados, la propiedad tiene su propio artículo para evitar el excesivo anidamiento. Cuando las propiedades requieren más explicación que la permitida por una tabla, son descritas en texto normal con cabeceras en negrita y cursiva. En ciertos casos el concepto principal se describe de la mejor forma bajo varias subdivisiones lógicas en lugar de hacerlo en una lista. En tales casos, las secciones adicionales están a continuación o reemplazan a la estructura de la subsección. Aunque se han usado varios mecanismos de organización, su estructura debería ser obvia para el lector.

Notación

Esta sección contiene una descripción detallada del concepto. Normalmente la sección de notación tiene un formato semejante a la sección de semántica, cuya referencia, y a menudo tiene las mismas divisiones. La sección de notación a menudo incluye uno o varios diagramas para ilustrar el concepto.

Ejemplo

Esta subsección contiene ejemplos de notación o ilustraciones del uso del concepto. También es frecuente que los ejemplos traten situaciones complicadas o potencialmente confusas.

Discusión

Esta sección describe asuntos sutiles, clarifica los puntos difíciles y habitualmente confusos, y contiene otros detalles que, de lo contrario, serían apartados de una sección de semántica más descriptiva. Hay pocos artículos que tengan una sección de discusión.

Esta sección también explica ciertas decisiones de diseño hechas durante el desarrollo de UML, en particular aquellas que pueden parecer menos intuitivas o que han causado gran controversia. Sólo una pequeña parte de los artículos tienen esta sección. Las diferencias simples por gustos, generalmente no se contemplan.

Elementos estándar

Esta sección lista las restricciones, etiquetas, estereotipos, y otras convenciones estándar predefinidas para el concepto del artículo. Esta sección es muy poco común.

Convenciones de sintaxis

Expresiones sintácticas. Las expresiones sintácticas se dan en un formato BNF modificado escrito en un tipo de letra distinto.

Los subíndices y barras superiores se usan como operadores sintácticos de la siguiente manera:

expresión$_{opc}$	La expresión es opcional.
expresión$_{lista,}$	Puede aparecer una lista de las expresiones separada por comas. Si hay cero o una repetición, no hay separador, Cada repetición puede tener una sustitución separada. Si aparece un signo de puntuación diferente a una coma en el subíndice, entonces es un separador.
$\overline{\text{= expresión}_{opc}}$	Una barra superior enlaza juntos uno o más términos que son considerados una unidad para la optatividad o la repetición de ocurrencias. En este ejemplo, el igual y la expresión forman una unidad que puede ser omitida o incluida. La barra superior es innecesaria si sólo hay un término.

Se evita que existan dos niveles de anidamiento.

Cadenas de literales. El texto ejecutable, las palabras clave del lenguaje, los nombres de elementos del modelo y las cadenas de ejemplo de modelos se muestran en un tipo de letra distinto.

Diagramas. En los diagramas, el texto y las flechas en azul son anotaciones, es decir, explicaciones de las notaciones de diagramas que no aparecen en un diagrama real. Cualquier texto y símbolos en tinta negra son notaciones de diagramas reales.

CD

Este libro se acompaña de un CD que contiene el texto completo en inglés del libro en formato "Adobe Acrobat Reader" (PDF). Usando "Adobe Acrobat Reader", el lector puede buscar fácilmente en el libro una palabra o frase. La versión en CD también contiene una tabla de

contenidos cuyos términos son enlaces al texto donde aparecen definidos índice, miniaturas e hiperenlaces en el cuerpo de los artículos. Basta con pulsar en uno de los enlaces para saltar al artículo de la enciclopedia para esa palabra o frase.

El CD también contiene el texto completo de las especificaciones de OMG de UML, incluidas con el permiso del Object Management Group.

Tenemos la impresión de que este CD será una referencia muy útil para usuarios avanzados.

Información adicional

Pueden encontrarse archivos fuente adicionales e información actualizada sobre trabajos posteriores en UML y temas relacionados en los servidores Web www.rational.com y www.omg.org.

Reconocimientos

Queremos dar las gracias a todos aquellos que han hecho posible UML. Primero, debemos agradecer a Rational Software Corporation, especialmente a Mike Devlin y Paul Levy, quien tuvo la visión de reunirnos, empezar el trabajo de unificación, y continuar durante los cuatro años que hicieron falta para llevar el trabajo a un final con éxito. También queremos dar las gracias al Object Management Group por brindar el marco que reunió muchos puntos de vista y los fusionó en un amplio consenso lo que superó todas las otras contribuciones.

Queremos dar las gracias en particular a Cris Kobryn, quien lideró al equipo técnico que preparó el estándar de UML y que se las arregló para alcanzar un consenso entre un grupo de personas extremadamente exaltadas (y nosotros tres no fuimos el menor de sus problemas). Su habilidad diplomática y equilibrio técnico evitó que el esfuerzo de UML fracasara en medio de las muchas opiniones diferentes. Cris también revisó el libro y nos ofreció incontables y útiles sugerencias.

Queremos agradecer a Gunnar Övergaard por la revisión concienzuda del libro, así como su perseverancia en completar muchas secciones de los documentos de UML, lo cual no fue divertido de escribir pero sí necesario para su corrección formal.

Queremos dar las gracias a Karin Palmkvist por una revisión extremadamente detallista del libro que descubrió muchos errores en el contenido técnico así como varios fallos de gramática, expresión y presentación.

También nos gustaría agradecer a Mike Blaha, Conrad Bock, Perry Cole, Bruce Douglass, Martin Fowler, Eran Gery, Pete McBreen, Guus Ramackers, Tom Schultz, Ed Seidewitz, y Bran Selic por sus provechosas revisiones.

Sobre todo, queremos dar las gracias a los, incluso, cientos de personas que contribuyeron a la comunidad de ideas de las cuales derivó UML -ideas en tecnología de orientación a objetos, metodología de software, lenguajes de programación, interfaces de usuario, programación visual, y muchas otras áreas de la informática. Es imposible enumerarlas todas o, en realidad, incluso hacer un seguimiento de las cadenas de mayor influencia sin hacer un gran esfuerzo erudito, y este es un libro de ingeniería, no una reseña histórica. Muchos son muy conocidos pero muchas buenas ideas vienen de algunos que no tienen la buena suerte de ser ampliamente reconocidos.

A nivel más personal, quiero dar las gracias al Profesor Jack Dennis, quien inspiró mi trabajo en el modelado y el trabajo de muchos otros estudiantes, hace más de veinticinco años. Las ideas de su Grupo de Estructuras de Computación en el MIT han dado muchos frutos, y no son la menor de las fuentes de UML. También debo dar las gracias a Mary Loomis y Ashwin Shah, con quien desarrollé las ideas originales de OMT, y a mis primeros colegas en el Centro de I+D en GE, Mike Blaha, Bill Premerlani, Fred Eddy, y Bill Lorensen, con quien escribí el libro de OMT.

Finalmente sin la paciencia de mi esposa, Madeline, y mis hijos, Nick y Alex, no habría habido UML ni libro sobre él.

James Rumbaugh
Cupertino, California
Noviembre 1998

Prólogo a la edición en español

UML: Una historia del futuro del desarrollo de software

UML es, probablemente, una de las innovaciones conceptuales en el mundo tecnológico del desarrollo de software que más expectativas y entusiasmos haya generado en muchos años, comparable a la aparición e implantación de los lenguajes COBOL, BASIC, Pascal, C++, y más recientemente Java o XML. Además, todas las expectativas se han cumplido y han generado a su vez nuevas expectativas. UML es ya un *estándar de la industria*, pero no sólo de la industria del software sino, en general, de cualquier industria que requiera la construcción de modelos como condición previa para el diseño y posterior construcción de prototipos.

Desde la mítica versión 0.8 del entonces denominado *Unified Method* versión 0.8, fruto de la unión de los creadores de las metodologías Booch'94 y OMT (Grady Booch y James Rumbaugh) y que presentaron a la comunidad informática en la primavera de 1995. Posteriormente se unió al equipo Ivar Jacobson, creador a su vez del método OOSE y, sobre todo, del ingenioso concepto *use case* (casos de uso). La unión del equipo de "3 amigos", como se les ha conocido, (incluso se les ha comparado con *los 3 tenores* que en el campo de la música formaron *Luciano Pavarotti, Plácido Domingo y José Carreras*); 3 amigos con inmenso prestigio en el mundo de la ingeniería del software que se propusieron construir un nuevo lenguaje de modelado, UML, cuya primera versión (1.1) se presentó para su estandarización al OMG (Object Management Group) en 1997 y que fue aceptada. Las últimas versiones 1.2 y 1.3 se han presentado posteriormente y nos encontramos en período de evaluación y pruebas de la versión 1.4. Estas versiones están disponibles en el sitio Web de Rational (www.rational.com).

Otra gran aportación, en este caso no sólo conceptual sino práctica en forma de herramientas, fue la creación de una herramienta CASE (ingeniería de software asistida por computador) denominada Rational CASE, cuya versión Rational'98 está muy extendida en la industria y que sigue todas las especificaciones de UML. Recientemente se ha presentado Rational 2000, que ha mejorado sensiblemente respecto de Rational'98 y promete ser una de las herramientas de referencia en el mundo de la ingeniería y, en particular, en la ingeniería de software.

¿Por qué UML es tan influyente en la industria del software?

UML ha nacido como un lenguaje, pero es mucho más que un lenguaje de programación. Aunque en su génesis se parece a C++ o a Java, en realidad se ha diseñado y construido un lenguaje que ha nacido con una madurez muy acentuada si se le compara, incluso, con los últimos desarrollos de HTML, Java y XML, los lenguajes por excelencia del mundo Internet.

UML se ha diseñado realizando combinaciones de una gran cantidad de estándares, si bien se rige a través de tres metodologías procedentes de la colaboración de los tres creadores de UML ya citados, J. Rumbaugh, G. Booch e I. Jacobson, así como del análisis y estudio de alrededor de 20 métodos estándares que a su vez se han integrado en otro estándar, en este caso, UML; esta fue una gran iniciativa de los tres creadores que pusieron las especificaciones de UML a la consideración de la comunidad informática mundial, antes de su publicación. El diseño de UML ha sido completo desde el principio, al contrario que HTML que ha cambiado gradualmente, de forma que XML ha tratado de resolver los problemas de HTML y Java, que sigue todavía en el proceso de estandarización con la nueva versión 2.0. Al contrario que HTML/XML que son lenguajes de marcación (*markup*), UML es un lenguaje para modelar, que es el procedimiento que emplean los ingenieros para el diseño de software antes de pasar a su construcción, al igual que sucede con cualquier producto manufacturado o fabricado en serie.

UML ayuda al usuario a entender la realidad de la tecnología y la posibilidad de que reflexione antes de invertir y gastar grandes cantidades en proyectos que no estén seguros en su desarrollo, reduciendo el coste y el tiempo empleado en la construcción de las piezas que constituirán el modelo.

Sin embargo, desde el punto de vista puramente tecnológico, UML tiene una gran cantidad de propiedades que han sido las que, realmente, han contribuido a hacer de UML el estándar de facto de la industria que es en realidad. Algunas de las propiedades de UML como lenguaje de modelado estándar son:

- *Concurrencia, es un lenguaje distribuido y adecuado a las necesidades de conectividad actuales y futuras.*
- *Ampliamente utilizado por la industria desde su adopción por OMG.*
- *Reemplaza a decenas de notaciones empleadas con otros lenguajes.*
- *Modela estructuras complejas.*
- *Las estructuras más importantes que soportan tienen su fundamento en las tecnologías orientadas a objetos, tales como objetos, clase, componentes y nodos.*
- *Emplea operaciones abstractas como guía para variaciones futuras, añadiendo variables si es necesario.*
- *Comportamiento del sistema: casos de uso, diagramas de secuencia y de colaboraciones, que sirven para evaluar el estado de las máquinas.*

UML, la *Web* y el *comercio electrónico*

Uno de los grandes anuncios del año 2000 en el mundo de UML, se ha producido el pasado 17 de mayo con el anuncio conjunto de Rational Software (empresa creadora de UML y de Rational CASE) y Commerce One (líder mundial en soluciones globales de comercio electrónico) de colaboración mutua para crear la especificación de la primera versión UML para la industria, que siga las especificaciones XML para el desarrollo del comercio electrónico. Es decir, se han unido dos grandes empresas con el objeto de construir un método estándar que reduzca drásticamente el tiempo de desarrollo e incremente la calidad de las aplicaciones de comercio electrónico basadas en XML.

Se trata de alcanzar la máxima de que el cambio rápido de Internet ha creado una paradoja para el desarrollo del conocido como *e-software* para las organizaciones que requieren la entrega de software de un modo mucho más rápido pero conservando una calidad alta. La versión de UML para especificaciones XML está disponible en el sitio Web de Commerce One (www.commerceone.com/xml/sox/index.html) y en el sitio Web de Rational (www.rational.com/uml/index.jtmpl).

Otra gran ventaja que está ofreciendo UML se refiere al desarrollo de aplicaciones globales para la Web, no sólo para comercio electrónico. UML está siendo utilizado por los gerentes de proyectos, desarrolladores y arquitectos de la Web que aplican técnicas orientadas a objetos para construir aplicaciones Web robustas, escalables y eficientes. UML permite a los desarrolladores modelar sus aplicaciones Web como parte de un sistema completo y la lógica de negocios que se debe reflejar en las aplicaciones.

El futuro ya no es lo que era sin UML

UML, como se ha comentado, ha generado y sigue generando un enorme entusiasmo comparable al nacimiento del desarrollo orientado a objetos a principios de los 90 y posteriormente el desarrollo de componentes en la segunda mitad de la década. Este entusiasmo se ha hecho una gran realidad y UML se ha convertido ya en una de las mejores herramientas para el diseño y desarrollo de software fiable, eficiente y de calidad.

Buena prueba de que este entusiasmo es una realidad creciente, se puede ver en la inmensa cantidad de bibliografía que ha generado y sigue generando UML, así como en los numerosos eventos de todo tipo (congresos, seminarios, simposium, jornadas, etc.) que se celebran a lo largo y ancho del mundo tecnológico y en los que brilla con luz propia UML (*léase* **ECOOP, Tools, Object Expo, OOPSLA, UML Conference, etc.**).

UML está predestinado a convertirse en el lenguaje estándar de la industria para especificar, visualizar, construir y documentar sistemas de software del siglo XXI. Iniciativas como las acordadas con Commerce One, con Microsoft para incorporar UML como herramienta de diseño de Office 2000, con Sun para el desarrollo de Java, etc., hacen que UML sea el nuevo buque insignia de la industria del futuro.

Madrid, mayo de 2000
Dr. Luis Joyanes Aguilar
Facultad de Informática
Universidad Pontificia de Salamanca campus de Madrid

Nota sobre la traducción

UML permite modelar sistemas de información, y su objetivo es lograr modelos que, además de describir con cierto grado de formalismo tales sistemas, puedan ser entendidos por los clientes o usuarios de aquello que se modela. Para ello, es muy importante que el idioma en el que estén las palabras y textos que aparezcan en tales modelos sea el propio de estas personas.

En cualquier comunidad hispanohablante, leer *Pedido* aclara mucho más que ver *Order*, y leer *hereda de* más que leer *inherits*, porque los términos en español evocan directamente al lector una semántica cercana a la que se pretende con su uso en el modelo, y esa evocación es precisamente la razón por la cual fueron elegidos los términos ingleses de ese modelo. Como lenguaje de modelado y descripción, UML permite que *todo el modelo* se cree en español o en cualquier otro lenguaje. Sin embargo, el inglés ha sido el idioma nativo en la creación de UML, por lo que las **palabras clave** incorporadas en el propio UML están en inglés. Para el cliente y el equipo de desarrollo, esto no es un problema, pueden usar los términos traducidos que entenderán mejor; pero cuando se quiere usar una herramienta de ingeniería de software que entiende semántica UML, ésta esperará encontrar los términos ingleses para *entender* lo que quiere decir, por ejemplo, esa línea o figura.

Las palabras que aparecen en este Manual de Referencia de UML pueden clasificarse de cara a su traducción, al igual que las que aparezcan en cualquier documento UML, en varias categorías, del siguiente modo:

- **Palabras clave UML**, es decir, términos con un significado específico asignado por el propio UML, no por un modelo en particular, que **no** son extensiones UML (aplicaciones de los mecanismos de extensión de UML). Ejemplo: *send*, *entry*, etc.
- **Palabras clave que son extensiones UML**. Por ejemplo: *<<trace>>*, *{xor}*.
- **Extensiones de UML**, *que no son palabras clave* UML pero están incorporadas al propio estándar por ser de uso común. Éstas, junto con las palabras clave componen los **elementos estándar** de UML y están listadas en el Capítulo 14.
- **Extensiones de UML** que *no son elementos estándar*, sino extensiones introducidas por los modelos usados como ejemplo.
- **Palabras que forman parte de un modelo UML** en particular, usado como ejemplo.
- **Texto explicativo**, que en el libro es el texto básico de los párrafos; y en un modelo los textos de acompañamiento.

En el Capítulo 13 de este libro, cuyo uso será similar al de un diccionario de acceso rápido, se ha optado por mantener en inglés, en figuras y texto, los **elementos estándar** UML que

corresponden a las tres primeras categorías de la clasificación anterior. Estos términos, en los capítulos 1 al 12 se han traducido, como regla general, puesto que su lectura será probablemente más secuencial y pausada.

Para su más rápida localización, los títulos de los términos del Capítulo 13 se han traducido al castellano, mientras que los del Capítulo 14 se mantienen en inglés, pero haciendo referencia a los términos adecuados del Capítulo 13. De este modo, los **elementos estándar** se pueden localizar bien en inglés o bien en su traducción al castellano.

Cuando se crean modelos UML para su uso con herramientas de ingeniería de software que entiendan UML, puede ser necesario que las tres primeras categorías estén en inglés, según el grado de *comprensión* de UML que tenga la herramienta. Cuando los implicados en un modelo incluyen colegas de habla no hispana, todas las categorías, incluso las tres últimas, deben estar en el idioma convenido. En ausencia de las condiciones anteriores, para modelar sistemas en los que todos los implicados hablen castellano no es necesario utilizar términos ingleses en ninguna de las categorías de palabras de la clasificación anterior.

Madrid, mayo de 2000
Carmelo R. Fernández
Director de Sistemas de Información
Cámara de Comercio, Industria y Navegación de Gipuzkoa
Universidad del País Vasco / Euskal Herriko Unibersitatea

Parte 1: Antecedentes

Esta parte describe los principios generales subyacentes en UML, incluyendo la naturaleza y propósito del modelado y aquellos aspectos de UML que impregnan todas las áreas funcionales.

1 Perspectiva general de UML

Este capítulo es una rápida descripción de UML y para qué es bueno.

Breve resumen de UML

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. UML incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Está pensado para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código así como generadores de informes. La especificación de UML no define un proceso estándar pero está pensado para ser útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos.

UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo. La estructura estática define los tipos de objetos importantes para un sistema y para su implementación, así como las relaciones entre los objetos. El comportamiento dinámico define la historia de los objetos en el tiempo y la comunicación entre objetos para cumplir sus objetivos. El modelar un sistema desde varios puntos de vista, separados pero relacionados, permite entenderlo para diferentes propósitos.

UML también contiene construcciones organizativas para agrupar los modelos en paquetes, lo que permite a los equipos de software dividir grandes sistemas en piezas de trabajo, para entender y controlar las dependencias entre paquetes, y para gestionar las versiones de las unidades del modelo, en un entorno de desarrollo complejo. Contiene construcciones para representar decisiones de implementación y para elementos de tiempo de ejecución en componentes.

UML no es un lenguaje de programación. Las herramientas pueden ofrecer generadores de código de UML para una gran variedad de lenguajes de programación, así como construir modelos por ingeniería inversa a partir de programas existentes. UML no es un lenguaje altamente formal pensado para probar teoremas. Hay varios lenguajes de ese tipo, pero no son fáciles de entender ni de usar para la mayoría de los propósitos. UML es un lenguaje de modelado de pro-

pósito general. Para dominios especializados, tales como la composición de IGU, diseño de circuitos VLSI, o inteligencia artificial basada en reglas, podría ser más apropiada una herramienta especializada con un lenguaje especial. UML es un lenguaje de modelado discreto. No se creó para modelar sistemas continuos como los basados en ingeniería y física. UML quiere ser un lenguaje de modelado universal, de propósito general, para sistemas discretos, tales como los compuestos por software, firmware o lógica digital.

Historia de UML

UML fue desarrollado en un esfuerzo para simplificar y consolidar el gran número de métodos de desarrollo orientado a objetos que habían surgido.

Los métodos de desarrollo orientado a objetos

Los métodos de desarrollo para los lenguajes de programación tradicionales, tales como Cobol y Fortran, emergieron en los años 70 y llegaron a ser ampliamente difundidos en los 80. Principalmente entre ellos estaba el Análisis estructurado y el diseño estructurado [Yourdon-79] y sus variantes tales como Diseño estructurado de tiempo real [Ward-85] y otros. Esos métodos originalmente desarrollados por Constantine, DeMarco, Mellor, Ward, Yourdon, y otros, alcanzaron cierta penetración en el área de los grandes sistemas, especialmente para los proyectos contratados por el gobierno en los campos aerospacial y de defensa, en los cuales los contratistas insistieron en un proceso de desarrollo organizado y en una amplia documentación del diseño e implementación del sistema. Los resultados no fueron siempre tan buenos como se esperaba —muchos sistemas de ingeniería de software asistidos por computador (CASE) fueron poco más que generadores de informes que extraían diseños después de que la implementación estuviera terminada— pero los métodos incluían buenas ideas que fueron usadas eficientemente en algunos casos para la construcción de grandes sistemas. Las aplicaciones comerciales fueron más reacias a adoptar grandes sistemas CASE y métodos de desarrollo. La mayoría de los negocios desarrollaban su software internamente según sus propias necesidades, sin la relación de enfrentamiento entre cliente y contratista que caracteriza los grandes proyectos del gobierno. Los sistemas comerciales se percibían como más simples, tanto si lo eran en verdad como si no, y por tanto había menos necesidad de una revisión por parte de una organización externa.

El primer lenguaje que es generalmente reconocido como orientado a objetos es Simula 67, desarrollado en 1967. Este lenguaje nunca tuvo un significativo seguimiento, aunque influyó notablemente en los desarrolladores de varios de los lenguajes orientados a objetos posteriores. El movimiento de la orientación a objetos se convirtió en activo con la amplia difusión de la disponibilidad del Smalltalk a principio de los 80, seguido por otros lenguajes orientados a objetos como Objetive C, C++, Eiffel, y CLOS. El uso real de los lenguajes orientados a objetos fue limitado al principio, pero la orientación a objetos atrajo mucho la atención. Aproximadamente 5 años después de que Smalltalk llegara a ser conocido, fueron publicados los primeros métodos de desarrollo orientado a objetos por Shlaer/Mellor [Shlaer-88] y Coad/Yourdon [Coad-91], seguidos muy de cerca por libros de Booch [Booch-91], Rumbaugh/Blaha/Premerlani/Eddy/Lorensen [Rumbaugh-91], y Wirfs-Brock/Wilkerson/Wiener [Wirfs-Brock-90] (nótese que los años de los derechos editoriales, a menudo empiezan en julio del año anterior). Esos libros, unidos a los primeros libros de diseño de lenguajes de programación escritos por Goldberg/Robson [Goldberg-83], Cox [Cox-86], y Meyer [Meyer-88], iniciaron el campo de la metodología

orientada a objetos. La primera fase se completó al final de 1990. El libro de Objectory [Jacobson-92] fue publicado ligeramente después, basado en publicaciones anteriores. Este libro tomó un acercamiento un poco diferente con su enfoque sobre los casos de uso y el proceso de desarrollo.

Durante los siguientes cinco años, aparecieron muchos libros de metodologías orientadas a objetos, cada uno con su propio conjunto de conceptos, definiciones, notación, terminología y procesos. Algunos añadieron nuevos conceptos útiles, pero en general hubo gran similitud entre los conceptos propuestos por los diferentes autores. Muchos de los nuevos libros partieron de uno o más de los métodos existentes e hicieron extensiones o cambios menores. Los autores originales tampoco estuvieron inactivos; la mayoría de ellos actualizaron su trabajo original, a menudo incorporando buenas ideas de otros autores. En general, surgió un núcleo de conceptos comunes, junto con una gran variedad de conceptos aceptados por uno o dos autores pero no utilizados ampliamente. Incluso en el núcleo de conceptos comunes, hay pequeñas discrepancias entre los métodos que hacían una comparación detallada algo capciosa, especialmente para el lector ocasional.

Esfuerzo de unificación

Hubo algunos intentos tempranos de unificar los conceptos entre métodos. Un ejemplo destacable fue Fusion por Coleman y sus colegas [Coleman-94], el cual incluyó conceptos de OMT [Rumbaugh-91], Booch [Booch-91], y CRC [Wirfs-Brock-90]. Como no involucró a los autores originales, permaneció como otro nuevo método en lugar de un reemplazo de varios de los métodos existentes. El primer intento exitoso de combinar y reemplazar los métodos existentes llegó cuando Rumbaugh se unió a Booch en Rational Software Corporation en 1994. Ellos empezaron combinando conceptos de los métodos OMT y Booch, obteniendo como resultado una primera propuesta en 1995. En ese momento Jacobson también se unió a Rational y comenzó a trabajar con Booch y Rumbaugh. Su trabajo conjunto fue llamado Lenguaje Unificado de Modelado (UML). El impulso ganado al tener a los autores de tres de los métodos más importantes trabajando juntos para unificar sus enfoques desplazó la balanza en el campo de las metodologías orientadas a objetos, donde había habido muy poco incentivo (o al menos poca voluntad) de los metodólogos de abandonar algunos de sus propios conceptos para alcanzar la armonía.

En 1996, el Object Management Group (OMG) publicó una petición de propuestas para un enfoque estándar sobre el modelado orientado a objetos. Los autores de UML (Booch, Jacobson y Rumbaugh) empezaron a trabajar con metodólogos y desarrolladores de otras compañías, para generar una propuesta atractiva a los miembros de OMG, así como también un lenguaje de modelado, que sería ampliamente aceptado por los fabricantes de herramientas, metodólogos, y desarrolladores, quienes serían los usuarios eventuales. Empezaron también varios esfuerzos competitivos. Finalmente, todas las propuestas se unieron a la propuesta final de UML que fue sometida a consideración del OMG en septiembre de 1997. El producto final es una colaboración entre muchas personas. Nosotros empezamos el esfuerzo de UML y aportamos unos pocas buenas ideas, pero las ideas contenidas en él son el producto de muchas mentes.

Estandarización

El Lenguaje Unificado de Modelado fue adoptado unánimemente por los miembros de OMG como estándar en noviembre de 1997 [UML-98]. OMG asumió la responsabilidad de futuros de-

sarrollos en el estándar de UML. Incluso antes de que se adoptara finalmente, se publicaron varios libros esbozando los puntos clave de UML. Muchos proveedores de herramientas anunciaron apoyo o planes para ofrecer UML, y muchos metodólogos anunciaron que usarían la notación UML en sus trabajos futuros. El surgir de UML parece ser atractivo a la generalidad del público informático porque consolida la experiencia de varios autores con un estado oficial que reducirá las diferencias gratuitas entre herramientas. Creemos que la estandarización apoyará tanto la expansión del uso del modelado orientado a objetos entre los desarrolladores como la aparición de un robusto mercado en herramientas de formación y utilización, ahora que ni los usuarios ni los proveedores tienen que pensar qué metodologías usar y mantener.

Equipo principal

Las siguientes personas formaron parte del equipo principal de desarrollo de la propuesta de UML o trabajaron en el equipo de revisión:

Data Access Corporation: Tom Digre

DHR Technologies: Ed Seidewitz

HP: Martin Griss

IBM: Steve Brodsky, Steve Cook, Jos Warmer

I-Logix: Eran Gery, David Harel

ICON Computing: Desmond D'Souza

IntelliCorp and James Martin &Co.: Conrad Bock, James Odell

MCI Systemhouse: Cris Kobryn, Joaquin Miller

ObjectTime: John Hogg, Bran Selic

Oracle: Guus Ramackers

Platinum Technologies: Dilhar DeSilva

Rational Software: Grady Booch, Ed Eykholt, Ivar Jacobson, Gunnar Övergaard, Karin Palmkvist, James Rumbaugh

SAP: Oliver Wiegert

SOFTEAM: Philippe Desfray

Sterling Software: John Cheesman, Keith Short

Taskon: Trygve Reenskaug

Unisys: Sridhar Iyengar, GK Khalsa

¿Qué significa unificado?

La palabra unificado tiene los siguientes significados relevantes para UML.

A través de los métodos históricos y notaciones. UML combina conceptos comúnmente aceptados por muchos métodos orientados a objetos, seleccionando una definición clara para cada concepto, así como una notación y una terminología. UML puede representar la mayoría de los modelos existentes tan bien o mejor que como lo hacían los métodos originales.

A través del ciclo de vida de desarrollo. UML no tiene saltos ni discontinuidades desde los requisitos a la implantación. Se puede utilizar el mismo conjunto de conceptos y notación en las diferentes etapas del desarrollo, incluso mezcladas en un solo modelo. No es necesario traducir de una etapa a otra. Esta continuidad es crítica para un desarrollo iterativo e incremental.

A través de los dominios de aplicación. UML está pensado para modelar la mayoría de los dominios de aplicación, incluyendo aquellos que implican sistemas grandes, complejos, de tiempo real, distribuidos, con tratamiento intensivo de datos o con cálculo intensivo, entre otras propiedades. Puede haber áreas especializadas en las cuales un lenguaje especial para ese propósito resulte más útil, pero UML pretende ser tan bueno o mejor que cualquier otro lenguaje de modelado de propósito general para la mayoría de las áreas de aplicación.

A través de los lenguajes de implementación y plataformas. UML está pensado para ser usado en sistemas desarrollados en varios lenguajes de implementación y plataformas, incluyendo lenguajes de programación, bases de datos, 4GLs, documentos de organización, firmware, y otros El trabajo de la capa superior debería ser idéntico o similar, mientras que el trabajo de la capa inferior diferirá en algo para cada medio.

A través de procesos de desarrollo. El UML es un lenguaje, no una descripción de un proceso de desarrollo detallado. Se pretende que sea usado como lenguaje de modelado subyacente a la mayoría de los procesos de desarrollo existentes o de nueva creación, de la misma forma que un lenguaje de programación de propósito general puede ser usado en varios estilos de programación. Está especialmente pensado para apoyar un estilo de desarrollo iterativo e incremental, que es el que recomendamos.

A través de los conceptos internos. En la construcción del metamodelo de UML, hicimos un esfuerzo deliberado para descubrir y representar las relaciones subyacentes entre varios conceptos, intentando captar conceptos de modelado de manera abierta, aplicable a muchas situaciones conocidas y desconocidas. Este proceso permitió comprender mejor los conceptos y hacerlos más aplicables. Éste no fue el propósito original de la unificación, pero sí uno de los resultados más importantes.

Objetivos de UML

Hubo varios objetivos detrás del desarrollo de UML. El primero y más importante, UML es un lenguaje de modelado de propósito general que pueden usar todos los modeladores. No tiene propietario y está basado en el común acuerdo de gran parte de la comunidad informática. Esto significa incluir conceptos de los métodos líderes para que UML pueda usarse como su lenguaje de modelado. Está pensado para reemplazar al menos los modelos de OMT, Booch y Objectory, así como aquéllos de otros participantes de la propuesta. Se pensó para ser tan familiar como sea posible, usar la notación de OMT, Booch, Objectory y otros métodos importantes. Esto significa incorporar buenas prácticas de diseño, tales como la encapsulación, separación de los temas, y la captura de la intención del modelo construido. Pretende abordar los problemas actuales del desarrollo de software, tales como gran tamaño, distribución, concurrencia, patrones, y desarrollo en equipo.

UML no pretende ser un método de desarrollo completo. No incluye un proceso de desarrollo paso a paso. Creemos que un buen proceso de desarrollo es crucial para el éxito de un desarrollo de software, y proponemos uno en un libro complementario [Jacobson-99]. Es importante darse cuenta de que UML y el proceso para usar UML son dos cosas independientes.

UML pretende trabajar correctamente con todos, o al menos con la mayoría de los procesos de desarrollo existentes. UML incluye todos los conceptos que consideramos necesarios para utilizar un proceso moderno iterativo, basado en construir una sólida arquitectura para resolver requisitos dirigidos por casos de uso.

Un objetivo final de UML era ser tan simple como fuera posible pero manteniendo la capacidad de modelar toda la gama de sistemas que se necesita construir. UML necesita ser lo suficientemente expresivo para manejar todos los conceptos que se originan en un sistema moderno, tales como la concurrencia y distribución, así como también los mecanismos de la ingeniería de software, tales como encapsulación y componentes. Debe ser un lenguaje universal, como cualquier lenguaje de programación de propósito general. Desafortunadamente eso significa que no puede ser pequeño si quiere manejar cosas que no sean sistemas de juguete. Los lenguajes modernos y los sistemas operativos modernos, son mucho más complicados hoy que hace 40 años, porque nosotros esperamos mucho más de ellos. UML tiene varios tipos de modelos; no es algo que uno pueda dominar en un día. Es más complicado que algunos de sus antecesores porque intenta ser más amplio. Pero no es necesario aprenderlo todo a la vez, no más que lo que exige un lenguaje de programación, un sistema operativo, o una compleja aplicación de usuario.

Áreas conceptuales de UML

Los conceptos y modelos de UML pueden agruparse en las siguientes áreas conceptuales.

Estructura estática. Cualquier modelo preciso debe primero definir el universo del discurso, esto es, los conceptos clave de la aplicación, sus propiedades internas, y las relaciones entre cada una. Este conjunto de construcciones es la vista estática. Los conceptos de la aplicación son modelados como clases, cada una de las cuales describe un conjunto de objetos discretos que almacenan información y se comunican para implementar un comportamiento. La información que almacenan es modelada como atributos; el comportamiento que realizan es modelado como operaciones. Varias clases pueden compartir una estructura común usando generalización. Una clase hija añade nuevas estructuras y comportamientos a las estructuras y comportamientos que obtiene por herencia de la clase padre. Los objetos también tienen conexión durante la ejecución con otros objetos individuales. Tales relaciones "Objeto a Objeto" son modeladas como asociaciones entre clases. Algunas relaciones entre elementos se agrupan como relaciones de dependencia, incluyendo las relaciones para modelar desplazamientos en los niveles de abstracción, enlace de parámetros del modelo, concesión de permisos, y uso de un elemento por parte de otro. Otras relaciones son la combinación de casos de uso y el flujo de datos. La vista estática se expresa con diagramas de clases y puede usarse para generar la mayoría de las declaraciones de estructuras de datos en un programa. Hay muchos tipos de elementos en los diagramas UML, tales como interfaces, tipos de datos, casos de uso y señales. En conjunto, se les llama clasificadores, y se comportan de forma muy parecida a las clases, con ciertas restricciones en cada tipo de clasificador.

Comportamiento dinámico. Hay dos formas de modelar el comportamiento. Una es la historia de la vida de un objeto, que muestra la forma en que interactúa con el resto del mundo; la otra son los patrones de comunicación de un conjunto de objetos conectados, que muestra cómo interactúan para implementar su comportamiento.

La visión de un objeto aislado es una máquina de estados —una vista de un objeto que muestra la forma en que responde a los eventos en función de su estado actual, realiza acciones

como parte de su respuesta y transiciones a un nuevo estado—. Las máquinas de estados se representan en un diagrama de estados.

La visión de la interacción de los objetos de un sistema es una colaboración: una vista, dependiente de contexto, de los objetos y los enlaces entre ellos, junto con el flujo de mensajes entre los objetos mediante los enlaces de datos. Este punto de vista unifica la estructura de los datos, el control de flujo y el flujo de datos en una sola vista. Las colaboraciones e interacciones se muestran mediante los diagramas de secuencia y los diagramas de colaboración. Guiando todas las vistas de comportamiento se encuentra un conjunto de casos de uso. Cada uno es una descripción de una porción de la funcionalidad del sistema como la percibe un actor, un usuario externo del sistema.

Construcciones de implementación. Los modelos de UML tienen significado para el análisis lógico y para la implementación física. Ciertos constructores representan elementos de implementación. Un componente es una parte física reemplazable de un sistema y es capaz de responder a las peticiones descritas por un conjunto de interfaces. Se pretende que sea fácilmente sustituible por otro componente que cumpla la misma especificación. Un nodo es un recurso computacional que define una localización durante la ejecución del sistema. Pueden contener componentes y objetos. La vista de despliegue describe la configuración de los nodos de un sistema en ejecución y la organización de los componentes y objetos en él, incluyendo posibles migraciones de contenido entre nodos.

Organización del modelo. Los ordenadores pueden manejar grandes modelos "planos", pero los humanos no. En un sistema grande, la información del modelo debe ser dividida en piezas coherentes, para que los equipos puedan trabajar en las diferentes partes de forma concurrente. Incluso en un sistema más pequeño, el conocimiento humano requiere que se organice el contenido del modelo en paquetes de tamaño modesto. Los paquetes son unidades organizativas, jerárquicas, y de propósito general de los modelos de UML. Pueden usarse para almacenamiento, control de acceso, gestión de la configuración, y construcción de bibliotecas que contengan fragmentos de código reutilizable. Una dependencia entre paquetes resume las dependencias entre los contenidos del paquete. Una dependencia entre paquetes puede ser impuesta por la arquitectura global del sistema. Por lo tanto, los contenidos de los paquetes deben adaptarse a las dependencias del paquete y a las impuestas por la arquitectura del sistema.

Mecanismos de extensión. No importa cuan completo sea el conjunto de "facilidades" de un leguaje, la gente querrá hacer extensiones. Hemos dotado a UML de una limitada capacidad de extensión, que creemos suficiente para la mayoría de las extensiones que requiere el "día a día", sin la necesidad de un cambio en el lenguaje básico. Un estereotipo es una nueva clase de elemento de modelado con la misma estructura que un elemento existente pero con restricciones adicionales, una interpretación diferente de un icono, y un tratamiento diferente por los generadores de código y otras herramientas de bajo nivel. Un valor etiquetado es un par arbitrario de cadenas etiqueta-valor, que pueden enlazarse a cualquier tipo de elemento de modelado, para almacenar información arbitraria, como información de gestión de proyecto, guías para los generadores de código, y valores requeridos por los estereotipos. La etiqueta y el valor son representadas como cadenas. Una restricción es una condición "bien formada" expresada en una cadena de texto en algún lenguaje restringido, tal como un lenguaje de programación, un lenguaje especial limitado, o lenguaje natural. UML incluye un lenguaje de restricciones llamado OCL[1]. Como con cualquier mecanismo de extensión, estos mecanismos deben usarse con cui-

[1] Object Constraint Language, "Lenguaje de Restricción de Objetos", *N. del T.*

dado debido al riesgo de producir un dialecto privado ilegible por los demás. Pero a la vez pueden evitar la necesidad de cambios más radicales.

Sintaxis de los diagramas y las expresiones

Este libro contiene expresiones y diagramas que muestran ejemplos de modelos reales, así como también la sintaxis de las expresiones y anotaciones explicando los diagramas. Para reducir el peligro de confundir las explicaciones con los ejemplos, se han usado ciertas convenciones de formato.

Dentro de los diagramas y las expresiones de texto, los fragmentos de diagrama o literales de texto, que aparecerán en la notación real, se muestran en letra "Helvética". Por ejemplo, un nombre de clase en negro es un nombre legal que puede aparecer en un modelo. Un paréntesis en una expresión de sintaxis es un paréntesis literal que aparecerá en la expresión real; no es parte de la "maquinaria" sintáctica. Por ejemplo:

Pedido.crear (cliente,cantidad)

Dentro del texto ejecutable, las palabras literales del modelo y los nombres de los elementos del modelo, también se muestran en letra "Helvética" negra, como **Pedido** o **cliente**.

En una expresión sintáctica, los nombres de las unidades sintácticas, que deben ser reemplazadas por las expansiones reales del texto, se muestran en letra "Helvética" azul, como nombre.

La aparición de texto negro en una expresión, representa un valor literal que aparecerá en la notación destino. El uso de cursivas o subrayado significa, que el texto que la reemplazará, tiene esa propiedad. Por ejemplo:

nombre.operación **(**argumento...,**)**

nombre-objeto: clase

En una expresión sintáctica, un subíndice azul y una barra superior azul, se usan para denotar ciertas propiedades sintácticas.

expresión$_{opc}$	La expresión es opcional.
expresión$_{lista,}$	Puede aparecer una lista de expresiones separadas por comas. Si hay cero o una repetición, no hay separador. Cada repetición puede tener una sustitución independiente. Si aparece un signo de puntuación diferente de la coma, éste es el separador.
$\overline{\text{= expresión}_{opc}}$	Una barra superior enlaza juntos uno a más términos, que son considerados una unidad para la optatividad o la repetición de ocurrencias. En este ejemplo, el igual y la expresión forman una unidad que puede ser omitida o incluida. La barra superior es innecesaria si sólo hay un término.

En un diagrama los textos y flechas en azul son anotaciones. No son parte de la notación real, pero se pensaron como explicaciones. Los símbolos y texto en negro son parte de la notación.

2 La naturaleza y propósito de los modelos

Este capítulo explica qué son los modelos, para qué son buenos, y cómo se usan. También explica varias clases de modelos: ideal, parcial y basado en herramientas.

¿Qué es un modelo?

Un modelo es una representación, en cierto medio, de algo en el mismo u otro medio. El modelo capta los aspectos importantes de lo que estamos modelando, desde cierto punto de vista, y simplifica u omite el resto. La ingeniería, la arquitectura y muchos otros campos creativos usan modelos.

Un modelo se expresa en un medio adecuado para el trabajo. Los modelos de construcciones pueden dibujarse en papel, las figuras tridimensionales son construidas con cartón y papel cartón, o las ecuaciones de elementos finitos en un computador. Un modelo de construcción de un edificio muestra la apariencia del edificio pero también puede usarse para hacer ingeniería y cálculos de coste.

Un modelo de un sistema software está construido en un lenguaje de modelado, como UML. El modelo tiene semántica y notación y puede adoptar varios formatos que incluyen texto y gráficos. El modelo pretende ser más fácil de usar para ciertos propósitos que el sistema final.

¿Para qué sirven los modelos?

Los modelos se usan para muchos propósitos.

Para captar y enumerar exhaustivamente los requisitos y el dominio de conocimiento, de forma que todos los implicados puedan entenderlos y estar de acuerdo con ellos. Diferentes modelos de un edificio captan requisitos sobre la apariencia, los patrones de tráfico, varios tipos de servicios, fortaleza frente al viento y terremotos, costes y otras cosas. Los implicados incluyen al arquitecto, aparejador, contratista, varias subcontratas, el dueño, los inquilinos y la ciudad.

Los diversos modelos de un sistema de software pueden capturar requisitos sobre su dominio de aplicación, las formas en que los usuarios lo utilizarán, su división en módulos, los patrones comunes usados en su construcción, y otras cosas. Los implicados incluyen al arquitecto, a los analistas, a los programadores, al encargado del proyecto, a los clientes, a inversores, a los usuarios finales, y a los operadores. Se utilizan los diferentes tipos de modelos de UML.

Para pensar del diseño de un sistema. Un arquitecto utiliza modelos en papel, sobre un computador, o como construcciones tridimensionales, para visualizar y experimentar con posibles diseños. La simplicidad de crear y de modificar modelos pequeños permite pensamiento creativo e innovación con poco coste.

Un modelo de un sistema software ayuda a los desarrolladores a explorar varias arquitecturas y soluciones de diseño fácilmente, antes de escribir el código. Un buen lenguaje de modelado permite que el diseñador consiga la arquitectura correcta antes de que comience el diseño detallado.

Para capturar decisiones del diseño en una forma mutable a partir de los requisitos. Un modelo de un edificio muestra el aspecto externo convenido con el cliente. Otro modelo muestra el encaminamiento interno de cables, de tuberías, y de conductos de ventilación. Hay muchas maneras de implementar estos servicios. El modelo final muestra un diseño que el arquitecto cree que es bueno. El cliente puede verificar esta información, pero, a menudo, los clientes no se preocupan por los detalles mientras funcionen.

Un modelo de un sistema software puede captar el comportamiento externo de un sistema y la información del dominio del mundo real representado por el sistema. Otro modelo muestra las clases y las operaciones internas, que implementan el comportamiento externo. Hay muchas maneras de implementar el comportamiento; el modelo final de diseño muestra un acercamiento que el diseñador cree correcto.

Para generar productos aprovechables para el trabajo. Un modelo de un edificio se puede utilizar para generar varias clases de productos. Éstos incluyen una cuenta de materiales, una simulación animada de un paseo, una tabla de desviaciones a varias velocidades del viento, o una visualización de la tensión en varios puntos en el armazón.

Un modelo de un sistema software se puede utilizar para generar las declaraciones de clase, los cuerpos de procedimiento, las interfaces de usuario, las bases de datos, los escenarios de uso válidos o los guiones de configuración.

Para organizar, encontrar, filtrar, recuperar, examinar, y corregir la información en grandes sistemas. Un modelo de un edificio organiza la información por servicio: estructural, eléctrico, fontanería, ventilación, decoración, etcétera. A menos que el modelo esté en un computador, no es tan fácil encontrar cosas y modificarlas. Si está en un computador, los cambios se pueden realizar y recordar fácilmente, y los múltiples diseños pueden ser explorados fácilmente mientras comparten algunos elementos comunes.

Un modelo de un sistema de software organiza la información en varias vistas: estructura estática, máquinas de estado, interacciones, requisitos, etc. Cada vista es una proyección del modelo completo para un propósito determinado. Mantener un modelo, de cualquier tamaño, es imposible sin tener una herramienta de edición, que maneje el modelo. Un editor gráfico e interactivo del modelo, puede presentar la información en diversos formatos, ocultar información que es innecesaria para un propósito dado, y mostrarla otra vez más adelante, agrupar operaciones relacionadas, realizar cambios en los elementos individuales, así como cambiar grupos de elementos con un comando, etc.

Para explorar económicamente múltiples soluciones. Las ventajas y los riesgos de diversos métodos de diseño para edificios, pueden no estar claros al principio. Por ejemplo, diversas subestructuras pueden interactuar de formas complicadas, que no se pueden evaluar en la cabeza

de un ingeniero. Los modelos pueden explorar los diversos diseños y permitir cálculos de costes y de riesgos, antes de que se construya el edificio real.

Los modelos de un sistema de software grande permiten que se propongan y comparen varios diseños. Los modelos no se construyen al detalle, por supuesto, pero incluso un modelo rudimentario puede exponer muchas cuestiones que el diseño final debe tener en cuenta. Modelar permite considerar varios diseños, con un coste pequeño al implementar cualquiera de ellos.

Para domesticar los sistemas complejos. Un modelo de ingeniería de un tornado acercándose a un edificio, proporciona un conocimiento que no es posible obtener de un edificio del mundo real. Un tornado verdadero no se puede producir a voluntad, y destruiría los instrumentos de medida, de todas formas. Muchos procesos físicos rápidos, pequeños, o violentos se pueden entender usando modelos físicos.

Un modelo de un sistema software grande permite ocuparse de la complejidad que es demasiado difícil de tratar directamente. Un modelo se puede abstraer a un nivel que sea comprensible a los seres humanos sin perderse en detalles. Un computador puede realizar complicados análisis en un modelo, en un esfuerzo para encontrar problemas tales como errores de sincronización o desbordamiento en los recursos. Un modelo puede determinar el impacto potencial de un cambio antes de que se haga, explorando dependencias en el sistema. Un modelo puede también mostrar cómo reestructurar un sistema para reducir tales efectos.

Niveles de los modelos

Los modelos adquieren diversas formas para diferentes propósitos, y aparecen en diversos niveles de abstracción. La cantidad de detalle del modelo debe adaptarse a uno de los siguientes propósitos:

Guías al proceso de pensamiento. Los modelos de alto nivel construidos al principio de un proyecto, sirven para enfocar el proceso del pensamiento de los participantes y destacar determinadas opciones. Capturan requisitos y representan un punto de partida hacia un diseño del sistema. Los primeros modelos ayudan a los autores a explorar las opciones posibles antes de converger en un concepto de sistema. Según progresa el diseño, los primeros modelos son sustituidos por otros modelos más exactos.

No hay necesidad de conservar cada giro y vuelta del proceso de exploración inicial. Su propósito es producir ideas. Sin embargo los modelos de pensamiento finales, se deben preservar incluso después de que nuestro foco de atención se desplace a tareas de diseño. Los primeros modelos no requieren el detalle o la precisión de un modelo de implementación, y no necesitan una gama completa de conceptos de implementación. Tales modelos utilizan un subconjunto de construcciones de UML, un subconjunto más limitado que los de modelos posteriores.

Cuando un primer modelo es una vista completa de un sistema con determinada precisión —por ejemplo, el modelo resultante del análisis de lo que debe ser hecho— debe ser preservado cuando el desarrollo pasa a la etapa siguiente. Hay una diferencia importante entre añadir detalles (en cuyo caso la cadena de razonamiento debe ser preservada) y el proceso normal de caminar al azar, explorando muchos callejones sin salida, antes de llegar a la solución correcta. En este último caso es generalmente abrumador e innecesario guardar la historia completa, excepto en las situaciones extraordinarias, en las cuales se requiere la capacidad completa de averiguar lo ocurrido.

Especificaciones abstractas de la estructura esencial de un sistema. Los modelos en el análisis o las etapas preliminares del diseño se centran en los conceptos y mecanismos claves del probable sistema. Se corresponden de cierta manera con el sistema final. Pero faltan los detalles, que se deben agregar explícitamente durante el proceso de diseño. El propósito de los modelos abstractos es conseguir que los aspectos de alto nivel estén correctos, antes de abordar los detalles más localizados. Estos modelos se piensan para evolucionar en los modelos finales, mediante un proceso cuidadoso que garantice que el sistema final implementa correctamente el objetivo de los modelos anteriores. Estos modelos esenciales deben corresponderse con los modelos completos; sino, no hay seguridad de que el sistema final incorpore correctamente las características clave, que el modelo esencial intentó mostrar. Los modelos esenciales se centran en la semántica. No necesitan la gama completa de opciones de implementación. De hecho, los detalles del funcionamiento a bajo nivel oscurecen a menudo la semántica lógica. La trayectoria de un modelo esencial a un modelo completo de implementación debe ser clara y directa, no importa si ha sido generada automáticamente por un generador de código o desarrollada manualmente por un diseñador.

Especificaciones completas de un sistema final. Un modelo de implementación incluye suficiente información para construir el sistema. Debe incluir, no solamente la semántica lógica del sistema y los algoritmos, las estructuras de datos y los mecanismos que aseguran funcionamiento apropiado, sino también las decisiones de organización sobre los artefactos del sistema que son necesarios, permitiendo así el trabajo cooperativo de las personas y el procesamiento por parte de las herramientas. Esta clase de modelo debe incluir las construcciones para empaquetar el modelo, para la comprensión de la persona, y para la conveniencia de la computadora. Éstas no son las características de la aplicación en sí misma. En realidad, son características del proceso de construcción.

Ejemplos de sistemas típicos o posibles. Algunos bien elegidos pueden facilitar el entendimiento a las personas, y pueden validar las especificaciones e implementación del sistema. Una gran colección de ejemplos, sin embargo, no elimina la necesidad de una descripción definitiva. En última instancia necesitamos los modelos que especifican el caso general; que es un programa, después de todo. Sin embargo, los ejemplos de las estructuras de datos típicas, de las secuencias de interacción, o de las historias de los objetos, pueden ayudar a las personas a intentar entender una situación complicada. Los ejemplos se deben utilizar con un cierto cuidado. Es lógicamente imposible inducir el caso general, a partir de un conjunto de ejemplos, pero los prototipos bien elegidos son la manera de pensar de la mayoría de la gente. Un modelo de ejemplo incluye instancias más que descriptores generales y por lo tanto, tiende a dar una visión diferente de la que da un modelo descriptivo genérico. Los modelos de ejemplo generalmente usan sólo un subconjunto de las construcciones de UML, las que se ocupan de instancias. Los modelos descriptivos y los modelos de ejemplo son útiles para modelar un sistema.

Descripciones completas o parciales de sistemas. Un modelo puede ser una descripción completa de un solo sistema, sin referencias externas. Más a menudo se organiza como un conjunto de unidades distintas, discretas, cada una de las cuales se puede almacenar y manipular por separado, como parte de la descripción completa. Tales modelos tienen conexiones que se deben enlazar a otros modelos en un sistema completo. Como las piezas tienen coherencia y significado, pueden ser combinadas con otras piezas de varias maneras para producir sistemas muy diversos. Lograr la reutilización es una meta importante del buen modelado.

Los modelos cambian con el tiempo. Los modelos con mayor grado de detalle se derivan de modelos más abstractos, y los modelos más concretos se derivan de modelos más lógicos. Por

ejemplo, un modelo pudo comenzar como vista de alto nivel del sistema entero, con algunos servicios clave poco detallados y ningún adorno. En un cierto plazo, se agrega mucho más detalle y se introducen las variaciones. También en un cierto plazo, el enfoque cambia de una visión centrada en el usuario y el aspecto exterior del sistema a una visión lógica centrada en su núcleo. A medida que los desarrolladores trabajan con un sistema y lo entienden mejor, el modelo se debe iterar en todos los niveles para capturar ese conocimiento: es imposible entender un sistema grande en una sola pasada lineal. No hay ninguna forma "correcta" para un modelo.

¿Qué hay en un modelo?

Semántica y presentación. Los modelos tienen dos aspectos importantes: Información semántica (semántica) y presentación visual (notación).

El aspecto semántico capta el significado de una aplicación como una red de construcciones lógicas, por ejemplo clases, asociaciones, estados, casos de uso, y mensajes. Los elementos semánticos del modelo llevan el significado del modelo, es decir, transportan la semántica. Los elementos semánticos del modelo se utilizan para la generación del código, la comprobación de la validez, las métricas de complejidad, etc. El aspecto visual es irrelevante para la mayoría de las herramientas que procesan modelos. La información semántica a menudo es llamada *el modelo*. Un modelo semántico tiene una estructura sintáctica, reglas para asegurar su corrección, y dinámicas de ejecución. Estos aspectos se describen a menudo por separado (como en los documentos de definición de UML), pero se correlacionan firmemente y parten de un solo modelo coherente.

La presentación visual muestra la información semántica de modo que se pueda ser considerada, hojeada y corregida por los seres humanos. Los elementos de la presentación llevan la presentación visual del modelo —esto es, lo muestran en una forma directamente comprensible por las personas—. No agregan significado, sino que organizan la presentación, para acentuar la organización del modelo de una manera útil. Por lo tanto, dirigen la comprensión humana del modelo. Los elementos de la presentación derivan su semántica de elementos semánticos del modelo. Pero, ya que la disposición de los diagramas la realizan las personas, los elementos de la presentación no son totalmente derivables de elementos lógicos. La disposición de los elementos de presentación puede contener connotaciones sobre las relaciones semánticas que son demasiado débiles o ambiguas para ser formalizadas en el modelo semántico pero que, sin embargo, sugieren algo a las personas.

Contexto. Los modelos son artefactos en un sistema informático, y se utilizan dentro de un contexto más grande que les dé significado completo. Este contexto incluye la organización interna del modelo, anotaciones sobre el uso de cada modelo en el proceso total del desarrollo, un sistema de valores por defecto y de suposiciones para la creación y la manipulación del elemento, y una relación al entorno en el cual se utilizan.

Los modelos requieren una organización interna que permita su uso simultáneo por varios grupos de trabajo, sin interferencias indebidas. Esta descomposición no se hace necesaria por razones semánticas —un modelo monolítico grande sería tan exacto como un sistema de modelos organizados en los paquetes coherentes, quizá aún más exacto porque los límites de organización complican el trabajo de definir la semántica exacta—. Pero los equipos de trabajadores no podrían trabajar con eficacia en un modelo monolítico grande sin entorpecerse constantemente unos a otros. Por otra parte, un modelo monolítico no tiene piezas que se puedan reutilizar en otras situaciones. Finalmente, los cambios en un modelo grande tienen consecuencias difíciles de de-

terminar. Los cambios en una pieza pequeña, aislada, de un modelo grande pueden ser manejables si el modelo se estructura correctamente en subsistemas con interfaces bien definidas. En cualquier caso, dividir los sistemas grandes en una jerarquía de piezas bien elegidas es la manera más fiable de diseñar sistemas grandes que los seres humanos han inventado en miles de años.

Los modelos capturan la información semántica sobre un sistema, pero también necesitan registrar mucha información sobre el proceso de desarrollo mismo, tal como el autor de una clase, el estado de depuración de un procedimiento, y el permiso de acceso de una persona a un diagrama. Tal información es, en el mejor de los casos, ajena a la semántica del sistema, pero es importante para el proceso de desarrollo. El modelo de un sistema, por lo tanto, necesita incluir ambos puntos de vista. Esto se consigue más fácilmente añadiendo información de gestión del proyecto, como anotaciones al modelo semántico (descripciones arbitrarias añadidas a los elementos del modelo pero cuyo significado está fuera del lenguaje de modelado). En UML se implementan como cadenas de texto.

Los comandos usados para crear y modificar un modelo no son parte de la semántica del lenguaje de modelado, del mismo modo en que los comandos de un editor de textos o de un navegador no son parte de la semántica de un lenguaje de programación. Las propiedades del elemento del modelo no tienen valores *prefijados*; en un modelo particular, simplemente tienen *valores*. Para el desarrollo práctico, sin embargo, las personas necesitan construir y modificar modelos, sin tener que especificar todo completo en detalle. Los valores prefijados están en el límite entre el lenguaje de modelado y la herramienta de edición que lo soporta. Son realmente valores por defecto en los comandos de la herramienta que crea un modelo aunque pueden trascender de una herramienta individual y convertirse en expectativas del usuario sobre la implementación del lenguaje por las herramientas en general.

Los modelos no se construyen y utilizan aislados. Son parte de un entorno más grande que incluye herramientas de modelado, lenguajes y compiladores, sistemas operativos, redes de computadores, restricciones de implementación, etc. La información sobre un sistema incluye la información sobre todas las partes del entorno. Parte será almacenado en un modelo aunque no es información semántica; por ejemplo, las anotaciones de gestión de proyecto (discutidas antes), las ayudas y directivas de la generación de código, empaquetado del modelo, y la configuración por defecto para herramienta de edición. El resto se puede almacenar por separado: código fuente del programa, comandos de configuración del sistema operativo, etc. Incluso si una cierta información es parte de un modelo, la responsabilidad de interpretarlo puede subyacer en varios lugares, incluyendo el lenguaje de modelado, la herramienta de modelado, el generador de código, el compilador, un lenguaje de comandos, etc. Este libro describe la interpretación de modelos hechos con el lenguaje UML, pero cuando funcionan en un entorno físico de desarrollo, otras fuentes pueden agregar interpretaciones adicionales a la información que es opaca para UML.

¿Cuál es el significado de un modelo?

Un modelo es un *generador* de potenciales configuraciones de sistemas; los posibles sistemas son sus *extensiones*, o valores. Idealmente, todas las configuraciones consistentes con el modelo deberían ser posibles. A veces, sin embargo, no es posible representar todas las restricciones dentro de un modelo. Un modelo es también una descripción de la estructura genérica y del significado de un sistema. Las descripciones son su *objetivo*, o significado. Un modelo es siempre una abstracción a un cierto nivel. Captura los aspectos *esenciales* de un sistema y omite algunos de los detalles. En los modelos hay que considerar los siguientes aspectos:

Abstracción frente a detalle. Un modelo captura los aspectos esenciales de un sistema y omite otros. Cuáles son esenciales es una cuestión de juicio que depende del propósito del modelo. Esto no es una dicotomía; puede haber un espectro de modelos de precisión creciente. Un lenguaje de modelado no es un lenguaje de programación.

Un lenguaje de modelado puede ser menos exacto a propósito, porque el detalle adicional es irrelevante para el propósito actual. A lo largo de la vida de un proyecto, se pueden utilizar modelos de diferentes niveles de precisión. Un modelo pensado para la generación del código requiere que se traten, por lo menos, algunos elementos del lenguaje de programación. Típicamente, los modelos tienen poca precisión durante el análisis. Ganan en detalle a medida que progresa el ciclo de desarrollo, por lo que los modelos finales, tienen un detalle y precisión considerables.

Especificación frente a implementación. Un modelo puede decir qué hace algo (*especificación*), y también cómo se logra la función (*implementación*). Estos aspectos deben ser separados en el modelado. Es importante determinar el *qué*, antes de invertir muchas horas en el *cómo*. Abstraerse de la implementación es una faceta importante del modelado. Puede haber una cadena de varias relaciones especificación-implementación, en las cuales cada implementación define las especificaciones para la capa siguiente.

Descripción frente a instancia. Los modelos son sobre todo descripción. Las cosas que describen son las instancias, que generalmente aparecen en los modelos sólo como ejemplos. La mayoría de las instancias existen solamente como parte de la ejecución. Algunas veces sin embargo, las instancias de tiempo de ejecución son a veces en sí mismas descripciones de otras cosas. Llamamos a estos objetos híbridos *metadatos*. Es poco realista insistir en que todo es una instancia o una descripción: algo es una instancia o una descripción solamente en referencia a algo más, y la mayoría de las cosas se pueden considerar desde múltiples puntos de vista.

Variaciones en la interpretación. Hay muchas interpretaciones posibles de modelos en un lenguaje de modelado. Uno puede definir ciertos *puntos de variación semántica* —lugares en los cuales son posibles diversas interpretaciones— y asignar a cada interpretación un nombre como *variación semántica*, de modo que uno pueda indicar qué variación está utilizando. Por ejemplo, los usuarios de Smalltalk pueden desear evitar herencia múltiple en un modelo de implementación porque no la tiene el lenguaje de programación. Los usuarios de otros lenguajes de programación lo necesitarían. Los puntos semánticos de variación permiten contemplar diversos modelos de ejecución.

Parte 2: Conceptos de UML

Esta parte contiene una descripción de los conceptos de UML para mostrar cómo encajan en el modelado de un sistema. No pretende describir los conceptos en completo detalle. Para consultar los detalles completos sobre un concepto de UML, véase la sección de la enciclopedia de este libro.

3 Un paseo por UML

Este capítulo presenta brevemente los conceptos y los diagramas de UML, usando un ejemplo simple. El propósito del capítulo es organizar los conceptos de alto nivel de UML, en un pequeño conjunto de vistas y de diagramas que presentan los conceptos visualmente. Muestra cómo se utilizan los diferentes conceptos describir un sistema, y cómo encajan las vistas unas con otras. Este resumen no pretende ser completo; se omiten muchos conceptos. Para más detalles, véanse los capítulos siguientes, que desarrollan las vistas semánticas de UML, así como el material de referencia detallado en el capítulo de la enciclopedia.

El ejemplo es una taquilla de teatro que ha automatizado sus operaciones. Es un ejemplo inventado con el propósito es destacar varias construcciones de UML en un espacio breve. Está simplificado y deliberadamente no se presenta por completo. La presentación de un modelo completo, de un sistema implementado, ni cabría en un espacio tan pequeño ni destacaría una gama suficiente de construcciones, sin excesiva repetición.

Vistas de UML

No hay ninguna línea entre los diferentes conceptos y las construcciones en UML, pero, por conveniencia, nosotros los dividimos en varias vistas. Una vista es simplemente un subconjunto de UML que modela construcciones que representan un aspecto de un sistema. La división en diversas vistas es algo arbitraria, pero esperamos que sea intuitiva. Una o dos clases de diagramas proporcionan una notación visual para los conceptos de cada vista.

En el nivel superior, las vistas se pueden dividir en tres áreas: clasificación estructural, comportamiento dinámico, y gestión del modelo.

La clasificación estructural describe los elementos del sistema y sus relaciones con otros elementos. Los clasificadores incluyen clases, casos del uso, componentes, y nodos y elementos proporcionan la base sobre la cual se construye el comportamiento dinámico. La clasificación de las vistas incluye la vista estática, la vista de casos de uso, y la vista de implementación.

El comportamiento dinámico describe el comportamiento de un sistema en el tiempo. El comportamiento se puede describir como serie de cambios a las fotos del sistema dibujadas a partir de la visión estática. Las vistas de comportamiento dinámico incluyen vista de la máquina de estados, la vista de actividad, y la vista de interacción.

La gestión del modelo describe la organización de los propios modelos en unidades jerárquicas. El paquete es la unidad genérica de organización para los modelos. Los paquetes especiales incluyen a los modelos y a los subsistemas. La vista de gestión del modelo cruza las otras vistas y las organiza para el trabajo de desarrollo y el control de configuración.

Tabla 3.1 Vistas y diagramas de UML

Área	*Vista*	*Diagramas*	*Conceptos Principales*
estructural	vista estática	diagrama de clases	clase, asociación, generalización, dependencia, realización, interfaz
	vista de casos de uso	diagrama de casos de uso	caso de uso, actor, asociación, extensión, inclusión, generalización de casos de uso
	vista de implementación	diagrama de componentes	componente, interfaz, dependencia, realización
	vista de despliegue	diagrama de despliegue	nodo, componente, dependencia, localización
dinámica	vista de máquina de estados	diagrama de estados	estado, evento, transición, acción
	vista de actividad	diagrama de actividad	estado, actividad, transición de terminación, división, unión
	vista de interacción	diagrama de secuencia	interacción, objeto, mensaje, activación
		diagrama de colaboración	colaboración, interacción, rol de colaboración, mensaje
gestión del modelo	vista de gestión del modelo	diagrama de clases	paquete, subsistema, modelo
extensión de UML	todas	todos	restricción, estereotipo, valores etiquetados

UML también contiene varias construcciones previstas para proporcionar una capacidad limitada, pero útil, de extensión. Estas construcciones incluyen restricciones, estereotipos, y valores etiquetados y son aplicables a los elementos de todas las vistas.

La Tabla 3.1 muestra las vistas de UML y los diagramas que las muestran, así como los principales conceptos relevantes de cada vista. Esta tabla no se debe tomar como un rígido sistema de reglas sino simplemente como guía para el uso normal, dado que se permite la mezcla de vistas.

Vista estática

La vista estática modela los conceptos del dominio de la aplicación, así como los conceptos internos inventados como parte de la implementación de la aplicación. Esta visión es estática porque no describe el comportamiento del sistema dependiente del tiempo, que se describe en otras vistas. Los componentes principales de la vista estática son las clases y sus relaciones: asociación, generalización, y varias clases de dependencia, tales como realización y uso. Una clase es la descripción de un concepto del dominio de la aplicación o de la solución de la apli-

cación. Las clases son el centro alrededor del cual se organiza la vista de clases; otros elementos pertenecen o se unen a las clases. La visión estática se exhibe en los diagramas de clases, llamados así porque su objetivo principal es la descripción de clases.

Las clases se dibujan como rectángulos. Las listas de atributos y de operaciones se muestran en compartimentos separados. Los compartimentos pueden ser suprimidos cuando no es necesario el detalle completo. Una clase puede aparecer en varios diagramas. Sus atributos y operaciones se suprimen, a menudo en todos menos en un diagrama.

Las relaciones entre clases se dibujan como las líneas que conectan rectángulos de clases. Las diversas clases de relaciones se diferencian por la textura de la línea y por los adornos en las líneas o en sus extremos.

La Figura 3.1 muestra un diagrama de clases de la aplicación de la taquilla. Este diagrama contiene parte del modelo del dominio "venta de entradas". Muestra varias clases importantes, tales como **Cliente**, **Reserva**, **Entrada**, y **Representación**. Los clientes pueden tener muchas reservas, pero cada reserva es hecha por un cliente. Las reservas son de dos clases: suscripción y reservas individuales. Ambos reservan entradas: en un caso, solamente una entrada; en el otro caso, varias entradas. Cada entrada es parte de una suscripción o de una reserva individual, pero

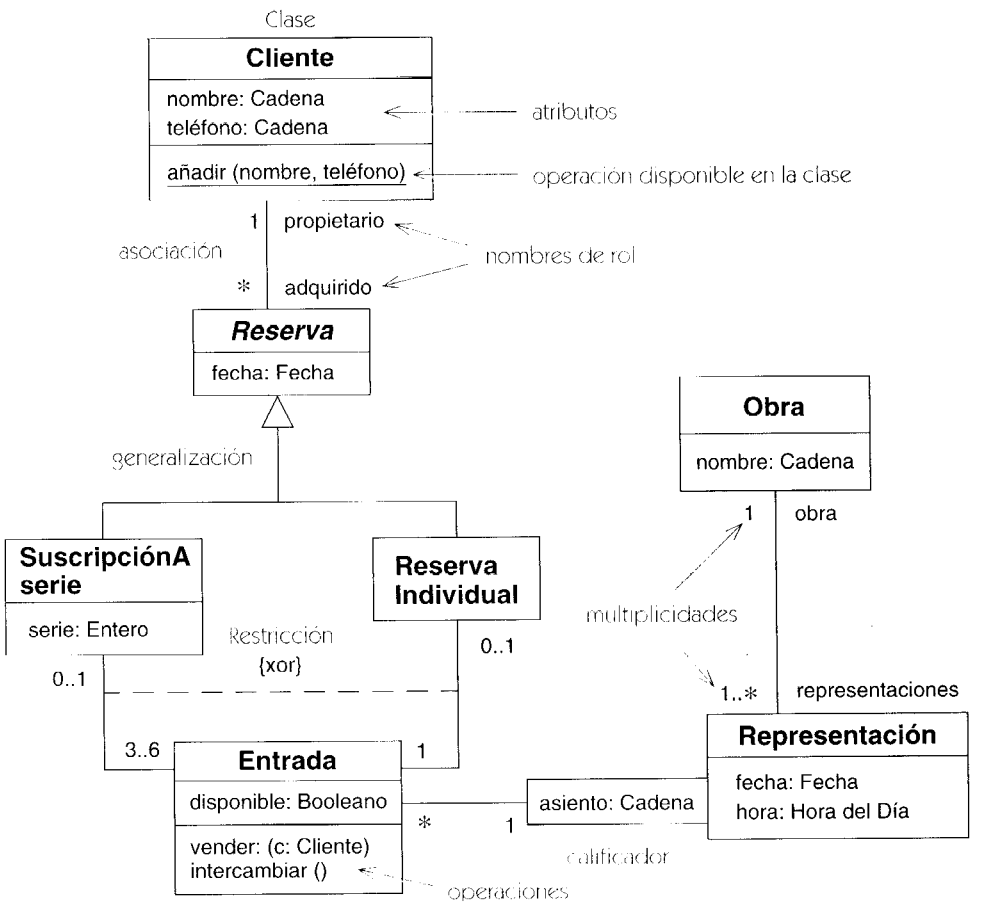


Figura 3.1 Diagrama de clases

no de ambas. Cada representación tiene muchas entradas disponibles, cada una con un número de asiento único. Una representación se puede identificar por una obra, una fecha, y una hora.

Las clases se pueden describir con varios niveles de precisión y concreción. Al empezar el diseño, el modelo captura los aspectos más lógicos del problema. En las fases posteriores, el modelo también capta decisiones de diseño y detalles de la implementación. La mayoría de las vistas tienen un comportamiento evolutivo similar.

Vista de los casos de uso

La vista de los casos de uso modela la funcionalidad del sistema según lo perciben los usuarios externos, llamados actores. Un caso de uso es una unidad coherente de funcionalidad, expresada como transacción entre los actores y el sistema. El propósito de la vista de casos de uso es enumerar a los actores y los casos de uso, y demostrar qué actores participan en cada caso de uso.

La Figura 3.2 muestra un diagrama de casos de uso para el ejemplo de la taquilla. Los actores son el vendedor, el supervisor, y el quiosco. El quiosco es otro sistema que acepta pedidos de un cliente. El cliente no es actor en la aplicación de la taquilla, porque no está conectado directamente con la aplicación. Los casos de uso incluyen el comprar entradas, a través del quiosco o del vendedor, comprar suscripciones (solamente a través del vendedor), y examinar las ventas totales (a petición del supervisor).

El comprar entradas y el comprar las suscripciones incluyen un fragmento común —que es hacer cargos al servicio de la tarjeta de crédito—. (La descripción completa de la taquilla im-

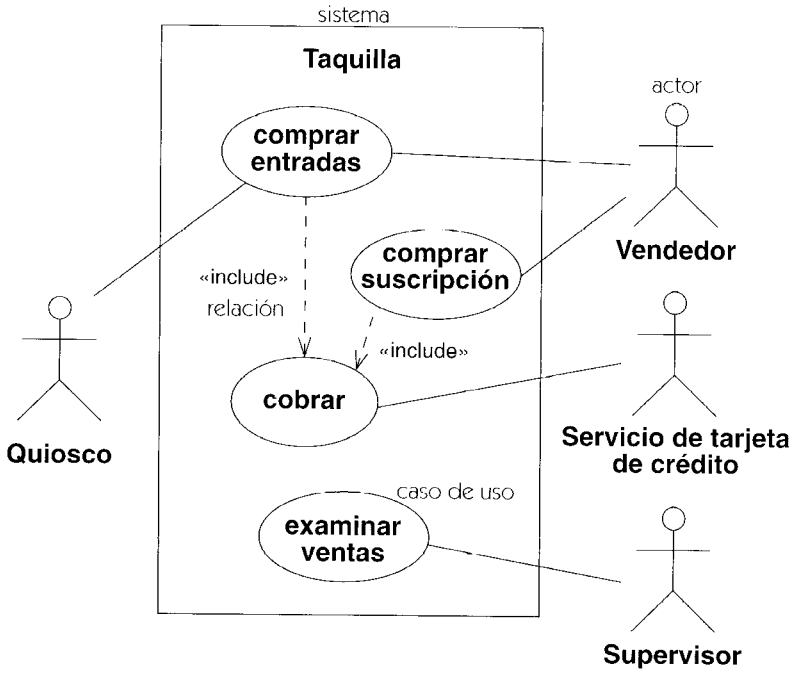


Figura 3.2 Diagrama de casos de uso

plicaría muchos otros casos de uso, tales como cambio de entradas y comprobación de disponibilidad.)

Los casos de uso se pueden también describir en varios niveles de detalle. Se pueden sacar partes como factor común y ser descritos en términos de otros casos de uso más simples. Un caso del uso se implementa como una colaboración en la vista de interacción.

Vista de interacción

La vista de interacción describe secuencias de intercambios de mensajes entre los roles que implementan el comportamiento de un sistema. Un rol de clasificador, o simplemente "rol", es la descripción de un objeto, que desempeña un determinado papel dentro de una interacción, distinto de los otros objetos de la misma clase. Esta visión proporciona una vista integral del comportamiento de un sistema —es decir, muestra el flujo de control a través de muchos objetos—. La vista de interacción se exhibe en dos diagramas centrados en distintos aspectos: diagramas de secuencia y diagramas de colaboración.

El diagrama de secuencia

Un diagrama de secuencia muestra un conjunto de mensajes, dispuestos en una secuencia temporal. Cada rol en la secuencia se muestra como una línea de vida, es decir, una línea vertical

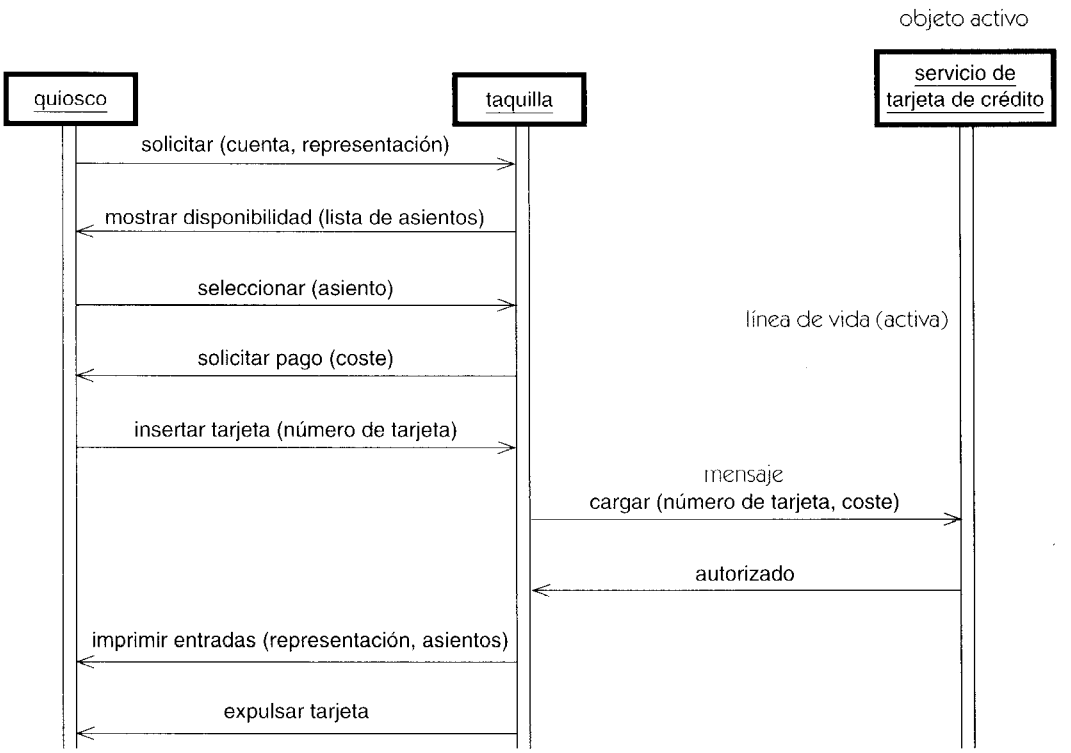


Figura 3.3 Diagrama de secuencia

que representa el rol durante cierto plazo de tiempo, con la interacción completa. Los mensajes se muestran como flechas entre las líneas de vida. Un diagrama de secuencia puede mostrar un escenario, es decir, una historia individual de una transacción.

Un uso de un diagrama de secuencia es mostrar la secuencia del comportamiento de un caso del uso. Cuando está implementado el comportamiento, cada mensaje en un diagrama de secuencia corresponde a una operación en una clase, a un evento disparador, o a una transición en una máquina de estados.

La Figura 3.3 muestra un diagrama de secuencia para el caso de uso **comprar entrada**. Este caso de uso lo inicia el cliente en el quiosco, comunicándose con la taquilla. Los pasos para el caso de uso **hacer cargos** se incluyen en la secuencia, que implica la comunicación con el quiosco y el servicio de la tarjeta de crédito. Este diagrama de secuencia está en una primera etapa de desarrollo y no muestra los detalles completos de la interfaz de usuario. Por ejemplo, la forma exacta de la lista de asientos y el mecanismo de especificar asientos, deben ser definidos aún, pero la comunicación esencial de la interacción ha sido especificada por el caso de uso.

El diagrama de colaboración

Una colaboración modela los objetos y los enlaces significativos dentro de una interacción. Los objetos y los enlaces son significativos solamente en el contexto proporcionado por la interacción. Un rol describe un objeto, y un rol en la asociación describe un enlace dentro de una colaboración. Un diagrama de colaboración muestra los roles en la interacción en una disposición geométrica (Figura 3.4). Los mensajes se muestran como flechas, ligadas a las líneas de la relación, que conectan a los roles. La secuencia de mensajes, se indica con los números secuenciales que preceden a las descripciones del mensaje.

Un uso de un diagrama de colaboración es mostrar la implementación de una operación. La colaboración muestra los parámetros y las variables locales de la operación, así como asociaciones más permanentes. Cuando se implementa el comportamiento, la secuencia de los mensajes corresponde a la estructura de llamadas anidadas y el paso de señales del programa.

La Figura 3.4 muestra un diagrama de colaboración para la interacción de la reserva de entradas en una fase ulterior del desarrollo. La colaboración muestra la interacción entre objetos internos en la aplicación para reservar entradas. La petición llega del quiosco y se utiliza para encontrar la base de datos de la representación deseada en el conjunto de todas las representaciones. El puntero **db** que es devuelto al objeto **vendedor de entradas** representa un enlace local transitorio a una base de datos de representaciones, que persiste durante la interacción y después se desecha. El vendedor de entradas solicita un número de asientos para la representación; se encuentra una selección de asientos en varias gamas de precios, se bloquea temporalmente, y se devuelve al quiosco para la selección de los clientes. Cuando el cliente hace una selección de la lista de asientos, se obtienen los asientos seleccionados y el resto son liberados.

Tanto los diagramas de secuencia como los diagramas de colaboración muestran interacciones, pero acentúan aspectos diferentes. Un diagrama de secuencia muestra secuencias en el tiempo como dimensión geométrica, pero las relaciones entre roles son implícitas. Un diagrama de colaboración muestra las relaciones entre roles geométricamente, y relaciona los mensajes con las relaciones, pero las secuencias temporales están menos claras, porque vienen dadas por los números de secuencia. Cada diagrama debe ser utilizado cuando su aspecto principal es el foco de atención.

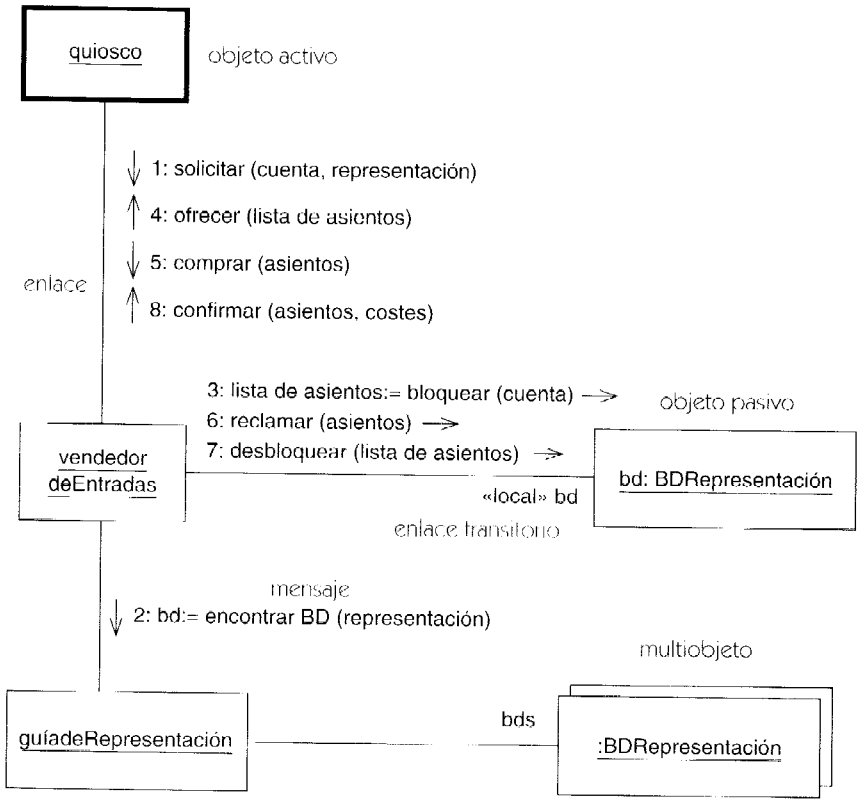


Figura 3.4 Diagrama de colaboración

Vista de la máquina de estados

Una máquina de estados modela las posibles historias de vida de un objeto de una clase. Una máquina de estados contiene los estados conectados por transiciones. Cada estado modela un período de tiempo, durante la vida de un objeto, en el que satisface ciertas condiciones. Cuando ocurre un evento, se puede desencadenar una transición que lleve el objeto a un nuevo estado. Cuando se dispara una transición, se puede ejecutar una acción unida a la transición. Las máquinas de estados se muestran como diagramas de estados.

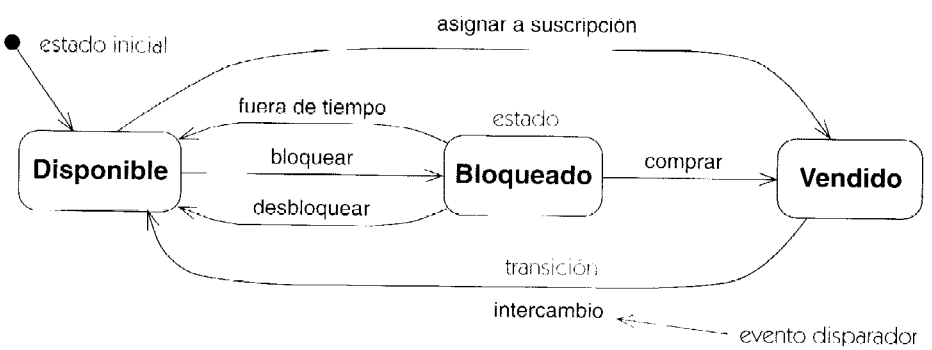


Figura 3.5 Diagrama de estados

La Figura 3.5 muestra un diagrama de estados de la historia de una entrada para una representación. El estado inicial de una entrada (representado por un punto negro) es el estado **disponible**. Antes del comienzo de la temporada, se asignan los asientos para los suscriptores de la temporada. Las entradas individuales compradas interactivamente primero se bloquean mientras el cliente hace una selección. Después de esto, se venden o se liberan si no se eligen. Si el cliente tarda demasiado en hacer una selección, finaliza el tiempo de la transacción y se libera el asiento. Los asientos vendidos para suscriptores de temporada, se pueden cambiar para otras representaciones, en cuyo caso, vuelven a estar disponibles otra vez.

Las máquinas de estados se pueden utilizar para describir interfaces de usuario, controladores de dispositivo, y otros subsistemas reactivos. También pueden usarse para describir los objetos pasivos que pasan por varias fases cualitativas distintas, durante su tiempo de vida, cada una de las cuales tiene su propio comportamiento especial.

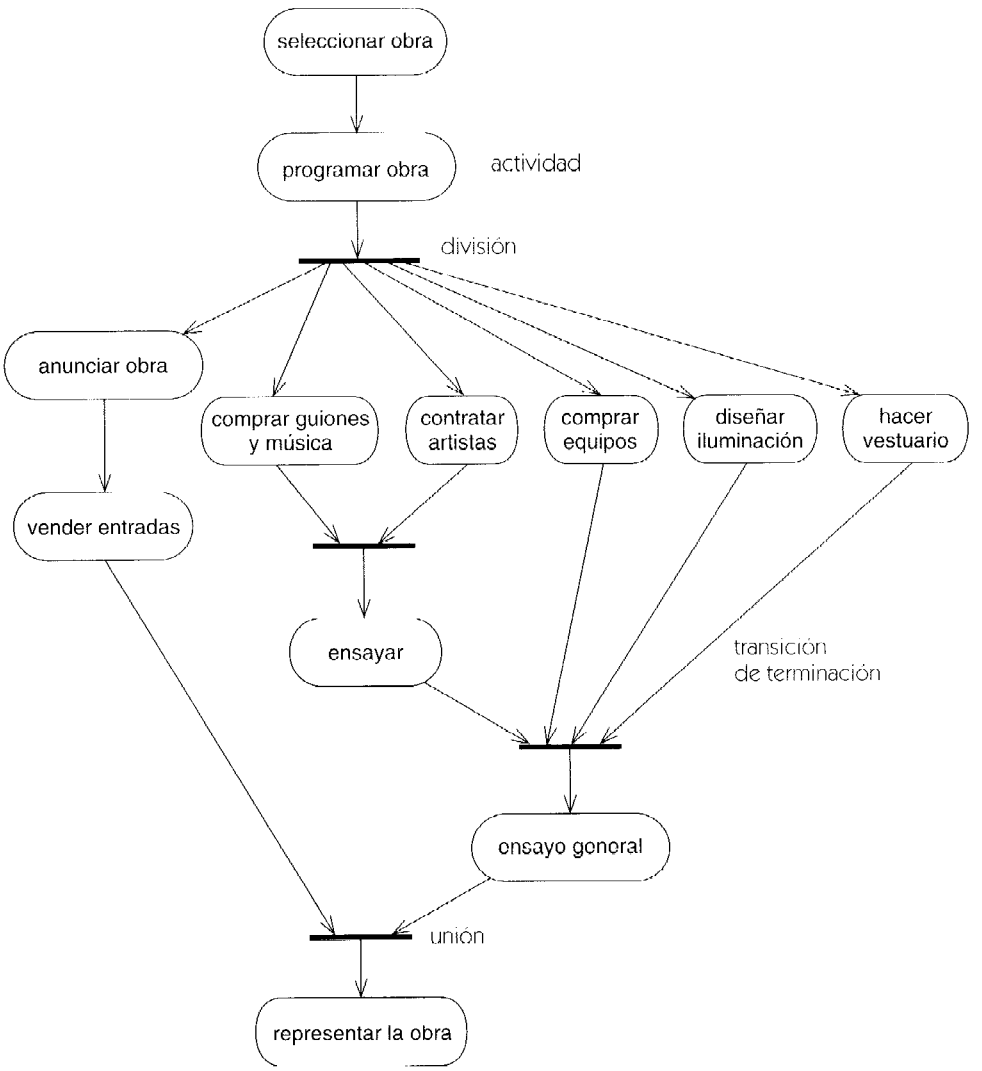


Figura 3.6 Diagrama de actividades

Vista de actividades

Un grafo de actividades es una variante de una máquina de estados, que muestra las actividades de computación implicadas en la ejecución de un cálculo. Un estado de actividad representa una actividad: un paso en el flujo de trabajo o la ejecución de una operación. Un grafo de actividades describe grupos secuenciales y concurrentes de actividades. Los grafo de actividades se muestran en diagramas de actividades.

La Figura 3.6 muestra un diagrama de actividades para la taquilla. Este diagrama muestra las actividades implicadas en montar una obra. (¡No tome este ejemplo demasiado en serio si tiene experiencia teatral!) Las flechas muestran dependencias secuenciales por ejemplo, las representaciones deben ser seleccionadas antes de poder planificarlas. Las barras horizontales representan bifurcaciones o uniones de control. Por ejemplo, después de planificar la obra, el teatro puede comenzar a darle publicidad, a comprar guiones, a contratar artistas, a construir decorados, a diseñar la iluminación, y a hacer los trajes, todo concurrentemente. Antes de que el ensayo pueda comenzar, sin embargo, se deben haber encargado los guiones y contratado a los artistas.

Este ejemplo muestra un diagrama de actividades, cuyo propósito es modelar los procesos reales de una organización humana. El modelado de tales negocios es un propósito importante de los diagramas de actividades, pero los diagramas de actividades se pueden también utilizar para modelar actividades software. Un diagrama de actividades es provechoso para entender el comportamiento de alto nivel de la ejecución de un sistema, sin profundizar en los detalles internos de los mensajes, lo que requeriría un diagrama de colaboración.

Los parámetros de entrada y de salida de una acción se pueden mostrar usando las relaciones de flujo que conectan la acción y un estado de flujo del objeto.

Vistas físicas

Las vistas anteriores modelan los conceptos de la aplicación desde un punto de vista lógico. Las vistas físicas modelan la estructura de la implementación de la aplicación por sí misma, su organización en componentes, y su despliegue en nodos ejecución. Estas vistas proporcionan una oportunidad de establecer correspondencias entre las clases y los componentes de implementación y nodos.

Hay dos vistas físicas: la vista de implementación y la vista de despliegue.

La vista de implementación modela los componentes de un sistema —a partir de los cuales se construye la aplicación— así como las dependencias entre los componentes, para poder determinar el impacto de un cambio propuesto. También modela la asignación de clases y de otros elementos del modelo a los componentes.

La vista de implementación se representa en diagramas componentes. La Figura 3.7 muestra un diagrama de componentes para el sistema de la taquilla.

Hay tres interfaces de usuario: la de los clientes que usan un quiosco, la de los vendedores que usan el sistema de reserva automatizado, y la de los supervisores que hacen consultas sobre las ventas de entradas. Hay un componente vendedor de entradas que ordena las peticiones de los quioscos y de los vendedores; un componente que procesa los cargos a la tarjeta de crédito; y la base de datos que contiene la información de la entrada. El diagrama de componentes mues-

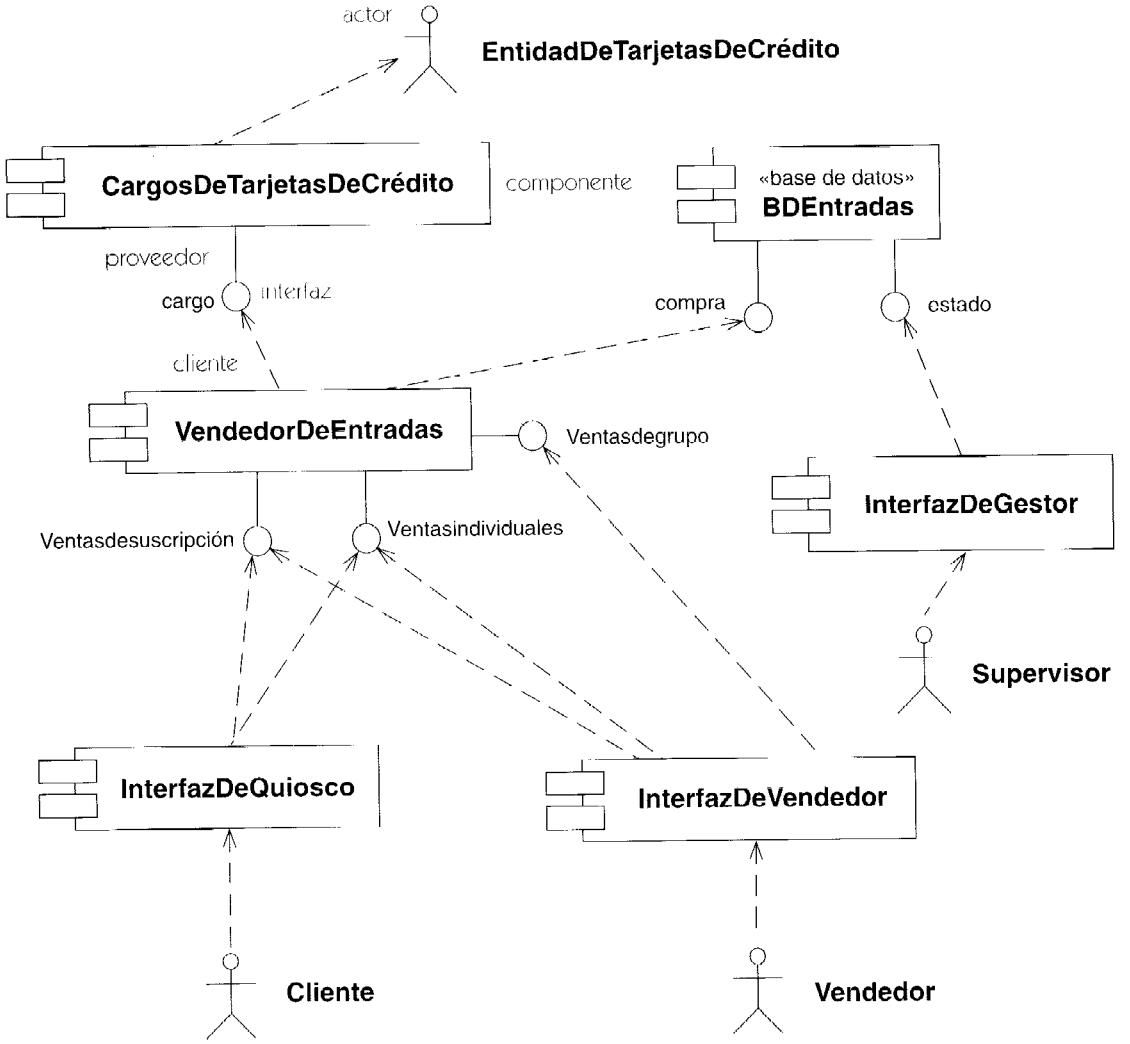


Figura 3.7 Diagrama de componentes

tra los tipos de componentes del sistema; una configuración particular de la aplicación puede tener más de una copia de un componente.

Un círculo pequeño con un nombre es una interfaz —un conjunto coherente de servicios—. Una línea sólida que va desde un componente a una interfaz, indica que el componente proporciona los servicios de la interfaz.

Una flecha de guiones de un componente a una interfaz indica que el componente requiere los servicios proporcionados por interfaz. Por ejemplo, las ventas de suscripciones y las ventas de grupos de entradas, son proporcionadas por el componente "vendedor de entradas"; las ventas de suscripciones son accesibles tanto para los quioscos como para los vendedores, pero las ventas de grupos sólo son accesibles para un vendedor.

La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos. Un nodo es un recurso de ejecución, tal como una computadora, un

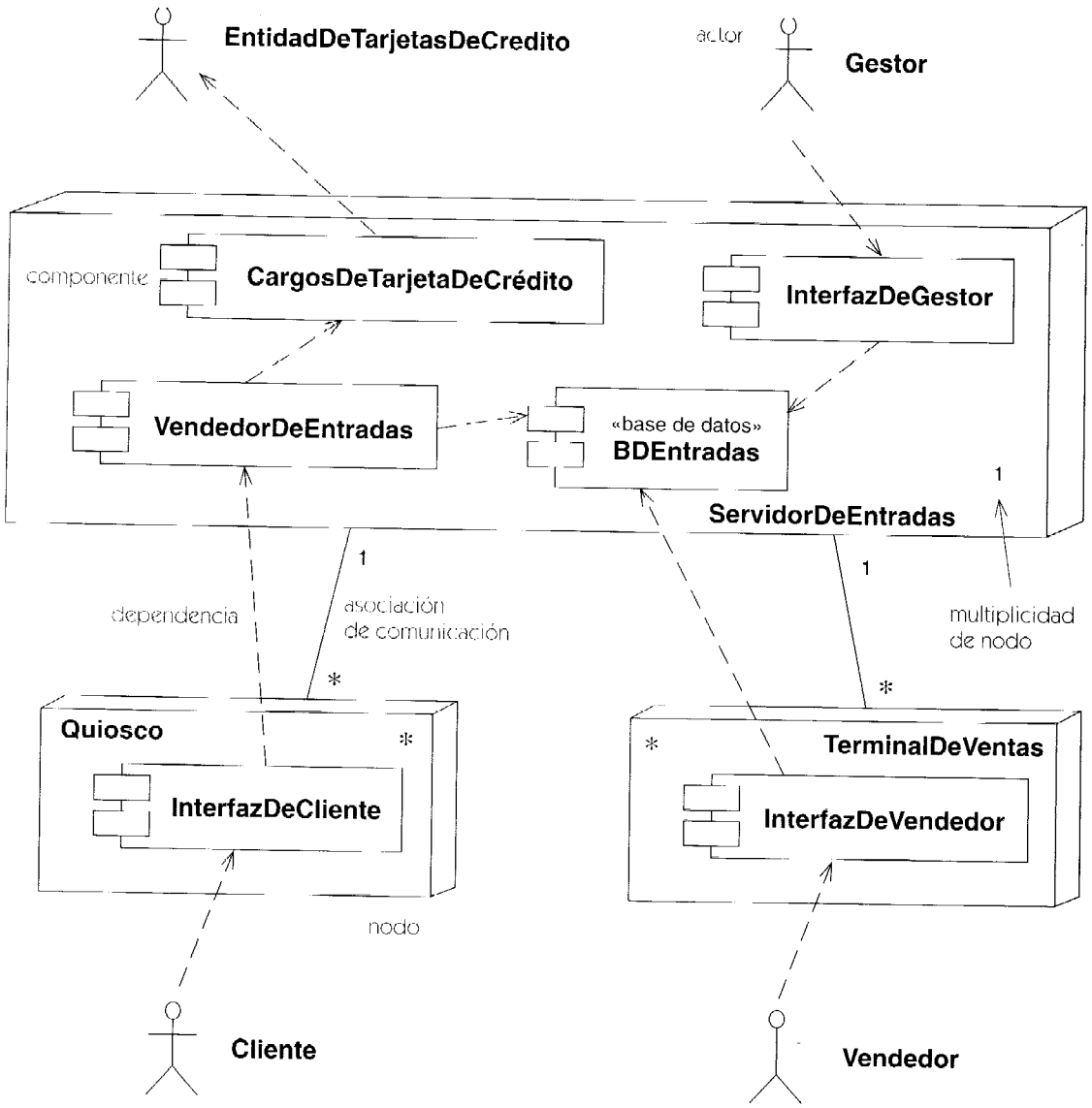


Figura 3.8 Diagrama de despliegue (nivel de descriptor)

dispositivo, o memoria. Esta vista permite determinar las consecuencias de la distribución y de la asignación de recursos.

La vista de despliegue se representa en diagramas de despliegue. La Figura 3.8 muestra un diagrama de despliegue del nivel de descriptor para el sistema de taquilla. Este diagrama muestra los tipos de nodos del sistema y los tipos de componentes que contienen. Un nodo se representa como un cubo.

La Figura 3.9 muestra un diagrama de despliegue del nivel de instancia, para el sistema de taquilla. El diagrama muestra los nodos individuales y sus enlaces, en una versión particular del sistema. La información de este modelo es consistente con la información del nivel de descriptor de la Figura 3.8.

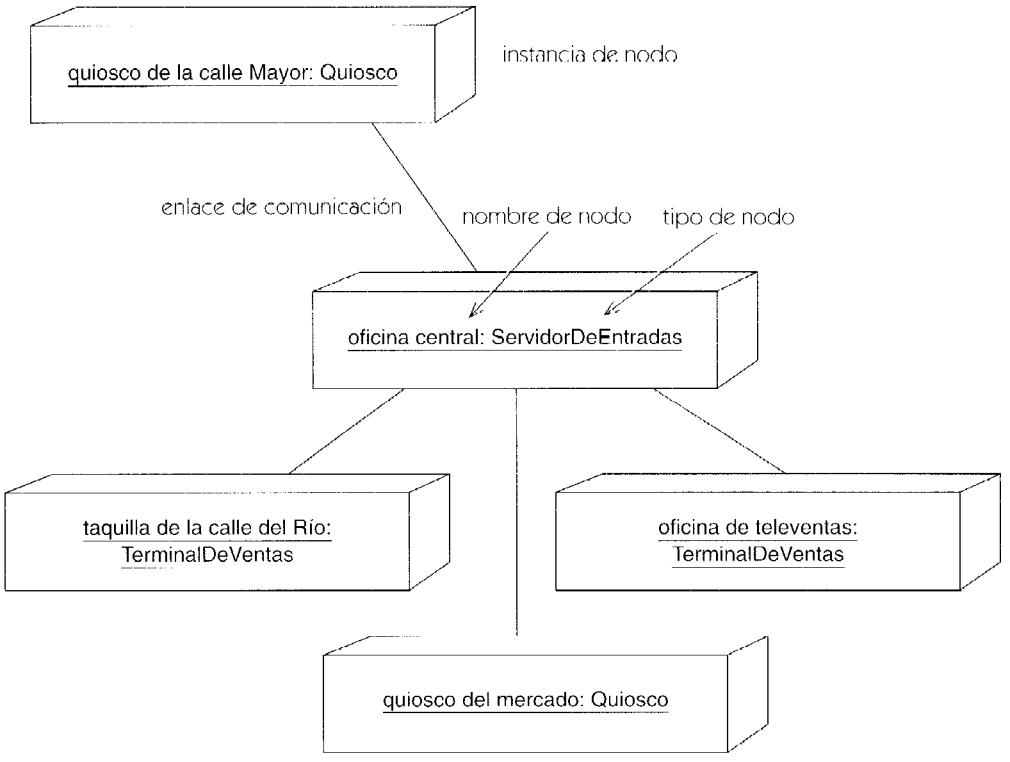


Figura 3.9 Diagrama de despliegue (nivel de instancia)

Vista de gestión del modelo

La vista de gestión del modelo modela la organización del modelo en sí mismo. Un modelo abarca un conjunto de paquetes que contienen los elementos del modelo, tales como clases, máquinas de estados, y casos de uso. Los paquetes pueden contener otros paquetes: por lo tanto, un modelo señala un paquete raíz, que contiene indirectamente todo el contenido del modelo. Los paquetes son unidades para manipular el contenido de un modelo, así como unidades para el control de acceso y el control de configuración. Cada elemento del modelo pertenece a un paquete o a otro elemento.

Un modelo es una descripción completa de un sistema, con una determinada precisión, desde un punto de vista. Puede haber varios modelos de un sistema desde distintos puntos de vista; por ejemplo, un modelo de análisis y un modelo de diseño. Un modelo se representa como una clase especial de paquete.

Un subsistema es otro paquete especial. Representa una porción de un sistema, con una interfaz perfectamente determinada, que puede ser implementado como un componente distinto.

Generalmente, la información de gestión del modelo se representa en diagramas de clases.

La Figura 3.10 muestra la descomposición de la totalidad del sistema de teatro en paquetes y sus relaciones de dependencia. El subsistema de taquilla incluye los ejemplos anteriores de este capítulo; el sistema completo también incluye operaciones del teatro y subsistemas de planificación. Cada subsistema consta de varios paquetes.

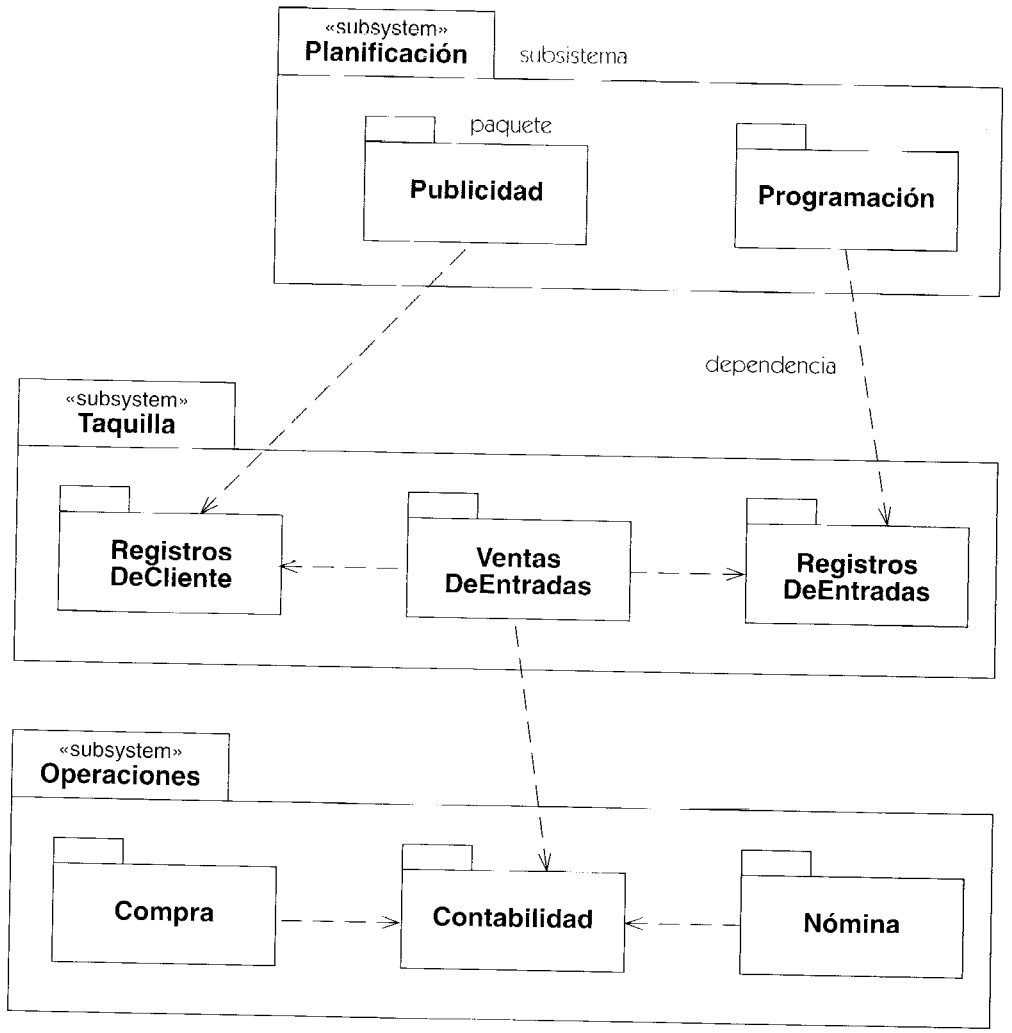


Figura 3.10 Paquetes

Construcciones de extensión

UML incluye tres construcciones principales de extensión: restricciones, estereotipos, y valores etiquetados. Una restricción es una declaración textual de una relación semántica expresada en un cierto lenguaje formal o en lenguaje natural. Un estereotipo es una nueva clase de elemento del modelo, ideada por el modelador, y basada en un tipo existente de elemento del modelo. Un valor etiquetado es una porción de información con nombre, unida a cualquier elemento del modelo.

Estas construcciones permiten muchas clases de extensiones a UML, sin requerir cambios al metamodelo básico de UML. Pueden ser utilizadas para crear versiones adaptadas a un área de aplicación.

La Figura 3.11 muestra ejemplos de restricciones, estereotipos, y de valores etiquetados. La restricción en la clase **Obra** asegura que los nombres de las obras sean únicos. La Figura 3.1

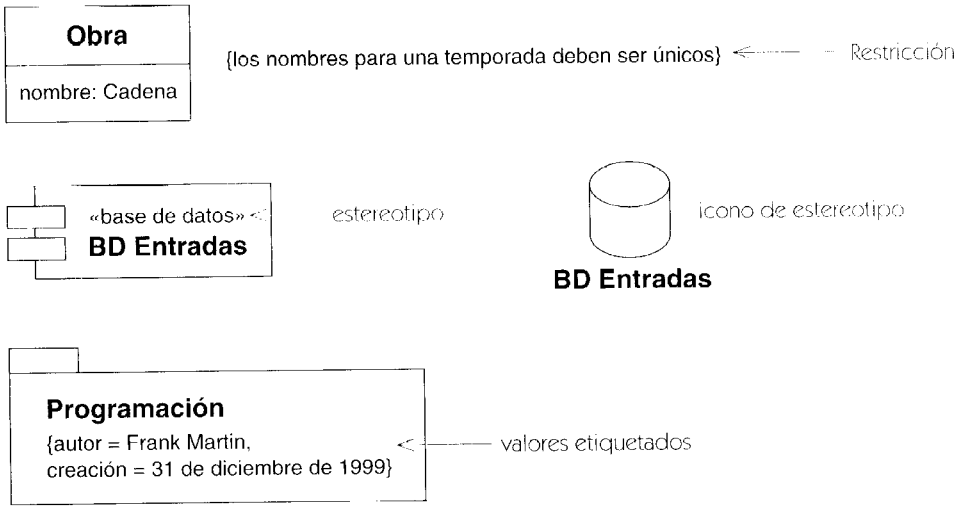


Figura 3.11 Construcciones de extensión

muestra una restricción **xor** entre dos asociaciones; un objeto sólo puede tener un enlace desde una de ellas a la vez. Las restricciones son útiles para representar sentencias que se pueden expresar en un lenguaje textual pero que no tienen soporte directo mediante construcciones de UML.

El estereotipo en el componente **DBEntrada** indica que el componente es una base de datos, lo que permite omitir qué interfaces soporta, puesto que son las soportadas por todas las bases de datos. Los modeladores pueden agregar nuevos estereotipos para representar elementos especiales. Un conjunto de restricciones implicadas, valores etiquetados, o características de la generación del código, pueden ser asociados a un estereotipo. Un modelador puede definir un icono para un nombre de estereotipo, como ayuda visual, según lo mostrado en el diagrama. No obstante, la forma textual se puede utilizar siempre.

Los valores etiquetados en el paquete **Programación** muestran que Frank Martin es responsable de acabarla antes del final del milenio. Se puede unir a un elemento del modelo cualquier pieza de información arbitraria, como un valor etiquetado bajo un nombre elegido por el modelador. Los valores de texto son especialmente útiles para la información de gestión de proyecto y para los parámetros de generación del código. La mayoría de los valores etiquetados serían almacenados, como información emergente, dentro de una herramienta de edición, y no aparecerán generalmente en las figuras impresas.

Conexiones entre vistas

Dentro de un modelo coexisten distintas vistas y sus elementos tienen muchas conexiones, algunas de las cuales se muestran en la Tabla 3.2. Esta tabla no pretende ser completa, sino mostrar algunas de las relaciones principales entre elementos de diversas vistas.

Tabla 3.2 Algunas Relaciones entre Elementos de Diferentes Vistas

Elemento	*Elemento*	*Relación*
clase	máquina de estados	propiedad
operación	interacción	realización
caso de uso	colaboración	realización
caso de uso	instancia de interacción	escenario de ejemplo
instancia de un componente	instancia de un nodo	localización
acción	operación	llamada
acción	señal	envío
actividad	operación	llamada
mensaje	acción	invocación
paquete	clase	propiedad
rol	clase	clasificación

4 La vista estática

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 subscripción, 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

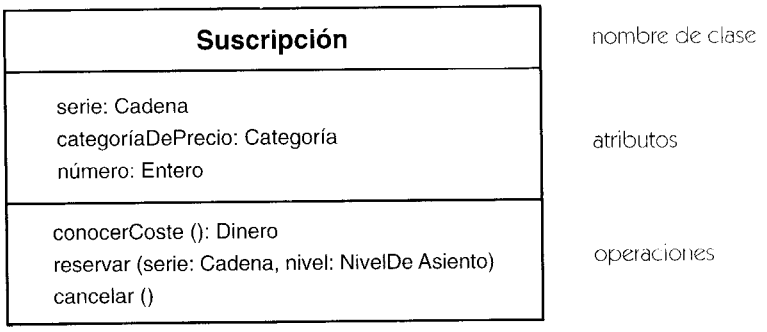
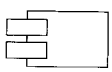


Figura 4.1 Notación de clases

Tabla 4.1 Tipos de Clasificadores

Clasificador	*Función*	*Notación*
actor	Un usuario externo al sistema	
clase	Un concepto del sistema modelado	**Nombre**
clase en un estado	Una clase restringida a estar en el estado dado	**Nombre[S]**
rol	Clasificador restringido a un uso particular en una colaboración	**rol:Nombre**
componente	Una pieza física de un sistema	
tipo de dato	Un descriptor de un conjunto de valores primitivos que carecen de identidad	**Nombre**
interfaz	Un conjunto de operaciones con nombre que caracteriza un comportamiento	**Inombre**
nodo	Un recurso computacional	
señal	Una comunicación asíncrona entre objetos	«signal»
subsistema	Un paquete que es tratado como una unidad con una especificación, implementación, e identidad	«subsystem»
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

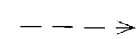
Las relaciones entre clasificadores son asociación, generalización, flujo, y varias clases de dependencia, que incluyen la realización y el uso (véase la Tabla 4.2).

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

Relación	*Función*	*Notación*
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.

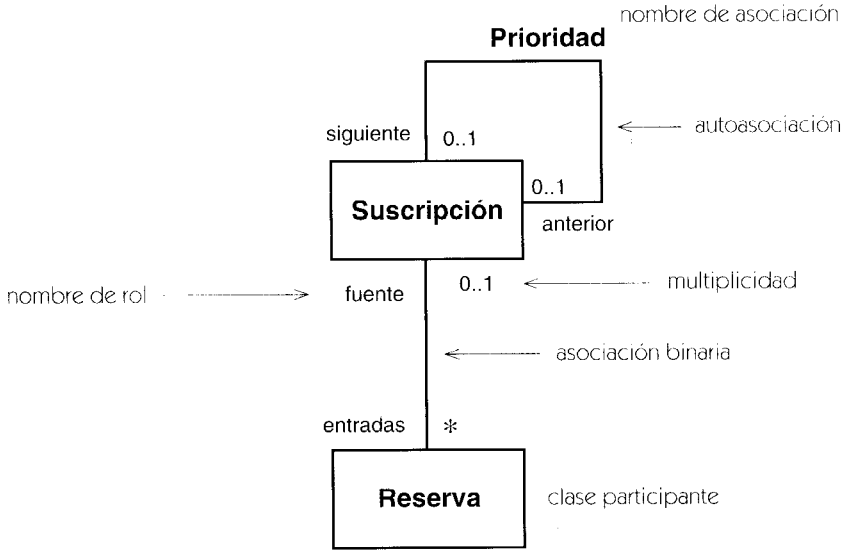


Figura 4.2 Notación de asociación

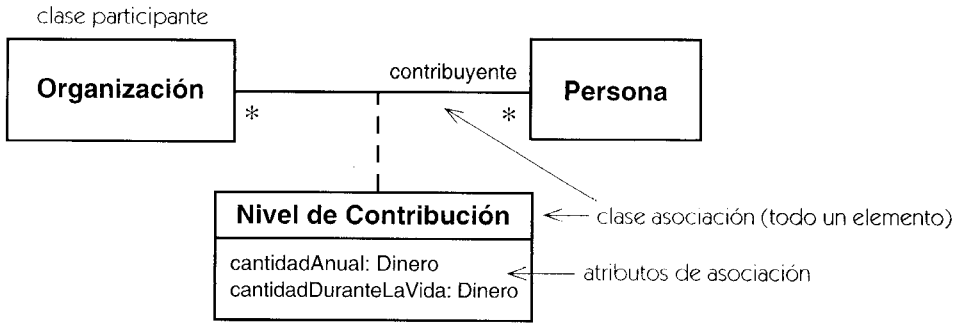


Figura 4.3 Clase de asociación

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 cualificadas. 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-

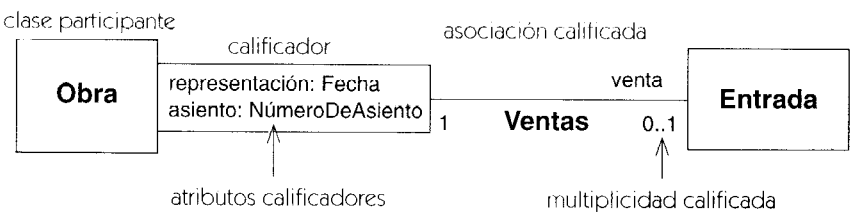


Figura 4.4 Asociación calificada

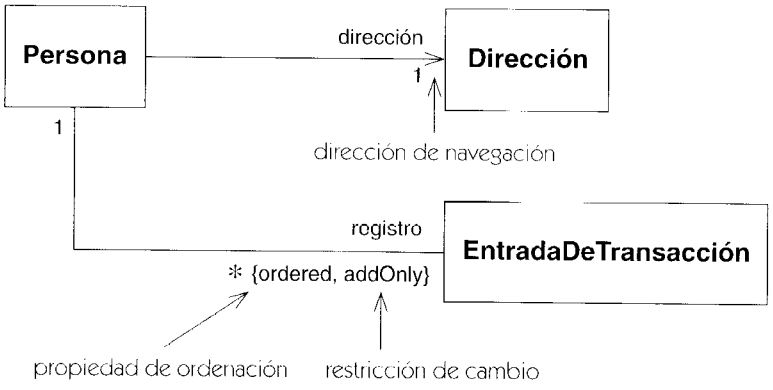


Figura 4.5 Propiedades del diseño de asociaciones

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-

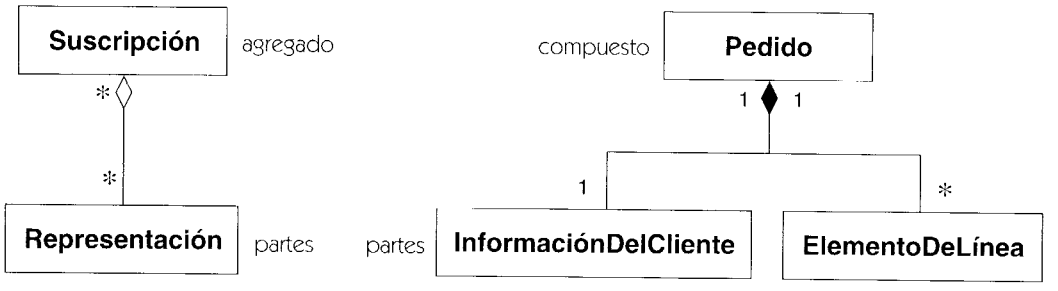


Figura 4.6 Agregación y Composición

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

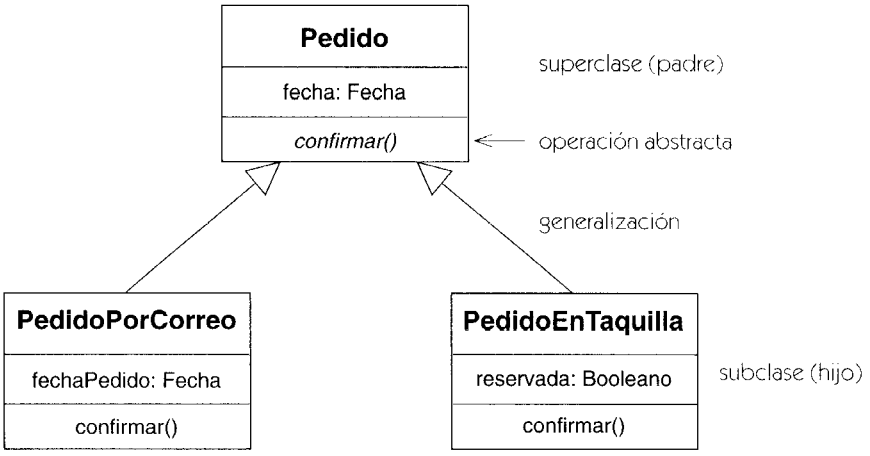


Figura 4.7 Notación de la generalización

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 signatura 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 signatura 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

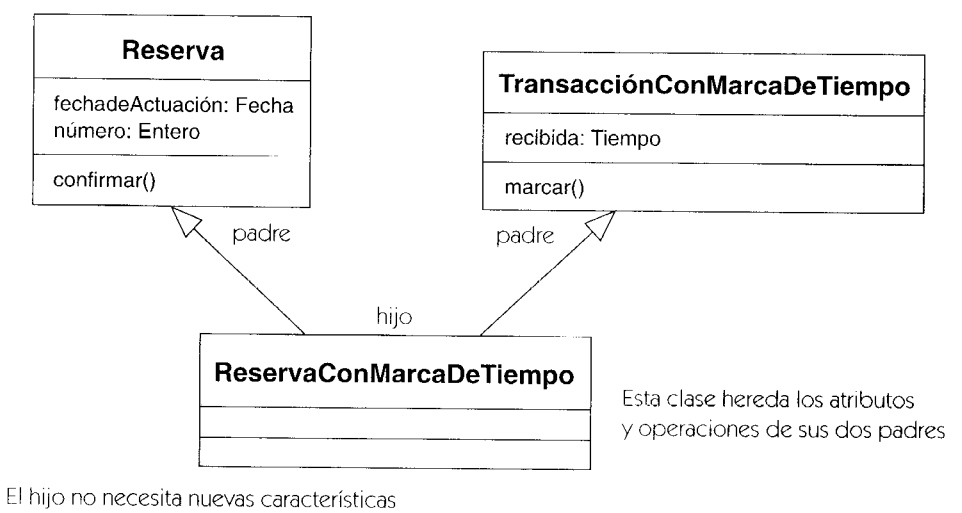


Figura 4.8 Herencia múltiple

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 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

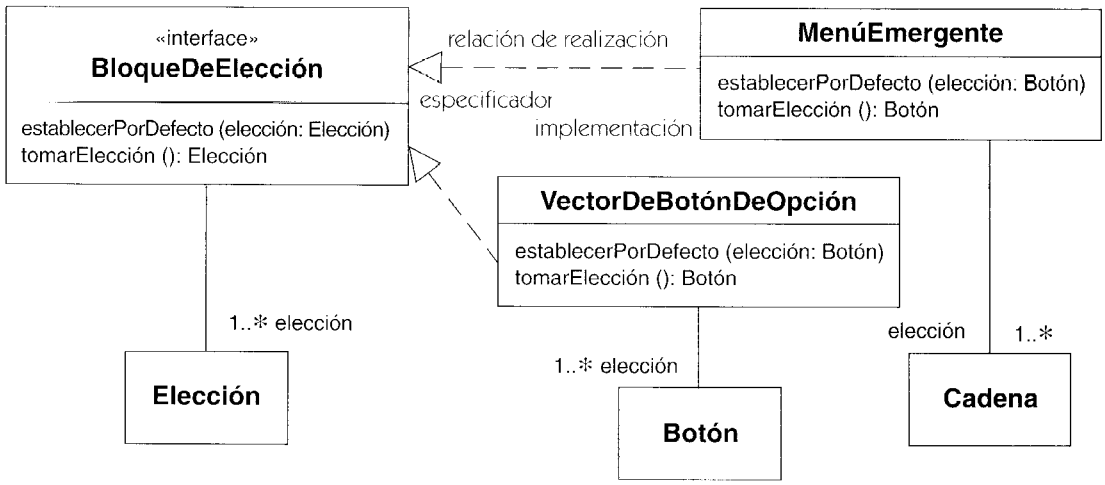


Figura 4.9 Relación de realización

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).

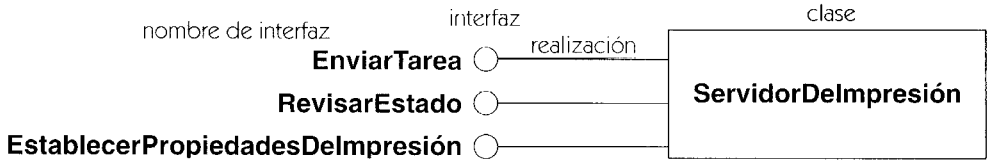


Figura 4.10 Interfaze iconos de realización

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

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

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.

[1] 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.*)

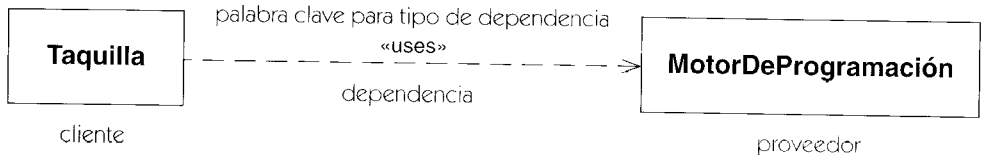


Figura 4.11 Dependencias

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.

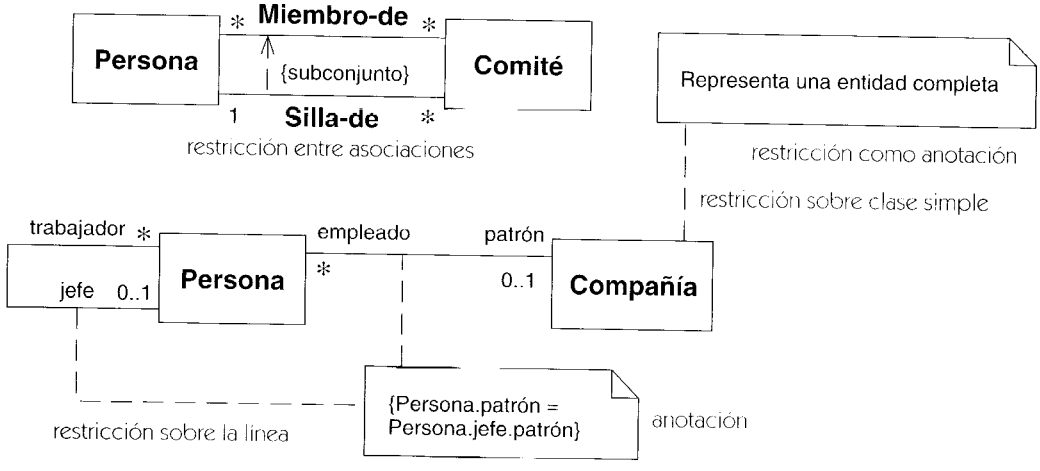


Figura 4.12 Restricciones

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.

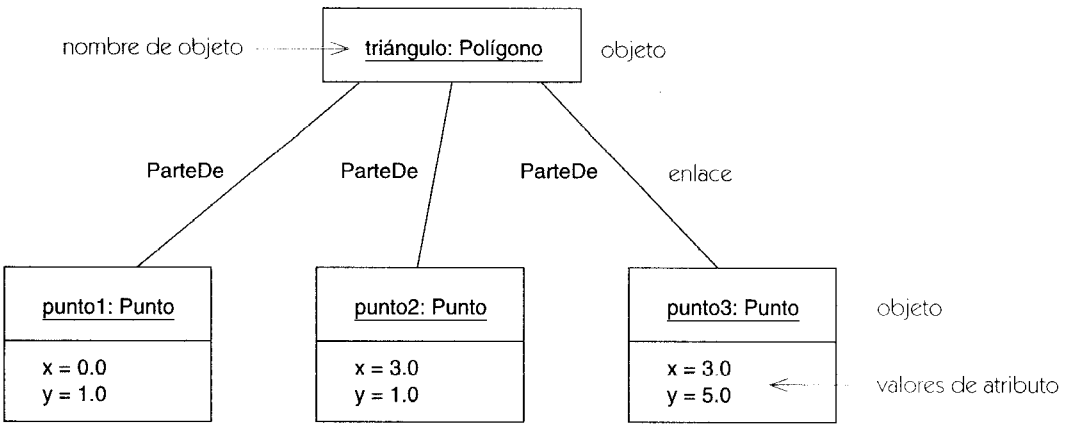


Figura 4.13 Diagrama de objetos

5 La vista de casos de usos

Descripción

La vista de casos de uso captura el comportamiento de un sistema, de un subsistema, o de una clase, tal como se muestra a un usuario exterior. Reparte la funcionalidad del sistema en transacciones significativas para los actores-usuarios ideales de un sistema. Las piezas de funcionalidad interactiva se llaman casos de uso. Un caso de uso describe una interacción con los actores como secuencia de mensajes entre el sistema y uno o más actores. El término actor incluye a los seres humanos, así como a otros sistemas informáticos y procesos. La Figura 5.1 muestra un diagrama de casos de uso, para una aplicación telefónica de venta por catálogo. El modelo se ha simplificado como ejemplo.

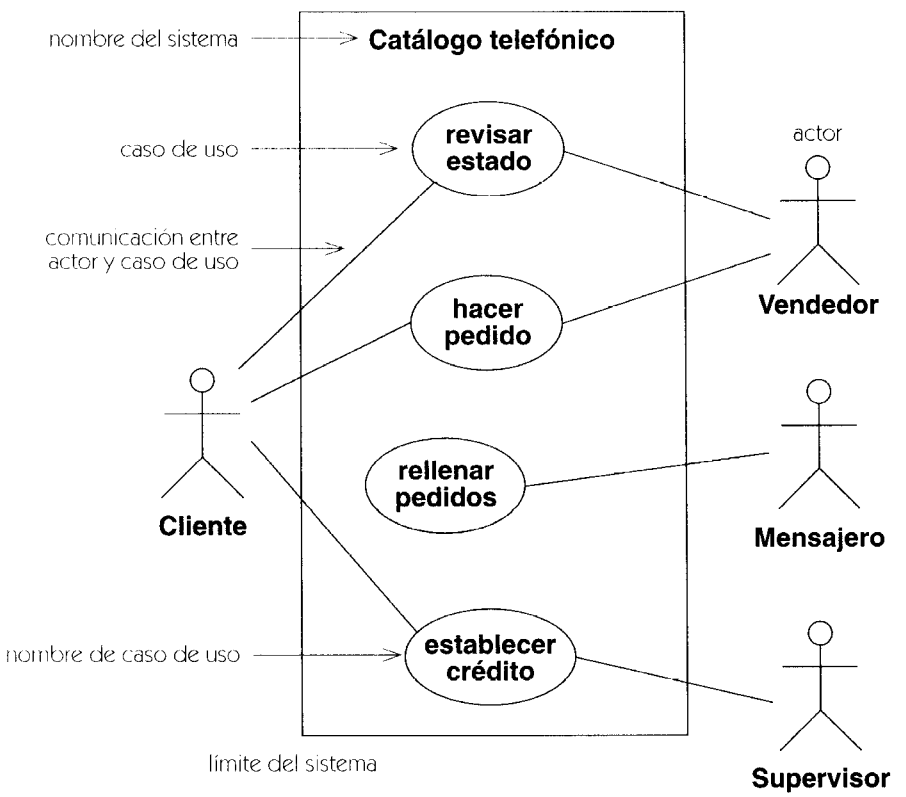


Figura 5.1 Diagrama de casos de uso

Actor

Un actor es una idealización de una persona externa, de un proceso, o de una cosa que interactúa con un sistema, un subsistema, o una clase. Un actor caracteriza las interacciones que los usuarios exteriores pueden tener con el sistema. En tiempo de ejecución, un usuario físico puede estar limitado a los actores múltiples dentro del sistema. Diferentes usuarios pueden estar ligados al mismo actor y por lo tanto pueden representar casos múltiples de la misma definición de actor.

Cada actor participa en uno o más casos de uso. Interactúa con el caso de uso (y por lo tanto con el sistema o la clase que posee el caso de uso), intercambiando mensajes. La implementación interna de un actor no es relevante en el caso de uso; un actor puede ser caracterizado suficientemente por un conjunto de atributos que definen su estado.

Los actores pueden ser definidos en jerarquías de generalización, en las cuales una descripción abstracta del actor es compartida y aumentada por una o más descripciones específicas del actor.

Un actor puede ser un ser humano, otro sistema informático, o un cierto proceso ejecutable.

Se dibuja a un actor como una persona pequeña con trazos lineales y el nombre debajo de él.

Caso de uso

Un caso de uso es una unidad coherente de funcionalidad, externamente visible, proporcionada por una unidad del sistema y expresada por secuencias de mensajes intercambiados por la unidad del sistema y uno o más actores. El propósito de un caso de uso es definir una pieza de comportamiento coherente, sin revelar la estructura interna del sistema. La definición de un caso de uso incluye todo el comportamiento que implica: las líneas principales, las diferentes variaciones sobre el comportamiento normal, y todas las condiciones excepcionales, que pueden ocurrir con tal comportamiento, junto con la respuesta deseada. Desde el punto de vista de los usuarios, éstas pueden ser situaciones anormales. Desde el punto de vista de los sistemas, son las variaciones adicionales que deben ser descritas y manejadas.

En el modelo, la ejecución de cada caso de uso es independiente de las demás, aunque una implementación de casos de uso puede crear dependencias implícitas entre ellas, debido a los objetos compartidos. Cada caso de uso representa una pieza ortogonal de la funcionalidad, cuya ejecución se puede mezclar con la ejecución de otros casos de uso.

La dinámica de un caso de uso se puede especificar por las interacciones de UML, mostradas como diagramas de estado, diagramas de secuencia, diagramas de colaboración, o descripciones informales de texto. Cuando se implementan, los casos de uso son realizados mediante colaboraciones entre clases del sistema. Una clase puede participar en múltiples colaboraciones y, por lo tanto, en múltiples casos de uso.

En el nivel de sistema, los casos de uso representan el comportamiento externo de todo sistema tal y como se muestra a los usuarios exteriores. Un caso de uso es como una operación de sistema, una operación invocable por un usuario exterior. Sin embargo, a diferencia de una operación, un caso de uso puede continuar recibiendo la entrada de sus actores durante su ejecución. Los casos de uso también se pueden aplicar internamente a unidades más pequeñas de un sistema, tales como subsistemas y clases individuales. Un caso interno de uso representa el comportamiento que una parte del sistema presenta al resto del sistema. Por ejemplo, un caso

de uso para una clase representa una pieza coherente de la funcionalidad que una clase proporciona a otras clases que desempeñen ciertos roles dentro del sistema. Una clase puede tener más de un caso de uso.

Un caso de uso es una descripción lógica de una parte de funcionalidad del sistema. No es una construcción manifiesta en la implementación de un sistema. En su lugar, cada caso de uso se debe corresponder con las clases que implementan un sistema. El comportamiento del caso de uso se corresponde con las transiciones y operaciones de las clases. Ya que una clase puede desempeñar roles múltiples en la implementación de un sistema, puede por lo tanto realizar porciones de múltiples casos de uso. Una parte de la tarea del diseño es encontrar las clases de implementación que combinen claramente los roles apropiados para implementar todos los casos de uso, sin introducir complicaciones innecesarias. La implementación de un caso de uso se puede modelar como un conjunto de una o más colaboraciones. Una colaboración es una realización de un caso de uso.

Un caso de uso puede participar en varias relaciones, además de poderse asociar con actores (Tabla 5.1).

Un caso de uso se dibuja como una elipse con su nombre dentro o debajo de ella. Se conecta por líneas con trazo continuo con los actores que se comunican con ella.

Aunque cada instancia de un caso de uso es independiente, la descripción de un caso de uso se puede descomponer en factores de otros casos de uso más simples. Esto es similar a la manera en que la descripción de una clase se puede definir incrementalmente a partir de la descripción de las superclases. Un caso de uso puede incorporar simplemente el comportamiento de otros casos de uso como fragmentos de su propio comportamiento. Esto se llama relación de inclusión. En este caso, el nuevo caso de uso no es un caso especial del caso de uso original, y no se puede sustituir por él.

Un caso de uso se puede también definir como una extensión incremental de un caso de uso base. Esto se llama relación de extensión. Puede haber varias extensiones del mismo caso de uso base, que pueden ser aplicadas conjuntamente. Las extensiones a un caso de uso base se agregan a su semántica; que es el caso de uso base instanciado, no los casos de uso de la extensión.

Las relaciones de inclusión y extensión se dibujan como flechas de líneas discontinuas con la palabra clave «**include**» o «**extend**». La relación de inclusión apunta al caso de uso a ser incluido; la relación de extensión señala el caso de uso que se extenderá.

Tabla 5.1 Tipos de Relaciones de Casos de Uso

Relación	*Función*	*Notación*
asociación	La línea de comunicación entre un actor y un caso de uso en el que participa	———
extensión	La inserción de comportamiento adicional en un caso de uso base que no tiene conocimiento sobre él	«**extend**» - - - >
generalización de casos de uso	Una relación entre un caso de uso general y un caso de uso más específico, que hereda y añade propiedades a aquél	——▷
inclusión	Inserción de comportamiento adicional en un caso de uso base, que describe explícitamente la inserción	«**include**» - - - >

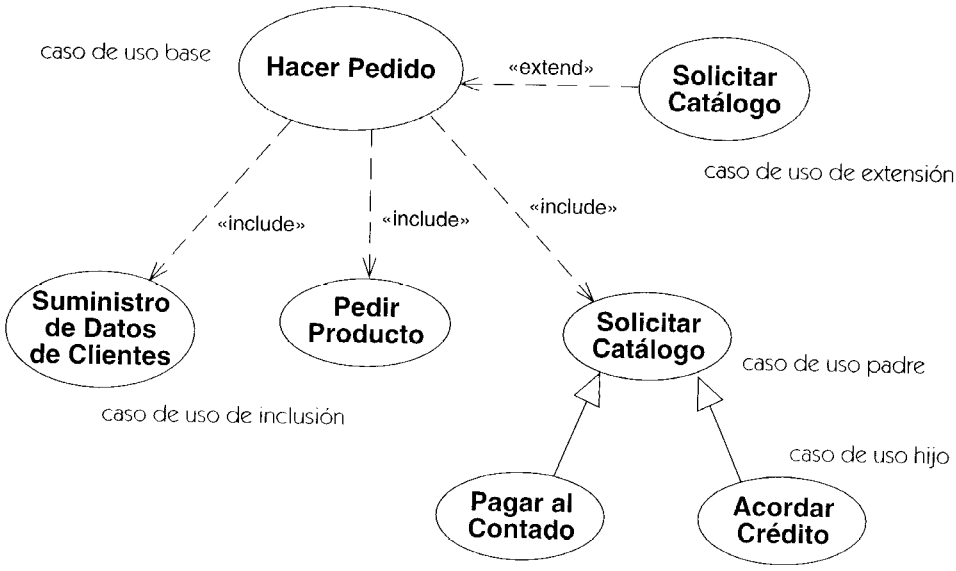


Figura 5.2 Relaciones de casos de uso

Un caso de uso también se puede especializar en uno o más casos de uso hijos. Ésta es la generalización de casos de uso. Cualquier caso de uso hijo se puede utilizar en una situación en la cual se espera el caso de uso padre.

La generalización de casos de uso se dibuja igual que cualquier generalización, utilizando una línea desde el caso de uso hijo al caso de uso padre con una punta de flecha triangular grande en el extremo del padre. La Figura 5.2 muestra las relaciones de casos de uso en la aplicación de venta por catálogo.

6 La vista de la máquina de estados

Descripción

La vista de la máquina de estados describe el comportamiento dinámico de los objetos, en un cierto plazo, modelando los ciclos de vida de los objetos de cada clase. Cada objeto se trata como una entidad aislada que se comunica con el resto del mundo detectando eventos y respondiendo a ellos. Los eventos representan las clases de cambios que un objeto puede detectar: la recepción de llamadas o señales explícitas desde un objeto a otro, un cambio en ciertos valores, o el paso del tiempo. Cualquier cosa que pueda afectar a un objeto se puede caracterizar como evento. Los sucesos del mundo real se modelan como señales del mundo exterior al sistema.

Un estado es un conjunto de los valores de un objeto para una clase dada, que tienen la misma respuesta cualitativa a los eventos que ocurren. Es decir, todos los objetos con el mismo estado reaccionan de la misma manera general a un evento, así que todos los objetos en un estado dado ejecutan la misma acción cuando reciben el mismo evento. Sin embargo, los objetos en diferentes estados, pueden reaccionar de diferente forma al mismo evento, realizando distintas acciones. Por ejemplo, una máquina de cajero automático reacciona de una forma al botón de cancelar cuando está procesando una transacción y de otra manera cuando está inactiva.

Las máquinas de estados describen el comportamiento de las clases, pero también describen el comportamiento dinámico de los casos de uso, de las colaboraciones, y de los métodos. Para uno de estos objetos, un estado representa un paso en su ejecución. Cuando hablamos de las máquinas de estado como descripción lo hacemos, la mayoría de las veces, en términos de clases y objetos, pero pueden ser aplicadas a otros elementos de manera directa.

Máquina de estados

Una máquina de estados es un gráfico de estados y de transiciones. Una máquina de estados se une a una clase y describe, generalmente, la respuesta de una instancia de la clase, a los eventos que recibe. Las máquinas de estados también se pueden unir a las operaciones, a los casos de uso, y a las colaboraciones para describir su ejecución.

Una máquina de estados es un modelo de todas las posibles historias de vida de un objeto de una clase. El objeto se examina aisladamente. Cualquier influencia externa del resto del mundo se resume como evento. Cuando el objeto detecta un evento, responde de una manera que depende de su estado actual. La respuesta puede incluir la ejecución de una acción y un cambio a un nuevo estado. Las máquinas de estados se pueden estructurar para heredar transiciones, y pueden modelar la concurrencia.

Una máquina de estados es una vista localizada de un objeto, una vista que lo separe del resto del mundo y examine su comportamiento aislado. Es una vista reduccionista de un sistema. Es una buena manera de especificar un comportamiento exacto, pero no es, a menudo, una buena manera de entender el funcionamiento total de un sistema. Para una idea más amplia de los efectos del comportamiento de un sistema, son más útiles las vistas de interacción. Sin embargo, las máquinas de estados son útiles para entender los mecanismos de control, tales como interfaces de usuario o controladores de dispositivo.

Evento

Un evento es una ocurrencia significativa que tiene una localización en tiempo y espacio. Ocurre en un punto en el tiempo y no tiene duración. Modela algo como un evento si su ocurrencia tiene consecuencias. Cuando utilizamos la palabra evento por sí mismo, queremos decir generalmente que es un descriptor, es decir una descripción de todas las ocurrencias individuales del evento, que tienen la misma forma general, del mismo modo que la palabra clase expresa todos los objetos individuales, que tienen la misma estructura. Una ocurrencia específica de un evento se llama instancia del evento. Los eventos pueden tener parámetros que caractericen cada instancia individual del evento, de la misma forma que las clases tienen atributos que caracterizan cada objeto. Como con las clases, las señales se pueden ordenar en jerarquías de generalización, para compartir la estructura común. Los eventos se pueden dividir en varios tipos explícitos e implícitos: eventos de señal, eventos de llamada, eventos de cambio y eventos de tiempo. La Tabla 6.1 es una lista de los tipos de eventos y sus descripciones.

Tabla 6.1 Tipos de Eventos

Tipo de evento	*Descripción*	*Sintaxis*
evento de llamada	Recepción de una petición explícita síncrona entre objetos que esperan por una respuesta	op (a:T)
evento de cambio	Un cambio en el valor de una expresión Booleana	**cuando** (expresión)
evento de señal	Recepción de una comunicación asíncrona explícita y con nombre entre objetos	nombreS (a:T)
evento de tiempo	La conclusión de un tiempo absoluto o el transcurso de una cantidad relativa de tiempo	**tras** (tiempo)

Evento de la señal. Una señal es una entidad con nombre, que se piensa explícitamente como vehículo de comunicación entre dos objetos; la recepción de una señal es un evento para el objeto receptor. El objeto que envía, crea explícitamente e inicializa una instancia de la señal y la envía a uno o a un conjunto de objetos explícitos. Las señales incorporan la comunicación unidireccional asíncrona, el tipo más importante. El remitente no espera a que el receptor se ocupe de la señal, sino que continúa con su propio trabajo independientemente. Para modelar la comunicación bidireccional se pueden utilizar varias señales, por lo menos una en cada dirección. El remitente y el receptor pueden ser el mismo objeto.

Las señales se pueden declarar en diagramas de clase como clasificadores, usando la palabra clave «**signal/señal**»[1]; los parámetros de la señal se declaran como atributos. Como clasificadores, las señales pueden tener relaciones de generalización. Las señales pueden ser hijas de

[1] Véase nota al pie pág. **51.**

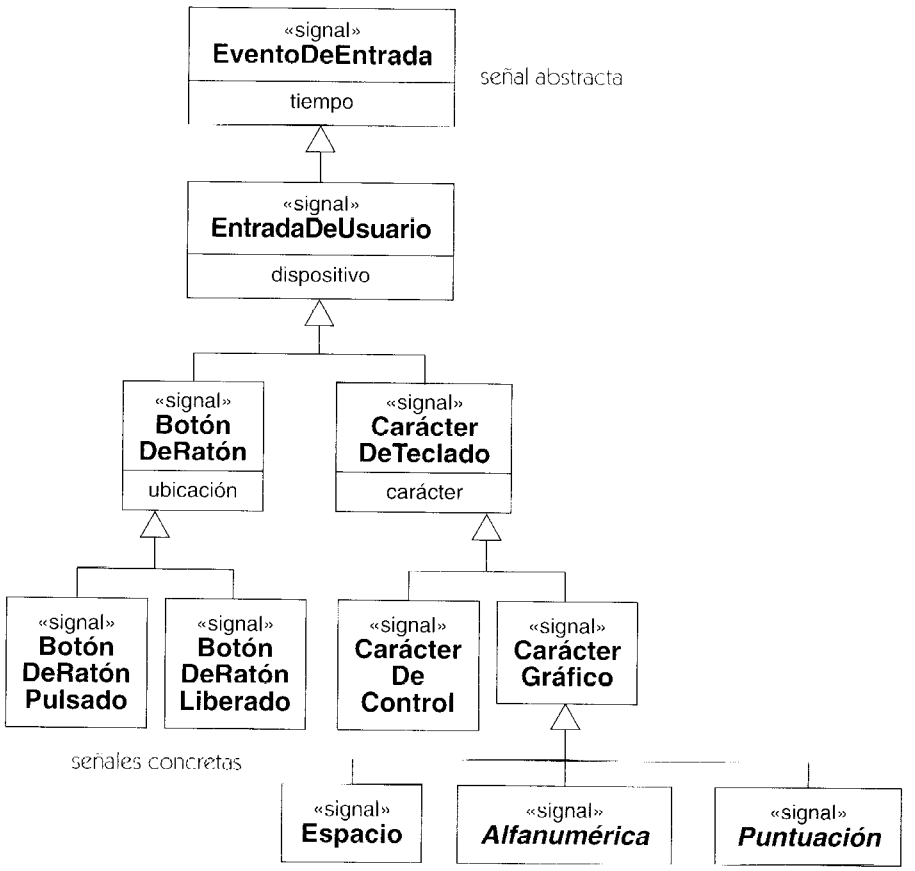


Figura 6.1 Jerarquía de señales

otras señales; heredando los parámetros de sus padres, y accionando las transiciones que dependen de la señal del padre (Figura 6.1).

Evento de llamada. Un evento de llamada es la recepción de una llamada por un objeto que elige poner una operación en ejecución, como transición de la máquina de estados en vez de como procedimiento fijo. Para el llamador, una llamada ordinaria (implementada por un método) es indistinguible de un evento de llamada. El receptor elige si una operación será implementada como un método o como disparador de un evento de llamada en una máquina de estados. Los parámetros de la operación son los parámetros del evento. Una vez que el objeto receptor procesa el evento de llamada tomando una transición accionada por el evento, o no pudiendo tomar ninguna transición, el control retorna al objeto llamador.

Sin embargo, a diferencia de una llamada ordinaria, el receptor de un evento de llamada puede continuar su propia ejecución en paralelo al llamador.

Evento de cambio. Un evento de cambio es la satisfacción de una expresión booleana que dependa de ciertos valores de un atributo. Esto es una manera declarativa de esperar hasta que una condición esté satisfecha, pero debe ser utilizada con cuidado, porque representa un cómputo continuo y potencialmente no local (acción en una distancia, porque el valor o los valores comprobados pueden ser distantes). Esto es bueno y malo. Es bueno porque centra el modelo en

la verdadera dependencia —un efecto que ocurre cuando se satisface una condición dada— y no en la mecánica de comprobación de condición. Es malo porque oscurece la relación causa-efecto entre la acción que cambia un valor subyacente y el efecto eventual. El coste de comprobar un evento de cambio es potencialmente grande, porque en principio es continuo. Sin embargo, hay en la práctica maneras de evitar el cómputo innecesario. Los eventos de cambio deben ser utilizados solamente cuando sea artificial una forma más explícita de comunicación.

Observe la diferencia entre una condición de guarda y un evento de cambio. Una condición de guarda se evalúa una vez cuando ocurre el evento disparador en la transición y el receptor maneja el evento. Si es falsa, la transición no se activa y la condición no se evalúa de nuevo. Un evento de cambio se evalúa continuamente hasta que llega a ser verdad, en cuyo caso se dispara la transición.

Evento de tiempo. Los eventos de tiempo representan el paso del tiempo. Un evento de tiempo se puede especificar de modo absoluto (hora) o de modo relativo (el tiempo transcurrió desde un evento dado). En un modelo de alto nivel, los eventos de tiempo se pueden pensar como eventos del universo; en un modelo de implementación, son causados por las señales de un cierto objeto específico, el sistema operativo o un objeto en la aplicación.

Estado

Un estado describe un período de tiempo durante la vida de un objeto de una clase. Puede ser caracterizado de tres maneras complementarias: como un conjunto de valores de objeto cualitativamente similares en cierta forma; como período de tiempo, durante el cual un objeto espera que ocurra algún evento o eventos; o como período de tiempo, durante el cual un objeto realiza una cierta actividad. Un estado puede tener un nombre, aunque a menudo es anónimo y viene descrito simplemente por sus acciones.

En una máquina de estados, un conjunto de estados está conectado mediante transiciones. Aunque las transiciones conectan dos estados (o más, si hay una división o unión de control), las transiciones son procesadas por el estado del que salen. Cuando un objeto está en un estado, es sensible a los eventos correspondientes a las transiciones que salen del estado.

Figura 6.2 Estado

Un estado se muestra como rectángulo con las esquinas redondeadas (Figura 6.2).

Transición

Una transición que deja un estado define la respuesta de un objeto en ese estado a la ocurrencia de un evento. En general, una transición tiene un evento que la activa (evento disparador), una condición de guarda, una acción, y un estado destino. La Tabla 6.2 muestra los tipos de las transiciones y de las acciones implícitas invocadas por transiciones.

Transición externa. Una transición externa es una transición que cambia el estado activo. Éste es el tipo más común de transición. Se dibuja como una flecha desde el estado origen al estado destino, con otras propiedades mostradas como una cadena de texto asociada a la flecha (Figura 6.3).

Tabla 6.2 Tipos de transiciones y acciones implícitas

Tipo de transición	*Descripción*	*Sintaxis*
acción de entrada	Una acción que se ejecuta cuando se entra en el estado	**entry/** acción **o** **entrada/** acción
acción de salida	Una acción que se ejecuta cuando se sale del estado	**exit/** acción **o** **Salida/** acción
transición externa	Una respuesta a un evento que causa un cambio de estado o una transición a sí mismo, junto con la acción especificada. También puede provocar la ejecución de la acción de entrada o salida en el caso de que haya estados en los que haya acciones asociadas a la entrada o a la salida	e(a:T)[exp]/acción
transición interna	Una respuesta a un evento que causa la ejecución de una acción pero no causa cambio de estado o ejecución de acciones de salida o entrada	e(a:T)[exp]/acción

Evento disparador. El disparador es un evento cuya ocurrencia permite la transición. El evento puede tener parámetros, que estarán disponibles para una acción en la transición. Si una señal tiene descendientes, cualquier descendiente de la señal permite la transición. Por ejemplo, si una transición tiene **BotónDeRatón**, como disparador, (véase la Figura 6.1), entonces **BotónDeRatónPulsado** accionará la transición.

Un evento no es algo continuo; ocurre en un punto en el tiempo. Cuando un objeto recibe un evento, guarda el evento si no es capaz de manejarlo. Un objeto maneja un evento a la vez. Debe activarse una transición cuando el objeto maneja el evento; el evento no es "recordado" hasta más adelante (excepto en el caso especial de los eventos diferidos, que se guardan hasta que accionan una transición o hasta que el objeto está en un estado donde no se difieren). Si ocurren

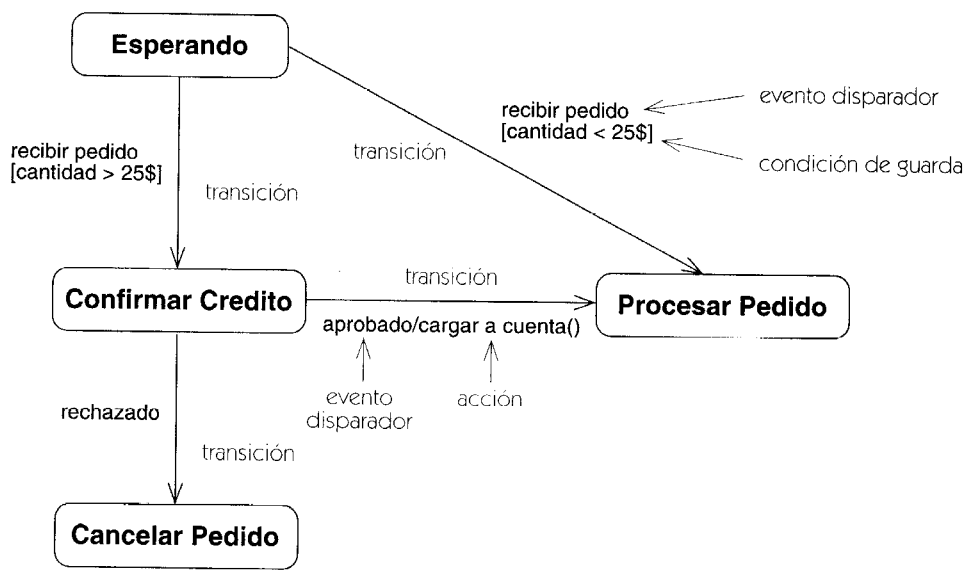


Figura 6.3 Transiciones externas

dos eventos simultáneamente, se manejan a la vez. Un evento que no acciona ninguna transición es simplemente ignorado y se pierde. Esto no es un error. Es mucho más fácil no hacer caso de los eventos no deseados que intentar especificarlos todos.

Condición de guarda. Una transición puede tener una condición de guarda, que es una expresión booleana. Puede referirse a los atributos del objeto que posee la máquina de estados, así como a los parámetros del evento del disparador. Se evalúa la condición de guarda cuando ocurre un evento disparador. Si la expresión se evalúa como verdadera, entonces se dispara la transición, es decir, ocurren sus efectos. Si la expresión se evalúa como falsa, entonces la transición no se dispara.

La condición de guarda se evalúa solamente una vez: cuando ocurre el evento disparador. Si la condición es falsa y llega a ser más adelante verdad, es demasiado tarde para disparar la transición.

El mismo evento puede ser un disparador para más de una transición que parte de un estado simple. Cada transición con el mismo evento debe tener una condición de guarda diferente. Si ocurre el evento, una transición accionada por el evento puede accionarse si su condición es verdadera. A menudo, el conjunto de condiciones de guarda cubre todas las posibilidades, de tal forma que está garantizado que la ocurrencia de un evento dispara una cierta transición. Si no se cubren todas las posibilidades y no se permite ninguna transición, entonces un evento simplemente se ignora. Una transición únicamente puede dispararse (dentro de un hilo de control) en respuesta a una ocurrencia de un evento. Si un evento permite más de una transición, solamente se dispara una de ellas. Una transición en un estado anidado tiene preferencia sobre una transición de uno de los estados que la incluyen. Si dos transiciones que están en conflicto se activan al mismo tiempo, alguna de ellas se dispara, pero queda indeterminada cuál. La opción puede ser aleatoria o puede depender de los detalles de la implementación, pero el modelador no debe contar con un resultado predecible.

Transición de finalización. Una transición que carece de evento disparador explícito es accionada por la terminación de la actividad del estado del que sale (esto es una transición de terminación). Una transición de terminación puede tener una condición de guarda, que se evalúa en el momento en que termina la actividad en ese estado (y no después).

Acción. Cuando se dispara una transición, su acción (si la hay), es ejecutada. Una acción es un cómputo atómico y normalmente breve, a menudo una declaración de asignación o un cómputo aritmético simple. Otras acciones incluyen enviar una señal a otro objeto, llamar a una operación, fijar los valores de retorno, crear o destruir un objeto, y las acciones indefinidas de control especificadas en un lenguaje externo. Una acción puede también ser una secuencia de acciones, es decir, una lista de acciones más simples. Una acción o una secuencia de acciones no se puede terminar o ser afectada por acciones simultáneas. Conceptualmente, su duración es insignificante comparada con la sincronización exterior del evento; por lo tanto, un segundo evento no puede ocurrir durante su ejecución. En la práctica, sin embargo, las acciones toman un cierto tiempo, y los eventos entrantes se deben poner en una cola.

El sistema total puede realizar múltiples acciones simultáneamente. Cuando decimos acciones atómicas, no implica que todo el sistema sea atómico. El sistema puede procesar interrupciones hardware y compartir el tiempo entre varias acciones. Una acción es atómica dentro de su propio hilo de control. Una vez que ha comenzado, debe terminar y no debe interactuar con otras acciones simultáneamente activas. Pero las acciones no se deben utilizar como mecanismos de transición. Su duración debe ser breve comparada con el tiempo de reacción necesario para los eventos externos. De lo contrario, el sistema puede no responder de manera oportuna.

Tabla 6.3 Tipos de acciones

Tipo de acción	*Descripción*	*Sintaxis*[2]
asignación	Asigna el valor de una variable	destino := expresión
creación	Crea un nuevo objeto	**new** NombreClase(arg,arg)
destrucción	Destruye un objeto	objeto.**destroy**()
enviar	Crea una instancia de una señal y la envía a un objeto destino o conjunto de objetos	Nombres(arg,arg)
llamada	Llama una operación en un objeto destino y espera la terminación de la ejecución de la operación. Puede devolver un valor	nombreop (arg,arg)
no interpretada	Acción específica del lenguaje, tal como un bucle o condición	[específico del lenguaje]
retorno	Especifica los valores a devolver al llamador	**return** valor
terminación	Autodestrucción del objeto	**terminate**

Una acción puede utilizar los parámetros del evento disparador y los atributos del objeto poseedor como parte de su expresión.

La Tabla 6.3 enumera las clases de acciones y sus descripciones.

Cambio de estado. Cuando se completa la ejecución de una acción, el estado destino de la transición pasa a ser el estado activo. Esto puede activar acciones de salida y acciones de entrada.

Estados anidados. Los estados se pueden anidar dentro de otros estados compuestos (véase el punto siguiente). Una transición que deja el estado más externo es aplicable a todos los estados anidados dentro de él. La transición es elegible para ser disparada siempre que cualquier estado anidado esté activo. Si se dispara, el estado destino de la transición pasa a ser el estado activo. Los estados compuestos son útiles para expresar condiciones de excepción y de error, porque las transiciones en ellos se aplican a todos los estados anidados sin necesidad de que cada estado anidado maneje la excepción explícitamente.

Acciones de entrada y salida. Una transición a través de uno o más niveles de anidamiento puede salir y entrar en estados. Un estado puede tener acciones que se realicen siempre que se entre o se salga del estado. Entrando al estado destino, se ejecuta una acción de entrada unida al estado. Si la transición sale del estado original, entonces su acción de salida se ejecuta antes de la acción, de la transición y de la acción de entrada en el nuevo estado.

Las acciones de entrada se utilizan, a menudo, para realizar la configuración necesaria dentro de un estado. Debido a que una acción de entrada no puede ser evadida, cualquier acción que ocurra dentro del estado, puede asumir que la configuración ha ocurrido, sin importar cómo se entre en el estado.

De manera similar, una acción de salida es una acción que ocurre siempre que se salga del estado, una oportunidad de realizar una limpieza. Es particularmente útil cuando hay transiciones

[2] Salvo si las herramientas usadas necesitan el uso de los términos ingleses, pueden utilizarse los términos traducidos: nuevo, destruir, devolver, terminar. (*N. de los R.T.*)

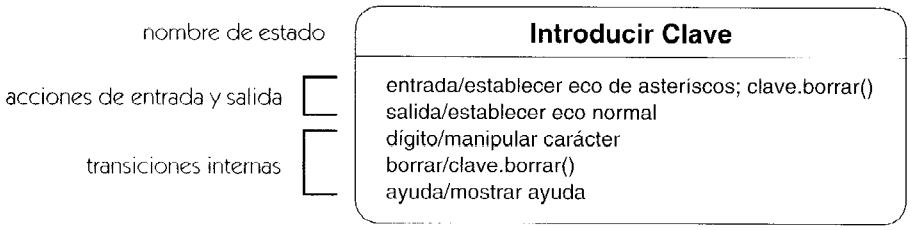


Figura 6.4 Transiciones internas y acciones de entrada y salida

de alto nivel que representan las condiciones de error que abortan estados anidados. La acción de salida puede limpiar tales casos de modo que el estado del objeto siga siendo consistente. Las acciones de entrada y de salida se podrían, en principio, dar como transiciones entrantes y salientes, pero el declararlas como acciones especiales del estado, permite que el estado sea definido independientemente de sus transiciones y por lo tanto esté encapsulado.

Transición interna. Una transición interna tiene un estado origen pero ningún estado destino. Las reglas de disparo para una transición interna son iguales que para una transición que cambie el estado. Una transición interna no tiene ningún estado destino, así que el estado activo no cambia como resultado de su activación. Si una transición interna tiene una acción, se ejecuta, pero no ocurre ningún cambio de estado, y por lo tanto no se ejecuta ninguna de las acciones de salida o de entrada. Las transiciones internas son útiles para modelar las acciones de interrupción que no cambian el estado (como recuento de ocurrencias de un evento o desplegar una pantalla de ayuda).

Las acciones de entrada y de salida utilizan la misma notación que las transiciones internas, pero usan las palabras reservadas **entry** y **exit** (**entrada** y **salida**) en lugar del nombre del evento disparado, aunque estas acciones son activadas por las transiciones externas que entran o salen del estado.

Una transición a sí mismo invoca acciones de salida y de entrada en su estado (conceptualmente, sale y entonces entra al estado de nuevo); por lo tanto, no es equivalente a una transición interna. La Figura 6.4 muestra acciones de entrada y salida así como transiciones internas.

Estados compuestos

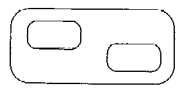
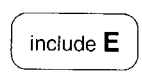
Un estado simple no tiene ninguna estructura, apenas tiene un conjunto de transiciones y posiblemente acciones de entrada y salida. Un estado compuesto es un estado que se ha descompuesto en subestados secuenciales o subestados concurrentes. La Tabla 6.4 enumera las distintas clases de estados.

Una descomposición en subestados disjuntos es un tipo de especialización de un estado.

Un estado externo se refina en varios estados internos, cada uno de los cuales hereda las transiciones del estado más externo. Únicamente puede ser activo a la vez un subestado secuencial. El estado externo representa la condición de estar en cualquiera de los estados internos.

Las transiciones hacia dentro o hacia fuera de un estado compuesto, invocan las acciones de entrada o de salida del estado. Si hay varios estados compuestos, una transición a través de varios niveles puede invocar múltiples acciones de entrada (primero las más exteriores) o de salida (primero las más internas).

Tabla 6.4 Tipos de Estado

Tipo de estado	*Descripción*	*Notación*
estado simple	Un estado sin estructura	
estado concurrente compuesto	Un estado que está dividido en dos o más estados concurrentes, todos los cuales están activos concurrentemente cuando el estado compuesto está activo	
estado secuencial compuesto	Un estado que contiene uno o más subestados disjuntos, exactamente uno de los cuales está activo en cada momento cuando el estado compuesto está activo	
estado inicial	Un pseudoestado que indica el estado inicial cuando es invocado el estado que lo engloba	
estado final	Un estado especial cuya activación indica que el estado que lo engloba ha completado su actividad	
estado de unión o conjunción	Un pseudoestado que encadena segmentos de transición en una sola transición atómica	
estado de historia	Un pseudoestado cuya activación restaura el estado activado previamente dentro de un estado compuesto	
estado de referencia a submáquina	Un estado que referencia a una submáquina, la cual está implícitamente insertada en el lugar del estado de referencia a submáquina	
estado abreviado externo	Un pseudoestado, dentro de un estado de referencia a submáquina, que identifica un estado en la referida	

Si hay una acción en la transición a sí mismo, la acción se ejecuta después de cualquier acción de salida y antes de que se ejecute cualquier acción de entrada.

Un estado compuesto también puede tener un estado inicial dentro de él. Una transición al límite del estado compuesto es implícitamente una transición al estado inicial. Un nuevo objeto comienza en su estado inicial de su estado más exterior. De la misma forma, un estado compuesto puede tener un estado final. Una transición al estado final acciona una transición de terminación (transición sin disparador) en el estado compuesto. Si un objeto alcanza el estado final de su estado más exterior, se destruye. Los estados iniciales, los estados finales, las acciones de entrada, y las acciones de salida permiten que la definición de un estado sea encapsulada independiente de las transiciones a y desde él.

La Figura 6.5 muestra una descomposición secuencial de un estado, incluyendo un estado inicial. Éste es el control para una máquina de venta de entradas.

Una descomposición en subestados concurrentes representa cómputos independientes. Cuando se entra en un superestado concurrente, el número de hilos de control aumenta. Cuando se

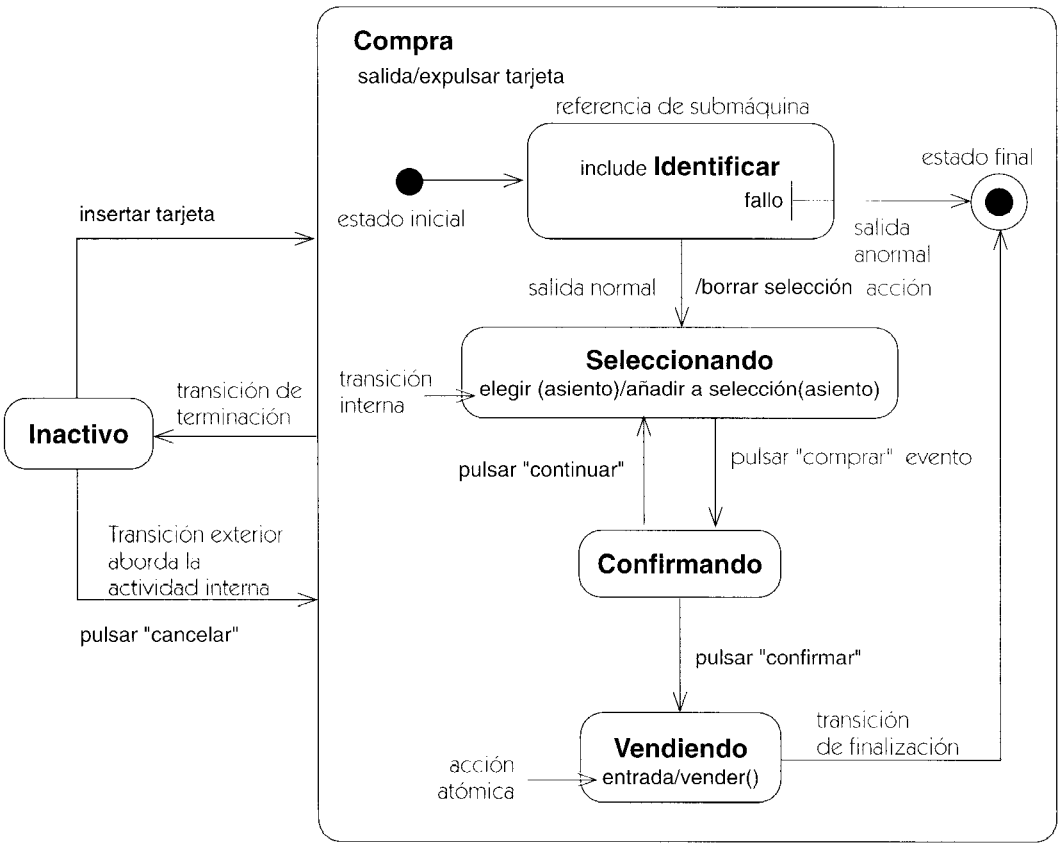


Figura 6.5 Máquina de estados

sale, el número de hilos de control disminuye. A menudo, la concurrencia es implementada por un objeto distinto para cada subestado, pero los subestados concurrentes pueden también representar concurrencia lógica dentro de un objeto. La Figura 6.6 muestra la descomposición concurrente de tomar una clase en la universidad.

A menudo, es conveniente reutilizar un fragmento de una máquina de estados en otras máquinas de estado. A una máquina de estados se le puede dar un nombre y hacer referencia a ella desde un estado de una o más máquinas. La máquina de estados de destino es una submáquina, y el estado que se refiere a él, se llama subestado de referencia a submáquina. Implica la sustitución (conceptual) de una copia de la máquina referida del estado en el lugar de la referencia, por una clase de subrutina de una máquina de estados. En vez de una submáquina, un estado puede contener una actividad, es decir, un cómputo o una ocurrencia continua que lleve un tiempo para terminarse y que pueda ser interrumpida por eventos. La Figura 6.7 muestra una referencia a submáquina.

Una transición a un estado de referencia a submáquina causa la activación del estado inicial de la submáquina destino. Para entrar en la submáquina en otros estados, se coloca uno o más estados externos en el estado de referencia a submáquina. Un estado externo identifica un estado en la submáquina.

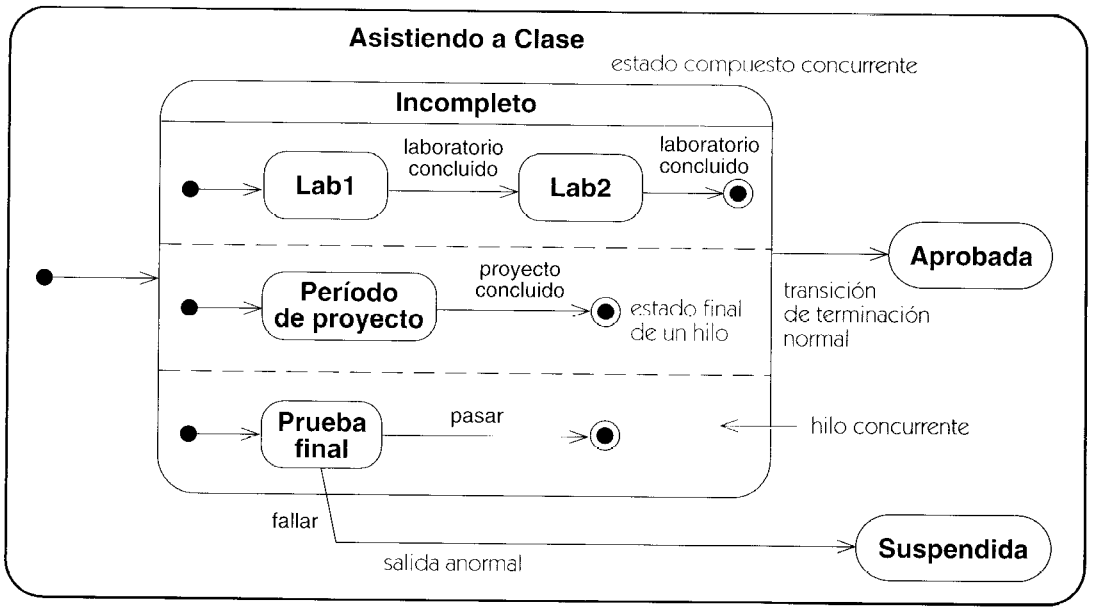


Figura 6.6 Máquina de estados con un estado compuesto concurrente

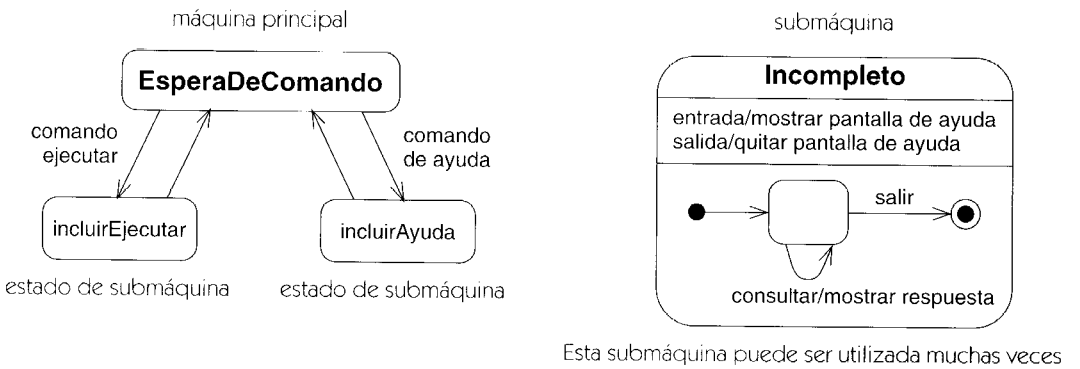


Figura 6.7 Submáquina de estados

7 La vista de actividades

Descripción

Un grafo de actividades es una forma especial de máquina de estados, prevista para modelar cómputos y flujo de trabajos. Los estados del grafo de actividades representan los estados de ejecución del cómputo, no los estados de un objeto ordinario. Normalmente, un grafo de actividades asume que los cómputos proceden sin interrupciones externas por eventos (si las hubiera puede ser preferible una máquina de estados ordinaria).

Un grafo de actividades contiene estados de actividad. Un estado de actividad representa la ejecución de una sentencia en un procedimiento, o el funcionamiento de una actividad en un flujo de trabajo. En vez de esperar un evento, como en un estado de espera normal, un estado de actividad espera la terminación de su cómputo. Cuando la actividad termina, entonces la ejecución procede al siguiente estado de actividad dentro del grafo. Una transición de terminación es activada en un diagrama de actividades cuando se completa la actividad precedente. Los estados de actividad no tienen generalmente transiciones con eventos explícitos, pero pueden ser abortados por transiciones en estados que los incluyen.

Un grafo de actividades puede también contener estados de acción, que son similares a los estados de actividad, excepto en que son atómicos y no permiten transiciones mientras están activos. Los estados de acción se deben utilizar generalmente para las operaciones cortas de mantenimiento.

Un diagrama de actividades puede contener bifurcaciones, así como divisiones de control en hilos concurrentes. Los hilos concurrentes representan actividades que se pueden realizar concurrentemente por los diversos objetos o personas en una organización. La concurrencia se presenta con frecuencia a partir de la agregación, en la cual cada objeto tiene su propio hilo concurrente. Las actividades concurrentes se pueden realizar simultáneamente o en cualquier orden. Un grafo de actividades es como un organigrama tradicional, excepto que permite control de concurrencia además de control secuencial: una gran diferencia.

Diagrama de actividades

Un diagrama de actividades es la notación para un grafo de actividades (Figura 7.1). Incluye algunos símbolos especiales abreviados por conveniencia. Estos símbolos pueden usarse en cualquier diagrama de estados, aunque mezclar la notación puede ser molesto.

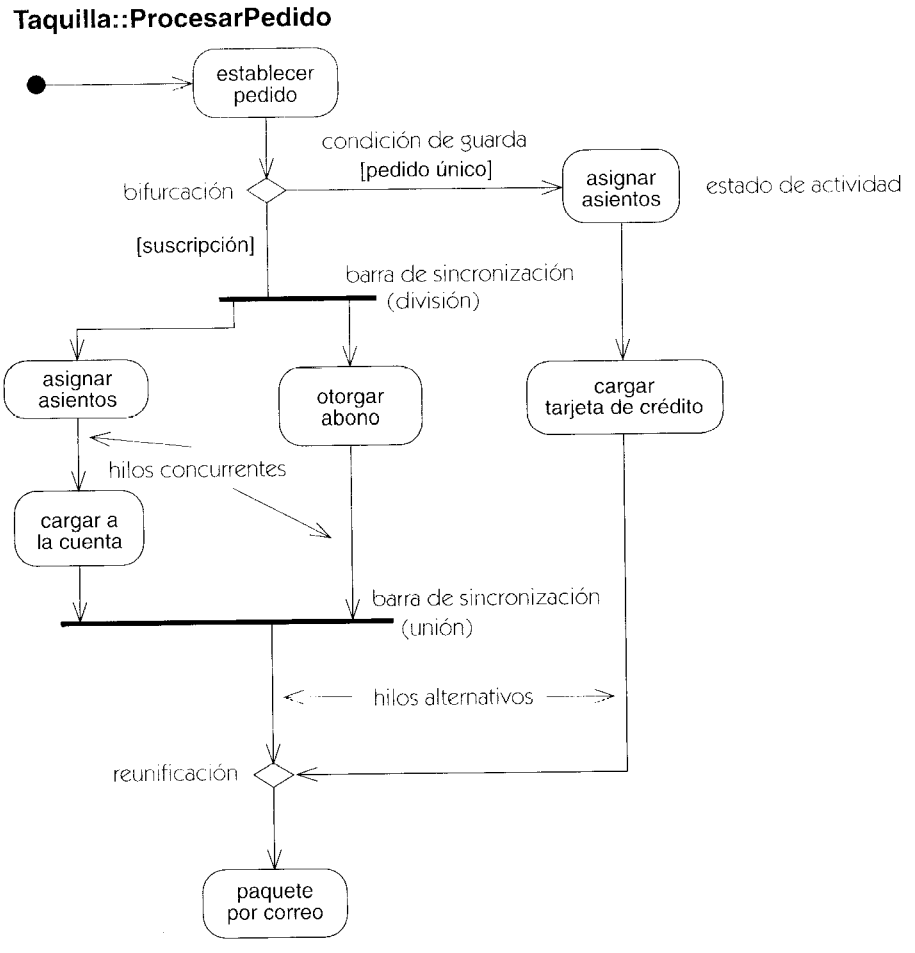


Figura 7.1 Diagrama de actividades

Un estado de actividad se representa como una caja con los extremos redondeados que contiene una descripción de actividad. (Las cajas normales del estado tienen lados rectos y esquinas redondeadas.) Las transiciones simples de terminación se muestran como flechas. Las ramas se muestran como condiciones de guarda en transiciones o como diamantes con múltiples flechas de salida etiquetadas. Una división o unión de control se representa de la misma manera que en un diagrama de estados, con múltiples flechas que entran o salen de una barra gruesa de sincronización. La Figura 7.1 muestra un diagrama de actividades para procesar un pedido por la taquilla.

Cuando es necesario incluir eventos externos, la recepción de un evento se puede mostrar como un disparador en una transición, o como un símbolo especial que denota la espera de una señal. Una notación similar muestra el envío de una señal. Sin embargo, si hay muchas transiciones dirigidas por eventos, probablemente es preferible un diagrama de estados ordinario.

Calles. A menudo es útil organizar las actividades en un modelo según su responsabilidad, por ejemplo, agrupando juntas todas las actividades manejadas por una organización del negocio. Esta clase de asignación puede mostrarse organizando las actividades en regiones dis-

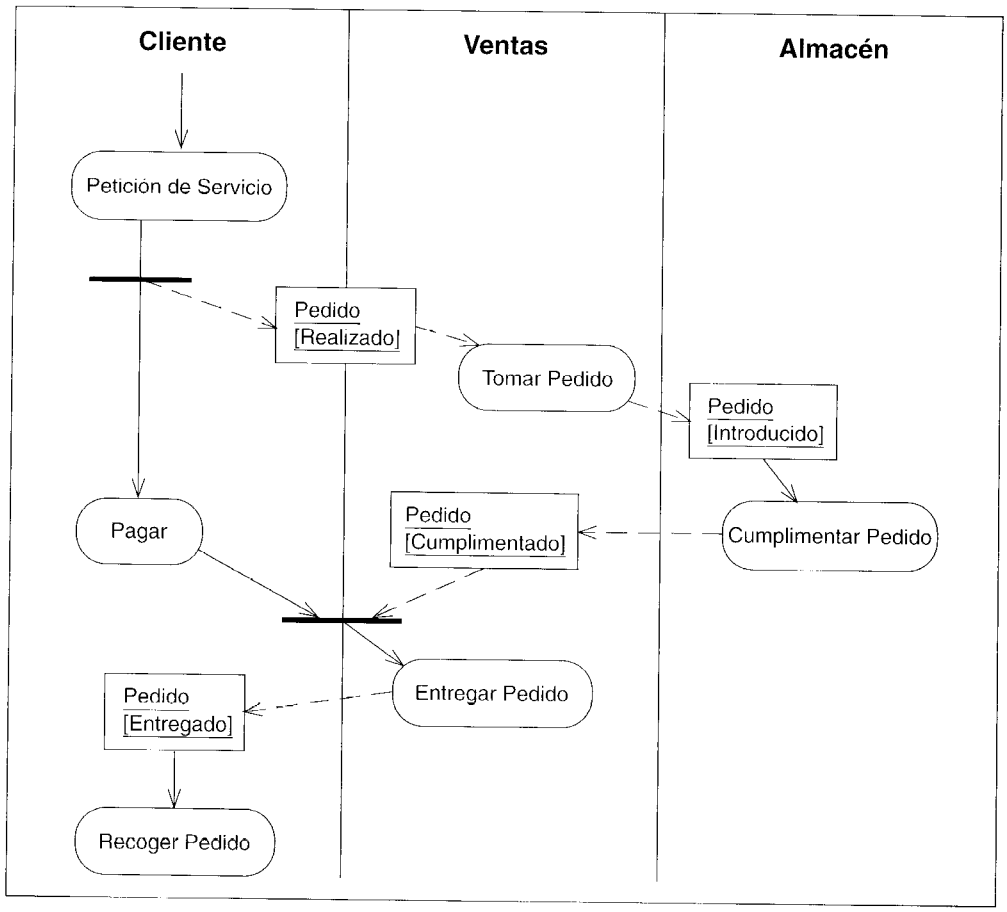


Figura 7.2 Calles y flujo de objetos

tintas separadas por líneas en el diagrama. Debido a su aspecto, cada región se llama calle[1]. La Figura 7.2 muestra las calles.

Flujo de objetos. Un diagrama de actividades puede mostrar el flujo de objetos como valores, además del flujo de objeto. Se puede representar un objeto que sea la entrada o la salida de una actividad dibujando un objeto; también el estado del objeto, para indicar su evolución o flujo, dibujando un estado de flujo de objeto. Para un valor de salida, se dibuja una flecha con línea discontinua desde una actividad al objeto. Para un valor de entrada, se dibuja una flecha con línea discontinua desde el objeto a una actividad. Si una actividad tiene más de un flujo de salida o flujo de objeto sucesor, las flechas se dibujan desde el símbolo de división. De la misma forma, las entradas múltiples se dibujan hacia un símbolo de unión.

La Figura 7.2 muestra un diagrama de actividades en el cual las actividades y los estados de flujo de un objeto se han asignado a las calles.

---

[1] Por su aspecto y sentido, se utiliza el término "calle" como en algunas pistas deportivas; franjas que indican el desplazamiento de una persona concreta por ellas. (*N. del Rev.*)

Actividades y otras vistas

Los grafo de actividades no muestran el detalle completo de un cómputo. Muestran el flujo de actividades pero no los objetos que realizan las actividades. Los grafo de actividades son un punto de partida para el diseño. Para terminar un diseño, cada actividad se debe ampliar como una o más operaciones, cada una de las cuales para ser implementada se asigna a una clase específica. Tal asignación da lugar al diseño de una colaboración que implemente el grafo de actividades.

8 La vista de interacción

Descripción

Los objetos obran recíprocamente para implementar comportamiento. Esta interacción se puede describir de dos maneras complementarias, una de ellas se centra en los objetos individuales y la otra en una colección de objetos cooperantes.

Una máquina de estados es una vista estrecha y profunda del comportamiento, una vista reduccionista que mira a cada objeto individualmente. Una especificación de la máquina de estados es exacta y conduce inmediatamente al código. Sin embargo, puede ser difícil entender el funcionamiento total de un sistema, debido a que una máquina de estados se centra en un solo objeto a la vez, y se deben combinar los efectos de muchas máquinas de estado para determinar el comportamiento de todo el sistema. La vista de interacción proporciona una visión más integral del comportamiento de un sistema de objetos. Esta vista es modelada por colaboraciones.

Colaboración

Una colaboración es una descripción de una colección de objetos que interactúan para implementar un cierto comportamiento dentro de un contexto. Describe una sociedad de objetos cooperantes unidos para realizar un cierto propósito. Una colaboración contiene ranuras que son rellenadas por los objetos y enlaces en tiempo de ejecución. Una ranura de colaboración se llama rol, porque describe el propósito de un objeto o de un enlace, dentro de la colaboración. Un rol de clasificador representa una descripción de los objetos que pueden participar en una ejecución de la colaboración; un rol de asociación representa una descripción de los enlaces que pueden participar en una ejecución de la colaboración. Un rol de clasificador es un clasificador que está limitado por tomar parte en la colaboración; una rol de asociación es una asociación que está limitada por tomar parte en la colaboración.

Las relaciones entre roles de clasificador y roles de asociación dentro de una colaboración solamente son significativas en ese contexto. En general, no se aplican las mismas relaciones a los clasificadores y a las asociaciones subyacentes fuera de la colaboración.

La vista estática describe las propiedades inherentes de una clase. Por ejemplo, un **Vehículo** tiene un dueño. Una colaboración describe las propiedades que una instancia de una clase tiene porque desempeña un rol particular en una colaboración.

Por ejemplo, un **VehiculoAlquiler** en una colaboración **CocheAlquiler** tiene un **ConductorArrendatario**, algo que no es relevante a un **Vehículo** en general, pero es una parte esencial de la colaboración.

Un objeto en un sistema puede participar en más de una colaboración: Colaboraciones que parecen no estar relacionadas directamente, pero cuya ejecución está conectada a través de un objeto compartido. Por ejemplo, una persona puede ser un **ConductorArrendatario** y un **HuespedHotel** como parte de un modelo **Vacaciones**. Menos frecuentemente, un objeto puede desempeñar más de un rol en la misma colaboración.

Una colaboración tiene un aspecto estructural y un aspecto de comportamiento. El aspecto estructural es similar a una vista estática: contiene un conjunto de roles y de relaciones que definen el contexto para su comportamiento.

El comportamiento es el conjunto de mensajes intercambiados por los objetos ligados a los roles. Tal conjunto de mensajes en una colaboración se llama interacción. Una colaboración puede incluir una o más interacciones, cada una de las cuales describe una serie de mensajes intercambiados entre los objetos en la colaboración para realizar un objetivo.

Mientras que una máquina de estados es estrecha y profunda, una colaboración es amplia pero más superficial. Captura una vista más integral del comportamiento en el intercambio de mensajes dentro de una red de objetos. Las colaboraciones muestran la unidad de las tres estructuras importantes de computación subyacentes: la estructura de datos, el flujo de objeto, y flujo de datos.

Interacción

Una interacción es un conjunto de mensajes dentro de una colaboración que son intercambiados por roles de clasificador a través de roles de asociación. Cuando una colaboración existe en tiempo de ejecución, los objetos ligados a roles de clasificador intercambian instancias de mensajes a través de los enlaces ligados a los roles de asociación. Una interacción modela la ejecución de una operación, caso de uso, u otra entidad de comportamiento.

Un mensaje es una comunicación unidireccional entre dos objetos, un flujo de objeto con la información de un remitente a un receptor. Un mensaje puede tener parámetros que transporten valores entre los objetos. Un mensaje puede ser una señal (una comunicación explícita entre objetos, con nombre, y asíncrona) o una llamada (la invocación síncrona de una operación con un mecanismo para el control, que retorna posteriormente al remitente).

La creación de un nuevo objeto se modela como un evento causado por el objeto creador y recibido por la propia clase. El evento de creación está disponible para la nueva instancia como evento actual en la transición desde el estado inicial del objeto.

Los mensajes se pueden ordenar en hilos secuenciales de control. Los hilos separados representan conjuntos de mensajes concurrentes. La sincronización entre los hilos se modela mediante restricciones entre mensajes en diversos hilos. Una construcción de sincronización puede modelar divisiones del control, uniones del control, y bifurcaciones.

El orden de los mensajes se puede mostrar en dos tipos de diagramas: un diagrama de secuencia (que se centra en las secuencias de tiempo de los mensajes) y un diagrama de colaboración (que se centra en las relaciones entre los objetos que intercambian los mensajes).

Diagrama de secuencia

Un diagrama de secuencia representa una interacción como un gráfico bidimensional. La dimensión vertical es el eje de tiempo, que avanza hacia abajo de la página. La dimensión horizon-

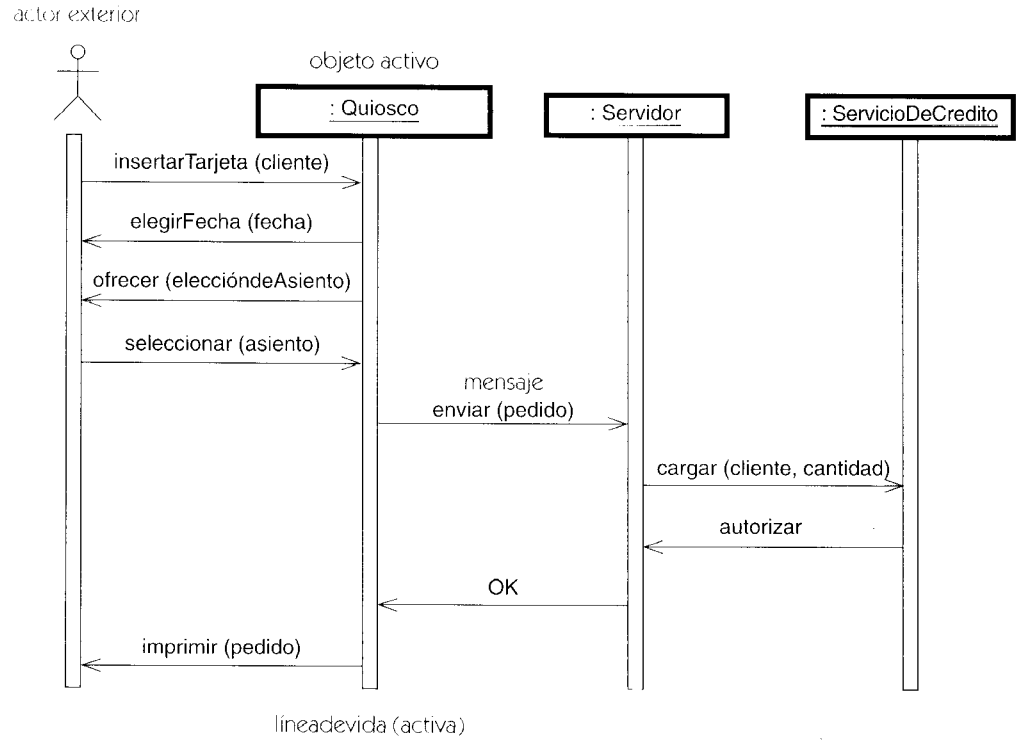


Figura 8.1 Diagrama de secuencia

tal muestra los roles de clasificador que representan objetos individuales en la colaboración. Cada rol de clasificador se representa mediante una columna vertical-línea de vida. Durante el tiempo que existe un objeto, el rol se muestra por una línea discontinua. Durante el tiempo que dura una activación de un procedimiento en el objeto, la línea de vida se dibuja como una línea doble.

Se muestra un mensaje como una flecha desde la línea de vida de un objeto a la del otro. Las flechas se organizan en el diagrama en orden cronológico hacia abajo.

La Figura 8.1 muestra un diagrama de secuencia típico con mensajes asíncronos.

Activación

Una activación es la ejecución de un procedimiento, incluyendo el tiempo que espera a los procedimientos anidados para ejecutarse. Se representa por una línea doble que sustituye la parte de la línea de vida en un diagrama de secuencia. Una llamada se representa por una flecha que apunta a la parte superior de la activación iniciada por la llamada. Ocurre una llamada recursiva cuando el control vuelve a entrar en una operación en un objeto, pero la segunda llamada es una activación separada de la primera. La recursión o una llamada anidada a otra operación en el mismo objeto, se representa en un diagrama de secuencia apilando las líneas de activación. La Figura 8.2 muestra un diagrama de secuencia, con el flujo de objeto de procedimientos, incluyendo una llamada recursiva y la creación de un objeto durante el cómputo.

Un objeto activo es un objeto que contiene la raíz de una pila de activaciones. Cada objeto activo tiene su propio hilo de control dirigido por eventos que se ejecuta en paralelo a otros ob-

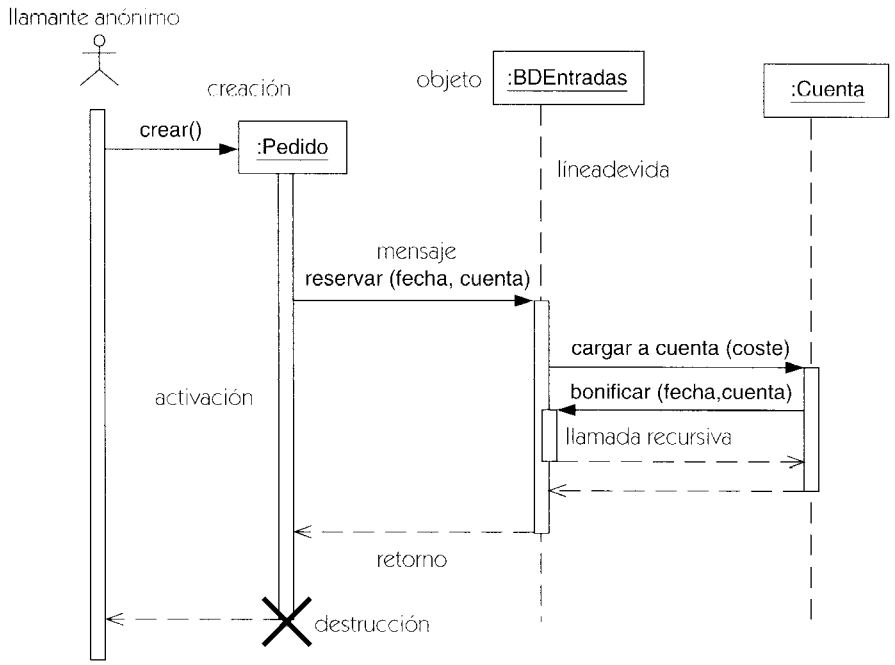


Figura 8.2 Cuando dos hilos van juntos en un solo objeto

jetos activos. Los objetos que son llamados por un objeto activo son objetos pasivos; reciben el control solamente cuando son llamados, y lo ceden cuando retornan.

Si varios hilos de control concurrentes tienen sus propios flujos de procedimientos de control usando llamadas anidadas, los diversos hilos deben distinguirse usando los nombres de los hilos, colores, u otros medios para evitar la confusión cuando dos hilos se unen en un único objeto (para una sincronización, por ejemplo). Generalmente, es mejor no mezclar llamadas a procedimientos con señales en un solo diagrama.

Diagrama de colaboración

Un diagrama de colaboración es un diagrama de clases que contiene roles de clasificador y roles de asociación en lugar de sólo clasificadores y asociaciones. Los roles de clasificador y los roles de asociación describen la configuración de los objetos y de los enlaces que pueden ocurrir cuando se ejecuta una instancia de la colaboración. Cuando se instancia la colaboración, los objetos están ligados a los roles de clasificador y los enlaces están ligados a los roles de asociación. El rol de asociación también puede ser desempeñado por varios tipos de enlaces temporales, tales como argumentos de procedimiento o variables locales del procedimiento. Los símbolos de enlace pueden llevar estereotipos para indicar enlaces temporales («**parameter**» o «**local**») o llamadas al mismo objeto («**self**»).

Solamente se representan los objetos que están implicados en la colaboración, aunque puede haber otros en el sistema completo. Es decir, un diagrama de colaboración modela los objetos y los enlaces implicados en la implementación de una interacción y no hace caso de las otras. La Figura 8.3 muestra un diagrama de colaboración.

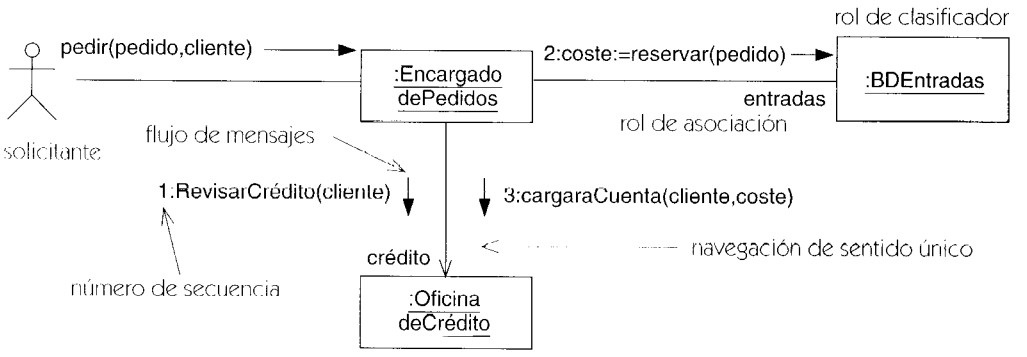


Figura 8.3 Diagrama de colaboración

Es útil marcar los objetos en cuatro grupos: los que existen con la interacción entera; los creados durante la interacción (restricción {**new**}); los destruidos durante la interacción (restricción {**destroyed**}); y los que se crean y se destruyen durante la interacción (restricción {**transient**}). Durante el diseño, se puede empezar mostrando los objetos y los enlaces disponibles al comienzo de una operación y después decidir cómo el control puede fluir a los objetos correctos dentro del gráfico para implementar la operación.

Aunque las colaboraciones muestran directamente la implementación de una operación, pueden también mostrar la realización de una clase entera. En este uso, muestran el contexto necesario para implementar *todas* las operaciones de una clase. Esto permite que el modelador vea los roles múltiples que los objetos pueden desempeñar en varias operaciones. Esta vista puede ser construida tomando la unión de todas las colaboraciones necesarias para describir todas las operaciones del objeto.

Mensajes. Los mensajes se muestran como flechas etiquetadas unidas a los enlaces. Cada mensaje tiene un número de secuencia, una lista opcional de mensajes precedentes, una condición opcional de guarda, un nombre y lista de argumentos, y un nombre de valor de retorno opcional. El número de serie incluye el nombre (opcional) de un hilo. Todos los mensajes del mismo hilo se ordenan secuencialmente. Los mensajes de diversos hilos son concurrentes a menos que haya una dependencia secuencial explícita. Pueden añadirse varios detalles de implementación, como por ejemplo una distinción entre los mensajes síncronos y asíncronos.

Flujos. Generalmente, un diagrama de colaboración contiene un símbolo para un objeto durante una operación completa. Sin embargo, a veces, un objeto tiene distintos estados que se deban hacer explícitos. Por ejemplo, un objeto pudo cambiar de localización, o sus asociaciones pudieron diferenciarse perceptiblemente en distintos momentos. Un objeto se puede mostrar con su clase y su estado —un objeto de una clase en un estado—. El mismo objeto se puede mostrar múltiples veces, cada uno con una localización o estado diferentes.

Los diferentes símbolos de objeto que representan un objeto se pueden conectar usando flujos «**become**» o «**conversión**». Un flujo «**become**» es una transición, a partir de un estado de un objeto a otro. Se dibuja como una flecha de línea discontinua con el estereotipo «**become**» o «**conversión**» y puede ser etiquetado con un número de serie para mostrar cuando ocurre (Figura 8.4). Un flujo de conversión también se utiliza para mostrar la migración de un objeto a partir de una localización a otra distinta.

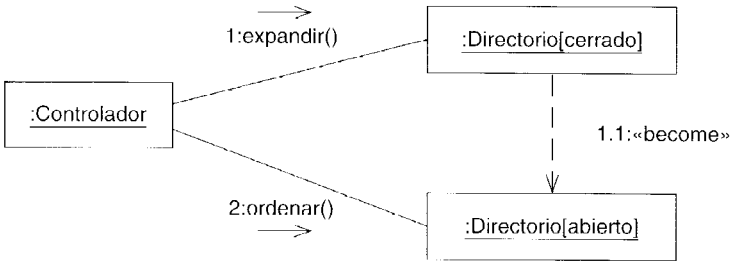


Figura 8.4 Flujos de conversión

Menos frecuentemente, el estereotipo «**copy**» o «**copia**» muestra un valor de objeto producido copiando otro valor de objeto.

La Tabla 8.1 muestra las tipos de relaciones de flujo de objeto.

Diagramas de colaboración y de secuencia. Los diagramas de colaboración y los diagramas de secuencia muestran interacciones, pero acentúan aspectos diferentes. Los diagramas de secuencia muestran secuencias de tiempo claramente pero no muestran explícitamente relaciones de objetos. Los diagramas de colaboración muestran las relaciones entre objetos claramente, pero las secuencias de tiempo se deben obtener a partir de números de secuencia. Los diagramas de secuencia son a menudo más útiles para mostrar escenarios; los diagramas de colaboración son a menudo más útiles para mostrar el diseño detallado de procedimientos.

Tabla 8.1 Tipos de Relaciones de Flujo

Flujo	*Función*	*Notación*
conversión	Transformación de un valor de un objeto a otro valor	«become» ⇢
copia	Copia de un objeto que a partir de entonces es independiente	«copy» ⇢

Patrones

Un patrón es una colaboración parametrizada, junto con las pautas sobre cuándo utilizarlo. Un parámetro se puede sustituir por diversos valores, para producir distintas colaboraciones. Los parámetros señalan generalmente las ranuras para las clases. Cuando se instancia un patrón, sus parámetros están ligados a clases reales de un diagrama de clase o a los roles dentro de una colaboración más amplia.

El uso de un patrón se representa como una elipse de línea discontinua conectada con cada una de sus clases por una línea discontinua, que se etiqueta con el nombre del rol. Por ejemplo, la Figura 8.5 muestra el uso del patrón **Observer** de [Gamma-95]. En este uso del patrón, **ColadeLlamadas** sustituye el rol del **sujeto** e **IconoDeBarraDeslizante** sustituye el rol del **manejador**.

Los patrones pueden aparecer en los niveles de análisis, arquitectura, diseño detallado e implementación. Son una manera de captar estructuras que ocurren frecuentemente para su reutilización. La Figura 8.5 muestra un uso del patrón Observer (Observador).

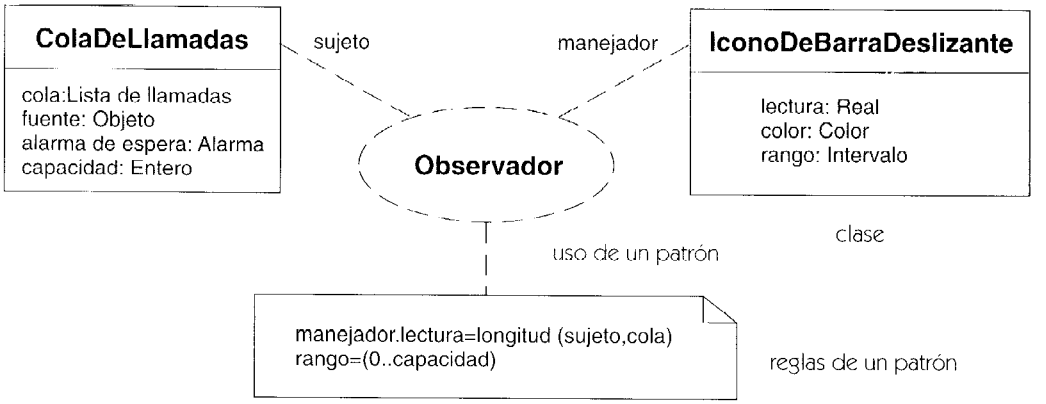


Figura 8.5 Uso del patrón

9 Vistas físicas

Descripción

Gran parte de un modelo de un sistema se piensa para mostrar los aspectos lógicos y de diseño del sistema, independiente de su empaquetado final en un medio de implementación. Sin embargo, los aspectos de implementación son importantes para los propósitos de reutilización y del rendimiento. UML incluye dos tipos de vistas para representar unidades de implementación: la vista de implementación y la vista de despliegue.

La vista de implementación muestra el empaquetado físico de las partes reutilizables del sistema en unidades sustituibles, llamadas componentes. Una vista de implementación muestra la implementación de los elementos del diseño (tales como clases) mediante componentes, así como sus interfaces y dependencias entre componentes. Los componentes son las piezas reutilizables de alto nivel a partir de las cuales se pueden construir los sistemas.

La vista de despliegue muestra la disposición física de los recursos de ejecución computacional, tales como computadores y sus interconexiones. Se llaman nodos. Durante la ejecución, los nodos pueden contener componentes y objetos. La asignación de componentes y de objetos a los nodos puede ser estática, o pueden migrar entre nodos.

La vista de despliegue puede mostrar cuellos de botella para el rendimiento si las instancias de los componentes con dependencias se ponen en distintos nodos.

Componente

Un componente es una unidad física de implementación con interfaces bien definidas pensada para ser utilizada como parte reemplazable de un sistema. Cada componente incorpora la implementación de ciertas clases del diseño del sistema. Los componentes bien diseñados no dependen directamente de otros componentes sino de las interfaces que ofrecen los componentes. En ese caso, un componente en un sistema se puede sustituir por otro componente que ofrezca las interfaces apropiadas.

Los componentes soportan más interfaces y requieren ciertas interfaces de otros componentes. Una interfaz es una lista de las operaciones que una pieza de software o de hardware ofrece y puede realizar. El uso de las llamadas interfaces permite evitar las dependencias directas entre componentes, facilitando una sustitución más fácil de nuevos componentes. La vista de componentes muestra la red de dependencias entre componentes. La vista de componentes puede aparecer de dos formas. Puede mostrar un conjunto de componentes disponibles (una biblioteca de componentes) con sus dependencias; éste es el material a partir del cual se puede ensam-

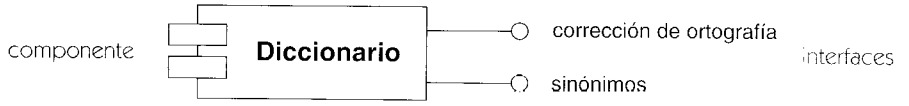


Figura 9.1 Componente con interfaces

blar un sistema. Puede también mostrar un sistema configurado, con la selección de componentes usados para construirlo (a partir de la biblioteca completa). De esta forma, cada componente se conecta a otros componentes cuyos servicios utiliza; estas conexiones deben ser consistentes con las interfaces de los componentes.

Un componente se dibuja como un rectángulo, con dos rectángulos pequeños a un lado. Puede ser unido por líneas sólidas a los círculos que representan sus interfaces (Figura 9.1).

Un diagrama de componentes muestra dependencias entre los componentes (Figura 9.2). Cada componente ofrece algunas interfaces y utiliza otras. Si las dependencias entre componentes se hacen a través de interfaces, los componentes se pueden sustituir por otros componentes que realicen las mismas interfaces.

Nodo

Un nodo es un objeto físico de ejecución que representa un recurso computacional, que generalmente tiene por lo menos memoria y a menudo también capacidad de proceso. Los nodos pueden tener estereotipos para distinguir diferentes tipos de recursos, tales como UCP, dispositivos, y memorias. Los nodos pueden contener objetos, instancias, instancias del componente.

Un nodo se representa mediante un cubo estilizado con el nombre del nodo y, opcionalmente, su clasificación (Figura 9.3).

Las asociaciones entre los nodos representan líneas de comunicación. Las asociaciones pueden tener estereotipos para distinguir diversos tipos de enlaces.

Los nodos pueden tener relaciones de generalización para relacionar una descripción general de un nodo con una variación más específica.

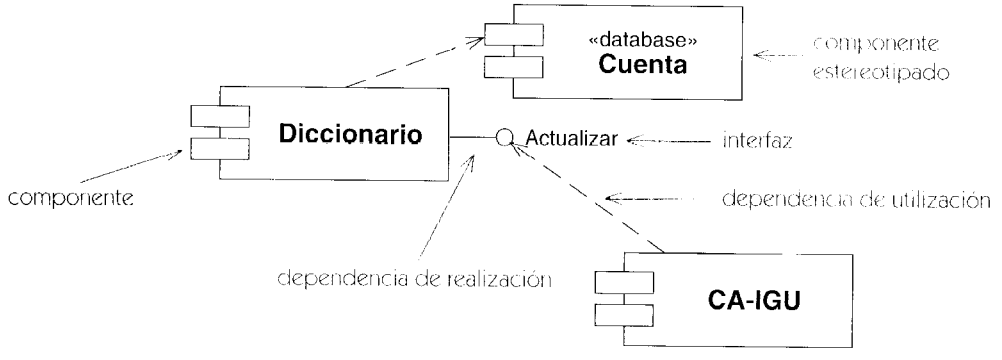


Figura 9.2 Diagrama de componentes

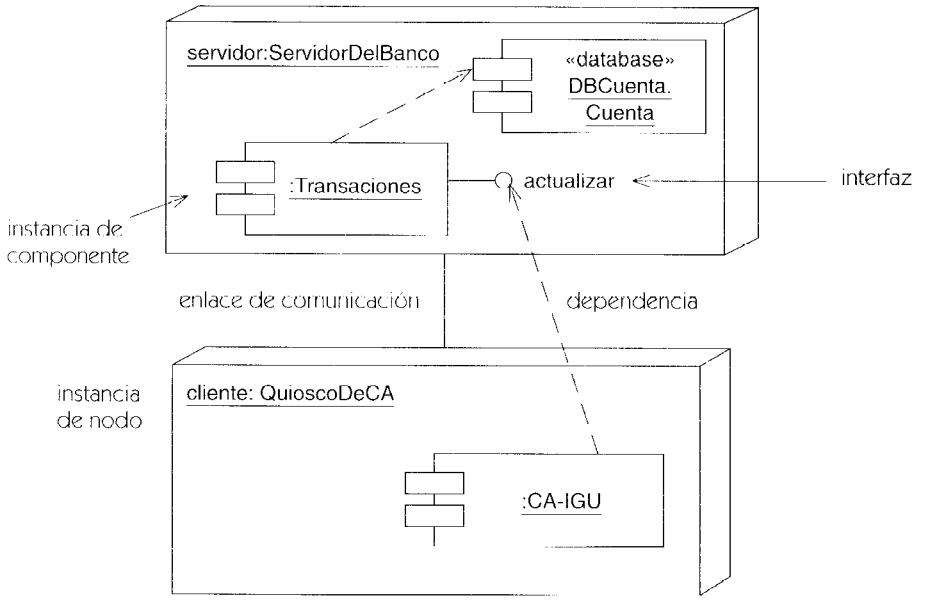


Figura 9.3 Diagrama de despliegue

La presencia de un objeto en un nodo se representa mediante el anidamiento físico del símbolo de objeto dentro del símbolo del nodo. Si eso no es conveniente, el símbolo de objeto puede contener la etiqueta **location**, cuyo valor sea el nombre del nodo en el que reside el objeto (su localización). La migración de objetos o de instancias de componentes entre nodos, puede representarse también.

Véase conversión.

10 La vista de gestión del modelo

Descripción

Cualquier sistema grande se debe dividir en unidades más pequeñas, de modo que las personas puedan trabajar con una cantidad de información limitada, a la vez y de modo que los equipos de trabajo no interfieran con el trabajo de los otros. La gestión del modelo consiste en paquetes (incluyendo tipos especiales de paquetes) y relaciones de dependencia entre paquetes.

Paquete

Un paquete es una parte de un modelo. Cada parte de un modelo debe pertenecer a un paquete. El modelador puede asignar el contenido de un modelo a un conjunto de paquetes. Pero, para ser funcional, la asignación debe seguir un cierto principio racional, tal como funcionalidad común, implementación estrechamente relacionada, y un punto de vista común. UML no impone una regla para componer los paquetes, pero una buena descomposición en paquetes realzará enormemente la capacidad de mantenimiento del modelo.

Los paquetes contienen elementos del modelo al más alto nivel, tales como clases y sus relaciones, máquinas de estado, diagramas de casos de uso, interacciones, y colaboraciones: cualquier elemento que no esté contenido en otro. Los elementos como atributos, operaciones, estados, líneas de vida, y mensajes están contenidos en otros elementos, y no aparecen como contenido directo de los paquetes. Cada elemento de nivel superior tiene un paquete en el cual se declara. Éste es su paquete de origen. Puede hacerse referencia a él en otros paquetes, pero el contenido del elemento pertenece al paquete de origen. En un sistema de control de configuración, un modelador debe tener acceso al paquete de origen para modificar el contenido de un elemento. Esto proporciona un mecanismo de control de acceso para trabajar en los modelos grandes. Los paquetes son también las unidades para cualquier mecanismo de versiones.

Los paquetes pueden contener otros paquetes. Hay un paquete raíz, que contiene indirectamente el modelo completo de un sistema. Hay varias maneras posibles de organizar los paquetes en un sistema. Pueden ser organizados por la vista, por la funcionalidad, o por cualquier otra base que el modelador elija. Los paquetes son unidades de organización jerárquica de uso general de los modelos de UML. Pueden ser utilizados para el almacenamiento, el control de acceso, la gestión de la configuración, y la construcción de bibliotecas que contengan fragmentos reutilizables del modelo.

Si se eligen bien los paquetes, reflejan la arquitectura de alto nivel de un sistema: su descomposición en subsistemas y sus dependencias. Una dependencia entre paquetes resume las dependencias entre los contenidos del paquete.

Dependencias en los paquetes

Las dependencias que se presentan entre elementos individuales, pero en un sistema de cualquier tamaño, deben ser vistas en un nivel más alto. Las dependencias entre los paquetes resumen dependencias entre los elementos internos a ellos, es decir, las dependencias del paquete son derivables a partir de las dependencias entre elementos individuales.

La presencia de una dependencia entre paquetes implica que existe en un enfoque ascendente (una declaración de existencia), o que se permite que exista más adelante en un enfoque descendente (una restricción que limita cualquier otra relación), por lo menos un elemento de relación con el tipo indicado de dependencia entre elementos individuales dentro de los paquetes correspondientes. Es una declaración de existencia y no implica que todos los elementos del paquete tengan la dependencia. Es un indicador para el modelador de que existe información adicional, pero la dependencia, cuando se trata de paquetes, no contiene más información por sí misma; es solamente un resumen.

El enfoque descendente refleja la arquitectura del sistema total. El enfoque ascendente se puede generar automáticamente a partir de los elementos individuales. Ambos enfoques tienen su lugar en el modelado, incluso en un solo sistema.

Las dependencias múltiples del mismo tipo entre elementos individuales se agregan a una sola dependencia entre los paquetes que contienen los elementos. Si las dependencias entre elementos individuales contienen estereotipos (tales como diferentes tipos de uso), el estereotipo se puede omitir en la dependencia de paquete, para dar una sola dependencia de alto nivel.

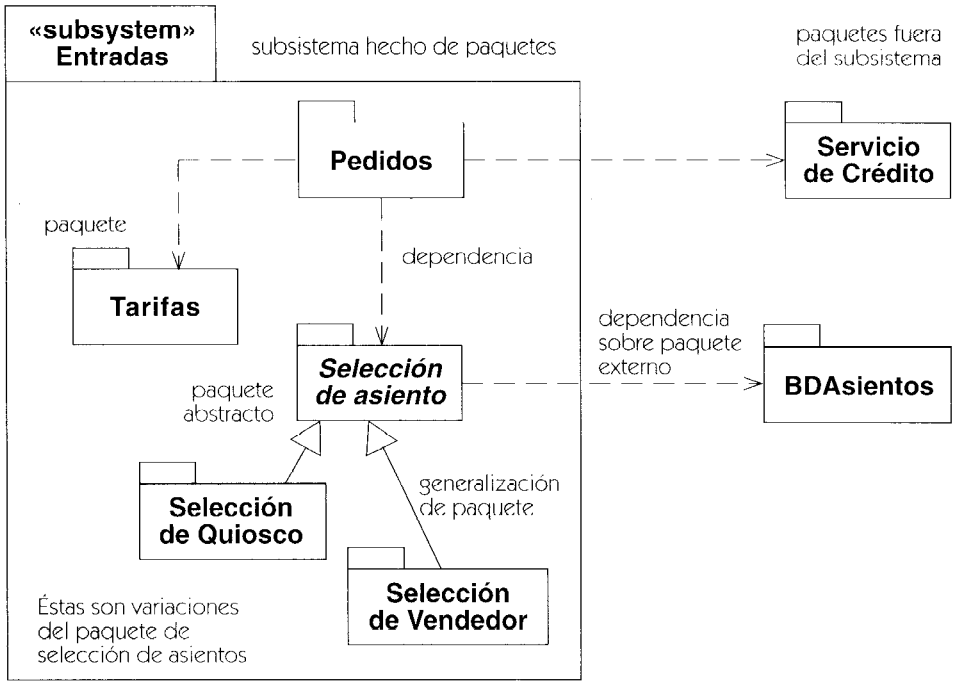


Figura 10.1 Paquetes y sus relaciones

Los paquetes se dibujan como rectángulos con lengüetas (iconos "carpeta"). Las dependencias se muestran como flechas con líneas discontinuas.

La Figura 10.1 muestra la estructura del paquete para un subsistema de reserva de entradas. Tiene dependencias en los paquetes exteriores y dos variaciones del paquete de **selección de asientos**. Cualquier implementación del subsistema incluiría solamente una variación.

Dependencia de acceso e importación

En general, un paquete no puede tener acceso al contenido de otro paquete. Los paquetes son opacos, a menos que sean abiertos por una dependencia de acceso o de importación. La dependencia de acceso se aplica directamente a los paquetes y a otros contenedores. En el nivel de paquete, la dependencia de acceso indica que el contenido del paquete del proveedor puede aparecer en referencias efectuadas por los elementos del paquete cliente o de los paquetes incluidos dentro del cliente. Un elemento en el proveedor debe tener suficiente visibilidad dentro de su paquete para permitir a un cliente verlo. En general, un paquete puede ver solamente los elementos de otros paquetes que tienen asignada visibilidad pública en el paquete que los contiene. Los elementos con visibilidad protegida son visibles solamente por los paquetes que son descendientes del paquete que contiene los elementos. Los elementos con visibilidad privada son visibles solamente por el paquete que los contiene, y por cualesquiera paquetes anidados en el interior de ese paquete. La visibilidad también se aplica al contenido de clases (atributos y operaciones). Un descendiente de una clase puede ver a miembros de su antecesor con visibilidad pública o protegida; cualquier otra clase puede ver solamente miembros con visibilidad pública. El permiso de acceso y la visibilidad apropiada son necesarios para hacer referencia a un elemento. Así que para que un elemento, en un paquete, vea a un elemento en un paquete sin relación, el primer paquete debe tener acceso o importar al segundo paquete, y el elemento destino debe tener visibilidad pública dentro del segundo paquete.

Un paquete anidado dentro de otro paquete es parte del contenedor y tiene acceso completo a su contenido, sin necesidad de accesos. El contenedor, sin embargo, puede no ver el interior de sus paquetes anidados si no tiene acceso. Se encapsula el contenido.

Observe que una dependencia de acceso no modifica el espacio de nombres del cliente ni crea las referencias automáticamente de ninguna otra forma. Simplemente concede permiso para establecer referencias. La dependencia de importación se utiliza para agregar nombres al espacio de nombres del paquete del cliente como sinónimos de los caminos completos.

Modelo y subsistema

Un modelo es un paquete que abarca una descripción completa de una vista particular de un sistema. Proporciona una descripción cerrada de un sistema a partir de un punto de vista. No tiene dependencias fuertes en otros paquetes, tales como dependencias de implementación o dependencias de herencia. La relación de traza es una forma débil de dependencia entre elementos, en distintos modelos, que observan la presencia de una cierta conexión sin implicaciones semánticas específicas.

Generalmente, un modelo se estructura con forma de árbol. El paquete raíz contiene anidados en sí mismo paquetes que constituyen el detalle completo del sistema desde un punto de vista dado.

Un subsistema es un paquete que tiene piezas separadas de especificación y de realización. Representa una unidad coherente del modelo, con interfaces limpias al resto del sistema. Representa generalmente la partición del sistema, en un límite funcional o de implementación. Los modelos y los subsistemas se dibujan como paquetes con las palabras clave de estereotipo (Figura 10.1).

11 Mecanismos de extensión

Descripción

UML proporciona varios mecanismos de extensión para permitir que los modeladores hagan algunas extensiones comunes sin tener que modificar el lenguaje de modelado subyacente. Se han diseñado estos mecanismos de extensión de modo que las herramientas puedan almacenar y manipular las extensiones aun sin entender su semántica completa. Por esta razón, las extensiones se pueden almacenar y manipular como cadenas. Para una herramienta que no entienda la extensión, es sólo una cadena, pero se puede introducir o almacenar como parte de un modelo, y ser pasada a otras herramientas. Se espera que las herramientas de base y los módulos adicionales sean escritos para procesar varios tipos de extensiones. Estas herramientas definirán una sintaxis y una semántica particulares para sus extensiones que sólo ellas necesitan entender.

Este acercamiento a las extensiones no resolverá probablemente todas las necesidades que se presenten, pero pensamos que se ajustará a una porción grande de las adaptaciones que necesitan la mayoría de los modeladores de una manera simple que es fácil de implementar. Los mecanismos de extensión son restricciones, valores etiquetados, y estereotipos. Tenga presente que una extensión, por definición, desvía la forma de estándar de UML y puede, por lo tanto, conducir a problemas de interoperatividad. El modelador debe sopesar cuidadosamente ventajas y costes antes de usar extensiones, especialmente cuando los mecanismos existentes trabajaran razonablemente bien. Típicamente, las extensiones están pensadas para los dominios de aplicaciones particulares o los entornos de programación, pero dan lugar a un dialecto de UML, con las ventajas y desventajas de todos los dialectos.

Restricción

Una restricción es una condición semántica representada como expresión textual. Cada expresión tiene un lenguaje implícito de interpretación, que puede ser una notación matemática formal, tal como notación de teoría de conjuntos; un lenguaje computarizado de restricción, tal como OCL; un lenguaje de programación, como C++; o pseudocódigo o el lenguaje natural informal. Por supuesto, si el lenguaje es informal, entonces su interpretación es también informal, y debe ser hecha por una persona. Incluso si una restricción se expresa en un lenguaje formal, no significa que se hará cumplir automáticamente. Las técnicas de mantenimiento y comprobación automática de verdades no pueden cubrir la mayoría de los casos, pero por lo menos la semántica será exacta.

Las restricciones pueden expresar limitaciones y relaciones que no se pueden expresar usando la notación de UML. Son particularmente útiles para indicar las condiciones globales o las condiciones que afectan a ciertos elementos.

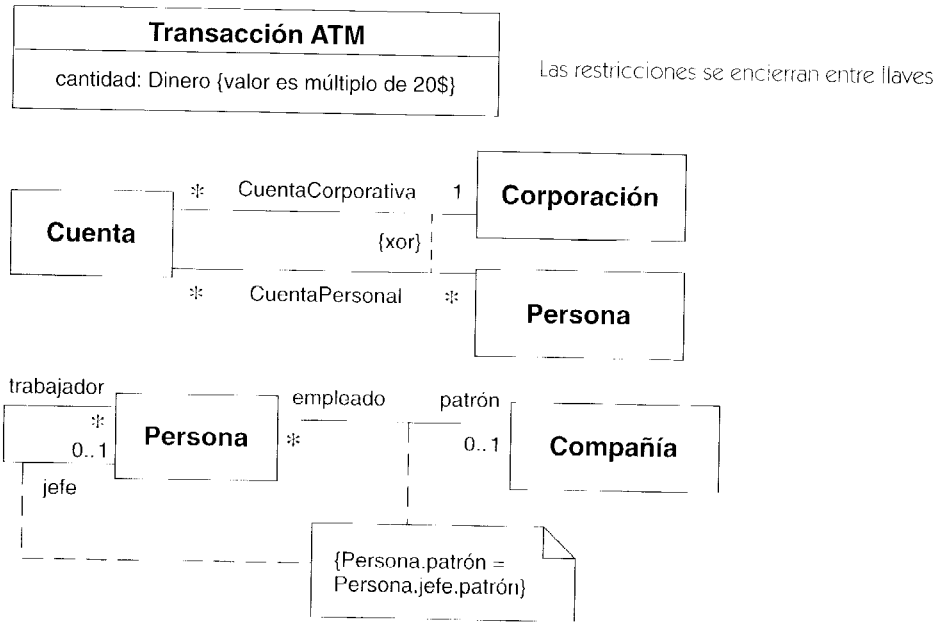


Figura 11.1 Restricciones

Las restricciones se muestran como expresiones de cadena entre llaves. Pueden ser añadidas a un elemento de una lista, ser asociadas a una dependencia, o ser incluidas en una nota. La Figura 11.1 muestra varios tipos de restricciones.

Valor etiquetado

Un valor etiquetado es un par de cadenas: una cadena de etiqueta y una cadena de valor, que almacena un elemento de información sobre un elemento. Un valor etiquetado se puede unir a cualquier elemento individual, incluyendo los elementos del modelo y los elementos de la presentación.

La etiqueta es un nombre de una cierta propiedad que el modelador desea registrar, y el valor es el valor de esa propiedad para el elemento dado. Por ejemplo, la etiqueta pudo ser **autor**, y el valor pudo ser el nombre de la persona responsable del elemento, tal como **Charles Babbage**.

Los valores etiquetados se pueden utilizar para almacenar cualquier información sobre los elementos. Son particularmente útiles para almacenar información de gestión del proyecto, tal como la fecha de creación de un elemento, de su estado de desarrollo, de fechas debidas, y del estado de la prueba. Cualquier cadena se puede utilizar como un nombre de etiqueta, excepto aquellos nombres de atributos incorporados al metamodelo que deben evitarse (porque las etiquetas y los atributos juntos pueden considerarse propiedades de un elemento y accederse de manera uniforme en una herramienta), y ciertos nombres de etiqueta están predefinidos (véase el capítulo 14, Elementos Estándar).

Los valores etiquetados también proporcionan una manera de añadir información dependiente de la implementación a los elementos. Por ejemplo, un generador de código necesita la información adicional sobre el tipo de código a generar a partir de un modelo. A menudo, hay

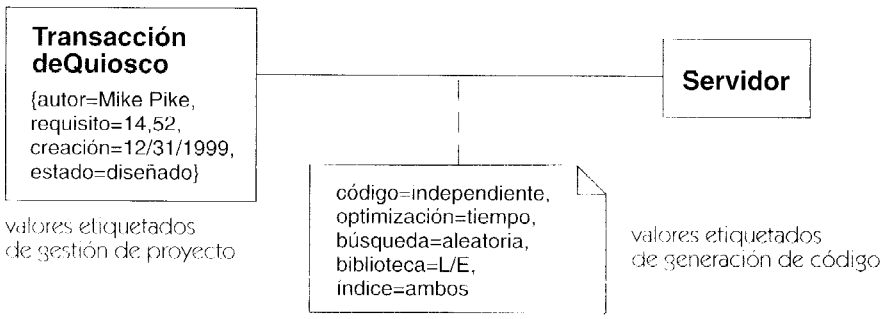


Figura 11.2 Valores etiquetados

varias maneras posibles de implementar correctamente un modelo; el modelador debe proporcionar las directrices sobre qué elección tomar. Ciertas etiquetas se pueden utilizar como indicadores para decir al generador de código qué implementación utilizar. Otras etiquetas se pueden utilizar para otros tipos de herramientas añadidas, tales como planificadores del proyecto y generadores de informes.

Los valores etiquetados también se pueden utilizar para almacenar información sobre elementos estereotipados del modelo (discutidos abajo).

Los valores etiquetados se muestran como cadenas con el nombre de la etiqueta, un igual, y el valor. Se colocan normalmente en listas entre llaves (Figura 11.2). Serán omitidos en los diagramas pero mostrados a menudo en listas y formularios emergentes.

Estereotipos

Muchos modeladores desean adaptar un lenguaje de modelado para un dominio de aplicación particular. Esto conlleva un cierto riesgo, porque el lenguaje adaptado no será universalmente comprensible, pero existe, sin embargo, la tentación de hacerlo.

Un estereotipo es un tipo de elemento del modelo definido en el propio modelo. El contenido de información y la forma de un estereotipo son igual que los de una clase existente de elemento del modelo base, pero su significado y uso son diferentes. Por ejemplo, los modeladores en el área de modelado de negocio a menudo quieren distinguir objetos de negocio y procesos del negocio como tipos especiales de elementos de modelado cuyo uso es distinto dentro de un proceso dado del desarrollo. Éstos se pueden tratar como tipos especiales de clases: tienen atributos y operaciones, pero tienen restricciones especiales en sus relaciones con otros elementos y en su uso.

Un estereotipo se basa en un elemento de modelado existente. El contenido de información del elemento estereotipado es igual que el elemento de modelo existente. Esto permite que una herramienta almacene y manipule el nuevo elemento de la misma forma que lo hace con el elemento existente. El elemento estereotipado puede tener sus propios iconos; es fácil que una herramienta lo permita. Por ejemplo, una "organización de negocios" podría tener un icono que parece un grupo de personas. El estereotipo puede también tener una lista de las restricciones que se aplican a su uso. Por ejemplo, quizás una "organización de negocios" se puede asociar solamente a otra "organización de negocios" y no a cualquier clase. No todas las restricciones se pueden verificar automáticamente por una herramienta de propósito general, pero

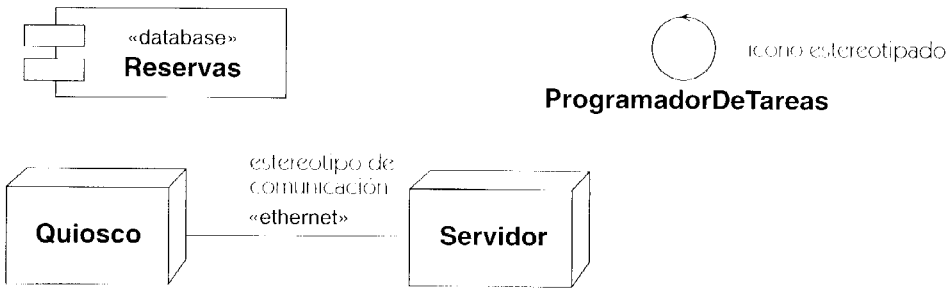


Figura 11.3 Estereotipos

pueden hacerse cumplir manualmente o ser verificadas por una herramienta añadida que entienda el estereotipo.

Los estereotipos pueden utilizar valores etiquetados para almacenar las propiedades adicionales que no se pueden modelar con el elemento base.

Los estereotipos se muestran como cadenas de texto rodeadas por comillas («») puestos dentro o cerca del símbolo normal del elemento. El modelador puede también crear un icono para un estereotipo particular, que sustituye el símbolo normal del elemento (Figura 11.3).

Adaptación de UML

Los mecanismos de extensión de restricciones, de valores etiquetados, y de estereotipos permiten adaptar los perfiles de UML para los dominios especiales de uso. Se han realizado varios perfiles, que se describen en el apéndice C, Extensiones de Proceso. Otros han sido propuestos por los usuarios. La capacidad de adaptación del lenguaje de modelado significa que los dominios de uso pueden adaptar el lenguaje de modelado a sus necesidades, sin embargo, todavía comparte la extensa preponderancia de los conceptos genéricos y comunes a todos los dominios.

12 El entorno de UML

Descripción

Los modelos de UML se utilizan dentro de un entorno. La mayoría de la gente utiliza el modelado como medio para un fin —vamos a llamarlo desarrollo de buenos sistemas— y no como fin en sí mismo. El propósito y la interpretación del modelo están afectados por el resto del entorno. Otras facilidades en el entorno más amplio incluyen los metamodelos que abarcan muchos lenguajes, herramientas de edición del modelo, lenguajes de programación, sistemas operativos y componentes importantes del sistema, y el mundo de los negocios y la ingeniería dentro del cual se utilizan los sistemas. La responsabilidad de dar significado a un modelo y de implementar su objetivo se resuelve con todas estas facilidades, incluyendo UML.

Los modelos existen en varios niveles de concreción. UML es un lenguaje de modelado, de propósito general, que incluye semántica y notación, pero es utilizable con diversas herramientas y lenguajes de implementación. Cada nivel de uso introduce ciertas consideraciones de modelado, que aparecen en UML en diferentes grados.

Responsabilidades semánticas

Un metamodelo es la descripción de un modelo. Un lenguaje de modelado describe modelos; por lo tanto, puede ser descrito por un metamodelo. Un metamodelo procura hacer un lenguaje exacto, definiendo su semántica, pero hay una tensión para permitir las extensiones para nuevas situaciones. La forma real del metamodelo es importante para la implementación de las herramientas, y el intercambio de modelos, pero no muy importante para la mayoría de los usuarios. Por lo tanto no la hemos incluido en este libro. Los que estén interesados pueden consultar los documentos originales del estándar [UML-9] disponibles en el CD adjunto.

Un metamodelo y un lenguaje deben cubrir muchos campos y acomodar muchas interpretaciones. Los sistemas existentes tienen modelos de ejecución y de memoria diferentes. Es imposible elegir uno de ellos como la interpretación correcta. De hecho, es probablemente engañoso incluso considerar tal opción. En su lugar, uno puede pensar en las diversas interpretaciones de los modelos de ejecución como puntos de variación semántica. Un punto de variación semántica es un punto de diferencia sobre la semántica detallada de la ejecución, pero que es ortogonal a otros aspectos de un sistema. Por ejemplo, un entorno puede elegir o no ofrecer la clasificación dinámica, la capacidad de un objeto de cambiar su clase en tiempo de ejecución. Hoy, la mayoría de los lenguajes de programación no lo permiten, principalmente por razones de implementación del lenguaje de programación, pero algunos sí. La diferencia es indistinguible en la semántica estática. La opción de la clasificación estática o de la clasificación dinámica se puede identificar

como un punto de variación semántica con dos opciones: clasificación estática o clasificación dinámica. Cuando existen tales opciones, la gente discute a menudo sobre cuál es la interpretación correcta. En vez de esto, observe que esto es una elección y asígnele un nombre para poder utilizarla.

Un metamodelo describe el contenido de un modelo bien formado, mientras que un lenguaje de programación describe un programa bien formado. Solamente un modelo bien formado tiene un significado y una semántica apropiada; no tiene sentido pedir el significado de un modelo mal formado. Sin embargo, la mayoría del tiempo, los modelos bajo desarrollo no están bien formados. Son incompletos y posiblemente inconsistentes. Pero esto es lo que deben contemplar las herramientas de modelado: modelos incompletos, no sólo modelos acabados. El metamodelo de UML describe modelos correctos, bien formados. Un metamodelo separado puede describir posibles fragmentos de un modelo. Dejamos a los fabricantes de la herramienta decidir dónde situar el límite en los fragmentos del modelo admitidos, y qué tipo de semántica dar a los modelos mal formados.

UML incluye algunos mecanismos de extensión incorporados para adaptar su uso en dominios especializados. Los mecanismos incluyen la capacidad de definir estereotipos y valores etiquetados. Estos mecanismos se pueden utilizar para adaptar una variante de UML, definiendo un sistema de estereotipos y de etiquetas, y adoptando las convenciones para su uso a la hora de construir un modelo. Por ejemplo, las variantes desarrolladas podrían centrarse en la semántica de la implementación de varios lenguajes de programación. La adición de extensiones puede ser poderosa, pero conlleva algunos peligros inherentes. Dado que su semántica no se define dentro de UML, UML no puede proveer su significado; la interpretación se delega al modelador. Además, si no se tiene cuidado, algunos significados pueden ser ambiguos o incluso inconsistentes. Las herramientas pueden proporcionar significados y funciones automatizadas para los estereotipos y las etiquetas definidas por las herramientas, pero no para las extensiones definidas por el usuario. Sin importar la ayuda para las extensiones, cualquier extensión tira del usuario lejos del centro común que el estándar del lenguaje proporciona y socava las metas de la capacidad de intercambio de modelos y de comprensibilidad de los modelos. Por supuesto, siempre que usted utilice una biblioteca de clases particular, usted diverge de la capacidad de intercambio perfecta de la nada. Así que, no se preocupe sobre lo abstracto. Utilice las extensiones cuando sean de ayuda, pero evítelas cuando no sean necesarias.

Responsabilidades de notación

La notación no agrega significado a un modelo, sino que ayuda al usuario a entender su significado. La notación no tiene semántica, sino que agrega a menudo las connotaciones para un usuario, tal como la afinidad percibida de dos conceptos basados en su proximidad en un diagrama.

Los documentos de UML [UML-98] y este libro definen una notación canónica de UML, que podría llamarse formato de publicación para modelos. Esto es similar a muchos lenguajes de programación en los cuales los programas dentro de artículos publicados se imprimen en un formato atractivo con disposición cuidadosa, palabras reservadas en negrita, y separan las figuras para cada procedimiento. Los verdaderos compiladores tienen que aceptar una entrada más sucia. Esperamos que esas herramientas de edición extiendan la notación a un formato de pantalla, incluyendo las cosas tales como el uso de tipografías y de color para resaltar elementos; la capacidad de suprimir y de filtrar fácilmente elementos que no son actualmente de interés, para

ampliar un diagrama, para mostrar los elementos anidados, para atravesar enlaces a otros modelos o vistas; y animación. Sería imposible intentar estandarizar todas estas posibilidades y absurdo intentarlo, porque no hay necesidad y limitaría la innovación útil. Esta clase de extensiones de notación es responsabilidad del constructor de la herramienta. En una herramienta interactiva hay menos peligro de ambigüedad, porque el usuario puede pedir siempre una aclaración. Esto es probablemente más útil que insistir en una notación que sea totalmente inequívoca en el primer vistazo. Lo importante es que una herramienta debe poder producir la notación canónica cuando ésta sea solicitada, especialmente en forma impresa, y que una herramienta interactiva debe suministrar extensiones razonables a este comportamiento mínimo.

Contamos con que las herramientas también permitan que los usuarios extiendan la notación de maneras limitadas pero útiles. Hemos especificado que los estereotipos pueden tener sus propios iconos. Podrían estar permitidas otras clases de extensiones de notación, pero los usuarios deben aplicar un buen criterio.

Observe que la notación es algo más que figuras; incluye la información en forma de texto y los hiperenlaces invisibles entre elementos de representación.

Responsabilidades del lenguaje de programación

UML debe trabajar con varios lenguajes de implementación, sin incorporarlos explícitamente. Pensamos que UML debe permitir el uso de cualquier (o por lo menos de muchos) lenguajes de programación, para la especificación y la generación del código final. El problema es que cada lenguaje de programación tiene muchos elementos semánticos, que no deseamos absorber en UML, porque se manejan mejor como elementos del lenguaje de programación, y hay variaciones considerables en la semántica de ejecución. Por ejemplo, la semántica de concurrencia se maneja de maneras diferentes entre los lenguajes (si es que se maneja en todos).

Los tipos de datos primitivos no se describen detalladamente en UML. Esto es deliberado pues no deseábamos incorporar la semántica de un lenguaje de programación en preferencia a todos los demás. Para la mayoría de los propósitos de modelado, esto no es un problema. Utilice el modelo semántico aplicable a su lenguaje. Éste es un ejemplo de un punto de variación semántico.

La representación de las propiedades detalladas del lenguaje para la implementación suscita el problema de capturar la información sobre propiedades de la implementación sin la construcción de su semántica en UML. La decisión tomada fue capturar las propiedades del lenguaje que van más allá de capacidades incorporadas en UML por medio de estereotipos y de valores etiquetados. Éstos se pueden asignar a las propiedades del lenguaje y a las opciones de la generación de código por una herramienta o un generador de código. Un editor genérico no necesita entenderlos. Un usuario podría, de hecho, crear un modelo usando una herramienta que no contemple el lenguaje a utilizar, y transferir el modelo final a otra herramienta para el procesamiento final. Por supuesto, si la herramienta no entiende los estereotipos y las etiquetas, no puede comprobarlas para saber si hay consistencia. Pero esto no es peor que la práctica normal con los editores de textos y los compiladores. En caso de necesidad, se puede crear una herramienta que utilice un conjunto particular de extensiones.

La generación de código y la ingeniería inversa requerirán, al menos en el futuro próximo, la participación del diseñador además de un modelo de UML. Las directivas y ayudas al generador de código se pueden proveer como valores etiquetados y estereotipos. Por ejemplo, el modelador

podría indicar qué tipo de clase contenedora se debe utilizar para implementar una asociación. Por supuesto, esto significa que los ajustes para generación de código en las herramientas pueden ser incompatibles, pero no creemos que haya actualmente suficiente acuerdo sobre la forma correcta de estandarizar los ajustes reales. En cualquier caso, diversas herramientas utilizarán sus generadores de código como su ventaja competitiva. Eventualmente, los ajustes por defecto pueden emerger y llegar a estar maduros para la estandarización.

Modelado con herramientas

Para modelar sistemas de tamaño real se necesitan herramientas. Las herramientas proporcionan maneras interactivas de ver y editar modelos. Proporcionan un nivel de organización, que está fuera del alcance del UML por sí mismo, pero que permite entender al usuario y ayuda en el acceso a la información. Las herramientas ayudan a encontrar la información en modelos grandes, buscando y filtrando lo que se presenta.

Algunos aspectos sobre el uso de herramientas

Las herramientas se ocupan de la organización y del almacenamiento físico de los modelos. Deben dar soporte múltiples equipos de trabajo en un sólo proyecto, así como permitir la reutilización entre proyectos. Los siguientes elementos están fuera del alcance del UML canónico, pero se deben considerar para el uso real de la herramienta.

Ambigüedades e información sin especificar. En las etapas iniciales faltan aún muchas cosas por decir. Las herramientas deben poder ajustar la precisión de un modelo y no forzar cada valor para ser específico. Véanse las siguientes secciones "Modelos inconsistentes para el trabajo en marcha" y "Valores sin especificar y nulos."

Opciones de presentación. Los usuarios no desean ver toda la información todo el tiempo. Las herramientas deben permitir filtrar y ocultar la información no deseada en un momento dado, así como visualizaciones alternativas, usando las capacidades de visualización del hardware disponible. Esto se ha descrito ya, en la sección "Responsabilidades de la Notación."

Gestión del modelo. El control de configuración, el control de acceso, y la gestión de versiones de las unidades del modelo está fuera del alcance de UML, pero son cruciales para el proceso de ingeniería del software y están en la cima del metamodelo.

Interfaces con otras herramientas. Los modelos necesitan ser manejados por los generadores de código, las calculadoras de métricas, los escritores de informes, los motores de ejecución, y otras herramientas de base. La información para otras herramientas necesita ser incluida en los modelos, pero no es información de UML. Los valores etiquetados son apropiados para contener esta información.

Modelos inconsistentes para el trabajo en marcha

La última meta del modelado es producir una descripción de un sistema en un cierto nivel de detalle. El modelo final debe satisfacer varias restricciones de validación para ser significativo. Sin embargo, como en todo proceso creativo, el resultado no se produce necesariamente de forma lineal. Los productos intermedios no satisfarán todos las restricciones de validez en cada paso. En la práctica, una herramienta debe manejar no sólo semánticamente los modelos válidos

que satisfacen las restricciones de validez, sino también los modelos sintácticamente válidos, que satisfacen ciertas reglas de construcción pero pueden violar algunas restricciones de validez. Los modelos semánticamente inválidos no son directamente utilizables. En su lugar pueden considerarse como "trabajos en marcha", que representan las trayectorias hacia el resultado final.

Valores sin especificar y nulos

Un modelo completo debe tener valores para todos los atributos de sus elementos. En muchos casos, *null o nulo* (ningún valor) es uno de los valores posibles, pero si un valor puede ser nulo o no, debe ser parte de la descripción del tipo del atributo; muchos tipos no tienen un valor nulo natural dentro de su gama de valores. Por ejemplo, nulo no tiene ningún sentido como límite superior en el tamaño de un conjunto. O el conjunto tiene un tamaño superior fijo o no hay límite, en cuyo caso su tamaño máximo es ilimitado, así que la posibilidad de ser nulo es un añadido al rango de posibles valores de un tipo de datos.

Por otra parte, durante las primeras etapas del diseño, un desarrollador puede no cuidar el valor de una propiedad en concreto. Puede ser que sea un valor que no es significativo en una etapa particular, por ejemplo, la visibilidad al hacer un modelo del dominio. O el valor puede ser significativo pero el modelador pudo no haberlo especificado todavía, y el desarrollador necesita recordar que aún es necesario elegir su valor. En este caso, el valor está sin especificar. Esto indica que un valor será eventualmente necesario, pero que todavía no se ha especificado. No es igual que un valor nulo, que puede ser un valor legítimo en el modelo final. En muchos casos, particularmente con las cadenas, un valor nulo es una buena manera de indicar un valor sin especificar, pero no son lo mismo. Un valor sin especificar no es significativo en un modelo bien formado. La definición de UML no maneja valores sin especificar. Son responsabilidad de las herramientas que soportan UML, y se consideran parte de un trabajo en marcha que, por necesidad, no tiene ningún significado semántico.

Parte 3: Referencia

abstracción

1. Acto de identificar aquellas características esenciales de una cosa que la distinguen del resto de cosas. La abstracción implica buscar parecidos en conjuntos de cosas, centrándose en las características esenciales comunes. Una abstracción siempre implica la perspectiva de quien la realiza, por lo que pueden existir diferentes abstracciones para la misma cosa según el propósito. Todo modelado implica abstracción, a menudo en diferentes niveles para propósitos distintos.

2. Tipo de dependencia que relaciona dos elementos que representan el mismo concepto en diferentes niveles de abstracción.

Véase derivación, realización, refinamiento, traza.

Semántica

Una dependencia de abstracción es una relación entre dos elementos que se encuentran en niveles distintos de abstracción, como las representaciones en diferentes modelos, en niveles distintos de precisión, en niveles distintos de concreción o en niveles distintos de optimización. Generalmente las dos representaciones no se utilizarán simultáneamente. Normalmente un elemento es más detallado que otro, donde el más detallado es el cliente y el otro el proveedor. Si no está claro qué elemento es más detallado, se puede modelar cualquiera de los dos elementos como cliente.

Los estereotipos de dependencia de abstracción son la traza (palabra clave «**trace**»), el refinamiento (palabra clave «**refine**»), la realización (palabra clave «**realize**») y la derivación (palabra clave «**derive**»).

Notación

Una dependencia de abstracción se representa mediante una flecha con línea discontinua que va del elemento cliente al elemento proveedor etiquetada con alguna de las palabras clave «**trace**», «**refine**» o «**derive**». La dependencia de realización tiene su propia notación, que consiste en una flecha con línea discontinua cuyo extremo es un triángulo cerrado que apunta al elemento proveedor.

La correspondencia entre elementos se puede adjuntar a la relación como restricción.

Elementos estándar

derive, refine, trace

abstracto/a

Clase, caso de uso, señal, otro clasificador u otro elemento generalizable que no puede ser instanciado directamente. También se utiliza para describir una operación que no tiene implementación. Antónimo: concreto/a.

Véase operación abstracta, elemento generalizable.

Semántica

Una clase abstracta es una clase no instanciable, es decir, que no puede tener instancias directas, bien porque su descripción es incompleta (le falta el método de una o más operaciones) o porque no ha sido pensada para ser instanciada aunque su descripción esté completa. El objetivo fundamental de las clases abstractas es la especialización. Para ser útil, una clase abstracta debe tener descendientes que puedan tener instancias: una clase hoja abstracta no sirve para nada. (Pueden aparecer clases abstractas como hojas en un marco de trabajo, pero al final deben especializarse.)

Una clase concreta no debe tener operaciones abstractas, pues de lo contrario sería necesariamente abstracta, pero una clase abstracta sí puede tener operaciones concretas. Las operaciones concretas son aquellas que pueden implementarse una vez y utilizarse tal cual en toda la jerarquía de subclases. En su implementación, las operaciones concretas sólo pueden utilizar características (atributos y operaciones) conocidas por la clase en la que se declaran. Uno de los propósitos de la herencia es situar estas operaciones en superclases abstractas para que puedan ser compartidas por todas las subclases. Una operación concreta puede ser polimórfica, es decir, puede ser sobrescrita por un método de una clase descendiente, pero no es necesario que lo sea (puede ser una operación hoja). Una clase cuyas operaciones están todas implementadas puede ser abstracta, pero debe ser declarada explícitamente como tal. Una clase con una o más operaciones no implementadas es automáticamente abstracta.

La misma semántica se aplica a los casos de uso. Un caso de uso abstracto define un fragmento de comportamiento que no puede aparecer por sí mismo, pero que puede aparecer en la definición de casos de uso concretos a través de relaciones de generalización, extensión o inclusión. Llevando el comportamiento común a un caso de uso abstracto, el modelo se reduce y es más fácil de comprender.

Existe una relación similar en otros clasificadores y elementos generalizables.

Notación

El nombre de una clase u operación abstracta debe aparecer en cursiva. También se puede utilizar la palabra **abstract** en la lista de propiedades que aparece después o debajo del nombre, por ejemplo, **Cuenta {abstract}**.

Véase también nombre de clase.

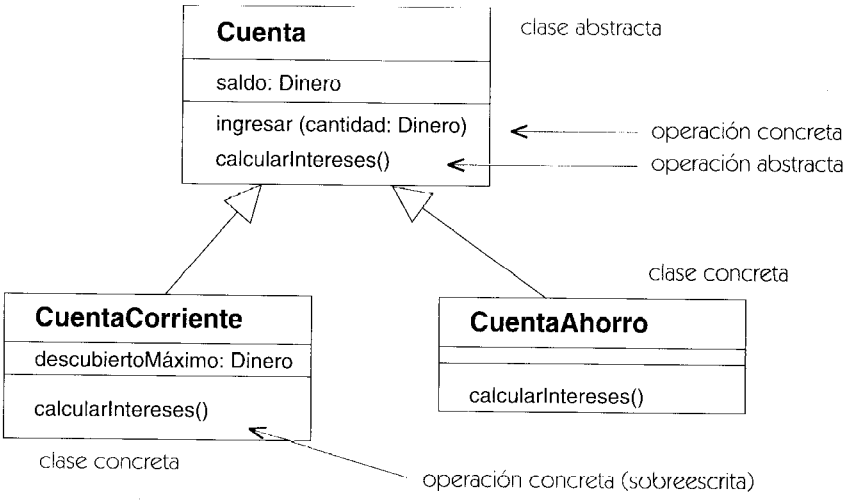


Figura 13.1 Clases abstractas y concretas

Ejemplo

La Figura 13.1 muestra una clase abstracta **Cuenta** con una operación abstracta —**calcularIntereses**— y una operación concreta —**ingresar**—, de la que se han declarado dos subclases concretas. Debido a que las subclases son concretas, cada una de ellas puede implementar la operación **calcularIntereses**. Los atributos son siempre concretos.

Discusión

La distinción entre modelar una clase como abstracta o como concreta no es tan importante como a primera vista podría parecer, pues es más una decisión de diseño sobre un modelo que una propiedad inherente al mismo. El estado de una clase puede cambiar con la evolución de un diseño. Una clase concreta puede modelarse como abstracta si se añaden subclases que incluyan todas sus posibilidades. Una clase abstracta puede ser modelada como concreta si nos encontramos con que las diferencias entre sus subclases son innecesarias y por tanto deben ser eliminadas, o si dichas diferencias pueden representarse utilizando diferentes valores de un atributo en lugar de subclases distintas.

Una forma de simplificar la decisión es adoptar el principio de diseño que afirma que todas las clases que no son hojas deben ser abstractas (y todas las clases hoja deben ser necesariamente concretas, excepto aquellas clases hoja pensadas para generalizaciones futuras). Esta regla no pertenece a UML: es un estilo que se puede o no adoptar. La razón de ser de esta "regla de la superclase abstracta" son las diferencias entre un método heredable de una superclase y un método de una clase concreta, cuyas necesidades no siempre están bien cubiertas con un único método. El método de la superclase debe hacer dos cosas: definir el caso concreto que deberán observar todos sus descendientes, e implementar el caso concreto para la clase específica en la que aparece. Estos objetivos generalmente entran en conflicto. Por otra parte, cualquier superclase no abstracta puede separarse en una superclase abstracta y una subclase hoja concreta. La superclase abstracta contendrá todos los métodos que deben ser heredados por

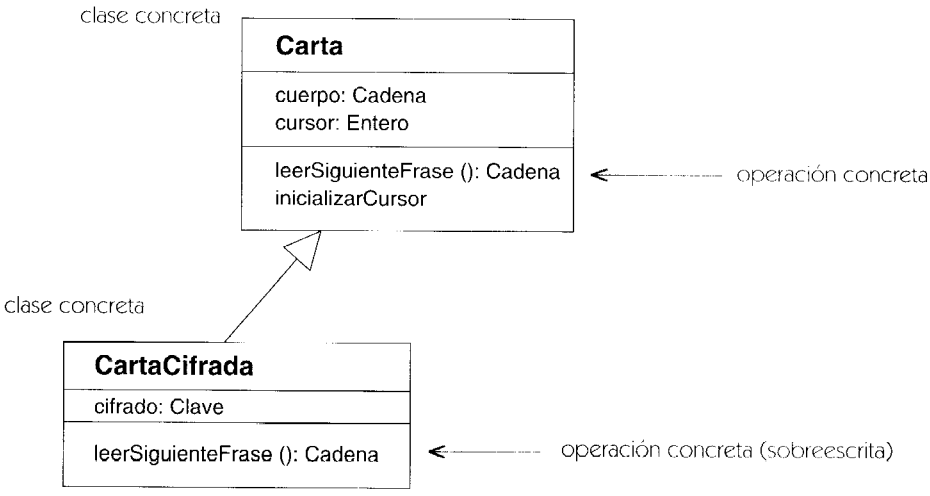


Figura 13.2 Una superclase concreta crea ambigüedad

las subclases, mientras que la subclase concreta contendrá los métodos necesarios para la clase instanciable específica. La regla de la superclase abstracta proporciona también una clara distinción entre una variable o parámetro que debe mantener el tipo concreto específico y una que puede tener el tipo de cualquiera de los descendientes de la superclase.

Considérese la declaración de la clase **Carta** de la Figura 13.2, la cual no sigue la regla de la superclase abstracta. Dicha clase tiene dos operaciones: **inicializarCursor** que pone el cursor al principio, y **leerSiguienteFrase** que devuelve el texto de la siguiente frase no leída. Sin embargo, la subclase **CartaCifrada** representa una carta cuyo texto ha sido cifrado y en la que la operación **leerSiguienteFrase** ha sido sobreescrita debido a que el texto debe ser descifrado antes de poder ser devuelto. La implementación de la operación es completamente diferente. El hecho de que **Carta** sea una superclase concreta no permite distinguir entre un parámetro que fuera una **Carta** ordinaria (no sobreescribible) y uno que pudiera ser tanto una **Carta** como una **CartaCifrada**.

El enfoque de la superclase abstracta permite distinguir la clase abstracta **Carta** (que puede ser tanto una carta cifrada como una no cifrada) y añadir la clase **CartaNoCifrada** para representar cada caso concreto, como se muestra en la Figura 13.3. En este caso, **leerSiguienteFrase** es una operación abstracta implementada por cada una de las subclases, mientras que **inicializarCursor** es una operación concreta, igual para las dos subclases. El modelo es simétrico.

Si se sigue la regla de la superclase abstracta, la declaración de clases abstractas puede determinarse automáticamente a partir de la jerarquía de clases, por lo que representarlo en diagramas es redundante.

Hay una excepción a la afirmación de que una clase hoja abstracta no sirve para nada: una clase abstracta declarada como espacio de nombres común de un conjunto de atributos y operaciones de ámbito global a la clase. Esta utilización es muy poco frecuente pero a veces es necesaria cuando se programa con lenguajes no orientados a objetos. Eso sí, debe evitarse siempre que sea posible, ya que las variables globales vulneran el espíritu del diseño orientado a objetos introduciendo dependencias globales. Una clase unitaria a menudo proporciona la misma funcionalidad de una manera más fácil de ampliar.

Véase [Gamma-95], Patrón Unitario (Singleton).

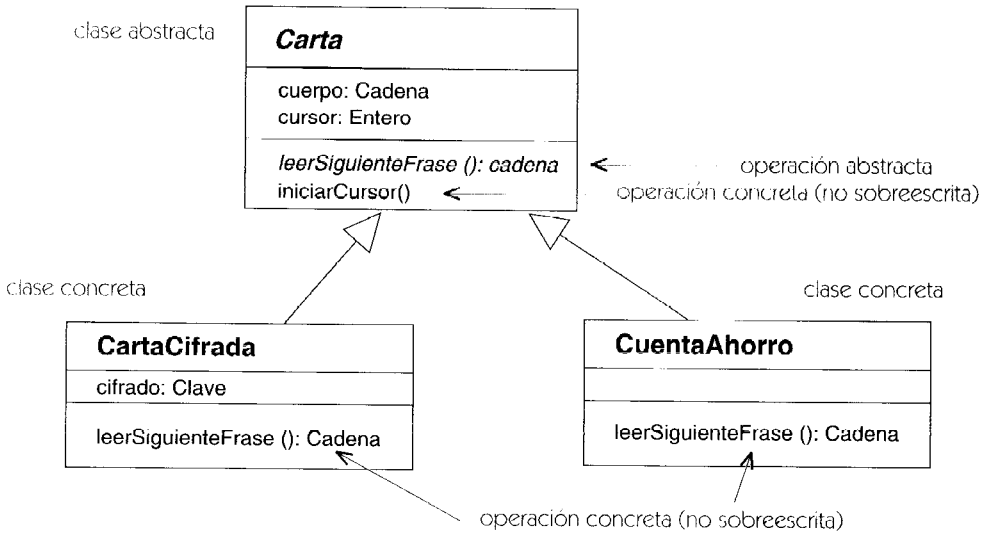


Figura 13.3 Una superclase abstracta elimina la ambigüedad

acceso

Dependencia de permiso que posibilita a un paquete hacer referencia a elementos de otro paquete.

Véase amigo/a, importar, visibilidad.

Semántica

Un paquete (cliente) que hace referencia a un elemento de otro paquete (provcedor) debe importar el paquete que contiene el elemento utilizando una dependencia «**access**» o «**import**» del paquete cliente hacia el paquete proveedor. Un paquete consigue implícitamente acceso a todos los paquetes importados por un paquete dentro del cual está anidado (esto es, los paquetes anidados pueden ver todo aquello que ven sus paquetes contenedores).

Un elemento de un paquete tiene acceso a todos los elementos que son *visibles dentro del paquete*. Las reglas de visibilidad se pueden resumir de la siguiente forma:

- Todo elemento definido en un paquete es visible dentro del mismo paquete.
- Si un elemento es visible dentro de un paquete, es visible en todos los paquetes anidados dentro del mismo.
- Si un paquete accede o importa otro paquete, todos los elementos definidos con visibilidad pública en el paquete accedido o importado son visibles dentro del paquete que los importa.
- Si un paquete es hijo de otro, los elementos definidos con visibilidad pública o privada en el paquete padre son visibles dentro del paquete hijo.

- Las dependencias de acceso e importación no son transitivas. Si A ve a B y B ve a C, no necesariamente A verá a C.

Una consecuencia de esto es que un paquete no puede ver el interior de sus propios paquetes anidados a menos que acceda a ellos y que los contenidos que haya dentro de dichos paquetes sean públicos. Algunas reglas más sobre visibilidad:

- Los contenidos de un clasificador, es decir, atributos, operaciones y clases anidadas, son visibles dentro del paquete si tienen visibilidad pública en el clasificador. Obsérvese que los contenidos no estructurados de un subsistema están gobernados por las reglas sobre visibilidad de los paquetes anteriormente establecidas, mientras que los atributos y operaciones se gobiernan por esta regla.
- Los contenidos de un clasificador son visibles dentro de un clasificador descendiente si tienen visibilidad pública o protegida en el clasificador.
- Todo lo que contiene un clasificador es visible para los elementos internos del clasificador, incluyendo métodos internos o máquinas de estados del clasificador.

El caso normal sencillo concierne a los elementos de paquetes que están al mismo nivel. En este caso, un elemento puede ver todos los elementos de su propio paquete y todos los elementos que tengan visibilidad pública de aquellos paquetes importados por su paquete. Una clase puede ver las características públicas de las clases que ve y también las características protegidas de sus ascendientes.

Notación

Una dependencia de acceso se representa mediante una flecha con línea discontinua que sale del paquete cliente y apunta al paquete proveedor. La flecha está etiquetada con la palabra clave «**access**».

Discusión

La Figura 13.4 muestra un ejemplo de acceso entre dos paquetes que se encuentran al mismo nivel. El paquete P puede acceder al paquete Q, pero el paquete Q no puede acceder a P. Las clases K y L del paquete P pueden ver la clase pública M del paquete Q, pero no la clase privada N. Las clases M y N no pueden ver ninguna clase del paquete P, aunque la visibilidad de la clase K es pública, debido a que Q no tiene acceso a P. Para que una clase de un paquete Q sea visible en un paquete P que está al mismo nivel que Q, la clase debe tener visibilidad pública, y Q debe ser accedido o importado por el paquete P.

La Figura 13.5 muestra un caso de visibilidad y declaraciones de acceso algo más complejo. El símbolo que precede al nombre del elemento representa la visibilidad del mismo fuera de su propio contenedor: + para visibilidad pública, # para protegida (visible sólo en descendientes) y – para visibilidad privada (no es visible fuera del contenedor).

La clase A puede ver las clases C y E porque está dentro de los paquetes X e Y.

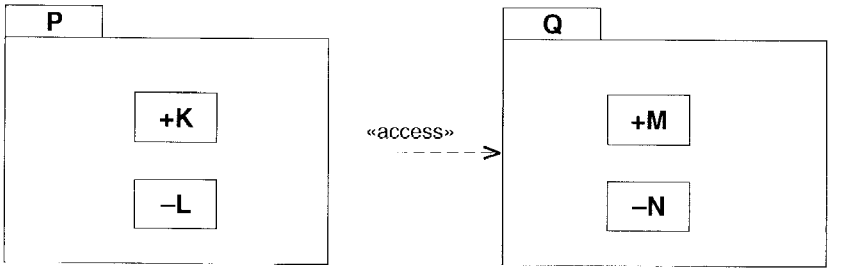


Figura 13.4 Acceso entre iguales

Las clases C y A pueden ver la clase D porque el paquete Y importa el paquete Z. La clase A está anidada dentro del paquete Y, por lo que puede ver todo aquello que ve Y.

Las clases A, C y E pueden ver a B porque están anidadas en el paquete X, que implementa el paquete V el cual contiene a B. Sin embargo, no pueden ver a F porque tiene visibilidad privada dentro de su propio paquete V. La clase F no es visible fuera de V.

La clase E no ve a D porque D está en el paquete Z, que no ha sido importado por el paquete X.

Ni la clase C ni la clase E pueden ver la clase A, debido a que A está en el paquete U, que no ha sido importado por ningún otro paquete.

Las clases B y F ven las clases D y E que están en paquetes que contienen al paquete V, al que ellas pertenecen. También pueden ver la clase C, que está en el paquete Y, pues dicho paquete es importado por el paquete X que engloba a V. El hecho de que F sea privada no limita el hecho de que pueda ver otras clases, pero evita que otras clases la vean a ella.

Las clases B y F se ven mutuamente porque están en el mismo paquete. La clase F es privada para las clases de otros paquetes, pero no para las que están en su mismo paquete.

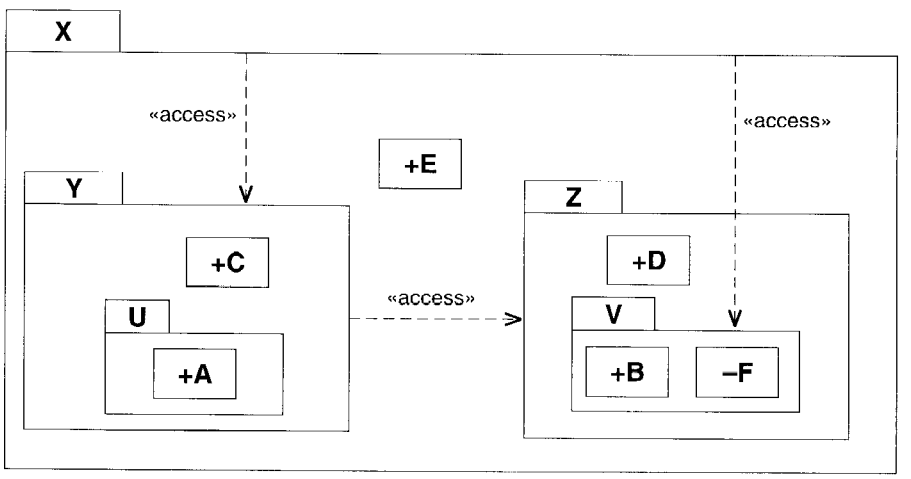


Figura 13.5 Reglas de acceso

acción

Computación atómica ejecutable que produce un cambio en el estado del modelo o que devuelve un valor. Contrastar con: actividad.

Véase también acción de entrada, acción de salida, transición.

Semántica

Una acción es una computación atómica, es decir, no puede ser finalizada por una orden de terminación externa. Puede asociarse a una transición en una máquina de estados (entre dos estados o bien dentro de un estado simple) o a un paso en una interacción. Generalmente es una operación primitiva o pseudo-primitiva sobre el estado de un sistema, a menudo sobre el estado de un objeto simple. Las acciones típicas son: asignación de un valor a un atributo, acceso a la información de atributos o enlaces, creación de nuevos objetos o enlaces, operaciones aritméticas sencillas y envío de señales a otros objetos. Las acciones son los pasos discretos con que se construye el comportamiento. Se supone que las acciones tienen que ser computaciones "rápidas" para no perjudicar el tiempo de respuesta del sistema. El sistema puede ejecutar varias acciones simultáneamente y hacer que compartan el tiempo entre ellas, pero su ejecución debe ser independiente.

Las acciones pueden estar asociadas a una transición, ejecutándose cuando la transición se dispara. También pueden aparecer como acciones de entrada o de salida de los estados. Hay acciones disparadas por transiciones que entran o dejan un estado. Todas las acciones son atómicas, esto es, se ejecutan completamente y sin interrupción de otras operaciones.

Una actividad es también una computación, pero puede tener una estructura interna y recibir una orden de terminación externa de una transición de un evento externo, y puede asociarse a un estado pero no a una transición. A diferencia de las acciones, las actividades pueden persistir indefinidamente hasta ser finalizadas externamente, aunque también pueden terminar por sí mismas. Una acción no puede ser finalizada externamente y debe estar asociada a una transición o a la entrada o salida de un estado, nunca a un estado en sí mismo.

Estructura

Una acción tiene un conjunto de objetos destino, una referencia a la señal a enviar o a la operación a realizar (a los que colectivamente se conoce como *solicitud*), una lista de valores para los argumentos y una expresión opcional de recurrencia que especifica la posible iteración.

Conjunto de objetos destino. Expresión de conjunto de objetos destino que produce un conjunto de objetos. En muchos casos, el conjunto contiene un objeto simple fijo. A todos los objetos del conjunto se les envía concurrentemente una copia del mensaje con la lista de argumentos correspondiente (es decir, se les comunica mediante difusión). Cada uno de ellos recibe y manipula independientemente una instancia distinta del mensaje. Si el conjunto está vació no ocurre nada.

Solicitud. Denota la declaración de una operación o una señal. O bien se envía la señal a los objetos o bien se llama a la operación (en el caso de que la operación devuelva un valor, el conjunto debe contener un único objeto).

Lista de argumentos. Una lista de argumentos. Cuando se evalúa, los valores de la lista de argumentos deben corresponderse con los parámetros de la señal o de la operación. Los argumentos se proporcionan como parte de la llamada o del envío de la señal.

Recurrencia. Expresión de una iteración, que especifica cuántas veces hay que realizar la acción, y que opcionalmente puede incluir variables de iteración. Esta expresión también puede describir una acción condicional (una iteración con una o ninguna repetición).

Tipos de acciones

Acción de asignación. Una asignación es una acción que da a un atributo de un objeto un valor determinado. La acción tiene una expresión para el objeto destino (el nombre de un atributo del objeto), y una expresión para el valor a asignar al atributo del objeto.

Acción de llamada. Una acción de llamada provoca una invocación de una operación sobre un objeto: es una invocación a una operación de un objeto. La acción tiene un nombre de mensaje, una expresión para la lista de argumentos, y una expresión de conjunto de objetos destino. El destino puede ser un conjunto de objetos, en cuyo caso la llamada se produce concurrentemente y la operación no deberá por tanto devolver ningún valor. Si la operación devuelve un valor, el destino debe ser un único objeto.

Las acciones de llamada son asíncronas. El que realiza la llamada debe esperar a que se complete la operación invocada antes de recibir de nuevo el control. Si la operación se implementa como un evento de llamada, el que realiza la llamada espera hasta que el receptor ejecute la transición disparada por la llamada para retomar el control. Si la ejecución de la operación devuelve valores, el que realiza la llamada los recibe cuando retoma el control.

Acción de creación. Una acción de creación produce la instanciación y creación de un objeto (véase creación). La acción tiene una referencia a una clase, y opcionalmente una operación presente en la clase con una lista de argumentos. La ejecución de la acción crea una nueva instancia de la clase, cuyos atributos obtienen los valores correspondientes a la evaluación de las expresiones de valor iniciales. Si existe una operación de creación explícita, se ejecuta. La operación puede sobrescribir la inicialización de los valores de los atributos, normalmente utilizando valores como argumentos de la acción de creación.

Acción de destrucción. Una acción de destrucción produce la destrucción de un objeto destino. No existen otros argumentos. El resultado de ejecutar la acción es la destrucción del objeto, junto con todos sus enlaces y con todas las partes de las que estuviera compuesto (véase composición).

Acción de retorno. Una acción de retorno provoca la transferencia del flujo de objeto a aquél que llamó a la operación. Esta acción sólo está permitida dentro de una operación invocada por una llamada. La acción tiene una lista opcional de valores devueltos que se pone a disposición de quien realizó la llamada cuando se le devuelve el control. Si la acción adjunta fue invocada de forma asíncrona, el que llamó a la operación debe escoger explícitamente entre recibir el mensaje de retorno (como una señal), o perderlo.

Acción de envío. Una acción de envío crea una instancia de una señal y la inicializa con los argumentos obtenidos al evaluar las expresiones de argumento de la acción. La señal se envía a los objetos del conjunto de objetos destino obtenido evaluando la expresión destino de la

acción. Cada objeto del conjunto recibe su propia copia de la señal. El que envía la señal tiene su propio hilo de control y continúa ejecutándose, ya que el envío de una señal es asíncrono. La acción tiene un nombre de señal, una lista de expresiones para los argumentos de la señal y una expresión de conjunto de objetos destino para los objetos destino.

Si se omite el conjunto de objetos destino, la señal se envía a uno o más objetos determinados por la señal y por la configuración del sistema. Por ejemplo, una excepción se envía a un ámbito cerrado que viene determinado por las políticas del sistema.

Acción de terminación. Una acción de terminación provoca la destrucción del objeto propietario de la máquina de estados que contiene la acción (esto es, se suicida). La destrucción de un objeto es un evento al que pueden responder otros objetos.

Acción no interpretada. Una acción no interpretada es una estructura de control u otra estructura no definida en UML.

Notación

UML no tiene un lenguaje fijo de acciones. Se espera que los que realizan los modelos elegirán un lenguaje de programación real para escribir las acciones. La siguiente adaptación de OCL se utiliza en este libro para escribir pseudocódigo, pero no es parte del estándar.

Acción de asignación

destino := expresión

Acción de llamada

conjunto-objetos-destino . nombre-operación (argumento$_{lista,}$)

Acción de creación

new nombre-clase (argumento$_{lista,}$)

Acción de destrucción

objeto . destroy ()

Acción de retorno

return expresión$_{lista,}$

Acción de envío

conjunto-objetos-destino . nombre-señal (argumento$_{lista,}$)

Acción de terminación

terminate

Acción no interpretada

if (expresión) **then** (acción) **else** (acción)

Si es necesario distinguir explícitamente entre llamada y envío, se pueden anteponer las palabras clave **call** o **send** a la expresión de forma opcional.

Discusión

La especificación de UML define un conjunto de acciones con la esperanza de que se añadirán otras en la implementación real de las herramientas que trabajen con UML. Esta decisión fue el resultado del compromiso entre el deseo de precisión y la necesidad de que los desarrolladores trabajaran con varios lenguajes, lo cual abarca un amplio espectro de conceptos semánticos. Existen muchas más variaciones en la semántica de ejecución entre lenguajes de programación de la que existe en las estructuras de datos o en el conjunto de estructuras de control disponibles. Las diferencias sutiles entre lenguajes son difíciles de plasmar de forma práctica, sin tener en cuenta si es posible hacerlo teóricamente. La selección de un lenguaje de programación como base para un lenguaje de acciones podría, sin embargo, tener el efecto de desanimar a los que utilizan otros lenguajes, lo cual no queríamos hacer. La semántica de acciones ha quedado pues algo incompleta y ambigua dentro del propio UML. Para precisar la semántica, hay que añadir a UML la semántica del lenguaje de acciones (a menudo un lenguaje estándar de programación) que se está utilizando. Algunos críticos se han quejado de que UML es impreciso debido a su libertad, pero lo es sólo hasta el grado de imprecisión del lenguaje de acción elegido. El auténtico defecto es que UML no impone una *lingua franca* de acciones y otras expresiones, lo cual es posible sólo a duras penas en el políglota mundo actual de la informática a pesar de las emotivas peticiones al respecto.

acción asíncrona

Solicitud en la cual el objeto emisor no hace una pausa para esperar los resultados. Envío.

Véase enviar, acción síncrona.

acción de entrada

Una acción realizada cuando se entra en un estado.

Véase también acción de salida, atómico/a, máquina de estados, transición.

Semántica

Un estado puede tener una acción de entrada opcional unida a él. Siempre que se entra en el estado se ejecuta la acción de entrada, después de las acciones asociadas a las transiciones entrantes o a la salida de los estados previos y antes de cualquier acción asociada a estados internos. La acción de entrada no se puede evadir por ningún medio. Está garantizado que se ha ejecutado siempre que el estado que la posee o un estado anidado dentro de él esté activo.

Orden de ejecución. En una transición entre dos estados con acciones de entrada y salida en las cuales la transición también tiene una acción, el orden de la ejecución es: Cualquier acción

de salida se ejecuta en el estado origen y los estados a los que engloba hacia fuera, pero no incluye a los estados que engloban tanto al estado de origen como al destino. Entonces se ejecuta la acción en la transición, después de lo cual se ejecutan las acciones de la entrada (las más externas primero) en los estados englobados dentro del estado, hacia abajo incluyendo el estado destino. La Figura 13.96 muestra algunas transiciones con acciones múltiples.

Notación

Una acción de entrada se codifica usando la sintaxis para una transición interna con el nombre de la **entrada** del evento simulado (que es por lo tanto una palabra reservada y no se puede utilizar como nombre real del evento).

entrada / secuencia-acción o entry / secuencia-acción

Sólo se puede unir una acción de entrada a un estado, pero la acción puede ser una secuencia, de modo que no se pierde generalidad.

Discusión

Las acciones de entrada y salida no son esenciales semánticamente (la acción de entrada se podría unir a todas las transiciones entrantes) pero facilitan la encapsulación de un estado para poder separar el uso externo de él de su construcción interna. Permiten definir acciones de inicialización y de finalización, sin la preocupación de que puedan evitarse. Son particularmente útiles con excepciones, porque definen las acciones que deben ser realizadas incluso si ocurre una excepción.

Una acción de entrada es útil para realizar una inicialización que deba hacerse cuando se entra al estado por primera vez. Un uso es inicializar las variables que capturan información durante un estado.

Por ejemplo, una interfaz de usuario para permitir la entrada, por un teclado numérico, de un número de teléfono o de un número de cuenta, limpiaría el número de la entrada. El reajuste de un contador de error, tal como el número de fallos en la contraseña, es otro ejemplo. La asignación de almacenamiento temporal, necesario durante el estado, es otro uso para una acción de entrada.

A menudo se utilizan juntas una acción de entrada y una acción de salida. La acción de entrada asigna recursos, y la acción de salida los libera. Incluso si ocurre una transición externa, se liberan los recursos. Esto es una buena manera de manejar errores de usuario y excepciones. Los errores de usuario accionan transiciones de alto nivel que abortan estados anidados, pero los estados anidados tienen una oportunidad de limpiar antes de perder el flujo de control.

acción de salida

Una acción que se realiza cuando se sale de un estado.

Véase también acción de entrada, atómico/a, máquina de estados, transición.

Semántica

Un estado puede tener asociada una acción de salida. Se ejecuta siempre que se salga del estado de cualquier manera, después de cualquier otra acción asociada a estados o transicciones más internos y antes de cualquier acción unida a estados más externos. La acción de salida no se puede evadir por ningún medio. Está garantizado que se ejecutará antes de que el flujo de objeto abandone el estado que la posee.

Las acciones de entrada y de salida no son esenciales semánticamente (la acción de salida se podría unir a todas las transiciones salientes), pero facilitan la encapsulación de un estado para poder separar el uso externo de él de su construcción interna. Permiten definir acciones de inicialización y de terminación, sin la preocupación de que puedan ser evitadas. Son particularmente útiles con excepciones, porque definen las acciones a realizar incluso si ocurre una excepción.

Notación

Una acción de salida se codifica usando la sintaxis para una transición interna con el evento simulado llamado **salida** (que es, por lo tanto, una palabra reservada y no se puede utilizar como nombre real del evento).

salida / secuencia-acción o exit / secuencia-acción

A un estado solamente se puede asociar una acción de salida, pero la acción puede ser una secuencia de acciones, así que no se pierde ninguna generalidad.

Discusión

Una acción de salida es útil para realizar una limpieza general cuando se sale de un estado. El uso más significativo de las acciones de salida es liberar el almacenamiento temporal y otros recursos asignados durante la ejecución del estado (generalmente, un estado con el detalle anidado).

A menudo se utilizan juntas una acción de entrada y una acción de salida. La acción de entrada asigna recursos, y la acción de salida los libera. Los recursos son liberados incluso si ocurre una excepción.

acción síncrona

Solicitud en la cual el objeto emisor se detiene a esperar una respuesta; también es una llamada. Contrastar con: acción asíncrona.

activación

Ejecución de una operación. Una activación (también conocida como foco de control) representa el período durante el cual un objeto realiza una operación, bien directamente o bien a

través de una operación subordinada. Modela tanto la duración de la ejecución como la relación de control entre la ejecución y los que hicieron la llamada a la misma. En un computador y lenguaje de programación convencionales, una activación se correspondería con un bloque de llamada en la pila.

Véase llamada, diagrama de secuencia.

Semántica

Una activación es una instancia de ejecución de una operación, que incluye el período durante el cual la operación llama a otras operaciones subordinadas (véase llamada). Su contexto comprende un conjunto de variables locales accesibles únicamente por la activación, una posición actual dentro del método (u otra descripción de comportamiento), y una referencia (la *referencia de retorno*) a la activación que representa el contexto de llamada, que retoma el control cuando termina la activación actual. Una activación sin referencia de retorno debe ser el resultado de una transición sobre la máquina de estados de la clase de un objeto activo; cuando se completa la activación, la máquina de estados simplemente espera el próximo evento.

Obsérvese que esta definición describe un procedimiento ordinario, tal y como se implementa en una típica máquina de von Neumann. Pero expresarlo de forma general significa aplicarlo también a un entorno distribuido, en el cual no hay memoria compartida y en el que el marco de pila está formado por una lista enlazada de activaciones en diferentes espacios de memoria.

Notación

Una activación se representa en un diagrama de secuencia utilizando un rectángulo alto y delgado (una barra vertical hueca) cuya parte superior está alineada con el momento de su iniciación y cuya parte inferior lo está con el momento de su finalización. La operación a realizar se representa mediante una etiqueta de texto al lado del símbolo de activación o en el margen izquierdo, dependiendo del estilo.

De forma alternativa, también es posible que el símbolo de mensaje entrante indique la operación, en cuyo caso se puede omitir la etiqueta en la propia activación. Si el flujo de objeto es de procedimientos, entonces el extremo superior del símbolo de activación estará en la punta de la flecha del mensaje entrante que inicia la acción, y del extremo inferior saldrá la flecha del mensaje de retorno.

Si hay actividad concurrente entre varios objetos, la activación muestra la ejecución de un objeto concurrente. A menos que los objetos se comuniquen, las actividades concurrentes son independientes, y no es relevante reflejar sus tiempos relativos de ejecución.

Cuando se trata de código de procedimientos, una activación muestra o bien la duración durante la cual un procedimiento está activo en el objeto o bien el tiempo durante el que está activo un procedimiento subordinado llamado por el procedimiento original, posiblemente en algún otro objeto. En otras palabras, se muestran simultáneamente todas las activaciones de procedimientos anidados. Este conjunto de activaciones anidadas simultáneas forma el marco de pila de la computación en un computador convencional. En caso de una segunda llamada a un objeto con una activación existente, el segundo símbolo de activación se dibuja ligeramente a la derecha del primero, de forma que den la sensación de estar apilados. Las llamadas

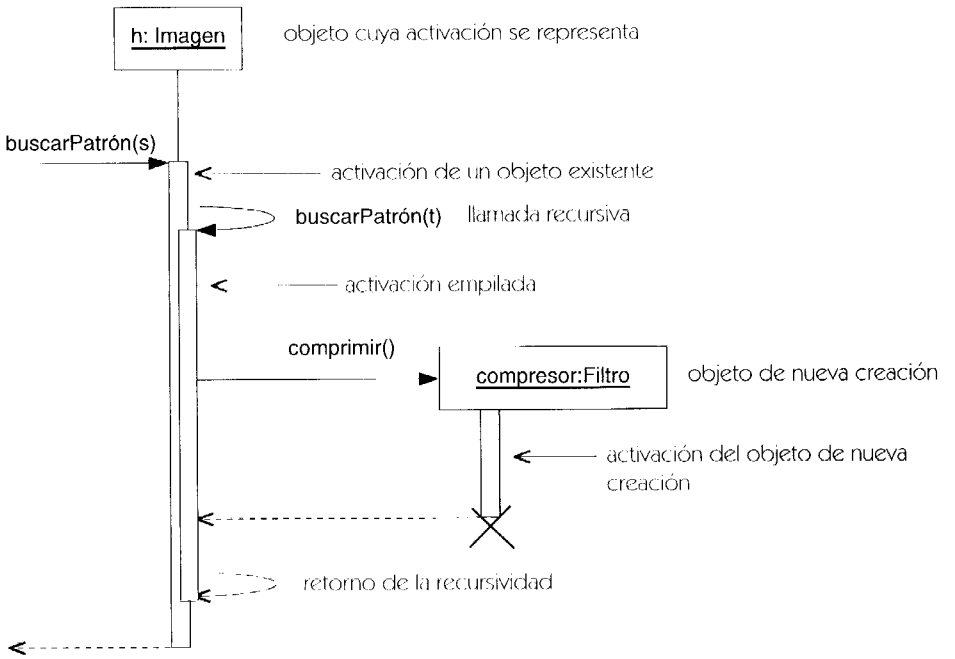


Figura 13.6 Activaciones

apiladas pueden tener cualquier nivel de anidamiento. Las llamadas pueden ser a la misma operación (llamada recursiva) o a diferentes operaciones del mismo objeto.

Ejemplo

La Figura 13.6 muestra las activaciones producidas a partir de las correspondientes llamadas, incluyendo una llamada recursiva.

actividad

Ejecución no atómica en curso dentro de una máquina de estados. Contrastar con: acción.

Véase también transición de finalización, estado.

Semántica

Una actividad es la ejecución de una subestructura dentro de una máquina de estados, pero de una subestructura que tiene una duración y posibles puntos de interrupción. Cualquier transición que fuerce la salida de la región de control aborta la actividad. Una actividad no termina por el hecho de que se dispare una transición interna, porque no hay cambio de estado, pero la acción asociada a la transición interna sí podría terminarla explícitamente.

Una actividad se puede modelar mediante estados anidados, utilizando una referencia a una submáquina o empleando una expresión de actividad.

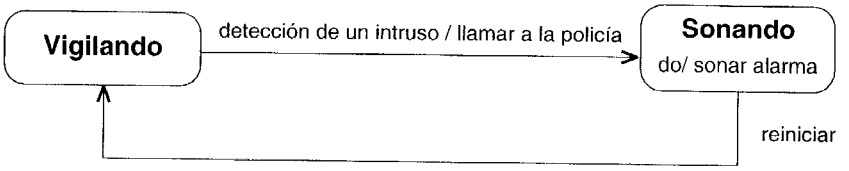


Figura 13.7 Acción y actividad

Ejemplo

La Figura 13.7 muestra un sistema de alarma que ilustra la diferencia entre una acción y una actividad. Cuando se produce el evento **detección de un intruso**, el sistema dispara una transición. Como parte de la transición, se lleva a cabo la acción **llamar a la policía**, que es una acción puesto que es algo atómico (y normalmente rápido). Mientras se ejecuta la acción no se acepta ningún evento. Después de realizar la acción el sistema pasa al estado **Sonando**, y mientras está en este estado realiza la actividad **sonar alarma**. Una actividad tarda un tiempo en realizarse, tiempo durante el cual la actividad puede ser interrumpida por algún evento. En este caso, la actividad **sonar alarma** no termina por sí sola, ya que continúa ejecutándose mientras el sistema se encuentre en el estado **Sonando**. Cuando se produce el evento **reiniciar**, se dispara la transición que devuelve al sistema al estado **Vigilando**.

Cuando el estado **Sonando** deja de estar activo, su actividad **sonar alarma** termina.

activo

Estado en el que se ha entrado y del que aún no se ha salido; estado que mantiene un objeto.

Véase también clase activa, objeto activo.

Semántica

Un estado pasa a estar activo cuando se dispara una transición de entrada, y deja de serlo cuando se dispara una transición de salida. Si un objeto tiene un hilo de control, entonces al menos hay un estado activo (en un caso extremo, una clase puede tener sólo un estado, lo que supone que la respuesta ante un evento es siempre la misma). Si un estado está activo dentro de la máquina de estados de la clase de un objeto, se dice que el objeto *mantiene* el estado, o que *está* en ese estado.

Un objeto puede tener más de un estado simultáneamente. El conjunto de estados activos se denomina configuración del estado activo. Si un estado anidado está activo, todos los estados que lo contienen están también activos. Si el objeto permite concurrencia, puede haber más de un subestado activo al mismo tiempo. Cada transición afecta, al menos, a unos pocos estados de la configuración del estado activo. Ante una transición, los estados activos no afectados por la misma permanecen activos.

Un estado compuesto puede ser secuencial o concurrente. Si es secuencial y está activo, entonces habrá exactamente uno de sus subestados activo. Si es concurrente y está activo, en-

tonces todos los subestados que contiene estarán activos. En otras palabras, un estado compuesto se expande en un árbol Y-O de subestados activos, de forma que en cada nivel hay siempre estados activos.

Una transición que atraviesa en la frontera de un estado compuesto debe estructurarse de forma que mantenga las restricciones de concurrencia. Una transición en un estado secuencial compuesto normalmente tiene un estado origen y otro destino, y disparar una transición de este tipo no cambia el número de estados activos. Sin embargo, una transición en un estado concurrente compuesto tendrá un estado origen y uno destino para cada una de las regiones en que se divide el estado concurrente compuesto, y una transición de este tipo se denomina división. Si se omiten una o más regiones como destino, el estado inicial de cada región omitida está presente de forma implícita como destino, pero si una de las regiones carece de estado inicial, entonces se dice que el modelo está mal modelado. El disparar una transición de este tipo incrementa el número de estados activos, situación que se invierte a la salida del estado concurrente compuesto.

Véase máquina de estados, donde se lleva a cabo una completa discusión sobre la semántica de los estados concurrentes y las transiciones complejas.

Ejemplo

La parte superior de la Figura 13.8 muestra un ejemplo de máquina de estados que tiene estados compuestos, tanto secuenciales como concurrentes. Se han omitido las transiciones para

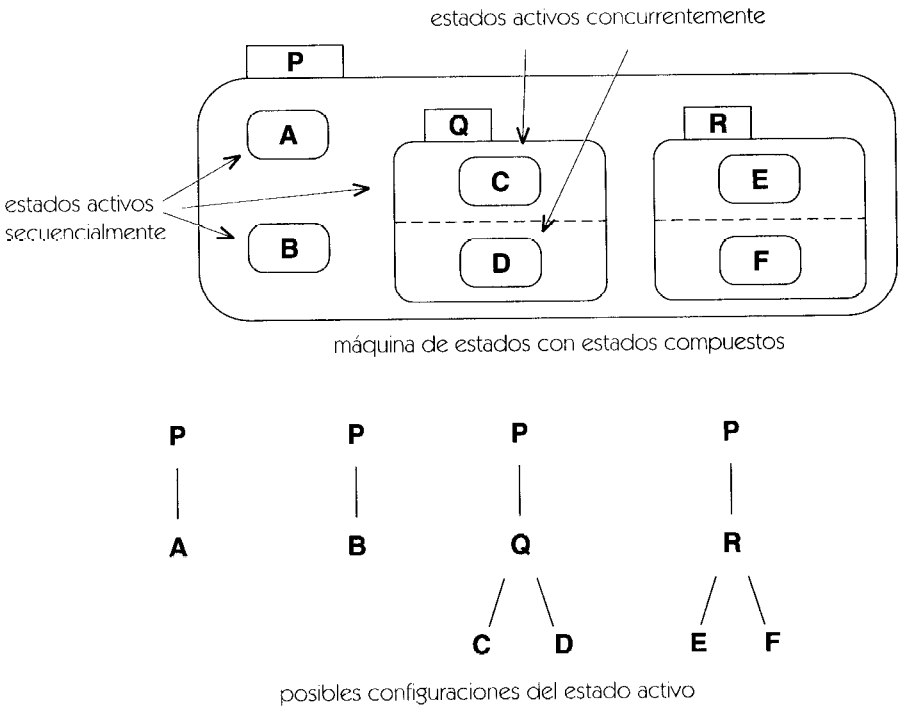


Figura 13.8 Estados activos concurrentemente

centrar la atención en los estados. La parte inferior de la figura muestra las diversas configuraciones de estados que pueden estar activas concurrentemente.

En este ejemplo hay cuatro posibles configuraciones de estados activos. Sólo los estados hoja son concretos, ya que los estados de más alto nivel son abstractos, es decir, un objeto no puede estar en uno de ellos sin estar también en uno de sus estados hoja anidados. Por ejemplo, un objeto no puede estar en el estado Q sin estar en alguno de los subestados de Q. Como Q es concurrente, tanto C como D tienen que estar activos si Q lo está. Cada estado hoja se corresponde con un hilo de control. En un ejemplo más amplio, el número de posibles configuraciones crecería exponencialmente y sería imposible mostrarlas todas, de aquí la ventaja de la notación.

actor

Abstracción de las entidades externas a un sistema, subsistemas o clases que interactúan directamente con el sistema. Un actor participa en un caso de uso o conjunto coherente de casos de uso para llevar a cabo un propósito global.

Véase también caso de uso.

Semántica

Un actor abstrae y caracteriza a un usuario externo o a un conjunto de usuarios externos relacionados que interactúan con el sistema o clasificador. Es una idealización con un propósito y significado concretos, que puede no corresponderse con objetos físicos. Un objeto físico puede combinar propósitos dispares y por tanto ser modelado por varios actores y viceversa, diferentes objetos físicos podrían incluir el mismo propósito, y por tanto ser modelados como el mismo actor. El objeto usuario puede ser una persona, un sistema informático, otro subsistema u otro tipo de objeto. Por ejemplo, el conjunto de actores de un sistema de redes de computadores podría incluir los actores **Operador**, **Administrador del Sistema**, **Administrador de Bases de Datos** y **Usuario** sin más. También podría haber actores no humanos como por ejemplo **Cliente Remoto**, **Reloj Maestro** o **Impresora de Red**.

Cada actor define un conjunto de roles que los usuarios de un sistema asumen cuando interactúan con el mismo. El conjunto completo de actores describe todas las diferentes formas de comunicación entre los usuarios externos y el sistema. Cuando se implementa el sistema, los actores se implementan como objetos físicos. Un objeto físico puede implementar más de un actor si puede llevar a cabo todos sus roles. Por ejemplo, una persona puede ser tanto vendedor como cliente de una misma tienda; estos actores no están inherentemente relacionados, pero ambos pueden ser implementados por una persona. Cuando se diseña un sistema, los diferentes actores se realizan mediante clases de diseño (véase realización).

Las diferentes interacciones de los actores con un sistema se cuantifican en casos de uso. Un caso de uso es una pieza coherente de funcionalidad que implica a un sistema y a sus actores para llevar a cabo algo que tiene sentido para los actores. Un caso de uso puede implicar a uno o más actores, de la misma forma que un mismo actor puede participar en más de un caso de uso. Los actores vienen determinados por los casos de uso que realizan y por los roles que

desempeñan en los diferentes casos de uso. Un actor que no participa en ningún caso de uso carece de sentido.

Un modelo de casos de uso caracteriza los tipos de comportamiento que proporciona una entidad, como por ejemplo un sistema, un subsistema o una clase, en sus interacciones con entidades externas. En el caso de un sistema, los actores pueden ser realizados tanto por usuarios humanos como por otros subsistemas. Cuando se trata de un subsistema o de una clase, los elementos externos pueden ser actores del sistema global o pueden ser otros elementos internos del sistema, como otros subsistemas u otras clases.

Las instancias de actores se comunican con el sistema mediante el envío y recepción de instancias de mensajes (señales y llamadas) desde y hacia instancias de casos de uso, y en el nivel de realización, desde y hacia los objetos que implementan el caso de uso. Todo lo anterior se expresa mediante asociaciones entre el actor y el caso de uso.

Un actor puede listar el conjunto de señales que envía y que recibe, y mantener una lista de las interfaces que ofrece y que requiere. Las interfaces de un actor deben ser compatibles con las de los casos de uso con los que se comunica. En otras palabras, un actor debe recibir todas las señales que un caso de uso sea capaz de enviarle y no debe mandarle al caso de uso señales que éste no sea capaz de recibir. Las interfaces de un actor restringen las clases con las que pueda corresponderse (que puedan implementar su comportamiento). Un actor puede también tener una lista de atributos que caracterice su estado.

Generalización

Pueden existir semejanzas entre dos o más actores; esto es, pueden comunicarse con el mismo conjunto de casos de uso de la misma manera. Esta semejanza se expresa mediante la generalización en otro actor, posiblemente abstracto, que modele los aspectos comunes. Los actores descendientes heredarán los roles y relaciones con casos de uso del actor antecesor. Una instancia de un actor descendiente siempre se puede utilizar en aquellos casos en los que se espera una instancia del actor antecesor (principio de capacidad de sustitución). Un descendiente incluye los atributos y operaciones de sus antecesores.

Notación

Un actor se puede representar mediante un símbolo de clase (rectángulo) con el estereotipo «**actor**». El icono estándar de un actor es el "monigote", con el nombre del actor debajo. El actor puede tener compartimentos que muestren los atributos y eventos que recibe, y también dependencias que muestren los eventos que envía, como cualquier clasificador normal (Figura 13.9).

Figura 13.9 Símbolo de actor

agregación

Forma de asociación que especifica una relación todo-parte entre un agregado (el todo) y las partes que lo componen.

Véase también composición.

Semántica

Una *asociación binaria* puede declararse como agregación, o lo que es lo mismo, como relación todo-parte. Un extremo de la asociación se designa como agregado mientras que el otro no se marca de forma especial. No pueden ser agregados (o compuestos) ambos extremos, pero sí que pueden estar los dos sin marcar, en cuyo caso no se trataría de una agregación.

Los enlaces instanciados de las asociaciones de agregación obedecen ciertas reglas. La relación de agregación es transitiva y antisimétrica a través de los enlaces de la agregación, incluso a través de aquellos que pertenecen a diferentes relaciones de agregación. La propiedad transitiva significa que tiene sentido decir que "B es parte de A" si existe una ruta de enlaces de agregación desde B hasta A en sentido directo (en este ejemplo, de las partes hacia el todo). La antisimetría obliga a que no existan bucles en las rutas dirigidas de los enlaces de agregación. Esto es, no es posible que ningún objeto sea parte de sí mismo directa o indirectamente. Uniendo los dos roles, el grafo de enlaces de agregación de todas las asociaciones de agregación es un grafo parcialmente ordenado, un grafo sin bucles (el caso más común de orden parcial sería un árbol). En la Figura 13.10 se muestra un ejemplo.

Una ruta dirigida de enlaces desde el objeto B hasta el objeto A implica que existe una ruta dirigida de asociaciones de agregación de la clase B a la clase A, pero la ruta de asociaciones puede implicar bucles en los que la misma clase aparezca más de una vez. Una ruta dirigida de asociaciones de agregación de una clase a sí misma es una *recursividad*.

Existe una relación más fuerte de agregación llamada composición. Un compuesto es un agregado con las restricciones adicionales de que un objeto puede ser parte sólo de un compuesto, y de que el objeto compuesto tiene la responsabilidad de la disponibilidad de sus partes a la hora de crearlas y destruirlas.

Véase composición para más detalles.

En la agregación simple, una parte puede pertenecer a más de un agregado, y lo que es más, puede existir independientemente del agregado. A menudo el agregado "necesita" las partes, en el sentido de que puede ser considerado como una colección de partes, pero las partes pueden existir por sí mismas, sin tener que ser vistas como partes únicamente. Por ejemplo, una ruta es poco más que un conjunto de segmentos, pero un segmento puede existir independientemente de si pertenece o no a una ruta, y además el mismo segmento puede pertenecer a más de una ruta.

Véase asociación y extremo de asociación para más información sobre las propiedades de la agregación.

Notación

Una agregación se representa mediante un rombo o diamante hueco en el extremo de la línea de la asociación que se conecta a la clase agregada (véase Figura 13.11). Si la agregación es una composición, su representación es un rombo relleno (véase Figura 13.54). No pueden tener indicadores de agregación ambos extremos.

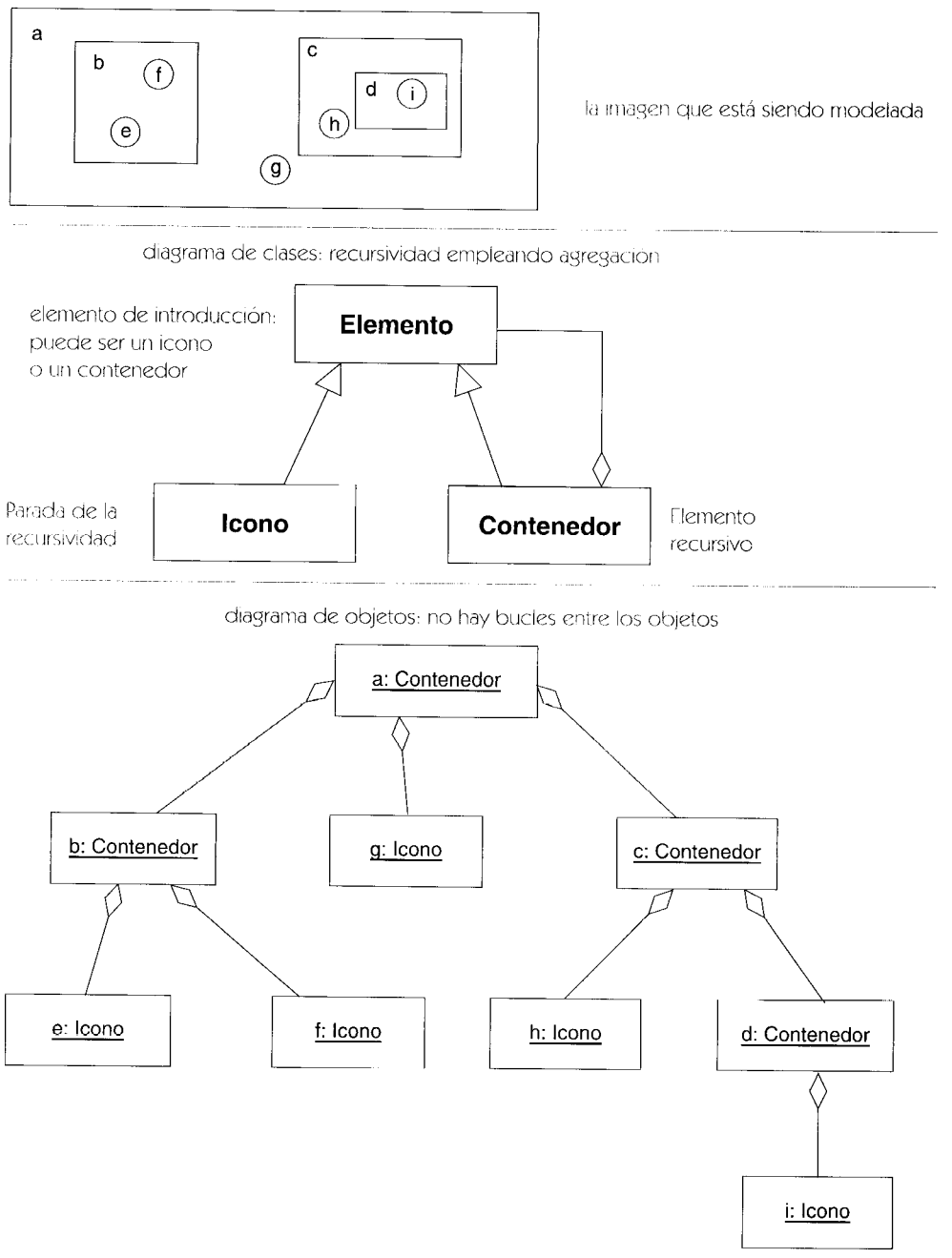


Figura 13.10 Las agregaciones de objetos son acíclicas

Una clase agregada tiene múltiples partes, pero cada relación entre la clase agregada y cada una de las partes es una asociación distinta (véase Figura 13.12).

Si hay dos o más asociaciones de agregación para la misma clase agregada, se pueden representar como un árbol combinando los extremos de la agregación en un único segmento

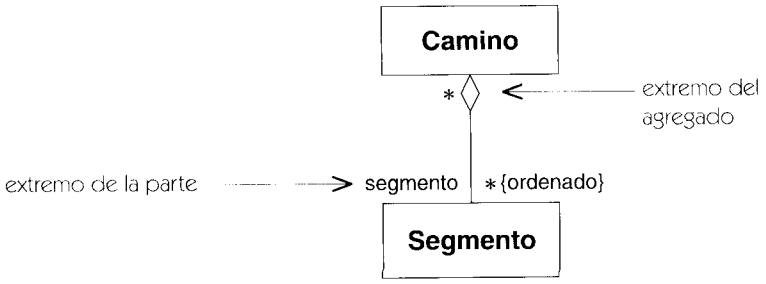


Figura 13.11 Notación de agregación

(véase Figura 13.13), pero esto requiere que todos los adornos de los extremos de la agregación sean consistentes; por ejemplo, deben tener la misma multiplicidad. Dibujarlo mediante un árbol es una mera cuestión de presentación que no aporta ninguna semántica adicional.

Discusión

La distinción entre asociación y agregación es muy a menudo un asunto de gustos más que una diferencia semántica real. Hay que recordar que la agregación es una asociación. La agregación conlleva la idea de que el agregado es la suma de sus partes, y en efecto, la única semántica adicional que aporta a la asociación es la restricción que impide que en los enlaces de agregación existan bucles, que es importante conocer. Otras restricciones, como la existencia de dependencia, las expresa la multiplicidad, no la agregación. A pesar del poco significado que

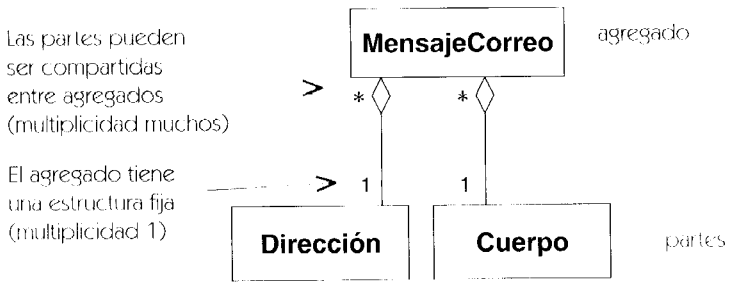


Figura 13.12 Un agregado con varias partes

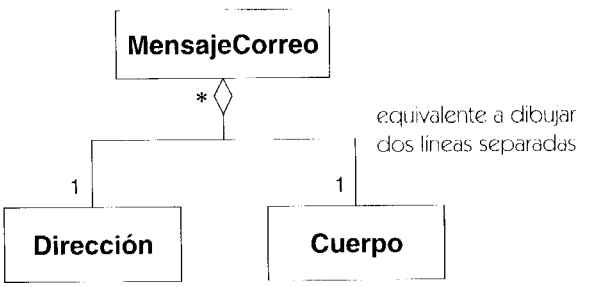


Figura 13.13 Representación en árbol para múltiples agregaciones de la misma clase

aporta, todo el mundo está convencido de que es necesaria (por diferentes razones). Piense en ello como en un placebo de modelado.

Hay algunas propiedades secundarias conectadas con la agregación, pero no son suficientes para requerir formar parte de la definición. Algunas de ellas son la propagación de operaciones desde el agregado a las partes (por ejemplo una operación mover), o la asignación de memoria compacta (de forma que el agregado y sus partes puedan cargarse eficientemente con una única transferencia de memoria). Algunos autores distinguen varios tipos de agregación, pero dichas distinciones son tan sutiles que probablemente sean innecesarias para un modelado común.

La agregación es una propiedad que trasciende una asociación particular. Se pueden componer agregaciones sobre diferentes pares de clases, y el resultado es una agregación. La agregación impone la restricción sobre las instancias de todas las asociaciones de agregación (incluyendo las asociaciones de composición) de que no pueden existir bucles entre los enlaces de agregación, incluyendo los enlaces de asociaciones diferentes. En cierto sentido, la agregación es una especie de generalización de la asociación en la cual las restricciones y algunas operaciones se aplican a asociaciones de muchos tipos específicos.

La composición tiene una semántica más específica, como el contenido físico y algunas nociones de propiedad. La composición resulta apropiada cuando cada parte es propiedad de un objeto y cuando las partes no tienen vida independiente separadas de su propietario. Es más útil cuando las partes tienen que ser asignadas e inicializadas en el momento en que se crea su propietario y no sobreviven a la destrucción del mismo. Los atributos de una clase tienen estas propiedades y pueden por ello ser considerados una forma de composición, aunque no se modelan explícitamente como tales. Utilizando la composición, se puede evitar la sobrecarga del gestor de memoria y el peligro de los punteros sueltos u objetos huérfanos. También resulta apropiado para situaciones en las cuales se han aislado una gran cantidad de atributos en una clase distinta por motivos de encapsulamiento y manipulación, pero los atributos realmente pertenecen a la clase principal. Las clases contenedoras utilizadas para implementar asociaciones son también candidatos obvios para partes compuestas, aunque por lo general, deberían ser generadas por un generador de código y no modeladas explícitamente. Obsérvese que una parte compuesta, como por ejemplo una clase contenedora, puede contener referencias (punteros) a partes no compuestas, pero los objetos referenciados no se destruyen cuando se destruye el objeto que los referencia.

agregación compuesta

Véase composición.

agregado

Clase que representa el todo en una asociación de agregación (todo-parte).

alcance

Extensión de un miembro de clasificador, tal como un atributo, operación o nombre de rol; esto es, si representa un valor en cada instancia o bien un valor que comparten todas las instancias

de un clasificador. Cuando se utiliza en solitario sin calificación alguna, indica un alcance de propietario.

Véase creación, alcance de propietario, alcance de destino.

Semántica

El alcance puede ser o bien de propietario o bien de destino.

Alcance de propietario. Indicación de si existe o no una ranura de atributo para cada instancia de una clase, o de si hay una sola ranura con el nombre dado para toda la clase.

instancia	Cada instancia del clasificador posee su propia copia independiente de una ranura de atributo o bien su propio conjunto de objetos asociados. Los valores de una ranura son independientes de los valores que haya en las otras ranuras. Ésta es la situación por defecto. Para una operación, la operación se le aplica a un objeto individual (en una operación normal).
clase	El clasificador en sí posee una copia de la ranura de atributo o del conjunto de objetos asociados. Todas las instancias del clasificador comparten el acceso a una sola ranura. Para una operación, la operación es aplicable a toda la clase, tal como una operación de creación o una operación que proporcione estadísticas relativas a todo un conjunto de instancias. Esta operación no se le puede aplicar a una instancia.

Alcance de destino. Una opción que indica si los valores de un atributo o los valores del destino de una asociación son o no instancias (por defecto) o clasificadores.

instancia	Cada ranura de atributo o cada enlace de la asociación contiene una referencia de una instancia del clasificador destino. El número de enlaces se ve limitado por la multiplicidad. Ésta es la situación por defecto.
clase	Toda ranura de atributo o todo enlace de la asociación contiene una referencia de la clase destino en sí. La información de la asociación queda por tanto determinada en tiempo de modelado y no cambia durante el tiempo de ejecución; de hecho no es necesario almacenarla en todos los objetos. Los enlaces implican a la clase en sí, no a sus instancias. Esto puede ser útil para cierta información de implementación, pero para la mayoría de los propósitos de modelado esta capacidad se puede ignorar.

Discusión

Los atributos o asociaciones de alcance de clase proporcionan valores globales para toda una clase y por tanto deberían utilizarse con cuidado o evitarse por completo, aunque los proporcionan casi todos los lenguajes de programación orientados a objetos. El problema es que implican una información global, lo cual viola el espíritu del diseño orientado a objetos. Además, la información global se vuelve problemática en sistemas distribuidos, porque impo-

ne unos accesos centrales en una situación en que los objetos de la clase pueden estar distribuidos en muchas máquinas. En lugar de utilizar las clases como si fuesen objetos con estado, es mejor introducir objetos explícitos para almacenar en ellos cualquier información compartida que pudiera precisarse. Tanto el modelo como los costes se vuelven más evidentes.

Los constructores (operación de creación, operaciones de fábrica) tienen necesariamente un alcance de código del nivel de clase, porque (todavía) no hay ninguna instancia sobre la que puedan operar. Esto es una aplicación correcta y necesaria del alcance de clase. Las otras clases de operaciones cuyo alcance original es del nivel de clase tienen las mismas dificultades que los atributos, a saber, implican una información global centralizada relativa a las instancias de las clases, lo cual no resulta práctico en sistemas distribuidos.

El alcance de destino tiene una utilidad limitada y sólo debería utilizarse en circunstancias especiales generalmente, sólo para propósitos detallados de programación.

alcance de destino

Especificación de si un valor es una instancia o un clasificador.

Véase alcance.

Discusión

El alcance de destino se usa sobre todo para almacenar clases como valores de atributos o como destinos de asociaciones. Tiene una utilidad limitada. La palabra alcance sin aditamentos significa alcance de propietario.

alcance de propietario

Se trata de una indicación de si la característica es aplicable a un objeto individual o si es compartida por toda una clase.

Véase también alcance, alcance de destino.

Semántica

El alcance de propietario indica si existe o no una ranura de atributo distinta para cada instancia de una clase, o si existe una única ranura para toda la clase. Para un operador, el alcance de propietario indica si una operación se aplica a una operación o bien a la clase en sí (tal como sucede en un operador de creación). En algunas ocasiones se denomina simplemente alcance. Los valores posibles son:

instancia	Cada instancia de clasificador posee su propia copia exclusiva de una ranura de atributo. Los valores de una ranura son independientes de los valores de otras ranuras. Ésta es la situación normal. Para un operador, el operador es aplicable a un objeto individual.

clase

El clasificador en sí posee una copia de la ranura de atributo. Todas las instancias del clasificador comparten el acceso a la única ranura. Si el lenguaje permite clases como objetos reales entonces esto es un atributo de la clase en sí como objeto.

Para un operador, el operador se aplica a toda la clase, como sucede en un operador de creación o en un operador que proporcione estadísticas relativas a todo un conjunto de instancias.

Notación

Los atributos u operadores con alcance de clase se subrayan (Figura 13.14). Los atributos u operadores con alcance de instancia no se subrayan.

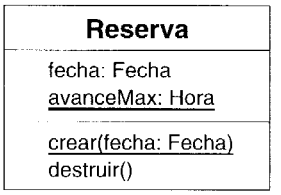


Figura 13.14 Atributo y operación con alcance de clase

Discusión

Para una asociación, se diría si la posición de origen de un enlace contiene instancias o clasificadores. Pero esta información se puede especificar como el alcance de destino en la otra dirección, así que el alcance de propietario es innecesario y, por tanto, no se usa para las asociaciones.

alcance original

Indicador de si una ranura es propiedad de una instancia o de una clase.

Véase alcance.

amigo/a

Una dependencia de uso que concede permiso al cliente para tener acceso al proveedor, incluso aunque el cliente no tenga la suficiente visibilidad para tener acceso al proveedor.

Véase también acceso, importar, visibilidad.

Semántica

La dependencia amigo se utiliza para conceder permiso a una clase u operación para usar el contenido de una clase, aunque el permiso fuera insuficiente. Es una excepción explícita a las

reglas normales de permiso entre los elementos afectados. Esta capacidad se debe utilizar cuidadosa y económicamente.

Notación

La dependencia amigo se representa como una flecha de línea discontinua desde la operación o la clase que adquiere el permiso a la clase cuyo contenido se hace disponible; la palabra clave de estereotipo «**friend**» se une a la flecha.

análisis

Etapa de un sistema que captura los requisitos y el dominio del problema. El análisis se centra en lo que hay que hacer, mientras que el diseño se centra en cómo hacerlo. En un proceso iterativo, estas etapas no tienen por qué realizarse de forma secuencial. El resultado de esta etapa se representa mediante modelos del nivel de análisis, especialmente la vista de casos de uso y la vista estática. Contrastar análisis con diseño, implementación y despliegue.

Véase fases de modelado, proceso de desarrollo.

antecesor

Elemento que se encuentra siguiendo una ruta en una o más relaciones padre.

Véase generalización, padre.

argumento

Valor específico de un parámetro.

Véase también ligadura, parámetro, principio de capacidad de sustitución.

Semántica

Una instancia de un mensaje en tiempo de ejecución tiene una lista de valores de los argumentos, cada uno de los cuales es un valor cuyo tipo debe ser consistente con el tipo declarado por el parámetro correspondiente en la declaración de la operación o señal. Un valor es consistente si su clase o tipo de dato es la misma o un descendiente de la declarada como tipo del parámetro. Según el principio de sustitución, se puede utilizar el valor de un descendiente en cualquier sitio donde se haya declarado un antecesor como tipo del parámetro. La implementación de un valor depende del simulador o entorno de ejecución en el que aparece.

En una colaboración o en una máquina de estados pueden aparecer expresiones que impliquen acciones. Dentro de estas expresiones, las llamadas y los envíos de mensajes requieren especificaciones de argumentos, que son también expresiones. Cuando se evalúan esas expresiones en tiempo de ejecución, deben evaluarse utilizando valores consistentes con los parámetros declarados con los que se corresponden.

Sin embargo, al realizar la ligadura de una plantilla, los argumentos aparecen dentro de un modelo UML durante el propio modelado. En estos casos, los argumentos se representan mediante expresiones en algún lenguaje, normalmente un lenguaje de restricciones o de programación. Los argumentos de las plantillas pueden incluir no sólo valores dato ordinarios u objetos, sino también clasificadores. En este último caso, el tipo de parámetro correspondiente deberá ser **Classifier** (clasificador) o cualquier otro metatipo. El valor de un argumento de plantilla debe fijarse en tiempo de modelado, y no es posible utilizarlo para representar un argumento en tiempo real. No utilice plantillas si no se han fijado los parámetros en tiempo de modelado.

argumento formal

Véase parámetro.

arquitectura

Estructura organizativa de un sistema que incluye su descomposición en partes, conectividad, mecanismos de interacción y principios de guía que proporcionan información sobre el diseño del mismo.

Véase paquete.

Semántica

La arquitectura es el conjunto de decisiones significativas sobre la organización de un sistema software. Incluye la selección de elementos estructurales y las interfaces mediante las que se conectan, la organización a gran escala de los elementos estructurales y la topología de su conexión, su comportamiento en las colaboraciones entre dichos elementos, los mecanismos importantes de que se dispone en el sistema y el estilo arquitectónico que guía su organización. Por ejemplo, la decisión de construir un sistema en dos capas, cada una de las cuales contiene un pequeño número de subsistemas que se comunican de una forma particular es una decisión arquitectónica. La arquitectura de un software no está relacionada sólo con la estructura y el comportamiento, sino también con el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, las restricciones y compromisos económicos y tecnológicos y con la estética.

Discusión

Las decisiones arquitectónicas sobre la descomposición de un sistema en partes se pueden capturar utilizando modelos, subsistemas, paquetes, y componentes. Las dependencias entre estos elementos son indicadores clave de la flexibilidad de la arquitectura y de la dificultad de modificar el sistema en el futuro.

Otra parte importante de una arquitectura son los mecanismos que proporciona para construir sobre ella, que deben ser capturados mediante patrones y colaboraciones.

Las decisiones no estructurales se pueden capturar utilizando valores etiquetados.

artefacto

Pieza de información utilizada o producida por un proceso de desarrollo de software, como un documento externo o el producto de un trabajo. Un artefacto puede ser un modelo, una descripción o un software.

asociación

Relación semántica entre dos o más clasificadores que implica la conexión entre sus instancias.

Véase también asociación binaria, asociación *n*-aria, clase asociación, extremo de asociación, generalización de asociaciones.

Semántica

Una asociación es una relación entre dos o más clasificadores que describe conexiones entre sus instancias. Los clasificadores que participan tienen posiciones ordenadas dentro de la asociación. La misma clase puede aparecer en más de una posición dentro de la asociación. Cada instancia de una asociación (denominada enlace) es una tupla (lista ordenada) de referencias a objetos, ya que la asociación es un conjunto de enlaces. Un objeto dado puede aparecer más de una vez en el conjunto de enlaces, o incluso más de una vez (en diferentes posiciones) en el mismo enlace si lo permite la definición de la asociación. Las asociaciones son el "pegamento" que mantiene unido un sistema, pues sin ellas sólo sería un conjunto de clases inconexas.

Estructura

Una asociación tiene un nombre opcional, pero la mayoría de sus descripciones se encuentran en una lista de extremos de asociación, cada uno de los cuales describe cómo participan los objetos de una determinada clase en la asociación. Téngase en cuenta que un extremo de una asociación es simplemente una parte de la descripción de una asociación, y no un concepto semántico o de notación separado.

Nombre. Una asociación puede tener opcionalmente un nombre: una cadena única con respecto a las clases y asociaciones del paquete que la contiene. Una clase asociación es tanto una asociación como una clase, pero sin embargo, las clases y las asociaciones comparten un único espacio de nombres. No es necesario que una asociación tenga un nombre, pues los nombre de roles de sus extremos proporcionan una forma alternativa de distinguir las asociaciones múltiples entre las mismas clases. Por convención, el nombre se lee en el orden en que las clases participantes aparecen en la lista: una Persona trabaja para una Empresa; un Vendedor vende un Coche a un Cliente.

Extremos de asociación. Una asociación contiene una lista ordenada de dos o más extremos de asociación. Por ordenado se entiende que los extremos pueden distinguirse y no son intercambiables. Cada extremo de una asociación define la participación de una clase en una posición dada (rol) en la asociación. La misma clase puede aparecer en más de una posición,

pero las posiciones no son intercambiables, en general. Cada extremo de asociación especifica propiedades que se aplican a los objetos que participan, como cuántas veces puede aparecer en la asociación un objeto simple (multiplicidad). Ciertas propiedades, como la posibilidad de navegación, se aplican sólo a las asociaciones binarias, pero la mayoría se aplican tanto a las binarias como a las *n*-arias.

Véase extremo de asociación para más detalles.

Instanciación

Un enlace es una instancia de una asociación, que contiene una ranura para cada extremo de la asociación. Cada ranura contiene una referencia a un objeto que es una instancia (directa o indirecta) de la clase especificada como clase del correspondiente extremo de la asociación. Un enlace no tiene identidad aparte de la lista de objetos que contiene. Los enlaces que forman una asociación forman un conjunto donde no pueden existir duplicados. El número de veces que aparece un objeto en un conjunto de enlaces debe ser compatible con la multiplicidad de cada extremo de la asociación. Por ejemplo, si la asociación **EntradasVendidas** conecta muchas entradas con un espectáculo, entonces cada entrada puede aparecer sólo una vez en un enlace, pero cada espectáculo puede aparecer muchas veces, cada una con una entrada diferente.

Se pueden crear y destruir enlaces según avance la ejecución de un sistema, ateniéndose a las restricciones de intercambio de los extremos de asociación. En algunos casos se puede crear o cambiar un enlace de un objeto de uno de los extremos de la asociación pero no del otro extremo. Un enlace se crea a partir de una lista de referencias. No tiene identidad por sí mismo, por lo que carece de sentido hablar de modificar su valor, aunque puede destruirse y crearse otro en su lugar. Un enlace de una clase asociación tiene uno o más valores de atributos además de la lista de objetos que define su identidad, y las operaciones pueden modificar los valores de los atributos mientras preserven las referencias a los objetos participantes.

Notación

Una asociación binaria se representa mediante una ruta continua que conecta los bordes de dos clases (véase Figura 13.15). Una asociación *n*-aria se representa mediante un rombo conectado

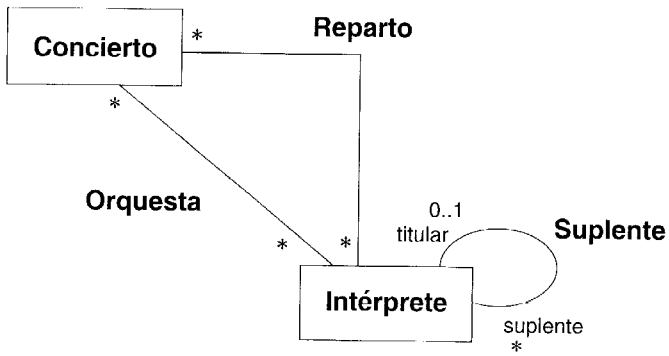


Figura 13.15 Asociaciones

a sus clases participantes mediante rutas (véase Figura 13.19). En la asociación binaria, el rombo se suprime por ser superfluo. Puede haber más de un final de ruta conectada a una misma clase.

Una ruta está formada por uno o más segmentos dibujados con línea continua conectados, normalmente segmentos de recta, aunque están permitidos los arcos y otras curvas, especialmente a la hora de mostrar una auto-asociación (una asociación en la que una clase aparece más de una vez). Los segmentos individuales no tienen semántica significativa. El usuario puede escoger el estilo de línea a utilizar.

Véase ruta.

Los extremos de las rutas tienen adornos que describen la participación de una clase en la asociación. Algunos adornos se representan en el extremo de la ruta, entre el segmento de línea y la caja de la clase. Si hay múltiples adornos, se sitúan secuencialmente comenzando por el extremo de la línea hasta el símbolo de la clase según el orden: flecha de navegación, rombo de agregación/composición y calificador (véase Figura 13.16).

Otros adornos, como las etiquetas de nombre, se sitúan cerca del elemento al que identifican. Los nombres de rol se sitúan cerca del final de la ruta.

Véase extremo de asociación para más detalles sobre la notación de los adornos.

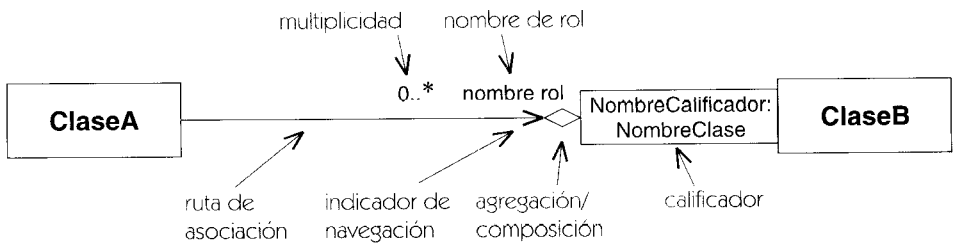


Figura 13.16 Orden de los adornos en un extremo de asociación

Nombre de asociación

El nombre de una asociación se sitúa cerca de la línea de la ruta, pero lo suficientemente lejos del extremo como para que no haya peligro de confusión. El peligro de confusión es meramente visual para las personas, ya que dentro de una herramienta gráfica no existe peligro de ambigüedad en la representación interna, por lo que la herramienta se encarga de determinar la distancia óptima. El nombre de la asociación se puede arrastrar de unos segmentos a otros de una asociación con múltiples segmentos sin que esto tenga ningún significado especial. El nombre de la asociación puede tener al lado un pequeño triángulo relleno que muestre el orden de las clases en la lista. Intuitivamente, la flecha del nombre muestra cómo leer el nombre. En la Figura 13.17, la asociación **TrabajaPara** entre la clase **Persona** y la clase **Empresa** debería tener el triángulo apuntando de **Persona** a **Empresa** y leerse "Persona trabaja para Empresa". Obsérvese que poner el triángulo en el nombre es un aspecto únicamente de notación que indica el orden en que finaliza la asociación. En el propio modelo, los extremos están inherentemente ordenados, por lo que el nombre del modelo no tiene ni necesita una propiedad de ordenación.

Un estereotipo en la asociación se indica mostrando el nombre del estereotipo entre comillas (« ») enfrente o en lugar del nombre de la asociación. Se puede situar una lista de propiedades después o debajo del nombre de la asociación.

Ejemplo

Figura 13.17 Nombre de asociación

Clase asociación

Una clase asociación se representa asociando un símbolo de clase a la ruta de asociación mediante una línea discontinua. En asociaciones *n*-arias, la línea discontinua se conecta con el rombo de la asociación. Las propiedades "de clase" de la asociación se muestran en el símbolo de clase, mientras que las propiedades "de asociación" de la misma se muestran en la ruta. Observe sin embargo, que la estructura de modelado subyacente es un único elemento, incluso aunque para representarla se utilicen dos estructuras diferentes.

Véase clase asociación para más detalles.

Restricción Xor

La restricción {**xor**} conecta dos o más asociaciones que están conectadas a una misma clase simple (la clase base) en un extremo. Una instancia de la clase base puede participar solamente en una de las asociaciones conectadas por la restricción. Debe respetarse la multiplicidad de la asociación elegida. Si la multiplicidad de alguna asociación incluye la cardinalidad 0, una instancia de la clase base podría no tener enlaces desde la asociación; si no, debería tener uno.

Una restricción **xor** se representa mediante una línea discontinua etiquetada con la cadena de restricción {**xor**} que conecta dos o más asociaciones que tienen una clase en común (Figura 13.18). Los nombres de roles de los extremos de la asociación más lejanos a la clase común deben ser diferentes, lo cual no es más que una utilización predefinida de la notación de restricción que emplea la restricción estándar de solapamiento.

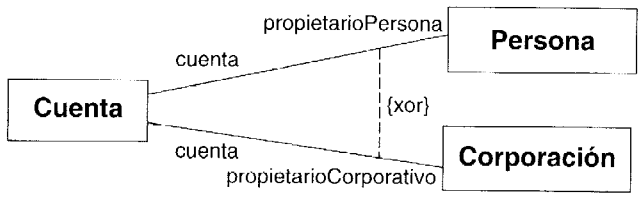


Figura 13.18 Asociación Xor

Discusión

No es necesario ponerle nombre a una asociación. Normalmente, los nombres de rol resultan más convenientes, ya que proporcionan nombres para la navegación y la generación de código a la vez que evitan el problema de saber en qué sentido leer el nombre. Si tiene nombre, debe ser único dentro de su paquete, pero si no lo tiene y hay más de una asociación entre un par (o conjunto) de clases, tienen que existir nombres de rol para distinguir las asociaciones. Si entre dos clases sólo hay una asociación, son suficientes los nombres de las clases para identificarla.

Se puede argumentar que los nombres de asociación son más útiles cuando el concepto tiene un nombre en el mundo real, tales como **Trabajo** o **Matrimonio**. Cuando un nombre de asociación está "dirigido" para ser leído en una determinada dirección, es generalmente más sencillo utilizar nombres de rol, que nunca son ambiguos se lean en la dirección que se lean.

Véase enlace transitorio, donde se discute sobre el modelado de relaciones de instancia que existen sólo durante la ejecución de un procedimiento.

Véase también composición, donde se muestra un ejemplo de generalización en que intervienen dos asociaciones.

Elementos estándar

implicit, persistence, xor.

asociación binaria

Asociación que se produce exactamente entre dos clases.

Véase también asociación, asociación *n*-aria.

Semántica

Una asociación binaria es una asociación con exactamente dos extremos, que es de lejos la asociación más común de todas. Como un extremo de una asociación binaria tiene otro extremo simple, las asociaciones binarias son particularmente útiles para especificar rutas de navegación entre objetos. Una asociación es navegable en un determinado sentido si puede atravesarse en ese sentido. Otras propiedades, como la multiplicidad, están definidas para asociaciones *n*-arias, pero son más intuitivas y útiles en las asociaciones binarias.

Notación

Una asociación binaria se representa mediante una ruta continua que conecta dos símbolos de clase. Se pueden asociar adornos a cada extremo y situar el nombre de la asociación cerca de la ruta, pero lo suficientemente lejos de los extremos como para que no haya confusiones con los nombres de rol. La notación de una asociación binaria es la misma que la de una asociación

n-aria excepto en que se suprime el rombo central. Sin embargo, las asociaciones binarias pueden tener adornos que no son aplicables a las asociaciones *n*-arias, como la posibilidad de navegación.

Para más detalles, *véase* asociación.

asociación de comunicación

Asociación que describe una relación de comunicación entre instancias de los elementos conectados. En una vista de despliegue, es una asociación entre nodos que implica una comunicación. En un modelo de casos de uso, una asociación entre un caso de uso y un actor constituye una asociación de comunicación.

Véase actor, caso de uso.

asociación *n*-aria

Denota una asociación entre tres o más clases. Por contrastar con: asociación binaria.

Semántica

Toda instancia de la asociación es una *n*-tupla de valores pertenecientes a las respectivas clases. Una clase puede aparecer en más de una posición de la asociación. Una asociación binaria es un caso especial que posee su propia notación simplificada y ciertas propiedades adicionales (tales como la navegabilidad) que carecen de sentido (o son inútilmente complicadas) para una asociación *n*-aria.

La multiplicidad de las asociaciones *n*-arias se puede especificar pero resulta menos evidente que la multiplicidad binaria. La multiplicidad del extremo de una asociación representa el número potencial de valores que puede haber en ese extremo, cuando están fijados los valores de los otros *n*-1 extremos. Obsérvese que esta definición es compatible con la multiplicidad binaria.

La agregación (incluyendo la composición) sólo tiene significado para las asociaciones binarias. Una asociación *n*-aria no puede contener el símbolo de agregación ni el de composición en ningún rol.

No existe diferencia semántica entre una asociación binaria y una asociación *n*-aria con dos extremos, independientemente de la representación. Una asociación con dos extremos se considera una asociación binaria y una que tenga más de dos extremos se considera asociación *n*-aria.

Notación

Una asociación *n*-aria se representa mediante un rombo grande (esto es, grande en comparación con el terminador de una ruta), existiendo una ruta desde el rombo hasta cada una de las clases participantes. El nombre de la asociación (si existe) se muestra junto al rombo. Pueden

aparecer adornos junto al extremo de cada ruta, tal como sucedería con una asociación binaria. Se puede indicar la multiplicidad, pero no se permiten los calificadores ni las agregaciones.

Se puede asociar un símbolo clase de asociación al rombo mediante una línea discontinua. Esto indica una asociación *n*-aria que posee atributos, operaciones y/o asociaciones.

Ejemplo

La Figura 13.19 muestra el registro de un equipo en las distintas temporadas de un determinado guardameta. Se supone que el portero puede ser traspasado durante la temporada y por tanto puede tener un registro con distintos equipos. En un libro de registros, cada enlace sería una línea diferente.

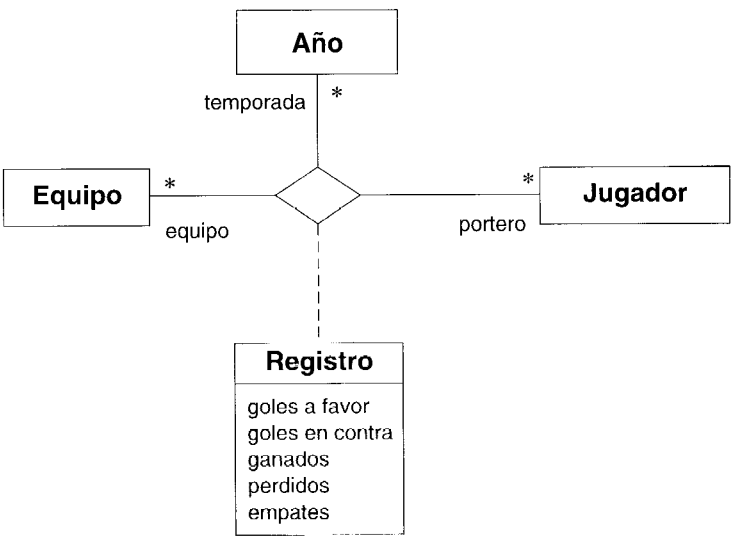


Figura 13.19 Una asociación ternaria que es además una clase asociación

Reglas de estilo

Normalmente, las líneas se dibujan saliendo de los vértices del rombo o desde el punto medio de una cara.

Discusión

En una asociación *n*-aria, la multiplicidad se define con respecto a los otros *n*-1 extremos. Por ejemplo, dada una asociación ternaria entre clases (A, B, C), la multiplicidad del extremo C indica el número de objetos C que pueden aparecer asociados a una pareja particular de objetos A y B. Si la multiplicidad de esta asociación es (muchos, muchos, uno) entonces para cada posible pareja (A, B) existe un único valor de C. Para una pareja dada (B, C) puede haber

muchos valores de A, sin embargo; de modo individual muchos valores de A, B y C podrían participar en la asociación. En una asociación binaria esta regla se reduce a la multiplicidad de cada extremo, que se define con respecto al otro extremo.

No tiene sentido definir la multiplicidad con respecto a un solo extremo (tal como han propuesto algunos autores) porque la multiplicidad sería muchos para cualquier asociación *n*-aria significativa. De no ser así, la asociación se podría fragmentar en una asociación binaria entre la clase única y una clase asociación que incluyera todas las clases restantes, mejorando así tanto la precisión como la eficiencia de la implementación. En general, lo mejor es evitar las asociaciones *n*-arias porque las asociaciones binarias son más fáciles de implementar y hacen posible la navegación. En general, las asociaciones *n*-arias sólo son útiles cuando se necesitan todos los valores para determinar un enlace de forma única. Una asociación *n*-aria se va a implementar en casi todas las ocasiones como una clase entre cuyos atributos se cuentan punteros de los objetos participantes. La ventaja de modelarla como una asociación es la restricción de que no puede haber enlaces duplicados dentro de una asociación.

Considere el ejemplo de un alumno que asiste a un curso de un profesor a lo largo de un curso (Figura 13.20). El alumno no va a seguir el mismo curso de más de un profesor, pero un alumno puede asistir a más de un curso de un único profesor, y el profesor puede impartir más de un curso. Las multiplicidades se muestran en el diagrama. La multiplicidad del **Profesor** es opcional (0..1); las otras multiplicidades son muchas (0..*).

Todo valor de multiplicidad es relativo a una pareja de objetos de otros extremos. Para toda pareja (curso, alumno) existe cero o un profesor. Para toda pareja (alumno, profesor) existen muchos cursos. Para toda pareja (curso, profesor) existen muchos alumnos.

Obsérvese que si esta asociación se materializa en una clase entonces sería posible tener más de una copia de la misma combinación (alumno, curso, profesor) lo cual no resulta deseable.

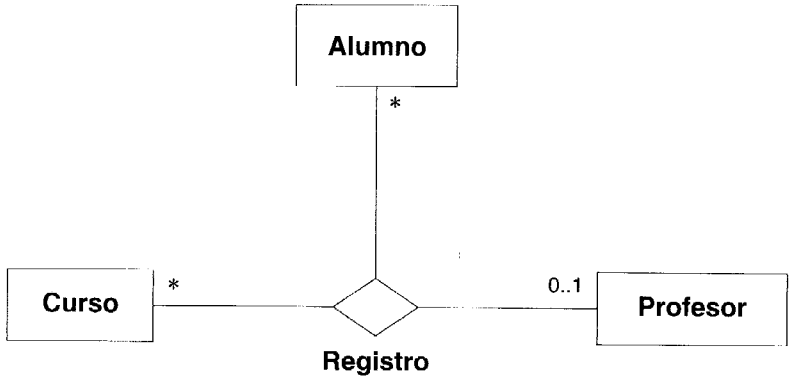


Figura 13.20 Multiplicidad en una asociación *n*-aria

atómico/a

Acción u operación cuya ejecución debe completarse como un todo, por lo que no puede ejecutarse parcialmente ni ser finalizada por un evento externo. Normalmente, las operaciones

atómicas son pequeñas y sencillas, como asignaciones, cálculos aritméticos sencillos u operaciones de tipo cadena. Un cómputo atómico ocurre en un punto definido de la secuencia de ejecución.

Véase también acción, actividad, ejecución a completar.

Semántica

Globalmente, el sistema puede realizar muchas acciones simultáneamente. Cuando decimos que una acción es atómica, eso no implica que el sistema completo sea atómico. El sistema puede procesar interrupciones del hardware y compartir el tiempo entre diversas acciones, ya que una acción es atómica sólo dentro de su hilo de control. Una vez iniciada, debe completar su ejecución sin interactuar con otras acciones que estén activas al mismo tiempo. El sistema puede procesar interrupciones y eventos, pero eso no debe afectar a la operación atómica, ya que dichas acciones no deberían usarse como un mecanismo para transacciones largas. Su duración debería ser breve comparada con el tiempo de respuesta necesario para responder a eventos externos porque, si no, el sistema podría ser incapaz de responder a tiempo.

atributo

Un atributo es la descripción de una ranura con nombre de un tipo especificado en una clase; cada objeto de la clase tiene un valor independiente para el atributo.

Semántica

Un atributo es una ranura con nombre dentro de un clasificador que describe los valores que pueden tener las instancias del clasificador. Todas las instancias del clasificador o de uno de sus descendientes tienen una ranura que guarda un valor del tipo dado. Todas las ranuras son distintas e independientes de las demás (excepto en el caso de los atributos con alcance de clase, que se describen más adelante). Cuando se lleva a cabo una ejecución, el valor de una ranura dentro de una instancia puede remplazarse por otro del mismo tipo, ya que los atributos son modificables.

Un clasificador forma un espacio de nombres para sus atributos, aunque también forman parte del espacio de nombres los pseudoatributos, los nombres de rol de las asociaciones que salen del clasificador y los discriminadores de generalizaciones en las que participe el clasificador.

Estructura

Un atributo tiene los siguientes componentes principales, que se describen con más detalle en sus propias entradas:

alcance de propietario En una clase, el valor descrito por un atributo puede ser distinto en cada objeto o compartido por todos los objetos. En el primer caso,

se trata de un atributo con alcance de instancia, mientras que en el último se trata de un atributo con alcance de clase. La mayoría de los atributos tienen alcance de instancia, esto es, mantienen información de estado acerca de un objeto en concreto. Los atributos con alcance de clase guardan información sobre una clase entera, ya que hay un único valor de la ranura para toda la clase.

Mientras que un atributo con alcance de instancia es una descripción de un valor sin existencia hasta que no se instancia el objeto, un atributo con alcance de clase representa la declaración de un valor discreto individual que existe a lo largo de toda la vida de un sistema.

alcance destino

El valor de un atributo puede ser una instancia o una clase. En el primer caso sería un alcance de instancia (valor por defecto), mientras que en el segundo es un alcance de clase, caso raro que normalmente implica alguna forma de metamodelado.

multiplicidad

El posible número de valores del atributo que pueden existir simultáneamente. El valor más común, "exactamente uno" denota un atributo escalar. El valor "cero o uno" denota un atributo con un valor opcional. La no existencia de valor es distinta de cualquier valor del dominio del tipo del atributo (en otras palabras, la ausencia de valor es distinta del valor cero, pues significa conjunto vacío). Otras multiplicidades denotan atributos que pueden tener múltiples valores. Si la multiplicidad no es un entero simple, es porque el número de valores del atributo puede variar. La multiplicidad "muchos" denota un conjunto ilimitado de valores.

nombre

El nombre de un atributo, una cadena que debe ser única dentro de la clase y sus antecesores. También debe ser única entre los nombres de los roles de asociación alcanzables desde la clase.

tipo

Designa una clase o tipo de dato del que son instancias los valores de la ranura. Un valor puede ser una instancia de un descendiente de la clase o tipo de dato dado.

valor inicial

Expresión que indica el valor de un atributo en un objeto justo después de haber sido inicializado. La expresión es una cadena de texto, junto con el nombre del lenguaje utilizado para evaluar la expresión. La expresión se evalúa en el contexto del lenguaje cuando se instancia el objeto. *Véase* expresión para más detalles. El valor inicial es opcional; si no lo hay, el modelo estático no especifica el valor para los objetos nuevos, aunque otra parte del modelo global puede proporcionar dicha información.

Obsérvese que un procedimiento de inicialización explícito, como un constructor, puede reemplazar una expresión de valor inicial.

El valor inicial de un atributo con alcance de clase se utiliza para inicializarlo una vez al principio de la ejecución. UML no especifi-

ca el orden relativo de inicialización de los diferentes atributos con alcance de clase.

variabilidad — Indica si el valor de la ranura puede cambiar tras la inicialización. Se utiliza una enumeración cuyo valor por defecto es **changeable**. Los posibles valores son:

- changeable — (modificable): no hay restricciones para modificarlo. Valor por defecto.
- addOnly — (sólo añadir): para atributos con multiplicidad mayor que uno, se pueden añadir valores adicionales al conjunto de valores del atributo, pero una vez creado, un valor no puede ser modificado ni eliminado.
- frozen — (congelado): el valor no se puede modificar una vez inicializado el objeto. No se pueden añadir valores adicionales a un conjunto de valores.

visibilidad — Indica si el atributo es visible desde otras clases. Se representa mediante una enumeración donde pueden aparecer las palabras clave **public**, **private** o **protected**. Se pueden añadir valores adicionales para modelar ciertos lenguajes de programación.

Notación

Un atributo se representa mediante una cadena de texto que puede dividirse en varias propiedades. La sintaxis por defecto es:

$$\overline{\text{«estereotipo»}}_{opc}\ \text{visibilidad}_{opc}\ \text{nombre}\ \text{mutiplicidad}_{opc}\ \overline{\text{: tipo}}_{opc}$$
$$\overline{\text{= valor-inicial}}_{opc}\ \overline{\{\text{cadena-de-propiedades}\}}_{opc}$$

Visibilidad. La visibilidad se representa mediante una marca de puntuación, aunque también se puede representar utilizando una palabra clave dentro de la cadena de propiedades, que se utiliza sobre todo para incluir opciones definidas por el usuario o dependientes del lenguaje. Las opciones predefinidas son:

+ (public)	Cualquier clase que vea la clase ve también sus atributos.
# (protected)	Sólo la propia clase o sus descendientes pueden ver los atributos.
– (private)	Sólo la propia clase puede ver los atributos.

Nombre. El nombre se representa mediante un identificador de tipo cadena.

Tipo. El tipo se representa mediante una expresión de cadena que denota un clasificador. El nombre de una clase o tipo de datos es una expresión legítima de tipo cadena que indica que los valores de un atributo deben ser de un tipo dado. La sintaxis adicional del tipo depende del

lenguaje de la expresión. Cada lenguaje tiene una sintaxis de construcción de nuevos tipos de datos a partir de los datos existentes. Por ejemplo, C++ tiene sintaxis para los punteros, para los arrays y para las funciones. Ada también tiene sintaxis para los subrangos. El lenguaje de la expresión es parte del modelo interno, pero normalmente no se indica en un diagrama, ya que se supone conocida en todo el diagrama u obvio a partir de su sintaxis.

La cadena de tipo se puede suprimir (pero sigue presente en el modelo).

Multiplicidad. La multiplicidad se representa mediante una expresión de multiplicidad encerrada entre corchetes y situada después del nombre del atributo. Si la multiplicidad es "exactamente uno", se puede omitir la expresión, incluyendo los corchetes. Esto indica que cada objeto tiene exactamente una ranura para guardar un valor del tipo especificado. Si no es así, debe mostrarse la multiplicidad. *Véase* multiplicidad, donde hay una completa discusión sobre la sintaxis. Por ejemplo:

colores [3]:Saturación	Un array de 3 saturaciones
puntos [2..*]:Punto	Un array de dos o más puntos

Obsérvese que una multiplicidad de 0..1 abre la posibilidad de valores nulos (la ausencia de valor como oposición a un valor concreto dentro del rango). Un valor nulo no es un valor del dominio de la mayoría de los tipos de datos: extiende el dominio con un valor extra que se encuentra fuera del mismo. Sin embargo, en el caso de los punteros el valor nulo es a menudo parte de la implementación (aunque, incluso entonces, se usa normalmente por convenio: por ejemplo el valor 0 en C o C++ es un convenio de direccionamiento de memoria). La siguiente declaración permite distinguir entre el valor *null* y la cadena vacía, una distinción presente en C++ y en otros lenguajes:

nombre [0..1]:String	Si no tiene nombre, es un valor nulo.

Valor inicial. El valor inicial se representa mediante una cadena. El lenguaje de evaluación normalmente no se indica explícitamente, aunque está presente en el modelo. Si no hay valor inicial, se omiten tanto la cadena como el signo igual. Si la multiplicidad del atributo incluye el valor 0 (es decir, opcional) y no se le ha dado explícitamente ningún valor, el atributo comienza teniendo un valor nulo (cero repeticiones).

Variabilidad. El valor de modificación se representa mediante una palabra clave que representa el valor elegido. Si no se elige ningún valor, se toma por defecto **changeable**.

Valor etiquetado. Un atributo puede tener asociados cero o más valores etiquetados (como cualquier elemento del modelo). Cada valor etiquetado se representa de la forma etiqueta = valor, donde etiqueta es el nombre de una etiqueta y valor es un valor literal. Los valores etiquetados se incluyen con las palabras clave de las propiedades como una lista entre corchetes de propiedades separadas por comas.

Alcance. El alcance de clase de un atributo se representa subrayando la cadena que expresa el tipo y nombre. Si no está subrayado, el atributo tiene alcance de instancia. La justificación de la notación es que un atributo con alcance de clase es un valor en el sistema en ejecución, al igual que un objeto es un valor de instancia, por lo que ambos deben subrayarse.

atributo-con-alcance-de-clase

La Figura 13.21 muestra la declaración de algunos atributos.

+tamaño: Área = (100,100)	público, valor inicial
#visibilidad: Boolean = invisible	protegido, valor inicial
+tamaño-por-defecto: Rectángulo	público
tamaño-máximo: Rectángulo	alcance de clase
–xptr: XWindowPtr {requisito = 4.3}	privado, valor etiquetado

Figura 13.21 Atributos

Opciones de presentación

Sintaxis del lenguaje de programación. La sintaxis de la cadena de un atributo puede ser la de un lenguaje de programación, como C++ o Smalltalk. Se pueden incluir en la cadena propiedades etiquetadas específicas.

Reglas de estilo

Los nombres de atributo se representan con un tipo de letra normal.

Discusión

Se utiliza una sintaxis similar para representar calificadores, parámetros de plantillas, parámetros de operaciones, y otros, aunque algunos de ellos omiten ciertos términos.

Obsérvese que un atributo es equivalente semánticamente a una asociación de composición, aunque el uso y la intención son totalmente diferentes. Utilice los atributos para tipos de datos, esto es, para valores sin identidad. Utilice las asociaciones para clases, o lo que es lo mismo, para valores con identidad. La razón es que en los objetos con identidad es importante ver la relación en ambas direcciones, mientras que en los tipos de datos, el tipo normalmente está subordinado al objeto y no sabe que existe.

Elementos estándar

persistence.

autotransición

Transición en la cual el estado origen y el estado destino son el mismo. Se considera un cambio de estado. Cuando se dispara, se sale del estado origen y se vuelve a entrar en él, de tal modo que se invocan las acciones de entrada y las de salida. No es equivalente a una transición interna, en la cual no se produce un cambio de estado.

become (se convierte en)

Tipo de dependencia de flujo, utilizada en una interacción, en la cual el objeto destino representa una nueva versión del objeto origen y a partir de entonces lo reemplaza.

Véase también clase en un estado, copia, localización.

Semántica

Una dependencia *become* es un tipo de dependencia de flujo que representa la derivación de un objeto a partir de otro dentro de una interacción. Después de ejecutarse un flujo *become*, el nuevo objeto reemplaza al original dentro del cómputo. Normalmente no es necesario utilizar esta relación para representar únicamente un cambio en el valor de un objeto. Por otra parte, esta relación es útil para representar un cambio cualitativo en un objeto, como un cambio de estado, un cambio de clase o un cambio de localización. En estas situaciones, el modelo contiene dos versiones del objeto, pero la relación *become* muestra que son realmente el mismo objeto, esto es, que tienen la misma identidad.

Una transición *become* dentro de una interacción puede tener un número de secuencia para indicar cuándo sucede con respecto a otras acciones.

Notación

Un flujo *become* se representa mediante una flecha discontinua con el estereotipo «**become**» que sale de la primera versión del objeto y apunta a la última. En una interacción, la flecha puede tener un número de secuencia cuando el cambio se produce con relación a otras acciones. Las transiciones *become* pueden aparecer en diagramas de colaboración, en diagramas de secuencia y en diagramas de actividades.

En un diagrama de actividades, la transición *become* se puede representar mediante una flecha discontinua desde o hacia el símbolo de flujo de objetos. Se puede omitir la palabra clave **become**.

Ejemplo

La Figura 13.22 muestra una orden para abrir un icono de directorio cerrado en un escritorio, seguido por otra que ordena los elementos del recién abierto directorio. El directorio se muestra dos veces como objeto de una clase en un estado, con una transición *become* entre las dos versiones.

La Figura 13.142 muestra un diagrama de despliegue en el que un objeto migra entre dos nodos.

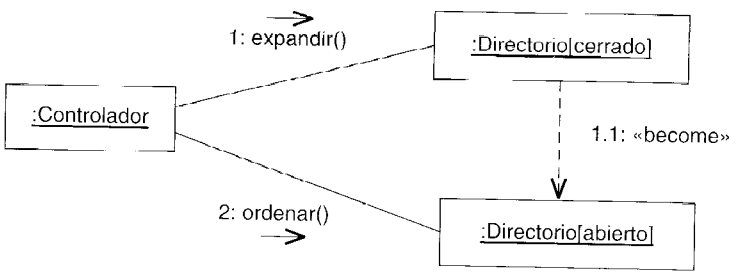


Figura 13.22 Flujo become

bien formado

Denota un modelo que está construido correctamente, que satisface todas las reglas predefinidas y establecidas por el modelo. Este modelo tiene una semántica significativa. Un modelo que no sea bien formado recibe el nombre de mal formado.

bifurcación

Elemento de una máquina de estados en el que un único disparador tiene más de un posible resultado, cada uno con su propia condición de guarda.

Véase también división, estado de unión o conjunción, unión.

Semántica

Si un mismo evento puede tener diferentes efectos dependiendo de las distintas condiciones de guarda, puede modelarse entonces como transiciones separadas con el mismo evento disparador. Sin embargo, en la práctica es conveniente permitir que un disparador simple dirija múltiples transiciones. Esto es especialmente cierto en el caso tan común en que las condiciones de guarda cubren todas las posibilidades, caso en el que se garantiza que una ocurrencia del evento disparará una de las transiciones. Una *bifurcación* es una parte de la transición que divide la ruta de la misma en dos o más segmentos, cada uno de ellos con una condición de guarda. El evento disparador se sitúa en el segmento común de la transición (el primer segmento). La salida de un segmento de una bifurcación puede conectarse con la entrada de otra bifurcación formando un árbol, de forma que cada ruta a través del árbol represente una transición diferente. La conjunción de todas las condiciones de una ruta de una transición es equivalente a una condición simple que se evalúe antes de que se dispare la transición. Una transición se dispara en un único paso, a pesar de su apariencia de conjunto de bifurcaciones, ya que el árbol no es más que un convenio de notación.

En un grafo de actividades, las actividades que salen de un estado de actividad son, generalmente, transiciones de finalización, esto es, carecen de eventos disparadores explícitos, y se disparan implícitamente cuando finaliza la actividad del estado. Si existen condiciones de guarda o bifurcaciones, es importante que cubran todas las posibilidades para que se dispare alguna transición, pues si no se congelaría la ejecución del grafo de actividades, ya que las transiciones de salida nunca volverían a estar disponibles.

Notación

Una bifurcación se puede representar repitiendo un evento disparador en los arcos de muchas transiciones con diferentes condiciones de guarda. También con transiciones de finalización, como en los diagramas de actividades.

Sin embargo, es más conveniente representar las bifurcaciones utilizando un rombo al que se conecta la punta de flecha de una transición. La flecha de la transición se etiqueta con el evento disparador, si lo hay, pero no debería llevar acciones asociadas. Las acciones irían en el segmento final de la transición.

De un rombo pueden salir dos o más flechas, cada una de las cuales se etiqueta con una condición de guarda. Se puede utilizar la palabra reservada **else** como condición de guarda, que será verdadera cuando las demás condiciones explícitas de guarda sean falsas. El final de una

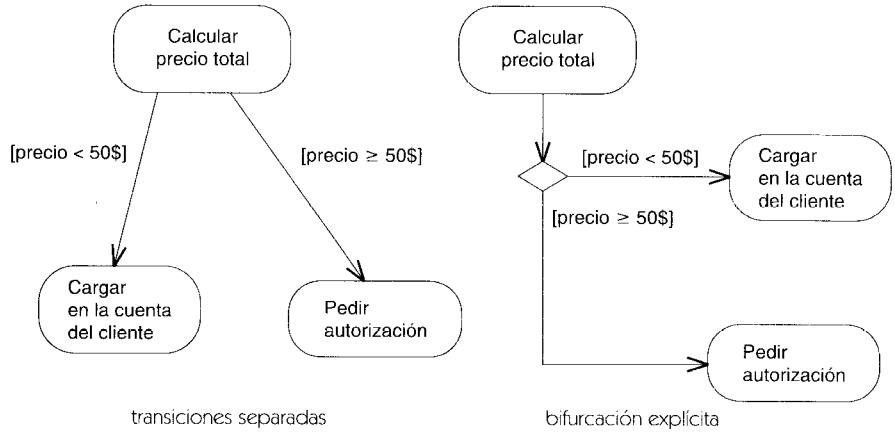


Figura 13.23 Dos formas de representar una bifurcación

flecha se puede conectar con otra bifurcación o con otro estado, y cuando se conecte a un estado, podrá llevar una acción asociada.

El efecto de árbol de bifurcaciones es el mismo que el que se produce expandiendo el árbol en un arco de transición separado para cada ruta del árbol, donde todos comparten el mismo evento disparador, pero cada uno de los cuales tiene su propia conjunción de condiciones de guarda, acciones y estados finales. La Figura 13.23 muestra dos formas de representar la misma situación.

Obsérvese que se puede utilizar también el símbolo del rombo para la reunificación (operación inversa de la bifurcación), donde dos rutas separados se unen, tal y como se muestra en la Figura 13.24. En el caso de la reunificación habrá dos o más flechas de entrada y una sola de salida. No son necesarias las condiciones de guarda.

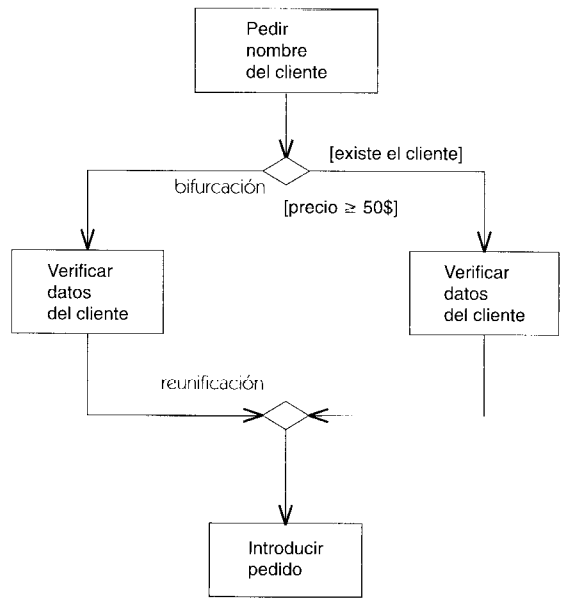


Figura 13.24 Bifurcación y reunificación

bind (ligar)

Palabra clave para expresar una dependencia de ligadura en la notación.

Véase ligadura.

booleano/a

Enumeración cuyos valores son **true** y **false**.

cadena

Secuencia de caracteres de texto. Los detalles de la representación de cadenas dependen de la implementación y pueden incluir conjuntos de caracteres que admitan gráficos y caracteres internacionales.

Semántica

Hay muchas propiedades semánticas, especialmente los nombres, que tienen una cadena como valor. Una cadena es una secuencia de caracteres pertenecientes a algún conjunto de caracteres adecuado, que se emplea para visualizar información relativa al modelo. Entre los juegos de caracteres se pueden incluir alfabetos y caracteres no latinos. UML no especifica la codificación de cadenas pero supone que la codificación es suficientemente general para permitir cualquier utilización razonable. En principio, la longitud de las cadenas debería ser ilimitada; toda limitación práctica debería ser suficientemente grande para que no resulte restrictiva. Las cadenas deberían incluir la posibilidad de admitir mostrar caracteres de diferentes idiomas humanos. Los identificadores (los nombres) deberían estar formados únicamente por caracteres de un conjunto finito de caracteres. Los comentarios y clases similares de cadenas descriptivas sin contenido semántico puede contener otras clases de medios de elementos, tales como diagramas, grafos, imágenes o segmentos de vídeo y otros tipos de documentos incrustados.

Notación

Una cadena gráfica es un elemento primitivo de notación con cierta flexibilidad de implementación. Se supone que se trata de una secuencia lineal de caracteres escrita en algún idioma, con la posible inclusión de documentos incrustados de distintas clases. Resulta deseable admitir la utilización de diferentes idiomas humanos, pero se dejan los detalles a las herramientas de edición para que sean éstas las que los implementen. Las cadenas gráficas pueden ir en líneas individuales, en listas o bien pueden ser etiquetas asociadas a otros símbolos.

Las cadenas se utilizan para mostrar propiedades semánticas que tienen cadenas como valores y también se emplean para codificar los valores de otras propiedades semánticas para su visualización. La correspondencia entre cadenas semánticas y cadenas de notación es directa.

La correspondencia de otras propiedades con las cadenas de notación está controlada por las gramáticas, que se describen en los artículos correspondientes a los distintos elementos. Por ejemplo, la notación de visualización para los atributos codifica el nombre, valor, visibilidad y alcance en una única cadena de visualización.

Es posible realizar extensiones no canónicas de las codificaciones —por ejemplo, se podría visualizar un atributo empleando la notación de C++—. Algunas de estas codificaciones pueden perder información del modelo, sin embargo, las herramientas deberían admitirlas como opciones seleccionables por el usuario, manteniendo ciertamente la notación canónica de UML.

El estilo y tamaño del tipo de letra son marcadores gráficos, que normalmente serán independientes de la cadena en sí. Se pueden codificar para distintas propiedades del modelo, algunas de las cuales se sugieren en este documento; otras se dejan al albedrío de la herramienta o del usuario. Por ejemplo, la cursiva muestra clases y operaciones abstractas y el subrayado muestra características con alcance de clase.

Las herramientas pueden tratar de distintas maneras las cadenas más largas; se pueden truncar hasta un tamaño dado, se puede hacer un salto de línea automático y se pueden insertar barras de desplazamiento. Se supone que existe alguna manera de obtener la cadena completa si se desea.

calificador

Denota una ranura de un atributo o lista de atributos en una asociación binaria, en la cual los valores de los atributos seleccionan un único objeto relacionado o un conjunto de objetos relacionados dentro de todo el conjunto de objetos relacionados con un objeto por esa asociación. Se trata de un índice para recorrer una asociación.

Véase clase asociación, extremo de asociación.

Semántica

Toda asociación binaria hace corresponder a un objeto un conjunto de objetos relacionados. En algunas ocasiones resulta deseable seleccionar un objeto del conjunto, proporcionando un valor que distingue a los objetos del conjunto. Este valor podría ser un atributo de la clase destino. En general, sin embargo, el valor selector podría ser parte de la asociación en sí, un atributo de la asociación cuyo valor es proporcionado por el creador cuando se añade un nuevo enlace a la clase asociación. Un atributo como éste, en una asociación binara, recibe el nombre de calificador. Un objeto, junto con un valor de calificador, determina un objeto único relacionado con él o bien (con menos frecuencia) un subconjunto de objetos relacionados. El valor califica la asociación. En un contexto de implementación, a estos atributos se les da el nombre de valores de índice.

Los calificadores se utilizan para seleccionar un objeto u objetos del conjunto de objetos relacionados con un objeto (que se denomina *objeto calificado*) mediante una asociación (Figura 13.25). El objeto seleccionado por el valor del calificador se denomina *objeto destino*.

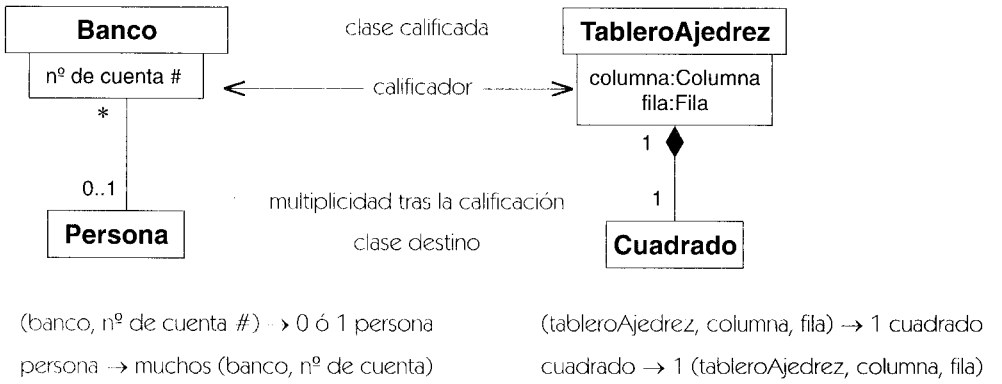


Figura 13.25 Asociaciones calificadas

Un calificador siempre actúa sobre una asociación cuya multiplicidad es *muchos* en la dirección del destino. En el caso más sencillo, cada valor del calificador selecciona un único objeto del conjunto destino de objetos relacionados. En otras palabras, un objeto calificado y un valor de calificador producen un único objeto destino relacionado. Dado un objeto calificado, cada valor del calificador se corresponde con un único objeto destino.

Hay muchas clases de nombres que son calificadores. Estos nombres dentro de un contexto se corresponden con un único valor. El objeto calificado proporciona el contexto y el objeto destino es el resultado. Toda ID u otro código único es un calificador; su propósito es seleccionar un valor de modo único. Las matrices se pueden modelar como asociaciones calificadas. La matriz es el objeto calificado, el índice de la matriz es el calificador y el elemento de la matriz es el objeto destino. Para una matriz, el tipo calificador es un intervalo de enteros.

Se puede emplear un calificador en una expresión de navegación para seleccionar un subconjunto de objetos relacionados con un objeto a través de una asociación, a saber, aquellos que tengan un valor concreto para el valor del atributo calificador o lista de valores. El calificador hace las veces de selector dentro del conjunto de objetos relacionados por la asociación. Descompone el conjunto en subconjuntos por valores del calificador. En la mayoría de los casos, el propósito del calificador es seleccionar un objeto único dentro del conjunto de objetos relacionados de tal modo que una asociación calificada se comporta como una tabla de búsqueda.

Estructura

Calificador. Un atributo calificador es una parte opcional del extremo de una asociación binaria. El calificador califica a la clase asociada al extremo de la asociación. Un objeto de la clase y un valor del calificador seleccionan un objeto o conjunto de objetos de la clase situada al otro extremo de la asociación binaria. Es posible que los dos extremos de la asociación binaria tenga calificadores, pero no suele suceder.

Un calificador es un atributo o lista de atributos de una asociación. Cada atributo posee nombre y tipo pero no valor inicial, porque los calificadores no son objetos independientes; el valor de cada calificador debe ser explícito siempre que se añade un enlace a la asociación.

Los calificadores no se utilizan en asociaciones *n*-arias.

Multiplicidad. La multiplicidad de la relación calificada se sitúa en el extremo opuesto de la asociación binaria respecto al calificador. (La regla mnemotécnica es que la clase calificada y el calificador, juntos, forman un valor compuesto que está relacionado con la clase destino.) En otras palabras, el calificador se asocia al extremo "próximo" de la asociación y la multiplicidad y el nombre de rol se asocian al extremo "lejano".

La multiplicidad asociada al extremo destino de la asociación denota el número de objetos destino que podrían ser seleccionados por una pareja (objeto origen, valor de calificador). Entre los valores comunes de multiplicidad se cuentan **0..1** (se puede seleccionar un único valor pero no todo posible valor del calificador selecciona un valor necesariamente), **1** (todo posible valor del calificador selecciona un único objeto destino, luego el dominio de valores del calificador debe ser finito) y * (el valor del calificador es un índice que descompone el objeto destino en subconjuntos).

En la mayoría de los casos, la multiplicidad es cero-o-uno. Esta opción significa que un objeto y un calificador pueden producir, como máximo, un único objeto relacionado. Una multiplicidad de uno significa que todo posible valor del calificador produce exactamente un objeto. Evidentemente, esto requiere que el tipo del calificador sea un dominio finito (al menos en la implementación para un computador). Esta multiplicidad puede ser útil para hacer corresponder tipos finitos enumerados; por ejemplo, un **Píxel** calificado por un **ColorPrimario** (enumeración de rojo, verde y azul) produciría el triplete de valores rojo-verde-azul para todos los píxeles de una imagen.

La multiplicidad de una asociación no calificada no se enuncia explícitamente. Pero suele suponerse que es muchos, o por lo menos mayor que uno. En caso contrario, no habría necesidad de tener un calificador.

Una multiplicidad de muchos en una asociación calificada no tiene un impacto semántico significativo, porque el calificador no reduce la multiplicidad del conjunto destino. Esta multiplicidad representa una sentencia de diseño consistente en que debe proporcionarse un índice para recorrer la asociación. En tal caso, el calificador descompone el conjunto de objetos destino en subconjuntos. Semánticamente, esto no añade nada más allá de tener un atributo de asociación, que también descompone (implícitamente) los enlaces. La connotación de diseño de un calificador en un modelo de diseño es que el recorrido debería ser eficiente, esto es, que no debe requerir una búsqueda lineal entre todos los valores destino. Normalmente se implementa mediante algún tipo de tabla de búsqueda. Un índice de una base de datos o de una estructura de datos se modela correctamente como un calificador.

En la dirección inversa a través de una asociación calificada (esto es, yendo desde la clase destino hacia el objeto calificado), la multiplicidad indica el número de parejas (objeto calificado, calificador) que se pueden relacionar con un objeto destino, no el número de objetos calificados. En otras palabras, si varias parejas (objeto calificado, calificador) se corresponden con el mismo objeto destino, entonces la multiplicidad inversa será muchos. Una multiplicidad inversa de uno entre destino y calificador significa que existe exactamente un emparejamiento de objeto calificado y valor calificador que está relacionado con el objeto destino.

Notación

Los calificadores se representan mediante un pequeño rectángulo asociado al extremo de una ruta de asociación entre el segmento final de la ruta y el símbolo de la clase calificada. El

rectángulo calificador forma parte de la ruta de asociación, no de la clase. El calificador está asociado a la clase que califica, esto es, un objeto de la clase calificada junto con un valor del calificador selecciona de modo único un conjunto de objetos de la clase destino situada en el otro extremo de la asociación.

Los atributos de calificador se enumeran dentro del cuadro calificador. Puede haber uno o más atributos en la lista. Los atributos de calificador tienen la misma notación de los atributos de clase, salvo que las expresiones de valor inicial no tienen sentido.

Opciones de presentación

Los calificadores no se pueden suprimir (proporcionan detalles esenciales cuya defecto modificaría el carácter inherente de la relación).

Las herramientas pueden utilizar una línea más fina para los rectángulos de calificador que para los rectángulos de clase, con objeto de distinguirlos claramente.

Preferiblemente, el rectángulo calificador debería ser más pequeño que el rectángulo de clase al que esté asociado, aunque esto no siempre es posible en la práctica.

Discusión

Las multiplicidades de una asociación calificada se tratan como si el objeto calificado y el calificador fueran una sola entidad, una clave compuesta. En la dirección hacia delante, la multiplicidad del extremo destino representa el número de objetos relacionados con el valor compuesto (objeto calificado + valor de calificador). En la dirección inversa, la multiplicidad describe el número de valores compuestos (objeto calificado + calificador) que están relacionados con cada objeto destino, no el número de objetos calificados relacionados con cada objeto destino. Ésta es la razón por la cual se pone el calificador en el mismo extremo de la ruta de asociación, junto al símbolo de la clase. Se puede pensar que la ruta de la asociación conecta el valor compuesto con la clase destino.

No se ha previsto especificar la multiplicidad de las relaciones no calificadas. Sin embargo, en la práctica suelen ser muchos en la dirección hacia delante. No tiene sentido tener una asociación calificada salvo que haya muchos objetos relacionados con un único objeto calificado. Para el modelado lógico, el propósito del calificador es reducir la multiplicidad a uno mediante la adición de un calificador, de tal modo que se asegure que una consulta pueda proporcionar un único valor en lugar de un conjunto de valores. La unicidad del valor del calificador suele ser frecuentemente una condición semántica crucial, que es difícil de capturar sin calificadores. Casi todas las asociaciones poseen muchas asociaciones calificadas. Hay muchos nombres que son realmente calificadores. Si un nombre es único en un determinado contexto, se trata de un calificador y el contexto debería ser identificado y modelado adecuadamente. No todos los nombres son calificadores. Los nombres de personas, por ejemplo, no son únicos. Dado que los nombres personales son ambiguos, la mayor parte de las aplicaciones de procesamiento de datos hace uso de algún tipo de número de identificación, tal como un número de cliente, un número de Seguridad Social o un número de empleado. Si una aplicación requiere la búsqueda de información o la recuperación de datos basada en claves de búsqueda, en general el modelo debería utilizar asociaciones calificadas. Todo contexto en el cual se

definan nombres o códigos de identificación para seleccionar cosas dentro de un conjunto deberá modelarse, normalmente, como una asociación calificada.

Observe que el valor del calificador es una propiedad del enlace, no del objeto destino. Considere un sistema de archivos Unix, en el cual cada directorio es una lista de entradas cuyos nombres son únicos dentro del directorio, aun cuando se pueden utilizar los mismos nombres dentro de otros directorios. Cada entrada denota un archivo, que puede ser un archivo de datos u otro directorio. Es posible tener más de una entrada que apunte a un mismo archivo. Si sucede esto, el archivo tiene varios alias. El sistema de directorios de Unix está modelado como una asociación muchos-a-uno en la cual el directorio calificado por un nombre de archivo produce un archivo. Observe que el nombre de archivo no forma parte del archivo; forma parte de la relación existente entre un directorio y un archivo. Un archivo no posee un único nombre. Puede tener múltiples nombres en otros tantos directorios (o incluso varios nombres en un mismo directorio). El nombre de archivo no es un atributo del archivo.

Una de las motivaciones más importantes para la existencia de asociaciones calificadas es la necesidad de modelar una importante situación semántica que posee una estructura de datos de implementación natural e importante. En la dirección hacia delante, una asociación calificada es una tabla de búsqueda; para un objeto calificado, cada valor del calificador produce un único objeto destino (o un valor nulo si el valor del calificador está ausente en el conjunto de valores). Las tablas de búsqueda se pueden implementar mediante estructuras de datos tales como tablas hash, árboles-b y listas ordenadas que proporcionan una eficiencia mucho mayor que las listas desordenadas. En casi todos los casos, son malos los diseños que utilizan una lista enlazada o alguna otra estructura desordenada para buscar nombres o códigos, aunque lamentablemente hay muchos programadores que las usan. El modelado de situaciones adecuadas empleando asociaciones calificadas y estructuras de datos eficientes para implementarlas resulta crucial para una buena programación.

Para un modelo lógico, tiene poco sentido tener una asociación calificada con una multiplicidad de muchos en la dirección hacia adelante, porque el calificador no añade ninguna información semántica que no pudiera mostrar un atributo de asociación. Sin embargo, en un modelo destinado al diseño de algoritmos y estructuras de datos, un calificador aporta una connotación adicional, a saber, la intención de que la selección sea eficiente. En otras palabras, una asociación calificada denota una estructura de datos indizada y optimizada para la búsqueda por valores del calificador. En este caso, puede ser útil una multiplicidad de muchos para representar un conjunto de valores al que deba poderse acceder en bloque a través de un valor de índice común, sin tener que buscar otros valores.

En general, no debe incluirse un atributo calificado como atributo de la clase destino, porque es suficiente su presencia en la asociación. En el caso de un valor índice, sin embargo, puede ser necesario tomar un valor que sea inherentemente un atributo de la clase destino y hacer que sea un valor de calificador redundante. Los valores de índice son inherentemente redundantes.

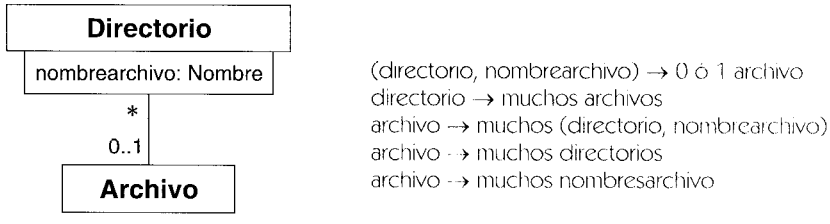


Figura 13.26 Calificador sencillo

Restricciones

Algunas situaciones complicadas no son fáciles de modelar con cualquier conjunto de relaciones no redundantes. Lo mejor es modelarlas empleando asociaciones calificadas para capturar las rutas básicas de acceso y enunciar explícitamente las restricciones adicionales. Dado que estas situaciones son poco frecuentes, estimamos que intentar incluirlas en una notación que capture directamente todas las posibles restricciones de multiplicidad no merecía la complejidad adicional.

Por ejemplo, considere un directorio en el que cada nombre de archivo identifica a un archivo único. Un archivo puede corresponder a múltiples parejas directorio-nombre de archivo. Éste es el modelo básico que se ha visto antes. El modelo se muestra en la Figura 13.26.

Ahora, sin embargo, deseamos añadir restricciones adicionales. Suponga que todo archivo debe estar en un único directorio, pero dentro de ese directorio puede tener múltiples nombres, esto es, que hay más de una forma de dar nombre al mismo archivo. Esto se podría modelar con una asociación redundante entre **Archivo** y **Directorio**, con una multiplicidad de un **Directorio** (Figura 13.27). La redundancia de las dos asociaciones se indica mediante la restricción {**igual**}, que implica que los dos elementos son el mismo pero con distintos niveles de detalle. Dado que estas asociaciones son redundantes, sólo se implementaría la asociación calificada; la otra se trataría como una restricción aplicada a sus componentes en tiempo de ejecución.

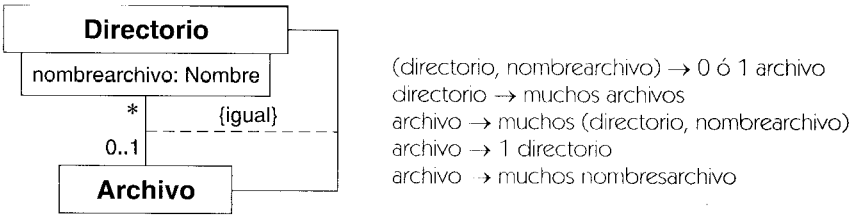


Figura 13.27 Un archivo con múltiples nombres en un directorio

Una restricción similar es que todo archivo puede aparecer en múltiples directorios pero siempre tiene el mismo nombre dondequiera que aparezca. Otros archivos podrán tener el mismo nombre pero tienen que estar en otro directorio. Esto se puede modelar haciendo que **nombrearchivo** sea un atributo de **Archivo** pero restringiendo el atributo de clase y el calificador para que sean iguales (Figura 13.28). Este patrón aparece con frecuencia como índice de

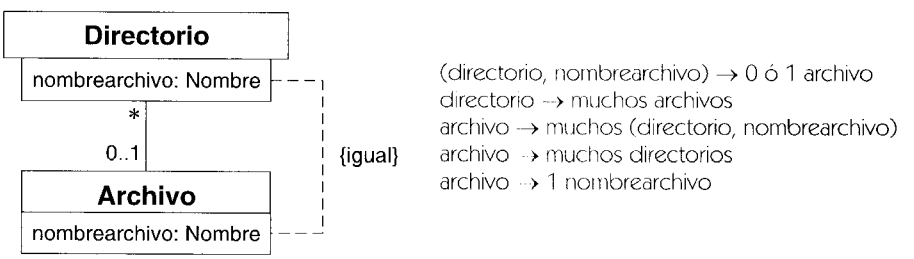


Figura 13.28 Un archivo con el mismo nombre en todos los directorios

búsqueda, aun cuando en un índice general la multiplicidad del destino calificado sería muchos. Por tanto, esta situación tiene más contenido sintáctico que un índice, que es un dispositivo de implementación.

Un tercer caso permitiría que un archivo apareciera en múltiples directorios con diferentes nombres, pero el archivo sólo podría aparecer una sola vez en cada directorio individual. Esto se podría modelar con una asociación calificada y una clase de asociación redundantes que compartieran el mismo atributo **nombrearchivo** (Figura 13.29).

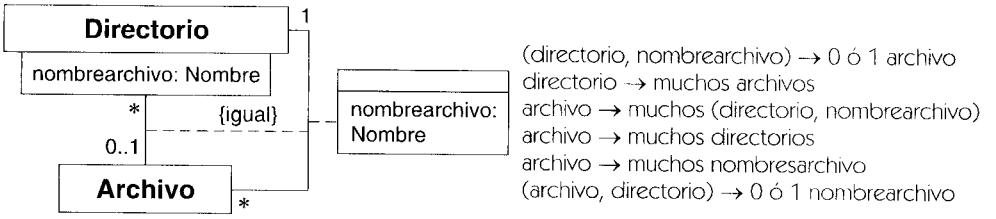


Figura 13.29 Un archivo con un nombre como máximo en cualquier directorio

Se han mostrado estos ejemplos con relaciones redundantes para ilustrar la naturaleza de las restricciones. Sin embargo, en la práctica suele resultar satisfactorio enunciar en forma de texto la restricción, mostrando gráficamente la asociación calificada.

calle [1]

Partición de los grafos de actividades que se emplea para organizar las responsabilidades de las actividades. Las calles no tienen un significado fijo, pero suelen corresponder a las unidades organizativas en los modelos de negocios.

Véase también grafo de actividades.

Semántica

Los estados de actividad dentro de un grafo de actividades se pueden organizar en particiones llamadas calles como consecuencia de su notación. Las calles son agrupamientos de estados para organizar los grafos de actividades. Cada calle representa alguna partición significativa de las responsabilidades de los estados —por ejemplo, la organización de negocios responsable de un paso en un flujo de trabajo—. Se pueden utilizar en la forma que prefiera el creador del modelo. Las calles, si existen, reparten entre ellas los estados del grafo de actividades.

Cada calle posee un nombre distinto del de las otras calles. No posee ninguna semántica adicional dentro de UML pero puede tener algunas implicaciones propias del mundo real.

---

[1] Se alude al término empleado en natación, *swimlane*. *N. del T.*

Notación

Se puede dividir visualmente el diagrama de actividades en calles, cada una de las cuales está separada de las calles contiguas mediante una línea vertical continua (Figura 13.30). Cada calle representa alguna responsabilidad de alto nivel para una parte de la actividad global, que puede ser implementada eventualmente por uno o más objetos. El orden relativo de calles no posee significación semántica pero puede indicar alguna afinidad en el mundo real. Cada estado de actividad se asigna a un calle y se sitúa visualmente en su interior. Las transiciones pueden cruzar las calles; la ruta de una transición no tiene significado especial.

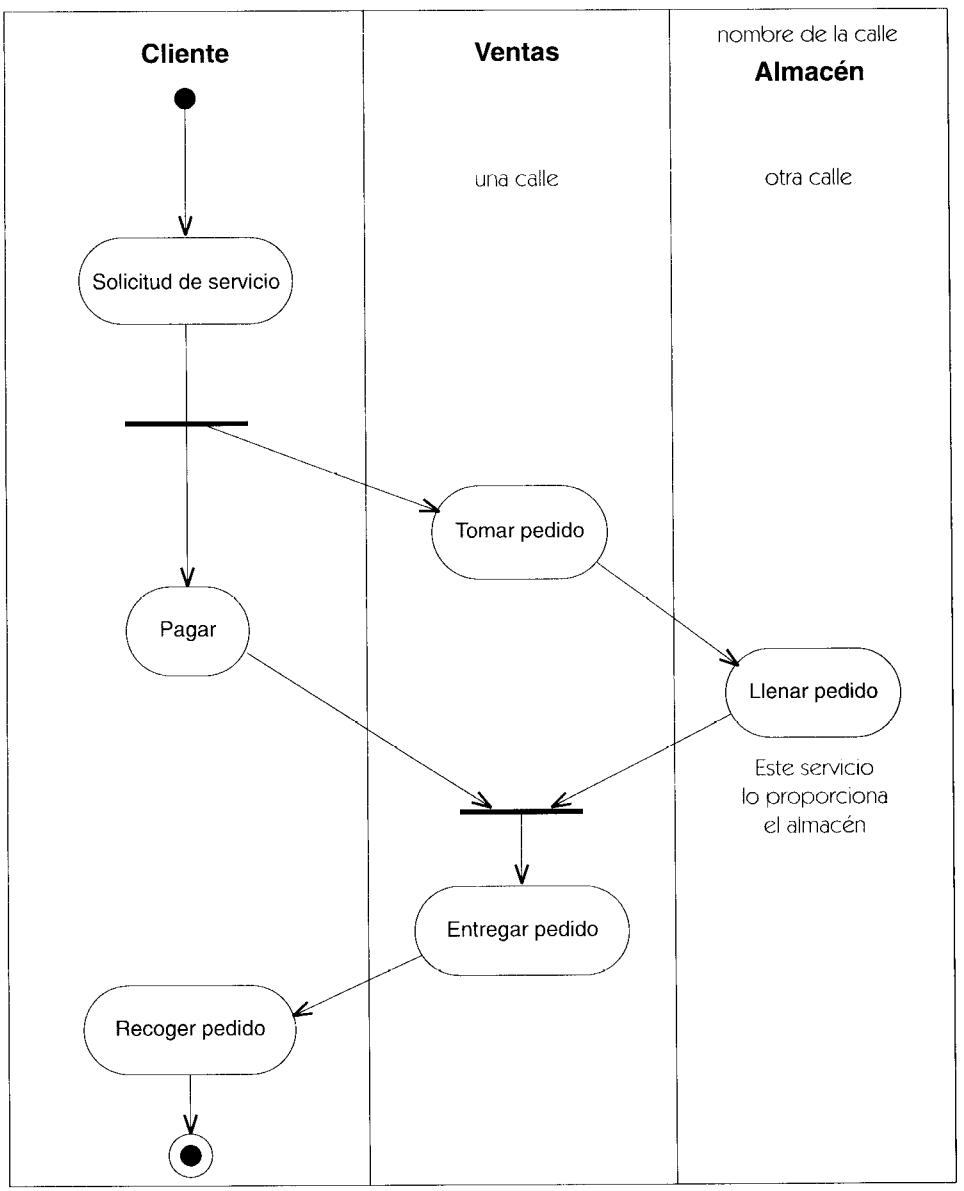


Figura 13.30 Calles de un diagrama de actividades

Como las calles no son otra cosa que particiones en categorías arbitrarias, pueden indicarse por otros medios si no resulta práctico agruparlas en regiones. Entre las posibilidades se cuenta el uso del color o utilizar simplemente valores etiquetados para mostrar la partición.

capa

Dícese de un patrón de arquitectura propio de los paquetes agrupados de un modelo, que poseen todos el mismo nivel de abstracción. Cada capa representa un mundo virtual en un determinado nivel de realidad.

característica

Una propiedad tal como una operación o atributo, que se encapsula como parte de una lista dentro de un clasificador, tal como una interfaz, una clase, o un tipo de dato.

característica de comportamiento

Elemento del modelo que expresa comportamiento dinámico, como una operación o método, y que puede ser una parte de un clasificador. La declaración de que un clasificador trata una señal, es también una característica de comportamiento.

Elementos estándar

create, destroy, leaf.

característica estructural

Característica estática de un elemento del modelo, tal como un atributo o una operación.

cardinalidad

Número de elementos de un conjunto. Es un número específico. Contrastar con multiplicidad, que es el rango de posibles cardinalidades que puede tener un conjunto.

Discusión

Obsérvese que muchos autores emplean el término *cardinalidad* de forma incorrecta queriendo hacer referencia a la multiplicidad, aunque el término cardinalidad tiene una definición matemática desde hace mucho tiempo como número, no como rango de números. Ésta es la definición que utilizamos aquí.

caso de uso

Especificación de las secuencias de acciones, incluyendo secuencias variantes y secuencias de error, que pueden ser efectuadas por un sistema, subsistema o clase por interacción con autores externos.

Véase también actor, clasificador.

Semántica

Un caso de uso es una unidad coherente de funcionalidad que proporciona un clasificador (un sistema, subsistema o clase) tal como lo manifiestan las secuencias de mensajes que se intercambian entre el sistema y uno o más usuarios externos (que se representan como actores), junto con acciones que realiza el sistema.

El propósito de un caso de uso es definir un cierto comportamiento de un clasificador (incluyendo un subsistema o todo el sistema) sin revelar la estructura interna del clasificador. Cada caso de uso especifica un servicio que proporciona el clasificador a sus usuarios, esto es, una forma específica de utilizar el clasificador que es visible desde el exterior. Describe una secuencia completa que es iniciada por un usuario (modelado por un actor) en términos de interacción entre los usuarios y el clasificador, así como las respuestas ofrecidas por el clasificador. La interacción sólo incluye las comunicaciones habidas entre el sistema y los actores. El comportamiento o implementación internos se ocultan. El conjunto completo de casos de uso de un clasificador o de un sistema particiona y abarca su comportamiento. Cada caso de uso representa un trozo cuantificado y significativo de funcionalidad que está disponible para los usuarios. Observe que entre los usuarios se cuentan los *seres* humanos así como los computadores y otros objetos. Un actor es la idealización del propósito de un usuario, no una representación de un usuario físico. Un usuario físico puede corresponderse con muchos actores y un actor puede representar el mismo aspecto de múltiples usuarios físicos.

Véase actor.

Los casos de uso incluyen el comportamiento normal y habitual que se tiene como respuesta a una solicitud del usuario, así como posibles variantes de la secuencia normal, tales como secuencias alternativas, comportamiento frente a excepciones y manejo de errores. El objetivo es describir un trozo de funcionalidad coherente en todas sus variaciones, incluyendo las condiciones de error. El conjunto completo de casos de uso de un clasificador especifica todas las formas distintas que hay de utilizar ese clasificador. Los casos de uso se pueden agrupar en paquetes por comodidad.

Un caso de uso es un descriptor; describe un comportamiento potencial. La ejecución de un caso de uso es una instancia de caso de uso. El comportamiento de un caso de uso se puede especificar mediante una máquina de estados asociada a él o bien por un código en forma de texto (que es equivalente a una máquina de estados). También se puede describir mediante una descripción informal en forma de texto. El comportamiento se puede ilustrar, pero no especificar formalmente, mediante un conjunto de escenarios. Esto puede resultar suficiente en las primeras fases del desarrollo.

Una instancia de caso de uso es una ejecución de un caso de uso, que será iniciada por un mensaje proveniente de una instancia de actor. Como respuesta al mensaje, la instancia de caso

de uso ejecuta una secuencia de acciones especificadas por el caso de uso, tales como enviar mensajes a instancias de actor, no necesariamente destinadas sólo al actor iniciador. Las instancias de actor pueden enviar mensajes a la instancia de caso de uso y la interacción prosigue hasta que la instancia ha respondido a todas las entradas. Cuando ya no espera más entradas, concluye.

Un caso de uso es una especificación del comportamiento de un sistema (u otro clasificador) como un todo en sus interacciones con actores exteriores. Las interacciones internas entre objetos internos del sistema que implementa el comportamiento se describen mediante una colaboración que realiza un caso de uso.

Estructura

Un caso de uso puede tener características y relaciones.

Características. Un caso de uso es un clasificador y por tanto tiene atributos y operaciones. Los atributos se emplean para representar el estado del caso de uso —esto es, el progreso de su ejecución—. Una operación representa un bloque de trabajo que puede realizar el caso de uso. No se puede invocar directamente desde el exterior, pero se puede emplear para describir el efecto del caso de uso sobre el sistema. La ejecución de una operación se puede asociar a la recepción de un mensaje procedente de un actor. Las operaciones actúan sobre los atributos del caso de uso e indirectamente sobre el sistema o clase al que esté asociado el caso de uso.

Asociaciones con actores. Una asociación entre un actor y un caso de uso indica que esa instancia de actor se comunica con la instancia de sistema o de clasificador para lograr algún resultado que sea de interés para el actor. Los actores modelan usuarios externos del clasificador. De este modo, si el clasificador es un sistema entonces sus actores son los usuarios externos del sistema. Los actores de subsistemas de nivel inferior pueden ser otras clases pertenecientes al sistema global.

Un actor se puede comunicar con varios casos de uso —esto es, el actor puede solicitar varios servicios diferentes del sistema— y un caso de uso puede comunicarse con uno o más actores cuando les proporciona sus servicios. Observe que dos casos de uso que especifiquen un mismo sistema no pueden comunicarse entre sí, porque cada uno de ellos describe individualmente un uso completo del sistema. Pueden interactuar indirectamente a través de actores compartidos.

La interacción entre actores y casos de uso se puede definir mediante interfaces. Una interfaz define las operaciones que puede admitir o utilizar un actor o caso de uso. Las distintas interfaces que ofrecen un mismo caso de uso no tienen necesidad de ser disjuntas.

Los casos de uso están relacionados con otros casos de uso mediante relaciones de generalización, extensión e inclusión.

Generalización. Una relación de generalización relaciona un caso de uso especializado con otro caso de uso más general. El hijo hereda los atributos, operaciones y secuencias de comportamiento del padre y puede agregar atributos y operaciones propios. El caso de uso hijo añade comportamiento al caso de uso padre insertando secuencias de acción adicionales en la secuencia del padre en puntos arbitrarios. También puede modificar algunas operaciones y secuencias heredadas, pero esto tiene que hacerse con igual cuidado que en una redefinición

para que la intención del padre se mantenga. Todas las relaciones de inclusión o extensión con el caso de uso hijo modifican también efectivamente el comportamiento heredado del caso de uso padre.

Extensión. Una relación de extensión es una especie de dependencia. El caso de uso cliente añade un comportamiento incremental al caso de uso base mediante la inserción de secuencias de acción adicionales a la secuencia base. El caso de uso cliente contiene uno o más segmentos separados de secuencia de comportamiento. La relación de extensión contiene una lista de nombres de puntos del caso de uso base, en número igual al número de segmentos que haya en el caso de uso cliente. Un punto de extensión representa una localización o conjunto de localizaciones dentro del caso de uso base en los cuales se podría insertar la extensión. Una relación de extensión también puede tener asociada una condición, que puede hacer uso de atributos del caso de uso padre. Cuando una instancia del caso de uso padre llega a una localización a la que hace referencia un punto de extensión de la relación de extensión, se evalúa la condición; si la condición es verdadera, entonces se ejecuta el correspondiente segmento de comportamiento del caso de uso hijo. Si no hay condición, se estima que es siempre cierta. Si la relación de extensión posee más de un punto de extensión, entonces la condición se evalúa únicamente en el primer punto de extensión antes de ejecutar el primer segmento.

Las relaciones de extensión no crean un nuevo caso de uso instanciable. Lo que hacen es añadir implícitamente comportamiento al caso de uso base original. El caso de uso base incluye implícitamente el comportamiento extendido. El caso de uso base original no extendido no está disponible en su forma no modificada. En otras palabras, si se extiende un caso de uso entonces no se puede instanciar explícitamente el caso de uso base sin la posibilidad de que haya extensiones. Un caso de uso puede tener múltiples extensiones que se aplicarán todas al mismo caso de uso base y se pueden insertar en una instancia de caso de uso si se satisfacen sus distintas condiciones. Por otra parte, un caso de uso de extensión puede extender varios casos de uso base (o el mismo en diferentes puntos de extensión), cada cual en su propio punto de extensión (o lista de puntos de extensión). Si hay varias extensiones en el mismo punto de extensión, su orden relativo de ejecución no es determinista.

Observe que el caso de uso de extensión no debe instanciarse; hay que instanciar el caso de uso base para obtener el comportamiento combinado de base más extensiones. El caso de uso de extensión puede ser o no instanciable pero en todo caso no incluye el comportamiento del caso de uso base.

Inclusión. Una relación de inclusión denota la inclusión de la secuencia de comportamiento del caso de uso proveedor en la secuencia de interacción de un caso de uso cliente, bajo el control del caso de uso cliente en una localización que especifique el cliente en su descripción. Se trata de una dependencia, no de una generalización, porque el caso de uso proveedor no puede ser reemplazado en aquellos casos en que aparece el caso de uso cliente. El cliente puede acceder a los atributos del caso de uso base para obtener valores y comunicar resultados. La instancia de caso de uso está ejecutando el caso de uso cliente. Cuando llega al punto de inclusión, comienza a ejecutar el caso de uso del proveedor hasta que éste finaliza. Después sigue ejecutando el caso de uso cliente más allá de la localización de la inclusión. Los atributos del caso de uso proveedor no tienen valores que persistan entre ejecuciones.

Un caso de uso puede ser abstracto, lo cual significa que se puede instanciar directamente en una ejecución de extensión. Define un fragmento de comportamiento que se especializa o se incluye en casos de uso concretos o quizá sea una extensión de un caso de uso base. También puede ser concreto si puede instanciarse por sí mismo.

Comportamiento. La secuencia de comportamiento de un caso de uso se puede describir mediante una máquina de estados, un grafo de actividades o un código de texto escrito en algún lenguaje ejecutable. Las acciones de la máquina de estados o las sentencias del código pueden llamar a las operaciones internas del caso de uso para especificar los efectos de la ejecución. Las acciones también pueden indicar el envío de mensajes a actores.

Un caso de uso se puede describir informalmente empleando escenarios o un texto normal, pero estas descripciones son imprecisas y sólo están destinadas a su interpretación por parte de seres humanos.

Las acciones de un caso de uso se pueden especificar en términos de llamadas a operaciones del clasificador que describe el caso de uso. Una operación puede ser invocada por más de un caso de uso.

Realización. La realización de un caso de uso se puede especificar mediante un conjunto de colaboraciones. Una colaboración describe la implementación del caso de uso por parte de objetos del clasificador que describe ese caso de uso. Cada colaboración describe el contexto entre constituyentes del sistema dentro del cual se produce una o más secuencias de interacción. Las colaboraciones y sus interacciones definen la forma en que interactúan los objetos del sistema para lograr el comportamiento externo especificado en el caso de uso.

Un sistema se puede especificar con casos de uso en diferentes niveles de abstracción. Un caso de uso que especifica un sistema, por ejemplo, se puede refinar para formar un conjunto de casos de uso subordinados, cada uno de los cuales especifica un servicio de un subsistema. La funcionalidad especificada por el caso de uso de rango superior se puede seguir por completo hasta la funcionalidad de los casos de uso subordinados (de rango inferior). Un caso de uso de rango superior y un conjunto de casos de uso de rango inferior especifican el mismo comportamiento en dos niveles distintos de abstracción. Los casos de uso subordinados cooperan para proporcionar el significado del caso de uso de rango superior. La cooperación de los casos de uso subordinados queda especificada por las colaboraciones del caso de uso de rango superior y se pueden representar mediante diagramas de colaboración. Los actores de un caso de uso de rango superior aparecen como actores de los casos de uso subordinados. Además, los casos de uso subordinados son actores entre sí. Esta realización por capas da lugar a un conjunto anidado de casos de uso y colaboraciones que implementan todo el sistema.

Notación

Los casos de uso se representan mediante una elipse que contiene el nombre del caso de uso. Si es preciso mostrar atributos o operaciones del caso de uso, el caso de uso puede dibujarse como un rectángulo clasificador con la palabra reservada «**use case**». La Figura 13.31 muestra un diagrama de caso de uso.

Un punto de extensión es una entidad con nombre perteneciente a un caso de uso que describe localizaciones en las cuales se pueden insertar secuencias de otros casos de uso. Proporciona un nivel de indirección entre las extensiones y el texto de secuencia de comportamiento. Un punto de texto hace referencia a una localización o conjunto de localizaciones dentro de la secuencia de comportamiento del caso de uso. Se puede cambiar la referencia independientemente de las relaciones de extensión que hagan uso del punto de extensión. Cada punto de extensión debe poseer un nombre único dentro del caso de uso. Los puntos de

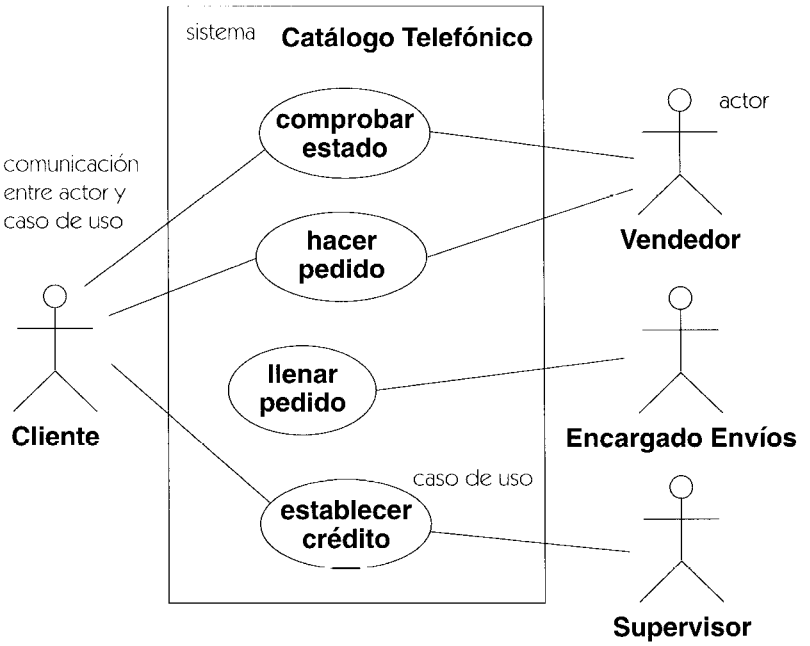


Figura 13.31 Casos de uso y actores

extensión se pueden enumerar en un compartimento del caso de uso, con el encabezado **puntos de extensión** (Figura 13.32).

Una relación de comunicación entre un caso de uso y un actor se muestra empleando un símbolo de asociación —una línea continua entre los símbolos del caso de uso y del actor—. Normalmente se puede omitir la palabra reservada «**communication**» porque éste es el único

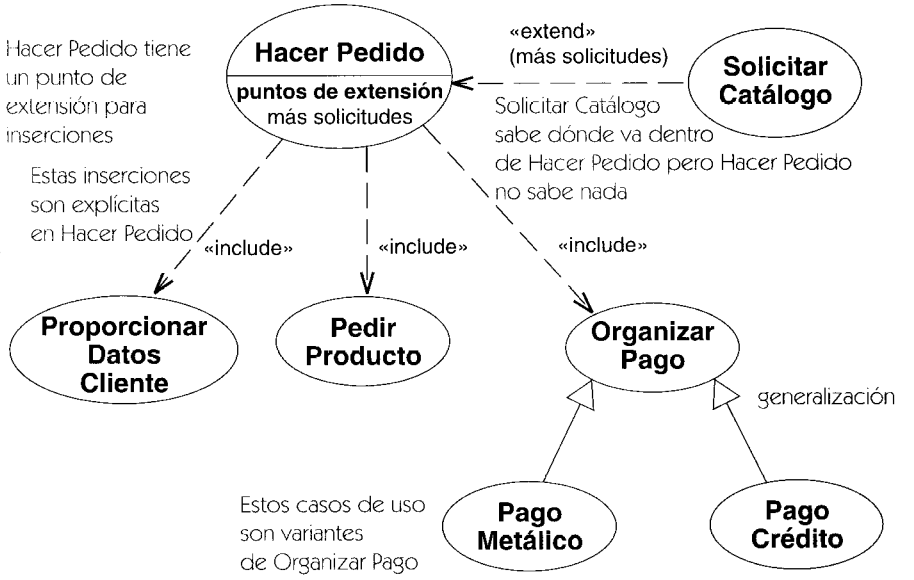


Figura 13.32 Relaciones de caso de uso

tipo de asociación que hay entre actores y casos de uso. Generalmente, no se ponen nombres ni nombres de rol en la línea, porque el actor y el caso de uso definen de modo único la relación.

Una relación de generalización se muestra mediante una flecha de generalización —una línea continua que va del caso de uso hijo al caso de uso padre, con una cabeza de flecha cerrada triangular en el caso de uso padre.

Una relación de extensión o de inclusión se muestran mediante una flecha de dependencia con la palabra reservada «**extend**» o «**include**» —una línea discontinua con cabeza de flecha hueca en el caso de uso cliente—. Las relaciones de extensión también tienen una lista de nombres de puntos de extensión (se pueden eliminar en el diagrama).

La Figura 13.32 muestra distintas clases de relaciones de casos de uso.

Especificación de comportamiento. La relación entre un caso de uso y sus secuencias externas de interacción suele representarse mediante un hipervínculo que llega a un diagrama de secuencias. El hipervínculo es invisible pero se puede recorrer mediante un editor. El comportamiento también se puede especificar mediante una máquina de estados o mediante un texto en algún lenguaje de programación asociado al caso de uso. Se puede emplear un texto en lenguaje natural como especificación informal.

Véase extensión para un ejemplo de secuencias de comportamiento.

La relación entre un caso de uso y su implementación se puede mostrar como una relación de realización que va del caso de uso a la colaboración. Pero dado que éstos suelen estar en modelos separados, suele representarse como un hipervínculo invisible. Se supone que una herramienta admitirá la posibilidad de "acercarse" a los casos de uso para contemplar sus escenarios y/o su implementación como una colaboración.

clase

Descriptor de un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y comportamiento. Una clase representa un concepto dentro del sistema que se está modelando. Dependiendo del tipo de modelo, el concepto puede ser del mundo real (en un modelo de análisis), o contener conceptos algorítmicos o de implementación en un computador (en un modelo de diseño). Un clasificador es una generalización del concepto de clase que incluye otros elementos similares a las clases como tipo de dato, actor y componente.

Semántica

Una clase es la descripción con nombre tanto de la estructura de datos como del comportamiento de un conjunto de objetos. Una clase se utiliza para declarar variables. Un objeto que es el valor de una variable debe tener una clase compatible con el tipo declarado para esa variable, o lo que es lo mismo, debe ser de la misma clase que la declarada o un descendiente de ella. Una clase también se utiliza para instanciar objetos; mediante operaciones de creación, se crean nuevas instancias de una clase.

Un objeto instanciado a partir de una clase es una instancia directa de la clase y una instancia indirecta de sus antecesores. El objeto contiene una ranura que mantiene un valor para

cada atributo; acepta todas las operaciones y señales de la clase, y puede aparecer en enlaces de asociaciones en los que participe la clase o un antecesor de la clase.

Algunas clases no pueden ser instanciadas directamente, ya que se emplean únicamente para describir estructuras compartidas por todos sus descendientes; a estas clases se las denomina clases abstractas. Una clase que puede ser instanciada directamente es una clase concreta.

Una clase también puede verse como un objeto global, siendo todos los atributos de la clase con alcance de clase los atributos de este objeto implícito. Estos atributos tienen un alcance global y cada uno de ellos tiene un único valor para todo el sistema. Una operación con alcance de clase se aplica a la clase, no al objeto, siendo las operaciones de creación las más comunes entre éstas.

En UML una clase es un tipo de clasificador. Los *clasificadores* son elementos similares a las clases, pero su expresión más clara es la *clase*.

Estructura

Una clase consta de un nombre y listas de operaciones, atributos y métodos. Una clase puede participar en una asociación, en una generalización, en dependencias y en relaciones de restricción. Se declara dentro de un espacio de nombres, como un paquete u otra clase, y tiene varias propiedades dentro de dicho espacio de nombres, como la multiplicidad o la visibilidad. Una clase tiene otras propiedades, como si es o no abstracta, o si es activa. Puede tener una máquina de estados que especifique su comportamiento reactivo, es decir, que especifique cómo responde a los eventos que recibe. Una clase puede declarar el conjunto de eventos (incluyendo excepciones) que es capaz de tratar. Puede proporcionar la realización del comportamiento especificado por cero o más interfaces o tipos proporcionando una implementación para dicho comportamiento. Una interfaz lista el conjunto de operaciones que ofrece una clase que promete realizar la interfaz.

Una clase contiene una lista de atributos y una lista de operaciones, cada una de las cuales forma un espacio de nombres dentro de la clase. Los atributos y las operaciones heredadas también aparecen dentro de sus respectivos espacios de nombres. Los espacios de nombres para atributos incluyen también los pseudoatributos, como los nombres de rol de las asociaciones que salen de la clase o los discriminadores de las relaciones de generalización en que participa la clase o alguno de sus antecesores. Un nombre debe declararse sólo una vez dentro de la clase y sus antecesores, pues de otro modo habría un conflicto y el modelo estaría mal formado. Los nombres de las operaciones se pueden repetir siempre que representen la misma operación, pues si no se produciría un conflicto de nombres.

Una clase también es un espacio de nombres y establece el alcance de las declaraciones de clasificadores anidados. Los clasificadores no son partes estructurales de las instancias de la clase. No existe relación entre los objetos de una clase y los objetos de clases anidadas. Una clase anidada es una declaración de una clase que puede ser utilizada por los métodos de la clase externa. Las clases declaradas dentro de otra clase son privadas a dicha clase, y por tanto no pueden ser accedidas desde fuera de la clase a menos que se las haga visibles explícitamente. No hay una notación para representar las declaraciones de clases anidadas, aunque es de esperar que en las herramientas pueda accederse a ellas mediante hipervínculos. Los nombres anidados deben ser referenciados utilizando la ruta del nombre.

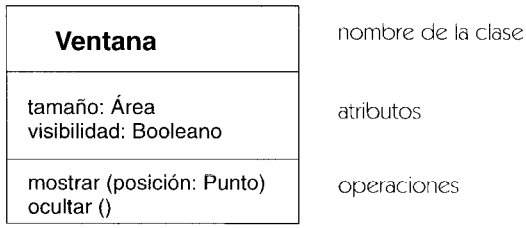


Figura 13.33 Declaración básica de clase

Notación

Una clase se representa mediante un rectángulo con tres partes o compartimentos separados por líneas horizontales. La parte superior contiene el nombre de la clase y otras propiedades que se aplican a la totalidad de la clase. La parte central contiene una lista de atributos, mientras que la parte inferior contiene una lista de operaciones. Estas dos últimas partes se pueden suprimir en un símbolo de clase.

Utilización. Las clases se declaran en diagramas de clases y se utilizan en muchos otros diagramas. UML proporciona una notación gráfica para la declaración y uso de clases, y una notación textual para hacer referencia a las clases dentro de las descripciones de otros elementos. La declaración de una clase en un diagrama de clases define el contenido de la misma: atributos, operaciones y otras propiedades. Otros diagramas definen relaciones adicionales y otros elementos asociados a la clase.

La Figura 13.33 muestra una declaración básica de clase con atributos y operaciones.

La Figura 13.34 muestra la misma declaración de clase con detalles adicionales, sobre todo información sobre la naturaleza de su implementación, como visibilidad, operaciones fuente de creación de alcance de clase y operaciones dependientes de la implementación.

En la Figura 13.35 se suprime toda la información interna sobre la clase. No obstante, la información sigue existiendo en el modelo interno y podría mostrarse en otros diagramas.

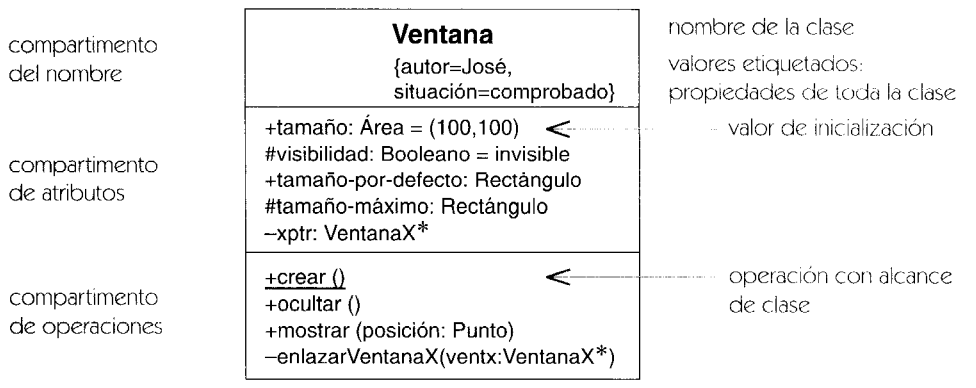


Figura 13.34 Declaración detallada de clase con visibilidad de características

Figura 13.35 Símbolo de clase con todos los detalles suprimidos

Opciones de representación

Supresión de compartimentos. Se puede suprimir cualquiera de los dos compartimentos ya citados (atributos y operaciones) o ambos, tal y como se muestra en la Figura 13.36. Cuando no se muestra un compartimento no se dibuja la línea que lo separa. En ese caso, no se puede inferir nada acerca de la presencia o ausencia de elementos en él. Téngase en cuenta que un compartimento vacío, es decir, uno con separador pero sin contenido, indica que no hay elementos en la lista correspondiente. Si se lleva a cabo algún tipo de filtrado, significa que no existe ningún elemento que satisfaga la condición de filtro. Por ejemplo, si sólo se desea mostrar las operaciones públicas, la presencia de un compartimento de operaciones vacío significa que no hay operaciones públicas, pero no se pueden sacar conclusiones acerca de las operaciones privadas.

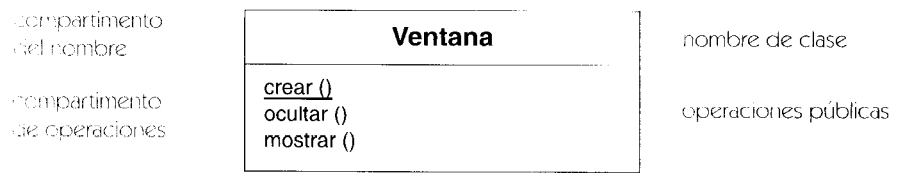


Figura 13.36 Declaración de clase con los atributos y operaciones no públicas suprimidas

Compartimentos adicionales. Se pueden añadir otros compartimentos para mostrar otras propiedades del modelo definidas por el usuario o predefinidas —por ejemplo, para mostrar reglas de negocio, variaciones, responsabilidades, tratamiento de señales, excepciones que genera, etcétera—. Un compartimento adicional se representa utilizando un nombre de compartimento en la parte superior que identifique de forma clara el contenido del mismo (Figura 13.37). Los compartimentos estándar (atributos y operaciones) no requieren nombre de compartimento, aunque pueden llevarlo para enfatizar o para mayor claridad cuando sólo hay un compartimento visible. La mayoría de los compartimentos son únicamente listas de cadenas, donde cada cadena representa una propiedad. Obsérvese que "cadena" puede ser también un icono o documentos empotrados, por ejemplo hojas de cálculo o gráficos. Es posible incluir formatos aún más complejos, pero UML no los especifica, ya que son responsabilidad del usuario y de las herramientas. Si la naturaleza del compartimento viene determinada por la forma de sus contenidos, se puede omitir el nombre del compartimento.

Véase uso de la tipografía, cadena.

Estereotipo. Un estereotipo se representa mediante un texto entre comillas (« ») encima del nombre de la clase, como se muestra en la Figura 13.38. Se puede utilizar también un icono en la esquina superior derecha del compartimento con nombre en lugar de la cadena de texto. Un símbolo de clase con un icono de estereotipo puede ser "comprimido" para mostrar sólo el icono de estereotipo, con el nombre de la clase dentro o debajo del icono (Figura 13.100). Se suprimen los demás contenidos de la clase.

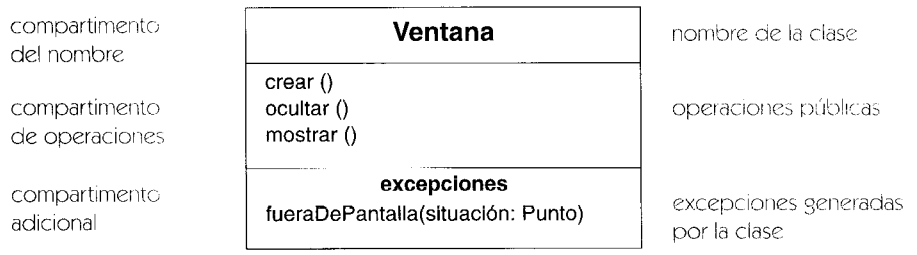


Figura 13.37 Compartimento adicional en la declaración de una clase

Véase estereotipo.

Reglas de estilo

- El nombre del estereotipo debe ir entre comillas, en letra normal y centrado sobre el nombre de la clase.
- El nombre de la clase debe ir en negrita, bien centrado o bien justificado a la izquierda.
- Los atributos y operaciones irán en letra normal justificados a la izquierda.
- El nombre de clases abstractas o la signatura de operaciones abstractas debe aparecer en cursiva.
- Muestre los compartimentos de atributos y operaciones cuando sea necesario (al menos una vez en todo el conjunto de diagramas), y suprímalos en otros contextos o referencias. Es útil definir un lugar "base" para una clase donde se muestre su descripción completa, mostrando la forma mínima en los demás sitios.

Discusión

El concepto de clase se aplica a un amplio espectro de diferentes utilizaciones en el modelado lógico y en la implementación, ya que incluye tanto el concepto de tipo como el concepto de clase de implementación. En UML, y bajo ciertos puntos de variación semántica, un objeto puede tener múltiples clases, así como ser capaz de cambiar de clase en tiempo de ejecución. Otras nociones más restrictivas de clase que aparecen en la mayoría de lenguajes de programación pueden asimilarse como tipos especiales de clases.

Elementos estándar

implementationClass, type.

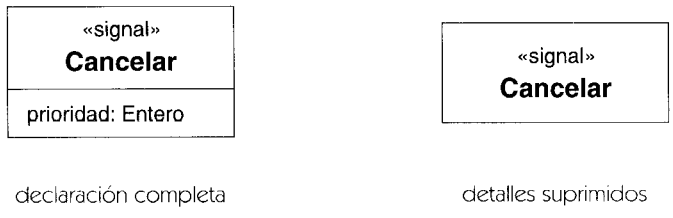


Figura 13.38 Clase con estereotipo

clase abstracta

Clase que no puede ser instanciada.

Véase abstracto/a.

Semántica

Una clase abstracta no puede tener instancias directas. Tiene que tener instancias indirectas a través de sus descendientes concretos.

Para una discusión sobre el tema, *véase* abstracto/a.

clase activa

Clase cuyas instancias son objetos activos.

Véase objeto activo para más detalles.

Semántica

Una clase activa es una clase cuyas instancias son objetos activos. Algunos estereotipos de clase activa son los procesos y los hilos.

Notación

Para indicar que una clase es activa se dibuja con un borde más grueso.

Ejemplo

La Figura 13.39 muestra un diagrama de clases con una clase activa y sus partes pasivas.

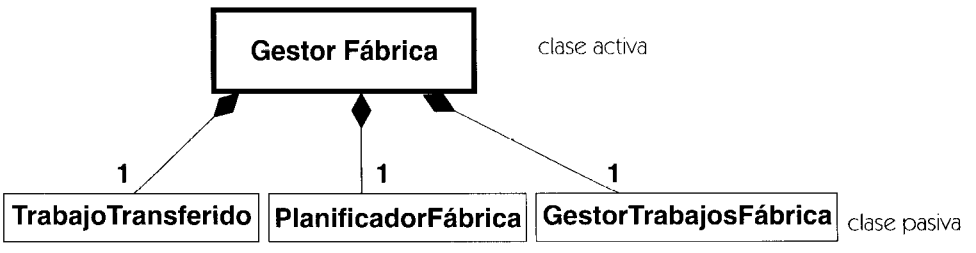


Figura 13.39 Clase activa y partes pasivas

La Figura 13.148 muestra una colaboración que contiene los objetos activos correspondientes a este modelo.

clase asociación

Una clase asociación es una asociación que también es una clase. Una clase asociación tiene propiedades tanto de las clases como de las asociaciones. Sus instancias son enlaces que tienen valores de los atributos a la vez que referencias a otros objetos. Aunque su notación esté formada por los dos símbolos, el de la asociación y el de la clase, es un único elemento del modelo.

Véase también asociación, clase.

Semántica

Una clase asociación tiene las propiedades de las asociaciones, pues conecta dos o más clases, y también tiene atributos y operaciones. Una clase asociación es útil cuando cada enlace debe tener sus propios valores para los atributos, operaciones propias, o sus propias referencias a objetos. Puede verse como una clase con una referencia de clase extra por cada extremo de la asociación, que es la forma obvia y normal de implementarlo. Cada instancia de una clase asociación tiene referencias a objetos, así como los valores para los atributos especificados por la parte de clase.

Una clase asociación C que conecta dos clases A y B no es lo mismo que una clase D con una asociación binaria a cada una de las clase A y B (véase la sección de discusión). Como todos los enlaces, un enlace de una clase asociación como C toma su identidad de las referencias a objetos que contiene. Los valores de los atributos no intervienen para proporcionarle su identidad, por lo que nunca dos enlaces de C podrán tener el mismo par de objetos (a, b) aunque los valores de sus atributos sean diferentes, pues tendrían entonces la misma identidad. Esto es, dado un atributo E, no está permitido que dos instancias de C tengan los valores (a, b, e1) y (a, b, e2) porque comparten la misma identidad (a, b). Sin embargo, los objetos tienen identidad inherente, por lo que dos objetos pueden tener idénticos valores de sus atributos o enlaces a los mismos objetos. En otras palabras, una asociación, incluyendo una clase asociación como C, es un conjunto de tuplas y no tiene duplicados entre sus referencias a objetos; mientras que una relación implícita como D es más como una bolsa, que puede tener duplicados. Para más detalles, véase la sección de discusión.

Las clases asociación pueden tener operaciones que modifiquen los atributos del enlace o que añadan o eliminen enlaces al propio enlace. Como la clase asociación es una clase, puede participar en asociaciones como tal clase.

Una clase asociación no puede tenerse a sí misma como una de sus clases participantes (aunque sin duda se podría encontrar un significado para una estructura recursiva de este tipo).

Notación

Una clase asociación se representa mediante un símbolo de clase (un rectángulo) unido por una línea discontinua a la ruta de una asociación (véase Figura 13.40). El nombre del símbolo de la

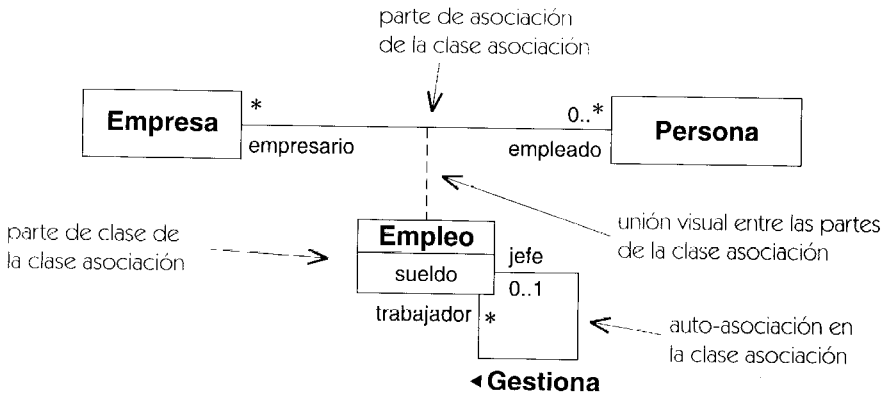


Figura 13.40 Clase asociación

clase y la cadena del nombre que acompaña a la ruta de la asociación son redundantes. La ruta de asociación puede tener los habituales adornos en los extremos. El símbolo de la clase puede tener atributos y operaciones, y participar en otras asociaciones como clase. No hay adornos en la línea discontinua, ya que no es una relación, sino simplemente parte del símbolo global de la clase asociación.

Reglas de estilo

El punto de unión no debería estar tan cerca de uno de los extremos de la asociación que pudiera dar lugar a pensar que está unido al final de la ruta o a alguno de los adornos de los roles.

Obsérvese que la ruta de asociación y la clase asociación son un único elemento del modelo, por lo que tienen un único nombre. El nombre puede aparecer sobre la ruta, en el nombre de la clase o en ambos. Si la clase asociación sólo tiene atributos pero no operaciones u otras asociaciones, el nombre puede mostrarse en la ruta de la asociación y omitirse en el símbolo de la clase asociación para enfatizar su "naturaleza de asociación", mientras que si tiene operaciones u otras asociaciones, es mejor no mostrar el nombre en la ruta y situarlo en el rectángulo de la clase para poner de relieve su "naturaleza de clase". En ninguno de los dos casos cambia la semántica.

Discusión

La Figura 13.40 muestra una clase asociación que representa el empleo. La asociación de empleo entre una empresa y una persona es muchos a muchos. Una persona puede tener varios

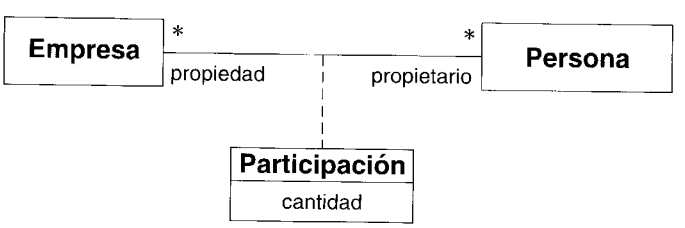


Figura 13.41 Clase asociación con atributo

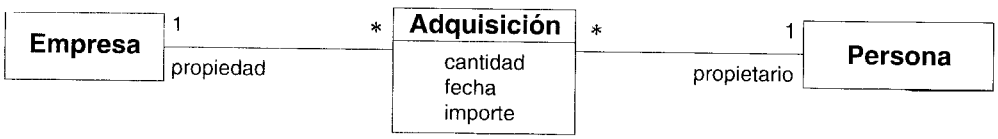


Figura 13.42 Asociación materializada

trabajos, pero sólo uno en una empresa determinada. El sueldo no es un atributo ni de la empresa ni de la persona puesto que la relación es muchos a muchos, por lo que debe ser un atributo de la relación.

La relación trabajador-jefe no es sólo una relación entre dos personas. Es una relación entre una persona en un trabajo y otra persona en otro trabajo. Es una asociación (**Dirige**) entre la clase asociación y ella misma.

El siguiente ejemplo muestra la diferencia entre una clase asociación y una relación materializada modelada como clase. En la Figura 13.41, la propiedad de la empresa se modela como una relación entre Persona y Empresa. El atributo **cantidad** de la clase asociación representa el número de acciones de la empresa que tiene una persona. La relación se modela como una clase asociación porque sólo debería haber una entrada para cada par **Persona-Empresa**.

Para modelar la adquisición de acciones, como se muestra en la Figura 13.42, no utilizamos una clase asociación, ya que puede haber más de una compra de acciones de una empresa por parte de la misma persona. Las compras deben distinguirse porque cada una es distinta y tiene su propia fecha y precio, además obviamente de la cantidad. La relación debe pues materializarse, esto es, dividirse en distintos objetos con identidad propia. La forma correcta de modelarlo es utilizar una clase ordinaria, ya que cada compra tiene su propia identidad, independiente de las clases **Persona** y **Empresa** que relaciona. Ésta es la forma de modelar una relación que es una bolsa en lugar de un conjunto.

clase compuesta

Una clase que está relacionada con una o más clases por una relación de composición.

Véase composición.

clase de implementación

En inglés, Implementation Class. Es un estereotipo de una clase que proporciona una implementación física, incluyendo los atributos, asociaciones a otras clases y métodos para las operaciones. La clase de implementación está orientada a lenguajes orientados a objetos tradicionales con clasificación estática simple. Un objeto de un sistema de estas características debe tener exactamente una clase de implementación como clase directa. Compárese con tipo, un estereotipo de una clase que permite clasificación múltiple. En un lenguaje convencional, tal como Java, un objeto puede tener una sola clase de implementación y muchos tipos. La clase de implementación debe ser consistente con los tipos.

Véase tipo, donde se comparan los tipos con las clases de implementación.

clase directa

La clase que describe lo más completamente posible un objeto.

Véase también clase, generalización, herencia, clasificación múltiple, herencia múltiple.

Semántica

Un objeto puede ser una instancia de muchas clases —si es una instancia de una clase, entonces también es una instancia de los antecesores de la clase. La clase directa es la descripción más específica de un objeto, la que lo describe lo más completamente posible. Un objeto es una instancia directa de su clase directa y una instancia indirecta de los antecesores de la clase directa. Un objeto no es una instancia de ningún descendiente de la clase directa (por definición).

Si la clasificación múltiple está permitida en el sistema, ninguna clase directa puede describir totalmente a un objeto. El objeto puede ser la instancia directa combinada de más de una clase. Un objeto es una instancia directa de cada clase que contenga parte de su descripción, con tal que ningún descendiente describa también al objeto.

En otras palabras, ninguna de las clases directas de un objeto tiene una relación ancestral con las otras.

Si se instancia una clase para producir un objeto, el objeto es una instancia directa de la clase.

clase-en-un-estado

Una clase junto con un estado en que los objetos de la clase pueden estar.

Véase también grafo de actividades.

Semántica

Una clase con una máquina de estados tiene muchos estados, cada uno de los cuales caracteriza el comportamiento, valores y restricciones de las instancias que se encuentran en dicho estado. En algunos casos, ciertos atributos, asociaciones u operaciones sólo son válidos cuando un objeto está en un determinado estado; en otros casos, un argumento de una operación debe ser un objeto en un determinado estado. A menudo estas distinciones son simplemente parte de los modelos de comportamiento, pero a veces es útil modelarlas directamente en vistas estáticas o en vistas de interacción.

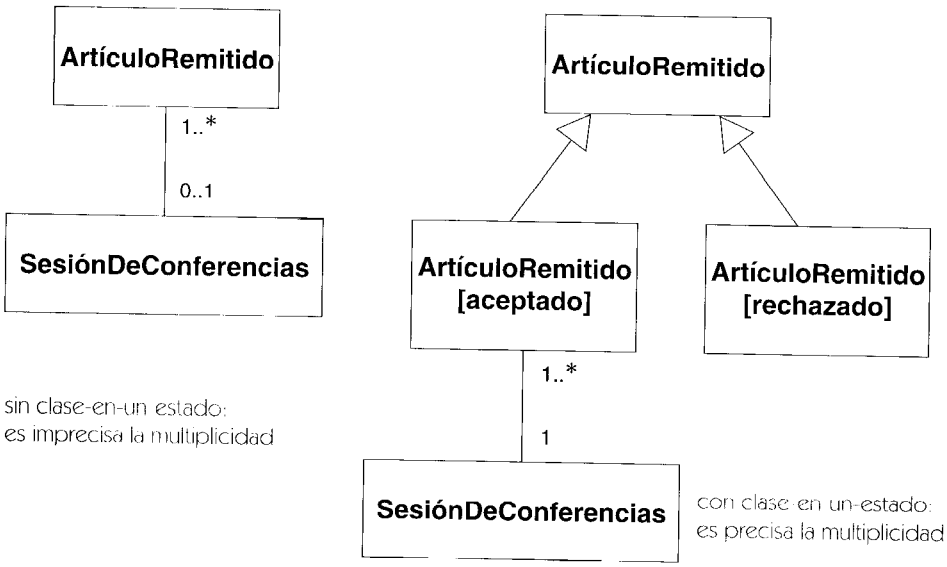


Figura 13.43 Clase-en-un-estado

Una clase en un estado es una clase, junto con uno de los estados válidos en los que los objetos de la clase pueden estar. Si la clase tiene subestados concurrentes, la especificación de estado puede ser un conjunto de subestados en los que la clase puede estar simultáneamente. Una clase en un estado puede utilizarse como clasificador; se comporta como una subclase de la propia clase; puede utilizarse como la clase de una variable o de un parámetro; y puede participar en asociaciones que sólo son válidas para objetos que estén en el estado dado. Considere la asociación **Asignación** de la Figura 13.43 entre las clases **SesiónDeConferencias** y **ArtículoRemitido**. Esta asociación es válida para un **ArtículoRemitido** en el estado **aceptado** (multiplicidad destino igual a uno) pero no en el estado **rechazado**. Para cualquier **ArtículoRemitido**, la multiplicidad destino es cero o uno, puesto que la clase incluye artículos tanto aceptados como rechazados. Sin embargo, si se modela la asociación entre **ArtículoRemitido** en el estado aceptado y **SesiónDeConferencias**, la multiplicidad destino tiene que ser exactamente uno.

Los elementos de una clase-en-un-estado son también útiles para mostrar los valores de entrada y salida de operaciones en grafos de actividades.

Notación

Una clase-en-un-estado se representa mediante un símbolo de clase en el cual el nombre de la clase va seguido del nombre del estado entre corchetes (NombreClase[nombreEstado]). Los corchetes también pueden contener una lista de nombres de estados concurrentes separados por comas para indicar que un objeto está en varios estados.

Discusión

Las clases-en-un-estado y la clasificación dinámica son dos formas de dar cumplimiento al mismo objetivo de permitir a las clases cambios en la estructura de los objetos durante su vida. Se utilizará el más conveniente de los dos dependiendo del entorno de implementación.

clase unitaria

Clase que posee (por declaración) una única instancia. Una clase unitaria es la forma de representar el conocimiento global en una aplicación, manteniéndolo en un marco de trabajo orientado a objetos.

Semántica

Toda aplicación debe poseer al menos una clase unitaria (frecuentemente, de forma implícita) para establecer el contexto de la aplicación. Con frecuencia, la clase unitaria equivale a la aplicación en sí y está implementada por la pila de control y el espacio de direcciones del computador.

Notación

Las clases unitarias se representan mediante un símbolo de clase con un pequeño "1" en la esquina superior derecha (Figura 13.44). Este valor representa la multiplicidad de la clase dentro del sistema.

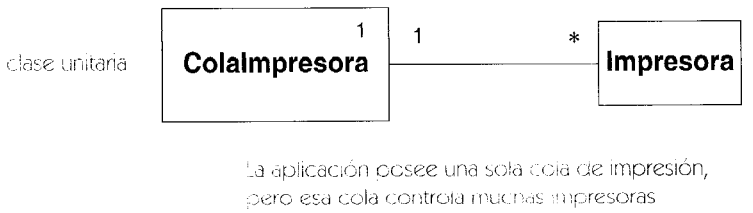


Figura 13.44 Clase unitaria

clasificación dinámica

Una variación semántica de la generalización en la que un objeto puede cambiar el tipo o el rol. Contrastar con: clasificación estática.

Véase también clasificación múltiple.

Semántica

En muchos lenguajes de programación, un objeto no puede cambiar la clase de la cual es instanciado. Esta restricción estática de la clasificación simplifica la implementación y la optimización de compiladores, pero no es una necesidad lógica. Por ejemplo, asumiendo clasificación estática, un objeto instanciado como un círculo debe seguir siendo un círculo; no se puede cambiar su tamaño en la dimensión X, por ejemplo. Suponiendo que hay clasificación dinámica, un círculo del que se cambia dimensión se convierte en una elipse. Esto no se considera un problema.

Cualquier asunción puede utilizarse en un modelo de UML. Éste es un ejemplo de un punto de variación semántica. Sorprendentemente, la elección afecta poco al modelo, aunque

las diferencias son importantes para la ejecución. En ambos casos se definen las mismas clases, pero sus operaciones pueden diferir.

clasificación estática

Variante semántica de la generalización en la cual un objeto no puede cambiar de tipo o no puede cambiar de rol. La selección de la clase estática frente a la clasificación dinámica es un punto de variación semántica.

clasificación múltiple

Se trata de una variación semántica de generalización en la cual un objeto puede pertenecer directamente a más de una clase.

Semántica

Éste es el punto de variación semántica en el cual un objeto puede ser instancia directa de más de una clase. Cuando se usa con clasificación dinámica, los objetos pueden adquirir y perder clases durante el tiempo de ejecución. Éste permite utilizar clases para representar roles temporales que puede desempeñar un objeto.

Aun cuando la clasificación múltiple se adapta bien a la lógica y al discurso habitual, complica la implementación de un lenguaje de programación y no es admitida por los lenguajes populares de programación.

clasificación simple

Régimen de ejecución en que cada objeto posee exactamente una clase directa. Es el modelo de ejecución de la mayoría de los lenguajes de programación orientados a objetos. Admitir la clasificación simple o la clasificación múltiple es un punto de variación semántica.

clasificador

Elemento del modelo que describe características estructurales y de comportamiento. Las clases, actores, componentes, tipos de datos, interfaces, nodos, señales, subsistemas y casos de uso son tipos de clasificadores. El tipo más general de clasificador es la clase. Otros tipos pueden tener un cierto parecido intuitivo con las clases, con ciertas restricciones sobre el contenido y la utilización, aunque cada tipo de clasificador está representado por su propia clase en el metamodelo. En general, la mayoría de las propiedades de las clases se aplican a los clasificadores, con ciertas restricciones según el tipo de clasificador.

Véase también elemento generalizable, vista estática.

Elementos estándar

enumeration, location, metaclass, persistence, powertype, process, semantics, stereotype, thread, utility.

cliente

Elemento que solicita un servicio a otro elemento. El término se utiliza para describir un rol dentro de una dependencia. En la notación, el cliente aparece en el origen de una flecha de dependencia. Antónimo: proveedor.

Véase dependencia.

colaboración

Descripción de una disposición general de objetos y enlaces que interactúan dentro de un contexto para implementar un comportamiento, como un caso de uso o una operación. Una colaboración tiene una parte estática y una parte dinámica. La parte estática describe los roles que pueden desempeñar los objetos y enlaces en una instancia de la colaboración. La parte dinámica está formada por una o más interacciones dinámicas que muestran flujos de mensajes en la colaboración a través del tiempo para realizar cómputos.

Véase también mensaje, interacción, rol de asociación, rol de clasificador.

Semántica

El comportamiento lo implementan grupos de objetos que intercambian mensajes en un contexto determinado para llevar a cabo un cierto propósito. Para comprender el mecanismo utilizado en un diseño, es importante fijarse en los objetos y mensajes implicados en la realización del propósito o del conjunto de propósitos relacionados, proyectados desde el sistema más amplio dentro del cual realizan también otros propósitos. Una disposición de objetos y enlaces que trabajan juntos para llevar a cabo un objetivo se denomina colaboración; una secuencia de mensajes que implementan un comportamiento dentro de una colaboración se denomina interacción. Una colaboración es una descripción de una "sociedad de objetos"; es un fragmento de un modelo más amplio y más completo, dentro del cual la colaboración es un propósito.

Por ejemplo, una venta representa una disposición de objetos que tienen ciertas relaciones entre ellos dentro de la transacción. Las relaciones no tienen sentido fuera de la transacción. Los participantes en una venta son el comprador, el vendedor y un intermediario. Para realizar una interacción específica, como la venta de una casa, los participantes intercambian una determinada secuencia de mensajes, como la presentación de una oferta o la firma de un contrato.

Los flujos de mensajes de una colaboración pueden especificarse de forma opcional utilizando una máquina de estados que muestre las secuencias válidas de comportamiento. Los eventos de la máquina de estados representan los mensajes que los roles intercambian dentro de la colaboración.

Una colaboración está formada por roles. Un rol es una parte que desempeña un clasificador o una asociación dentro de la colaboración. Un rol es una ranura que puede contener una instancia de un clasificador o de una asociación. El mismo clasificador o la misma asociación pueden desempeñar diferentes roles, cada uno de los cuales podría ser ocupado por un objeto o enlace diferente. Por ejemplo, dentro de una transacción comercial, una parte puede ser el vendedor y otra el comprador, incluso aunque ambos sean empresas. El **comprador** y el **vendedor** son instancias de la clase **Empresa** dentro de la colaboración **Venta**. Los roles sólo tienen significado dentro de una colaboración; fuera de ella carecen de sentido. Además, en otras colaboraciones se pueden intercambiar los roles: un objeto puede ser **comprador** en una instancia de la colaboración y **vendedor** en otra. El mismo objeto puede desempeñar múltiples roles en colaboraciones diferentes. Contraste el restringido alcance de un rol con una asociación. Una asociación describe una relación con significado completo para una clase en todos los contextos, participe o no realmente un objeto en la asociación. Una colaboración define relaciones que se encuentran restringidas a un contexto y que por tanto no tienen sentido fuera del mismo.

Realización. Una colaboración realiza una operación o un caso de uso. Describe un contexto en el que se ejecuta la implementación de una operación o caso de uso —esto es, la disposición de objetos y enlaces existente en el momento de inicio de la ejecución, y las instancias que se crean y se destruyen durante la misma—. Se pueden especificar las secuencias de comportamiento en interacciones, representándolas como diagramas de secuencia o como diagramas de colaboración.

Una colaboración también puede realizar la implementación de una clase. Una colaboración para una clase es la unión de las colaboraciones para sus operaciones. Se pueden concebir diferentes colaboraciones para la misma clase, sistema o subsistema, de forma que cada colaboración muestre el subconjunto de atributos, operadores y objetos relacionados relevantes en una determinada vista de la entidad, como la implementación de una operación concreta.

Patrones. Una colaboración parametrizada representa una construcción de diseño que puede ser reutilizada en varios diseños. Generalmente los clasificadores base son parámetros. Una colaboración parametrizada de este tipo captura la estructura de un *patrón*.

Véase plantilla.

Un patrón de diseño se instancia proporcionando clasificadores reales para los parámetros de clasificadores base.

Cada instanciación produce una colaboración entre un conjunto específico de clasificadores del modelo. Un patrón puede ser ligado una o más veces con diferentes conjuntos de clasificadores dentro de un modelo, evitando así la necesidad de definir una colaboración para cada ocurrencia. Por ejemplo, un patrón de una vista del modelo define una relación general entre elementos del modelo, por lo que puede ligarse con muchas parejas de clases para representar parejas de vistas del modelo. Cada pareja de clases vista del modelo representará pues una ligadura del patrón. Una pareja como ésta podría ser una casa y una fotografía de la casa, otra pareja podría ser un stock y una gráfica que mostrara el precio actual del stock.

Obsérvese que un patrón implica también la existencia de unas directrices para su uso, ventajas y desventajas. Estas directrices pueden ponerse en notas o en documentos de texto independientes.

Capas de colaboraciones. Una colaboración se puede expresar en diferentes niveles de granularidad. Una colaboración de grano grueso puede refinarse para producir otra colaboración de grano más fino, expandiendo una o más operaciones de una colaboración de alto nivel en diferentes colaboraciones de más bajo nivel, una por cada operación.

Una colaboración se puede implementar utilizando colaboraciones subordinadas, de forma que cada colaboración subordinada implemente una parte de la funcionalidad global y tenga su propio conjunto de roles. Cada rol de la colaboración global puede ligarse a uno o más roles de las composiciones anidadas. Si un rol de nivel externo se liga a más de un rol de nivel interno, entonces implícitamente conecta el comportamiento de las colaboraciones de más bajo nivel. Ésta es la única forma de interacción entre casos de uso. Un enfoque de diseño significa trabajar de dentro hacia fuera: primero construir los roles más internos y reducidos, y luego irlos combinando para producir roles más amplios y externos que tengan múltiples responsabilidades.

Ligadura en tiempo de ejecución. En tiempo de ejecución, los objetos y enlaces están ligados a los roles de la colaboración. Un objeto puede estar ligado a uno o más roles, normalmente en diferentes colaboraciones. Si un objeto está ligado a múltiples roles, entonces se dice que representa una interacción "accidental" entre los roles —esto es, una interacción que no es inherente a los propios roles, sino un efecto lateral de su uso en un contexto más amplio—. A menudo, un objeto desempeña roles en más de una colaboración como parte de una colaboración más amplia. Este solapamiento entre colaboraciones proporciona un flujo implícito de control y de información entre ellas.

Estructura

Roles. Una colaboración contiene un conjunto de roles, cada uno de los cuales es una referencia a uno o más clasificadores base (rol de clasificador) o asociaciones base (rol de asociación). Un rol es una ranura en la colaboración que describe un uso de un clasificador o una asociación dentro de la colaboración. Un rol es también un clasificador, pues un objeto ligado al rol en una instancia de la colaboración es una instancia transitoria del rol. Dentro de una instancia de la colaboración, se liga un objeto a cada rol de clasificador y un enlace a cada rol de asociación. Se puede ligar un objeto a más de un rol de clasificador de una misma instancia de una colaboración, aunque no es lo normal, e incluso puede ser deseable evitarlo mediante una restricción. Los objetos de la misma clase pueden aparecer en múltiples roles de la misma instancia de colaboración. Cada clasificador puede mantener una lista de las características de clasificador utilizadas en la colaboración. Las otras características son irrelevantes en la colaboración, si bien pueden ser utilizadas en otras colaboraciones. Si un mismo clasificador base está implicado en múltiples roles, éstos deberían tener un nombre que permitiera distinguirlos, mientras que si sólo hay un uso del clasificador en una colaboración, el rol puede ser anónimo ya que basta el clasificador para identificarlo. Los roles definen la estructura de la colaboración.

Si hay clasificación múltiple, el rol puede tener múltiples clasificadores base, de forma que un objeto ligado al rol de clasificador será una instancia de cada uno de ellos.

Generalizaciones. Una colaboración también puede incluir generalizaciones y restricciones, que se añaden a las relaciones que puedan tener los clasificadores participantes fuera de la colaboración. Una generalización es necesaria en una colaboración sólo cuando los clasificadores de la colaboración sean parámetros; si no, su estructura de generalización se especificará como parte de su definición como clasificadores y no podrá ser alterada por la colaboración. En una colaboración parametrizada (un patrón) algunos de los roles de clasificador pueden ser

parámetros. Una generalización entre dos roles de clasificador parametrizados indica que todos los clasificadores ligados a los roles deben satisfacer la relación de generalización (el clasificador ligado al rol padre debe ser un antecesor del clasificador ligado al rol hijo, pero no es necesario que sean padre e hijo).

Por ejemplo, el patrón **Composite** (**Compuesto**) de [Gamma-95] representa un árbol de objetos recursivo en el que la clase **Componente** es un elemento genérico del árbol, **Compuesto** es un elemento recursivo y **Hoja** es un elemento hoja (Figura 13.45). **Componente** es el padre de **Hoja** y **Compuesto**, el último de los cuales es un agregado de elementos **Componente** (he aquí la recursividad). **Componente**, **Compuesto** y **Hoja** son parámetros del patrón que son sustituidos por clases reales cuando se utiliza el mismo. Cualquier conjunto de clases reales ligado al patrón debe observar la relación antecesor-sucesor entre **Componente** y sus hijos **Compuesto** y **Hoja**. Algunos ejemplos de sustitución podrían ser **Gráfico**, **Dibujo** y **Rectángulo**; **EntradaDeDirectorio**, **Directorio** y **Archivo**; y otras clases recursivas. Si una ligadura no cumple la restricción de generalización, estará mal formada.

Restricciones. Las restricciones se pueden definir sobre roles parametrizados y no parametrizados. Estas restricciones se añadirán a las que puedan existir sobre los clasificadores ligados a los roles. Las restricciones se aplican a todas las instancias de la colaboración.

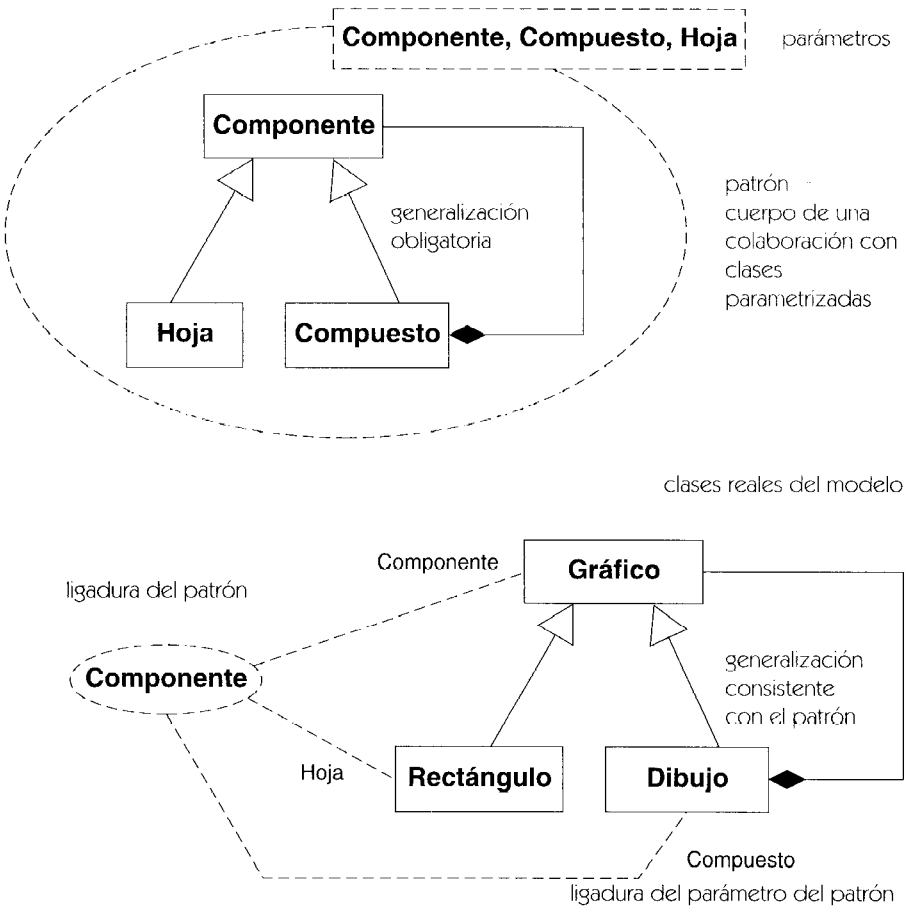


Figura 13.45 Patrón: una colaboración parametrizada

Mensajes. Una colaboración puede tener un conjunto de mensajes para describir su comportamiento dinámico. Una colaboración con mensajes es una interacción. Puede haber múltiples interacciones, cada una de las cuales describe parte de una misma colaboración. Una interacción puede describir la implementación de una operación mientras otra describe otra operación, por ejemplo. Cada mensaje contiene información que indica el orden del mismo en la secuencia de mensajes; dicha información equivale a la especificación de una máquina de estados disparada por mensajes.

Origen. Una colaboración puede representar una clase, caso de uso o método (una colaboración es la implementación de una operación, no su especificación). La colaboración describe el comportamiento del objeto origen.

Notación

Un diagrama de colaboración es un grafo de símbolos de clase (rectángulos) que representan roles de clasificador y rutas de asociación (líneas continuas) que representan roles de asociación, con símbolos de mensaje asociados a sus caminos de roles de asociación. Un diagrama de colaboración sin mensajes muestra el *contexto* en el que pueden ocurrir interacciones, sin mostrar ninguna interacción. Puede utilizarse para mostrar el contexto de un evento u operación simple para todas las operaciones de una clase o grupo de clases. Si existen mensajes asociados a las líneas de asociación, el diagrama muestra una interacción. Típicamente, una interacción representa la implementación de una operación o caso de uso.

Un diagrama de colaboración muestra ranuras para los objetos implicados en la misma, mediante roles de clasificador. Un rol de clasificador se distingue de un clasificador porque tiene un nombre y una clase, con la sintaxis nombreRol : nombreClase. Se puede omitir el nombre de la clase o el del clasificador, pero los dos puntos son necesarios. Un diagrama también muestra los enlaces entre los objetos como roles de asociación, incluyendo los enlaces transitorios que representan argumentos de procedimientos, variables locales y *auto*-enlaces. Varios roles de asociación pueden tener el mismo nombre de asociación, suponiendo que conectan diferentes roles de clasificador. Las flechas situadas en las líneas de los enlaces indican la dirección de navegación, de manera que una punta de flecha sobre una línea que une cajas de objetos indica un flujo de mensajes que fluye en la dirección dada, no pudiendo fluir los mensajes en sentido contrario al especificado en un enlace de un solo sentido, por lo que los flujos de mensajes deben ser compatibles con las flechas de navegación.

Generalmente, no se muestran de forma explícita los valores de atributos individuales en los roles de clasificador. Si hay que enviar mensajes a valores de los atributos, dichos atributos deberían modelarse como objetos que utilizan asociaciones.

Se pueden emplear herramientas u otras aplicaciones gráficas para sustituir o complementar las palabras clave. Por ejemplo, cada tipo de vida podría mostrarse con un color diferente. Es más, la herramienta podría incluso utilizar la animación para mostrar la creación y destrucción de elementos y el estado del sistema en diferentes momentos.

Implementación de una operación

Una colaboración que muestra la implementación de una operación incluye símbolos para el rol de objeto origen y para los roles de otros objetos que éste utiliza, directa o indirectamente, para

llevar a cabo la operación. Los mensajes que aparecen sobre los roles de asociación representan el flujo de objeto de una interacción.

Una colaboración que describe una operación también incluye símbolos de rol que representan los argumentos de la operación, y variables locales creadas durante su ejecución. Los objetos creados durante la ejecución pueden designarse mediante {**new**}, los objetos destruidos durante la misma se pueden designar mediante {**destroyed**}, y los creados durante la ejecución y destruidos durante su transcurso se pueden designar utilizando {**transient**}. Los objetos sin palabra clave existen ya al iniciarse la operación y siguen existiendo tras su terminación.

Los mensajes internos que implementan un método están numerados comenzando a partir del número 1. En un flujo de objeto procedimental, los números asociados a los mensajes emplean una secuencia de números anidada utilizando la notación de puntos en concordancia con los niveles de anidamiento de llamadas. Por ejemplo, el segundo paso del nivel superior llevará asociado el número 2; el primer paso subordinado dentro de éste será el mensaje 2.1. En los mensajes asíncronos intercambiados entre objetos concurrentes, todos los números de secuencia están al mismo nivel (es decir, no están anidados).

Véase mensaje, donde se incluye una completa descripción de la sintaxis de los mensajes que incluye los números de secuencia.

Un diagrama de colaboración completo muestra los roles de todos los objetos y enlaces utilizados por la operación. Si no se muestra un objeto, se asume que no se utiliza. Sin embargo, no es seguro asumir que una operación utiliza todos los objetos que aparecen en un diagrama de colaboración.

Ejemplo

En la Figura 13.46 se invoca la operación **actualizarVisualización** sobre un objeto **Controlador**. En el momento de invocarse la operación, ésta ya tiene un enlace al objeto **Ventana** donde se va a mostrar la imagen. También tiene un enlace a un objeto **Cable**, el objeto cuya imagen será visualizada en la ventana.

La implementación de más alto nivel de la operación **actualizarVisualización** tiene un solo paso —la invocación a la operación **mostrarPosiciones** del objeto **cable**—. Esta operación tiene el número de secuencia 1 puesto que se trata del primer método de nivel superior. Este mensaje fluye a través de una referencia al objeto **Ventana** que se necesitará más adelante.

La operación **mostrarPosiciones** llama a la operación **dibujarSegmento** del propio objeto **cable**. La llamada, etiquetada con el número de secuencia 1.1, se trata a través del enlace implícito **self**. El asterisco indica una llamada iterativa a la operación cuyos detalles se muestran entre corchetes.

Cada operación **dibujarSegmento** accede a dos objetos **SegmentoCable**, uno indexado por el valor **i-1** y otro por el valor **i**. Aunque sólo hay una asociación entre **Cable** y **SegmentoCable**, en el contexto de esta operación son necesarios dos enlaces a dos objetos **Cable**. Los objetos están etiquetados como **izquierdo** y **derecho**, que son los roles de clasificador en la colaboración. Cada mensaje fluye por cada uno de los enlaces, etiquetándose como mensaje 1.1.1a y 1.1.1b, lo que indica que son pasos de la operación 1.1. Las letras del final indican que los dos mensajes pueden ser tratados concurrentemente. En una implementación normal, probablemente no

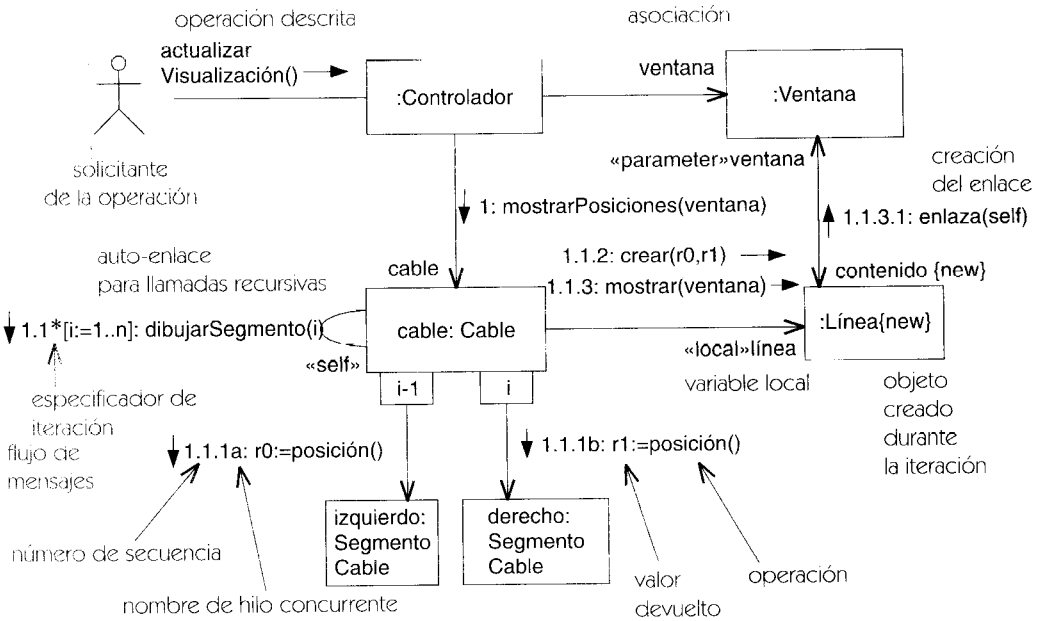


Figura 13.46 Diagrama de colaboración con flujo de mensajes

se ejecutarían en paralelo, pero como se han declarado como concurrentes pueden ejecutarse en cualquier orden de secuencia.

Cuando ambos valores (**r0** y **r1**) han sido devueltos, puede continuar el siguiente paso de la operación 1.1. El mensaje 1.1.2 es un mensaje de creación que se envía a un objeto **Línea**. Realmente se envía a la propia clase **Línea** (al menos en principio), que crea un nuevo objeto Línea enlazado al objeto que envió el mensaje. El nuevo objeto tiene la etiqueta {**new**} para indicar que ha sido creado durante la operación pero que sigue viviendo después. El nuevo enlace tiene la etiqueta «**local**» para indicar que se trata de una variable local al procedimiento. Las variables locales son inherentemente transitorias, por lo que desaparecen cuando finaliza el procedimiento y por tanto no es necesario etiquetarlas con la palabra clave {**transient**}.

El paso 1.3 utiliza el enlace recién creado para enviar un mensaje mostrar al recién creado objeto **Línea**. El puntero al objeto **ventana** se pasa como argumento, de forma que el objeto **Línea** pasa a tener acceso al mismo a través de un enlace «**parameter**». Obsérvese que el objeto **Línea** tiene un enlace al mismo objeto **ventana** que está asociado con el objeto original **Controlador**; esto es importante para la operación y por ello se muestra en el diagrama. En el paso final, 1.3.1, se solicita al objeto **Ventana** la creación de un enlace al objeto **Línea**. Este enlace *es* una asociación, por lo que lleva el nombre de rol **contenido**, y está etiquetado como {**new**}.

El estado final del sistema puede observarse eliminando mentalmente todos los enlaces temporales. Existe un enlace desde **Controlador** hasta **cable** y desde **cable** hasta sus **SegmentoCable**, un enlace de **Controlador** a **ventana** y otro desde **ventana** hasta su **contenido**. Una vez terminada la operación, una **Línea** no tiene acceso a la **Ventana** que la contiene, por lo que el enlace en esa dirección es transitorio y desaparece al finalizar la operación. De forma similar, un objeto **Cable** no tiene por qué seguir teniendo acceso a los objetos **Línea** utilizados para mostrarlo.

colaboración entre instancias

Las colaboraciones se pueden expresar como descriptores y como instancias, al igual que muchos otros elementos del modelo. Una colaboración entre descriptores muestra una relación potencial entre objetos, pudiendo instanciarse muchas veces la colaboración para producir instancias de colaboración. Cada instancia de colaboración muestra una relación entre objetos específicos: es una colaboración entre instancias.

Pero, ¿qué forma debería utilizarse para modelar una determinada situación? Si el diagrama muestra contingencia, debe emplearse un diagrama de descriptores, ya que los diagramas de objetos no tienen contingencia, es decir, no tienen estructuras condicionales ni bucles, ya que estos elementos forman parte de las descripciones genéricas. Una instancia no tiene un rango de valores ni un posible conjunto de caminos de control: tiene un valor concreto y una historia de control.

Si el diagrama muestra valores específicos de atributos o argumentos, si muestra un número específico de objetos o enlaces de entre los muchos posibles en una multiplicidad de tamaño variable, o si muestra una elección particular de bifurcaciones y bucles durante una ejecución, debe ser un diagrama de instancias.

En muchos casos es posible utilizar las dos formas, generalmente cuando la ejecución no tiene bucles. En estos casos cualquier ejecución es prototípica, y no hay mucha diferencia entre la forma de descriptor y la forma de instancia.

combinación

Tipo de relación que relaciona dos partes de la descripción de un clasificador, combinándolas para producir el descriptor completo del elemento.

Véase extender, incluir.

Semántica

Una de las potentes capacidades de la orientación a objetos es la posibilidad de combinar descripciones de elementos del modelo a partir de piezas incrementales. La herencia combina clasificadores relacionados por la generalización para producir el descriptor efectivo completo de una clase.

Otras formas de combinación de descriptores son las relaciones de extensión e inclusión. Estas relaciones se modelan como variaciones de la relación de combinación. La generalización también podría englobarse en esta misma categoría, aunque debido a su importancia se trata como una relación fundamental diferente.

Notación

Una relación de combinación se representa mediante una flecha con línea discontinua con una palabra clave de estereotipo asociada. *Véase* extender e incluir para ver los detalles específicos de cada una.

Discusión

Son posibles otros tipos de relación de combinación. Algunos lenguajes de programación, como CLOS, implementan algunas potentes variantes de combinación de métodos.

comentario

Anotación asociada a un elemento o colección de elementos. Un comentario no tiene significado directo, pero puede mostrar información semántica u otra información significativa para el que realiza el modelado o para la herramienta, por ejemplo una restricción o el cuerpo de un método.

Véase también nota.

Semántica

Un comentario está formado por una cadena de texto pero también puede incluir documentos embebidos si lo permite la herramienta de modelado utilizada. Un comentario puede asociarse con un elemento del modelo, con un elemento de presentación o con un conjunto de elementos. Proporciona una descripción textual de información arbitraria, pero no tiene impacto semántico. Los comentarios proporcionan información a quienes realizan el modelo y pueden utilizarse para buscar modelos.

Notación

Los comentarios se muestran en símbolos de nota, que se representan mediante rectángulos con la esquina superior derecha doblada en forma de "oreja de perro" asociados mediante una línea o líneas discontinuas al elemento o elementos al que se aplica el comentario (Figura 13.47). Las herramientas de modelado pueden proporcionar libremente formas adicionales de mostrar comentarios y navegar por ellos, como comentarios *pop-up*, tipos de letra especiales, etc.

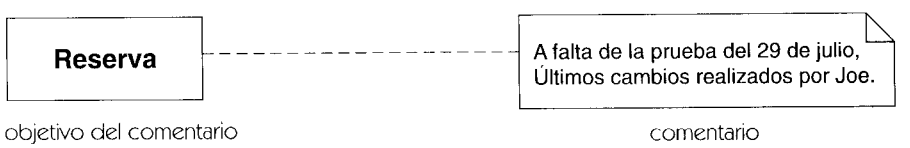


Figura 13.47 Comentario

Elementos estándar

requirement, responsibility.

comillas

Los corchetes angulares pequeños (« ») se emplean para denotar cita en Francés, Italiano, Español y otros idiomas. En la notación UML se emplean para denotar palabras reservadas y nombres de estereotipos. Por ejemplo: «**bind**», «**instanceOf**». Las comillas están disponibles en casi todas las tipografías así que realmente no hay excusa para no utilizarlas, pero quienes tengan problemas tipográficos pueden reemplazarlas por dos corchetes angulares (<< >>) si fuera necesario.

Véase también uso de la tipografía.

compartimento

División gráfica de un símbolo cerrado; por ejemplo un símbolo de clase se divide verticalmente en rectángulos más pequeños. Cada compartimento muestra las propiedades del elemento que representa. Los compartimentos pueden ser de tres tipos: fijos, listas y regiones.

Véase también clase, clasificador.

Notación

Un *compartimento fijo* tiene un formato fijo de partes gráficas y de texto para representar un conjunto fijo de propiedades. El formato depende del tipo de elemento. Un ejemplo sería un compartimento de nombre de clase, que contiene un símbolo de estereotipo y/o nombre, un nombre de clase, y una cadena de propiedades que muestra varias propiedades de la clase. Dependiendo del elemento, puede suprimirse alguna información.

Un *compartimento lista* contiene una lista de cadenas que codifican las partes que constituyen el elemento. Un ejemplo sería una lista de atributos. Los elementos de la lista se pueden mostrar en su orden natural dentro del modelo u ordenados por una o más propiedades (en cuyo caso, el orden natural no sería visible). Por ejemplo, una lista de atributos podría ordenarse primero por visibilidad y en segundo lugar por nombre. Las entradas de la lista pueden mostrarse o suprimirse basándose en las propiedades de los elementos del modelo. Un compartimento de atributos, por ejemplo, podría mostrar únicamente atributos públicos. Se pueden aplicar estereotipos y palabras clave a los constituyentes individuales colgándolos de la correspondiente entrada de la lista. Se pueden aplicar estereotipos y palabras clave a todos los constituyentes que siguen a uno dado situándolos sobre la lista de entradas, de forma que afecten a todas las entradas subsiguientes de la lista hasta el final de la misma o hasta que se encuentre otra declaración similar. La cadena «**constructor**» situada en una línea separada en una lista de operaciones provocaría que se aplicara el estereotipo de constructor a todas las operaciones subsiguientes, pero si apareciera la cadena «**query**» más adelante en la lista, revocaría la primera declaración y la remplazaría por el estereotipo «**query**».

Una *región* es un área que contiene una subimagen gráfica que muestra una subestructura del elemento, a menudo potencialmente recursivo. Un ejemplo sería una región de estado anidado. La naturaleza de la subimagen es característica de cada elemento del modelo. Se pueden incluir en un mismo símbolo regiones y compartimentos de texto, pero puede resultar

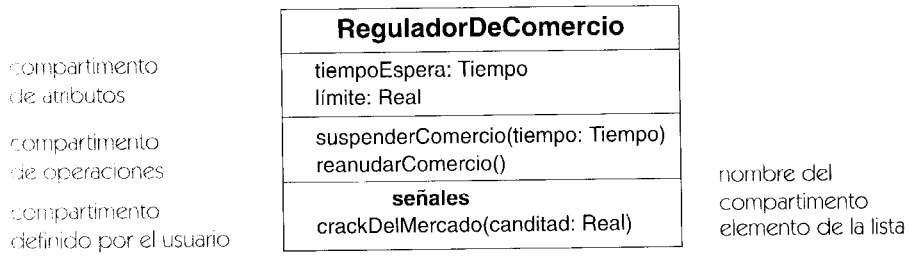


Figura 13.48 Compartimento con nombre en una clase

desordenado. Las regiones se utilizan frecuentemente en elementos recursivos, mientras que el texto se emplea en elementos hoja sin subestructura recursiva.

Una clase tiene tres compartimentos predefinidos: nombre, que es un compartimento fijo; atributos, que es un compartimento lista; y operaciones, que es otro compartimento lista. El que realiza el modelo puede añadir otro rectángulo y situar su nombre en la parte superior del compartimento con un tipo de letra distintivo (por ejemplo en pequeño y en negrita).

La sintaxis gráfica depende del elemento y del tipo de compartimento. La Figura 13.48 muestra un compartimento para señales. La Figura 13.166 muestra un compartimento para responsabilidades.

compendio

Resultado de filtrar, combinar y abstraer las propiedades de un conjunto de elementos dentro de su contenedor para dar una visión más abstracta, de más alto nivel, de un sistema.

Véase paquete.

Semántica

Los contenedores, tales como los paquetes y las clases, pueden tener propiedades y relaciones derivadas que resuman las propiedades y relaciones de su contenido. Esto permite al creador de modelos tener una mejor comprensión del sistema en un nivel más alto y con menos detalle, que resulta más fácil de comprender. Por ejemplo, una dependencia entre dos paquetes indica que existe una dependencia entre al menos una pareja de elementos de los dos paquetes. El compendio tiene menos detalle que la información original. Puede haber una pareja de dependencias individuales, o muchas parejas, que están representadas por dependencias del nivel de paquetes. En todo caso, el creador del modelo sabe que un cambio efectuado en un paquete *puede* afectar al otro. Si se necesitan más detalles, siempre se puede examinar detalladamente el contenido una vez que se ha observado el compendio de alto nivel.

Análogamente, una dependencia de uso entre dos clases suele indicar una dependencia entre sus operaciones, tal como el caso de un método de una clase que llama a una operación (¡No a un método!) de otra clase. Muchas dependencias del nivel de clases se derivan de dependencias entre operaciones o atributos.

En general, las relaciones que se resumen en un contenedor indican la existencia de al menos una aparición de la relación entre contenidos. Normalmente, no indican que todos los elementos contenidos participen en la relación.

componente

Una parte física reemplazable de un sistema que empaqueta su implementación, y es conforme a un conjunto de interfaces a las que proporciona su realización.

Semántica

Un componente representa una pieza física de la implementación de un sistema, incluyendo el código del software (fuente, binario, o ejecutables) o equivalentes, tales como guiones o archivos de comandos. Algunos componentes tienen identidad y pueden poseer entidades físicas, que incluyen objetos en tiempo de ejecución, documentos, bases de datos, etcétera. Los componentes existen en el dominio de la implementación son unidades físicas en los computadores que se pueden conectar con otros componentes, sustituir por componentes equivalentes, trasladar, archivar, etcétera. Los modelos pueden representar dependencias entre componentes, tales como dependencias del compilador y de tiempo de ejecución o dependencias de la información en una organización humana. Una instancia de un componente se puede utilizar para representar las unidades de implementación que existen en tiempo de ejecución, incluyendo su localización en instancias de un nodo.

Los componentes tienen dos características. Empaquetan el código que implementa la funcionalidad de un sistema, y algunas de sus propias instancias de objetos que constituyen el estado del sistema. Los llamamos últimos componentes de la identidad, porque sus instancias poseen identidad y estado.

Código. Un componente contiene el código para las clases de implementación y otros elementos. (La palabra código se interpreta de forma amplia e incluye los guiones, las estructuras de hipertexto, y otras formas de descripciones ejecutables.) Un componente de código fuente es un paquete para el código fuente de las clases de implementación. Algunos lenguajes de programación (tales como C++) distinguen archivos de declaración de los archivos de método, pero todos son componentes. Un componente de código binario es un paquete para el código compilado. Una biblioteca del código binario es un componente.

Un componente ejecutable contiene código ejecutable. Cada tipo de componente contiene el código para las clases de implementación que realizan algunas clases e interfaces lógicas. La relación de realización asocia un componente con las clases y las interfaces lógicas que implementan sus clases de implementación. Las interfaces de un componente describen la funcionalidad que aporta. Cada operación en la interfaz debe hacer referencia eventualmente a un elemento de la implementación disponible en el componente.

La estructura estática, ejecutable de una implementación de un sistema se puede representar como un conjunto interconectado de componentes. Las dependencias entre componentes significan que los elementos de la implementación en un componente requieren los servicios de los elementos de la implementación en otros componentes. Tal uso requiere que los elementos del proveedor sean públicamente visibles en sus componentes. Un componente puede también

tener elementos privados, pero éstos no pueden ser proveedores directos de servicios para otros componentes.

Los componentes pueden ser contenidos por otros componentes. Un componente contenido es sólo otro elemento de la implementación dentro de su contenedor.

Una instancia de componente es una instancia de un componente en una instancia del nodo. Las instancias de componentes de código fuente y de código binario pueden residir en instancias particulares de un nodo, pero las instancias de componentes ejecutables son las más útiles. Si una instancia de componente ejecutable está situada en una instancia de un nodo, los objetos de las clases de implementación presentes en el componente pueden ejecutar operaciones cuando están situados en la instancia del nodo. Si no, un objeto situado en una instancia de un nodo no puede ejecutar operaciones y se debe mover o copiar a otra instancia del nodo para ejecutar dichas operaciones.

Si un componente carece de identidad, todas sus instancias son iguales. No importa cuál de ellas esté ejecutando un objeto. Todos se comportan igual. Como no tienen ninguna identidad, las instancias de componentes no tienen valores o estado por sí mismas.

Identidad. Un componente de identidad tiene identidad y estado. Posee los objetos físicos, que están situados en él (y, por lo tanto, en la instancia del nodo que contiene la instancia del componente). Puede tener atributos, relaciones de composición con los objetos poseídos, y asociaciones a otros componentes. Desde este punto de vista, es una clase. Sin embargo, la totalidad de su estado debe hacer referencia a las instancias que contiene. Eso es lo que le hace un componente, en lugar de una clase ordinaria. Una clase de implementación se utiliza a menudo para representar el componente en su totalidad. Esto se llama clase dominante y se compara a menudo con el componente mismo, aunque no son la misma cosa.

Un objeto que solicita servicios de un componente de identidad debe seleccionar una instancia específica del componente, generalmente teniendo una asociación a uno de los objetos poseídos por el componente. Debido a que cada instancia de componente de identidad tiene estado, las distintas peticiones a instancias pueden producir resultados diferentes.

Ejemplo

Por ejemplo, un corrector ortográfico puede ser un componente. Si tiene un diccionario fijo, puede ser modelado como un componente sin identidad. Todas las instancias de él producen los mismos resultados y no hay memoria de peticiones anteriores. Si el diccionario puede ser actualizado, entonces debe ser modelado como un componente de identidad.

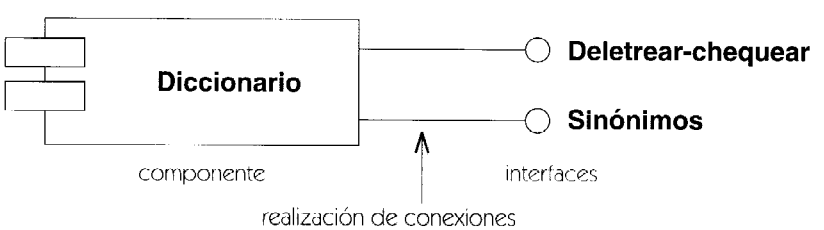


Figura 13.49 Componente

Puede haber diversas versiones del diccionario, que corresponden a diversas instancias del componente del corrector ortográfico. Un solicitante debe dirigir su petición a una instancia específica del componente. El componente destino de la petición es a menudo implícito dentro de un contexto particular, pero esto es una opción que debe ser parte del diseño.

Estructura

Un componente *ofrece* un conjunto de elementos de implementación, tales como clases de implementación. Esto significa que el componente proporciona el código para los elementos. Varios componentes pueden ofrecer un elemento de implementación dado.

Un componente puede tener operaciones e interfaces, que deben ser implementados por sus elementos de implementación.

Un componente de identidad es un *contenedor* físico para las entidades físicas, tales como objetos de tiempo de ejecución y bases de datos. Para proporcionar manejadores para sus elementos contenidos, puede tener los atributos y asociaciones salientes, que deben ser implementados por sus elementos de implementación. Un componente de identidad puede señalar una clase dominante que ofrezca todos sus atributos y operaciones públicas, pero tal clase es uno más de sus elementos de implementación.

Notación

Un componente se representa como un rectángulo con dos rectángulos más pequeños que sobresalen de un lado. El nombre del tipo del componente se pone dentro (Figura 13.49). Una instancia de componente tiene un nombre individual separado de un nombre del tipo de componente por dos puntos. La cadena del nombre se subraya para distinguirla de un tipo de componente (Figura 13.50). Un símbolo de instancia de componente se puede dibujar dentro del símbolo de un nodo para indicar que la instancia del componente está localizada en la instancia de un nodo (Figura 13.51).

Si el componente no tiene identidad, el nombre de la instancia se omite generalmente. Los objetos poseídos por una instancia de componente de identidad se pueden dibujar dentro de él. Un componente que contiene atributos u objetos es automáticamente un componente de identidad.

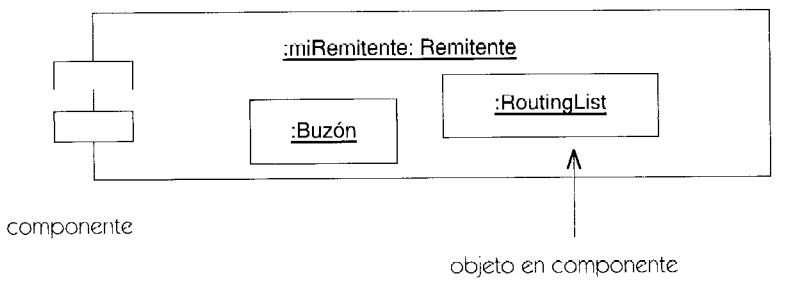


Figura 13.50 Instancias de componentes de identidad con objetos residentes

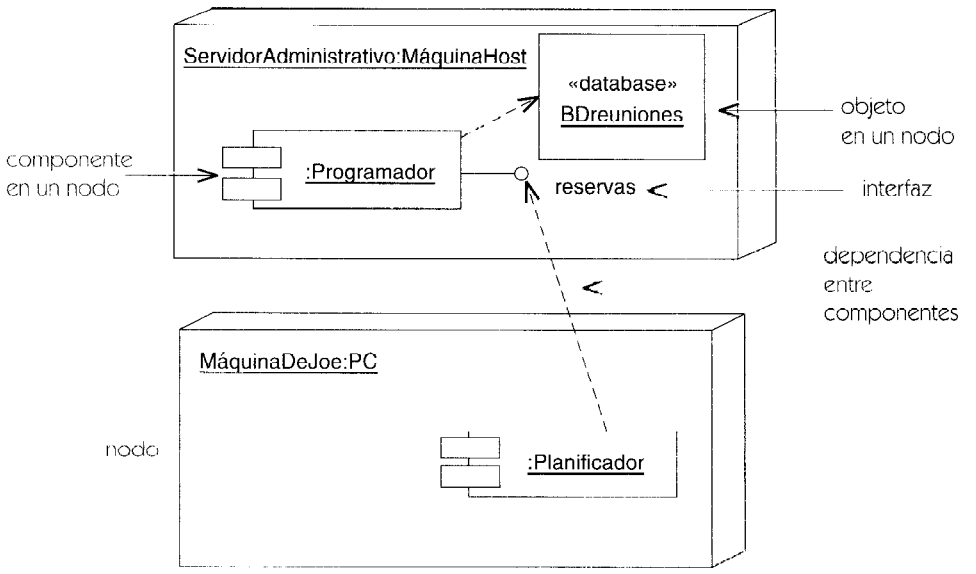


Figura 13.51 Instancias de componentes en nodos

Una clase dominante incluye la interfaz del componente. Puede ser dibujada como clase con un símbolo de componente en la esquina superior derecha como un icono de estereotipo.

En este caso, el componente y su clase dominante comparten la misma interfaz, y cualquier objeto contenido en el componente es accesible desde la clase dominante por los enlaces de composición. La Figura 13.52 presenta un ejemplo.

Las operaciones y las interfaces disponibles para los objetos exteriores se pueden representar directamente en el símbolo de clase. Éstos son su comportamiento como clase. Los contenidos del subsistema se representan en un diagrama separado. Cuando la característica de un subsistema de ser instanciable no necesita ser representada, se puede utilizar la notación ordinaria del paquete.

Las dependencias de un componente con otros componentes o elementos del modelo se representan usando líneas discontinuas con las puntas de flecha en los elementos del proveedor (Figura 13.53). Si un componente es la realización de una interfaz, se puede utilizar la notación taquigráfica de un círculo unido al símbolo del componente por un segmento de línea. Realizar una interfaz implica, por lo menos, que los elementos de la implementación en el componente proporcionen todas las operaciones de la interfaz. Si un componente utiliza una interfaz de otro

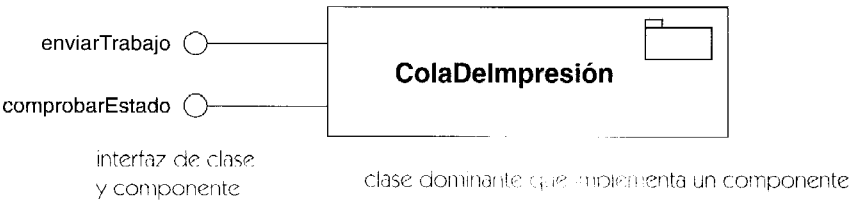


Figura 13.52 Clase dominante para un componente

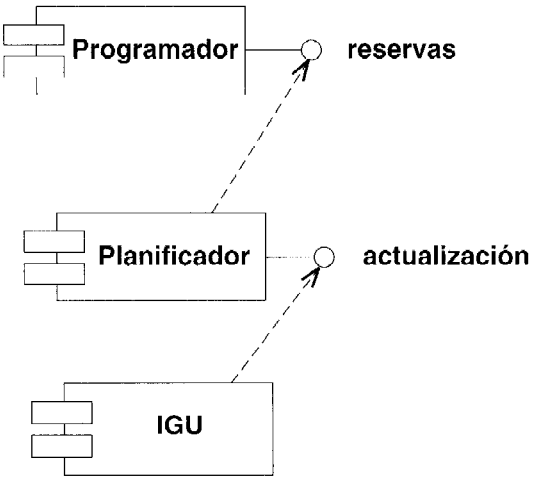


Figura 13.53 Dependencias entre componentes

elemento, la dependencia se puede representar por una línea discontinua con una punta de flecha en el símbolo de la interfaz. Usar una interfaz implica que los elementos de la implementación en el componente no requieren más operaciones de un componente proveedor que las que están enumeradas en el interfaz (pero también el cliente puede depender de otras interfaces).

Discusión

La siguiente definición ampliada explica la intención subyacente en un componente y las consideraciones implicadas en decidir si una parte de un sistema debe considerarse un componente.

- Un componente no es trivial. Es funcional y conceptualmente más grande que una sola clase o una sola línea del código. Típicamente, un componente abarca la estructura y comportamiento de una colaboración de clases.
- Un componente es casi independiente de otros componentes. Aunque raramente se encuentra solo. Más bien, un componente dado colabora con otros componentes y en ese empeño, asume un contexto arquitectónico.
- Un componente es una parte reemplazable de un sistema. Es sustituible, permitiendo sustituirlo por otro que se amolde a las mismas interfaces. El mecanismo de insertar o de sustituir un componente para formar un sistema ejecutable es típicamente transparente al usuario del componente, y es habilitado por los modelos de objetos que requieren pocas o ninguna transformación o por las herramientas que automatizan el mecanismo.
- Un componente satisface una función clara y es lógica y físicamente cohesivo. Denota así un pedazo significativo bien estructuralmente o bien por el comportamiento, para un sistema más grande.
- Un componente existe en el contexto de una arquitectura bien definida. Representa un bloque constructivo fundamental en el que puedan ser diseñados y compuestos los sistemas.

Esta definición es recurrente. Un sistema en un nivel de abstracción puede simplemente ser un componente en un nivel más alto de la abstracción.

- Un componente nunca se encuentra solo. Todo componente presupone una arquitectura y/o el contexto tecnológico en el cual se piensa utilizar.
- Un componente está formado por un conjunto de interfaces. Un componente que se amolda a una interfaz satisface el contrato especificado por la interfaz y se puede sustituir en cualquier contexto en el que se aplique esa interfaz.

Elementos estándar

document, executable, file, library, location, table.

comportamiento

Efecto observable de una operación o evento que incluye sus resultados.

composición

Una forma de asociación de agregación con fuerte sentido de posesión y tiempo de vida coincidente de las partes con el conjunto. Una pieza puede pertenecer a solamente una composición. Las partes con multiplicidad no fija, se pueden crear después del elemento compuesto. Pero una vez creadas, viven y mueren con él (es decir, comparten tiempo de vida). Tales piezas se pueden también quitar explícitamente antes de la muerte del elemento compuesto. La composición puede ser recurrente.

Véase también agregación, asociación, objeto compuesto.

Semántica

Hay una forma de asociación de agregación muy fuerte llamada composición. Una composición es una asociación de agregación con las restricciones adicionales de que un objeto sólo puede ser parte de un compuesto a la vez y que el objeto compuesto es el único responsable de la disponibilidad de todas sus partes. Como consecuencia de la primera restricción, el conjunto de todas las relaciones de composición (incluidas *todas* las asociaciones con la propiedad de la composición) forma un bosque de árboles hechos de objetos y de enlaces de composición. Una parte compuesta no se puede compartir por dos objetos compuestos. Esto concuerda con la intuición normal de la composición física por partes —una no puede ser parte directa de dos objetos (aunque puede indirectamente ser parte de objetos múltiples en diversos niveles de granularidad en el árbol)—.

Por tener la responsabilidad de la disponibilidad de sus partes, entendemos que el elemento compuesto es responsable de su creación y destrucción. En términos de implementación, es responsable de su asignación de memoria. Durante su instanciación, un elemento compuesto debe asegurarse de que hayan sido instanciadas y unidas correctamente a él todas sus partes. Puede crear una pieza por sí mismo o puede asumir la responsabilidad de una pieza existente.

Pero durante la vida del elemento compuesto, ningún otro objeto puede tener esta responsabilidad sobre ellos. Esto significa que el comportamiento para una clase compuesta se puede diseñar con el conocimiento de que ninguna otra clase destruirá o desasignará sus piezas. Un elemento compuesto puede agregar piezas adicionales durante su vida (si lo permite la multiplicidad), siempre que asuma una responsabilidad única sobre ellas. Puede quitar piezas, siempre que la multiplicidad lo permita y la responsabilidad de ellas sea asumida por otro objeto. Si se destruye el elemento compuesto, debe o destruir todas sus piezas o bien dar la responsabilidad de ellas a otros objetos.

Esta definición abarca la mayoría de las intuiciones lógicas e implementaciones comunes de la composición. Por ejemplo, un registro que contiene una lista de valores es una implementación común de un objeto y de sus atributos. Cuando se asigna el registro, la memoria para los atributos se asigna automáticamente también, pero los valores de los atributos pueden necesitar ser inicializados. Mientras existe el registro, ningún atributo se puede eliminar tampoco. Cuando se desasigna el registro, la memoria de los atributos se desasigna también. Ningún otro objeto puede afectar a la asignación de algún atributo dentro del registro. Las características físicas de un registro hacen cumplir los requisitos de un elemento compuesto.

Esta definición de la composición funciona bien con la recolección de basura. Si se destruye el elemento compuesto en sí mismo, el único indicador a la pieza se destruye y la pieza se convierte en inaccesible y está sujeta a la recolección de basura. Sin embargo, la recuperación de piezas inaccesibles es simple incluso con la recolección de basura, lo cual es una razón para distinguir la composición de otra agregación.

Observe que una pieza no necesita ser implementada como una parte física de bloque de memoria con el elemento compuesto. Si la pieza está aparte del elemento compuesto, entonces el compuesto tiene la responsabilidad de asignar y de desasignar la memoria para la pieza, según lo necesita. En C++, por ejemplo, los constructores y destructores facilitan la implementación de elementos compuestos.

Un objeto puede ser parte solamente de un objeto compuesto a la vez. Esto no imposibilita a una clase para ser una parte compuesta de más de una clase en diferentes momentos o en diferentes instancias, pero solamente puede existir un enlace de composición al mismo tiempo para un objeto. Es decir hay una restricción OR entre los posibles compuestos a los que una pieza pudo pertenecer. Un objeto también puede ser parte de diversos objetos compuestos durante su vida, pero solamente uno a la vez.

Estructura

La propiedad de agregación en un extremo de la asociación puede tener los siguientes valores.

ninguno	El clasificador unido no es un agregado o un compuesto.
agregado	El clasificador unido es un agregado. El otro extremo es una pieza.
compuesto	El clasificador unido es un compuesto. El otro extremo es una pieza.

Un extremo de una asociación debe tener por lo menos el valor **ninguno**.

Notación

La composición se representa por un adorno que consiste en un diamante sólido en el extremo de la asociación unida al elemento compuesto (Figura 13.54). La multiplicidad se puede mostrar de manera normal. Debe ser 1 ó 0..1.

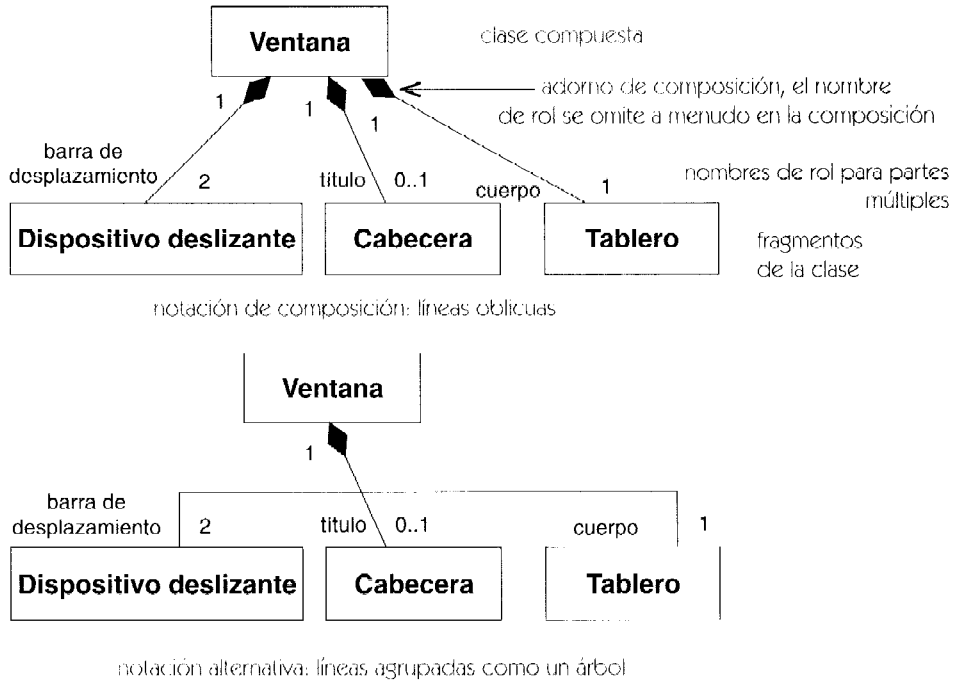


Figura 13.54 Notación de la composición

Alternativamente, la composición puede ser representada gráficamente anidando los símbolos de las partes dentro del símbolo del elemento compuesto (Figura 13.55). Un clasificador anidado puede tener multiplicidad dentro de su elemento compuesto. La multiplicidad se

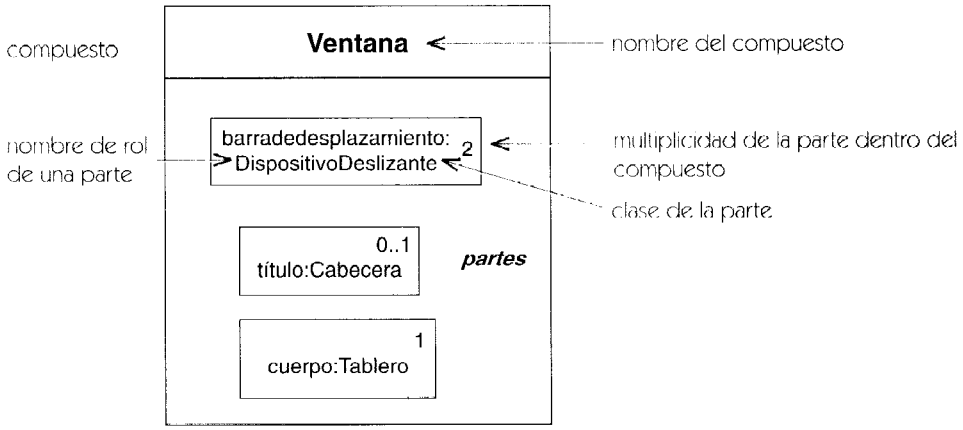


Figura 13.55 Composición como anidamiento gráfico

representa por una cadena de la multiplicidad en la esquina superior derecha del símbolo para la pieza. Si se omite la marca de la multiplicidad, la multiplicidad por defecto son muchas. Un elemento anidado puede tener un nombre de rol dentro de la composición. El nombre se muestra delante de su tipo con la sintaxis

```
nombre-de-rol : nombre-de-clase
```

El nombre de rol es el nombre de rol en una asociación implícita de composición del compuesto a la pieza.

Una asociación dibujada enteramente dentro de un borde del compuesto se considera que es parte de la composición. Cualquier objeto conectado por un solo enlace de la asociación debe pertenecer al mismo compuesto. Una asociación dibujada de modo que su línea rompa la frontera del elemento compuesto no se considera parte de la composición. Cualquier objeto en un solo enlace de la asociación puede pertenecer al mismo o a los diversos compuestos (Figura 13.56).

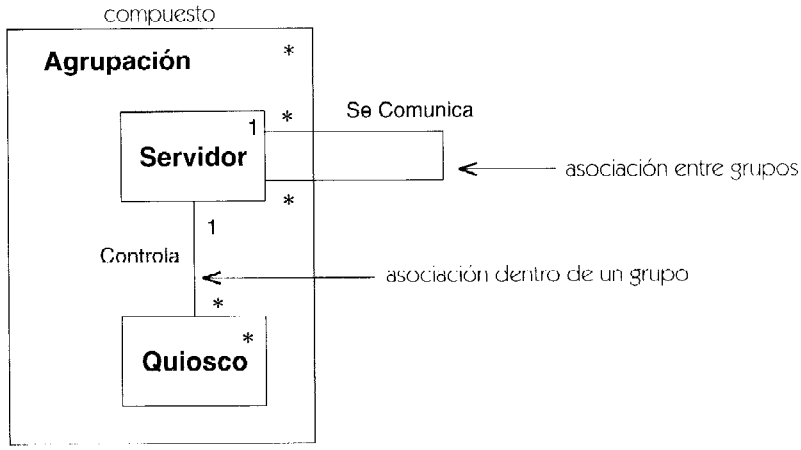


Figura 13.56 Asociación en y entre componentes

Observe que los atributos son, en efecto, relaciones de la composición entre una clase y las clases de sus atributos (Figura 13.57). En general, sin embargo, los atributos deben ser reservados para los valores primitivos de los datos (tales como números, cadenas, y fechas) y no las referencias a las clases, porque ninguna otra relación de las clases que son parte pueden considerarse en la notación de los atributos.

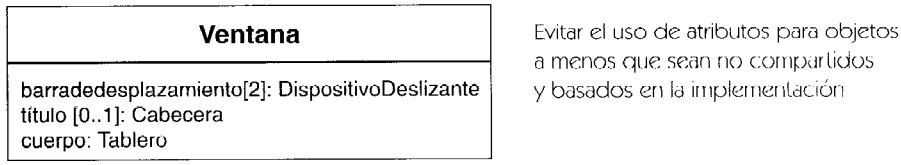
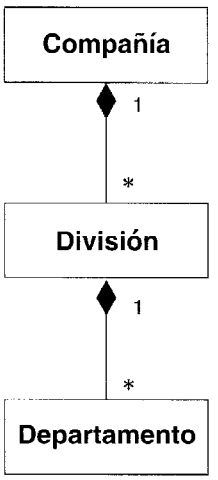


Figura 13.57 Los atributos son una forma de composición

Observe que la notación para la composición se asemeja a la notación para la colaboración. Una composición se puede pensar como una colaboración en la cual todos los participantes sean parte de un solo objeto compuesto.

La Figura 13.58 muestra composición a múltiples niveles.



La composición es transitiva:

Departamento es una parte compuesta indirecta de Compañía

Figura 13.58 Composición multinivel

Discusión

(*Véase también* la discusión en la agregación para las pautas sobre cuándo son apropiadas la agregación, la composición, y la asociación simple.)

La composición y la agregación son metarrelaciones —superan asociaciones individuales para imponer restricciones en el sistema de asociaciones completo—. La composición es significativa a través de relaciones de composición. Un objeto puede tener en la mayoría un enlace de composición (a un elemento compuesto) aunque puede ser que potencialmente venga de más de una asociación de composición. El gráfico completo de la composición y los enlaces y los objetos de la agregación debe ser acíclico, incluso si los enlaces vienen de diversas asociaciones. Observe que estas restricciones se aplican a la instancia del dominio —las asociaciones de agregación forman ciclos a menudo por ellas mismas, y las estructuras recurrentes requieren siempre ciclos de asociaciones.

Considere el modelo en la Figura 13.59. Cada **Autentificación** es una parte compuesta de exactamente una **Transacción**, que puede ser una **Compra** o una **Venta**. Sin embargo no toda **Transacción** necesita tener una **Autentificación**. De este fragmento, tenemos bastante información para concluir que una **Autentificación** no tiene ninguna otra asociación de la composición. Cada objeto de autentificación debe ser parte de un objeto transacción (la multiplicidad es uno); un objeto puede ser parte en de un compuesto como máximo (por la definición); es ya parte de un compuesto (según el dibujo); por lo tanto una **Autentificación** no debería ser parte de ninguna otra asociación de composición. No hay peligro que una **Autentificación** pudiera tener que manejar su propio almacenaje. Una transacción siempre está disponible para tomar esa responsabilidad, aun-

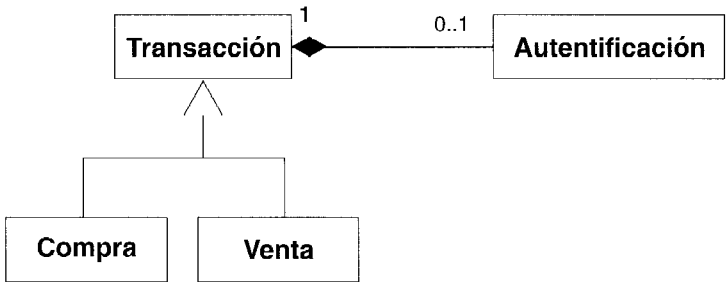


Figura 13.59 Composición a una clase abstracta compuesta

que no todas las **Transacciones** tienen las **Autentificaciones** que necesita manejar. (Por supuesto, la **Autentificación** puede manejarse por sí misma si el diseñador lo desea.)

Ahora considere la Figura 13.60. Un **Autógrafo** puede opcionalmente ser parte de una **Transacción** o de una **Carta**. No puede ser parte de ambos al mismo tiempo (por las reglas de la composición). Este modelo no evita que un Autógrafo comience como parte de una carta y se convierta en parte de una **Transacción** (en cuyo caso, debe dejar de ser parte de la **Carta**). De hecho, un **Autógrafo** no necesita ser parte de ninguna cosa. También, de este fragmento modelo, no podemos impedir la posibilidad que el **Autógrafo** es opcionalmente parte de una cierta clase que no se muestra en el diagrama o que se pudo agregar más adelante.

Figura 13.60 Partes compartidas de clases

¿Qué ocurre si es necesario indicar que cada **Autógrafo** debe ser parte de una **Carta** o una **Transacción**? Entonces el modelo se debe replantear, como en la Figura 13.59.

Se puede agregar una nueva superclase abstracta que incluye a **Carta** y a **Transacción** (llamada **Documento**), y la asociación de composición con **Autógrafo** se puede desplazar a ella desde las clases originales. Al mismo tiempo, la multiplicidad del **Autógrafo** al **Documento** se hace uno.

Hay un problema de menor importancia con este acercamiento: La multiplicidad del **documento** al **Autógrafo** se debe hacer opcional, lo que debilita la inclusión obligatoria original del **Autógrafo** dentro de la **Transacción**. La situación se puede modelar usando la generalización de la asociación de composición por sí misma, como en la Figura 13.61. La asociación de composición entre el **Autógrafo** y la **Transacción** se modela como hijo de la asociación de composición entre el **Autógrafo** y el **Documento**. Pero sus multiplicidades se clarifican para el hijo (nótese que siguen siendo coherentes con las heredadas, así que el hijo es sustituible por el padre). Alternativamente, el modelo original puede ser utilizado agregando una restricción entre las dos composiciones, de las cuales siempre debe mantenerse una.

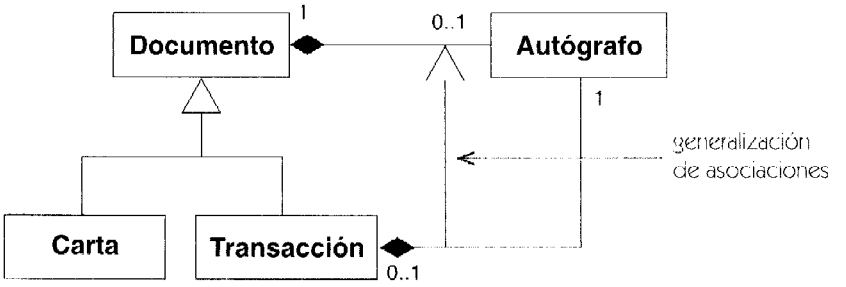


Figura 13.61 Generalización de asociación de composición

concreto

Un elemento generalizable (tal como una clase) que puede ser instanciado directamente. Por lo tanto, su implementación debe ser especificarse completamente. Para una clase, todas sus operaciones deben ser implementadas (por la clase o un antecesor). Antónimo: abstracto.

Véase también clase directa, instanciación.

Semántica

Sólo los clasificadores concretos pueden ser instanciados. Por lo tanto todas las hojas de una jerarquía de generalización deben ser concretas. Es decir, todas las operaciones abstractas y otras propiedades abstractas se deben implementar eventualmente en algún descendiente. (Por supuesto, una clase abstracta puede no tener ningún descendiente concreto, si el programa está incompleto, por ejemplo un marco previsto para la extensión del usuario, pero tal clase no puede utilizarse en una implementación hasta que sus descendientes no sean concretos.)

Notación

El nombre de un elemento concreto aparece en letra normal. El nombre de un elemento abstracto aparece en letra cursiva.

concurrencia

El funcionamiento de dos o más actividades durante el mismo intervalo del tiempo. No hay implicación sobre que las actividades estén sincronizadas. En general, funcionan independientemente a excepción de puntos explícitos de la sincronización. La concurrencia puede lograrse intercalando o ejecutando simultáneamente dos o más hilos.

Véase transición compleja, estado compuesto, hilo.

concurrencia dinámica

Un estado de actividad que representa múltiples ejecuciones concurrentes cuando se activa.

Véase grafo de actividades.

condición de guarda

Es una condición que debe satisfacerse para permitir que se dispare la transición asociada.

Véase también bifurcación, hilo condicional, estado de unión o conjunción, transición.

Semántica

Una condición de guarda es una expresión booleana que forma parte de la especificación de una transición. Cuando se recibe el evento disparador de una transición, ese evento se guarda hasta que la máquina de estados ha finalizado cualquier posible paso que deba ejecutarse hasta finalizar. Entonces se procesa el evento y se evalúa la condición de guarda. Si la condición es verdadera, se permite que se dispare la transición (pero si se activa más de una transición, sólo se dispara una de ellas). La comprobación se produce en el momento en que se procesa el evento disparador. Si la condición de guarda toma el valor falso al ser evaluada cuando se procesa el evento, esa condición no se vuelve a evaluar a no ser que vuelva a producirse el evento disparador, aun cuando la condición pudiera pasar a ser verdadera posteriormente.

Una condición de guarda tiene que ser una consulta, esto es, no puede modificar el valor del sistema o de su estado, ni puede poseer efectos secundarios.

Una condición de guarda puede aparecer en una transición de finalización. En tal caso, seleccionará una de las ramas de la bifurcación.

Notación

La condición de guarda forma parte de la cadena correspondiente a una transición. Adopta el aspecto de una expresión booleana entre corchetes.

[expresión booleana]

Los nombres que se utilicen dentro de la expresión deben estar a disposición de la transición. Serán o bien parámetros del evento disparador o bien atributos del objeto poseedor.

configuración del estado activo

Conjunto de estados activos existente en un momento dado en una máquina de estados. Cuando se dispara una transición se producen cambios en unos cuantos estados del conjunto, pero los demás permanecen inalterados.

Véase activo/a, máquina de estados, transición compleja.

conflicto

Situación en que atributos u operaciones con el mismo nombre se heredan de más de una clase, o cuando el mismo evento permite más de una transición, o cualquier situación similar en la cual las reglas normales produzcan resultados potencialmente contradictorios. Dependiendo de la semántica para cada tipo de elemento del modelo, un conflicto se puede resolver por una regla de resolución del conflicto, puede ser legal producir un resultado no determinista, o se puede indicar que el modelo está mal formado.

Discusión

Es posible evitar conflictos definiéndolos sin tener en cuenta las reglas de la resolución de conflictos, por ejemplo: Si el mismo atributo está definido es más de una superclase, utilice la definición que aparece en superclases anteriores (esto requiere que las superclases estén ordenadas). UML no especifica generalmente las reglas para resolver conflictos, en principio es peligroso contar con tales reglas. Son fáciles de pasar por alto y con frecuencia son el síntoma de problemas más profundos en un modelo.

Es mejor forzar al modelador a ser explícito en lugar de depender de reglas sutiles y posiblemente confusas. En una herramienta o un lenguaje de programación, tales reglas tienen su lugar, únicamente para permitir obtener un significado determinista. Pero sería provechoso que las herramientas proporcionen advertencias cuando se utilicen las reglas de modo que el modelador esté enterado del conflicto.

construcción

Tercera fase de un proceso del desarrollo del software, durante la cual se hace el diseño detallado y el sistema se implementa y se prueban los elementos de software, hardware o circuitería. Durante esta fase, la vista del análisis y la vista del diseño se terminan substancialmente, junto con la mayor parte de la vista de implementación y de parte de la vista del despliegue.

Véase proceso de desarrollo.

constructor

Una operación de ámbito de clase que crea e inicializa una instancia de una clase. Puede ser utilizado como estereotipo de operación.

Véase creación, instanciación.

consulta

Dícese de una operación que proporciona un valor pero no altera el estado del sistema; denota una operación sin efectos secundarios.

contenedor

Un objeto que existe para contener a otros objetos, y que proporciona operaciones para acceder o iterar sobre su contenido, o una clase que describe tales objetos.

Por ejemplo, arrays, listas, y conjuntos.

Véase también agregación, composición.

Discusión

Generalmente es innecesario modelar explícitamente los contenedores. Son la implementación más común para el extremo "muchos" de una asociación. En la mayoría de los modelos, una multiplicidad mayor de uno es suficiente para indicar la semántica correcta. Cuando un modelo de diseño se utiliza para generar código, la clase del contenedor usado para implementar la asociación puede especificarse a un generador de código usando valores etiquetados.

contexto

Una vista de un conjunto de elementos de modelado que están relacionados para un propósito, tal como ejecutar una operación o formar un patrón. Un contexto es un pedazo de un modelo que restringe o proporciona el ambiente para sus elementos. Una colaboración proporciona un contexto para su contenido.

Véase colaboración.

copia

Un tipo de relación de flujo usada en una interacción, en la cual el objeto destino representa una copia del objeto fuente; después de esto ambos son independientes.

Véase también become.

Semántica

La relación de copia es un tipo de relación de flujo que indica la derivación de un objeto desde otro objeto dentro de una interacción. Representa la acción de hacer una copia. Después de que se ejecute un flujo de copia, hay dos objetos independientes cuyos valores pueden evolucionar independientemente.

Una transición de copia dentro de una interacción puede tener un número de secuencia para indicar cuándo ocurre en relación con otras acciones.

Notación

Un flujo de copia se representa por una flecha con línea discontinua desde el objeto original a la nueva copia (en la punta de flecha). La flecha lleva la palabra clave de estereotipo «**copy**» y

puede tener un número de secuencia. Las transiciones de copia pueden aparecer en diagramas de colaboración, de secuencia, y de actividad.

Ejemplo

La Figura 13.62 muestra un archivo que tiene una copia de seguridad en otro nodo. Primero se hace una copia («**copy**»), después la copia se mueve al nodo secundario ("**become**" con un cambio de localización).

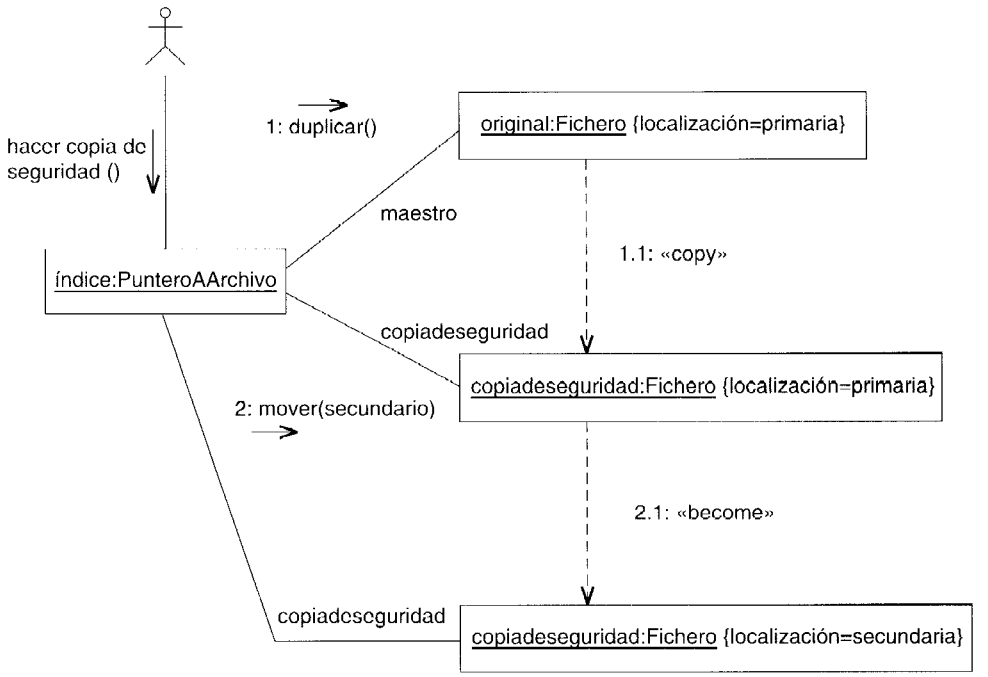


Figura 13.62 Flujo copy y become

creación

La instanciación y la inicialización de un objeto o de otra instancia (como una instancia de un caso de uso). Antónimo: destrucción.

Véase también instanciación.

Semántica

La creación de un objeto es el resultado de un mensaje que instancia el objeto. Una operación de creación puede tener parámetros que se utilicen para la inicialización de la nueva instancia.

Al finalizar la operación de creación, el nuevo objeto obedece a las restricciones de su clase y puede recibir mensajes.

Una operación de creación, o constructor, se puede declarar como operación del ámbito de la clase. El destino de la operación es (al menos conceptual) la propia clase. En un lenguaje de programación como Smalltalk, una clase se implementa como objeto de ejecución real y la creación por lo tanto se realiza como mensaje normal a tal objeto. En un lenguaje como C++, no hay un objeto de ejecución real. La operación se puede pensar como mensaje conceptual que se ha optimizado dejándolo al margen de la ejecución. El enfoque de C++ imposibilita la oportunidad de computar la clase que va a ser instanciada. Si no, cada acercamiento se puede modelar como mensaje enviado a una clase.

Las expresiones del valor inicial para los atributos de una clase son (conceptualmente) evaluados en la creación, y los resultados se utilizan para inicializar los atributos. El código para una operación de creación puede sustituir explícitamente estos valores, así que las expresiones del valor inicial se deben tomar como valores por defecto que pueden ser modificados.

Dentro de una máquina de estados, los parámetros de la operación de construcción que creó el objeto están disponibles como evento actual implícito en la transición que sale del estado inicial del nivel superior.

Notación

En un diagrama de clases, una operación de creación (constructor) se incluye como una de las operaciones en la lista de operación de la clase. Puede tener una lista de parámetros, pero el valor de vuelta es implícitamente una instancia de la clase y puede omitirse. Como operación del ámbito de la clase, su cadena con el nombre debe estar subrayada (Figura 13.63). Puede tener también el estereotipo «**constructor**».

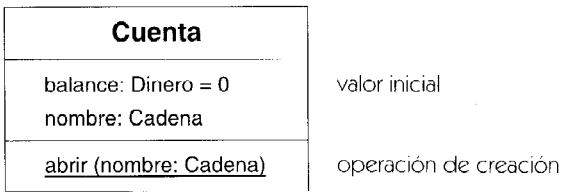


Figura 13.63 Operación de creación

Una ejecución de una operación de creación dentro de un diagrama de secuencia se representa dibujando una flecha de mensaje, con su punta de flecha en el símbolo del objeto (rectángulo con el nombre del objeto subrayado). Debajo del objeto, el símbolo es la cuerda de salvamento para el objeto (línea llena de línea discontinua o doble, dependiendo de si es activa), que continúa hasta la destrucción del objeto o el final del diagrama (Figura 13.64).

La ejecución de una operación de creación, dentro de un diagrama de colaboración, se representa incluyendo un símbolo de objeto con la propiedad {**new**}. El primer mensaje al objeto es implícitamente el mensaje que crea el objeto. Aunque el mensaje se dirige realmente a la propia clase, este flujo generalmente se omite y se muestra el mensaje inicializando (instanciado, pero sin inicializar) la instancia, tal y como lo muestra la Figura 13.65.

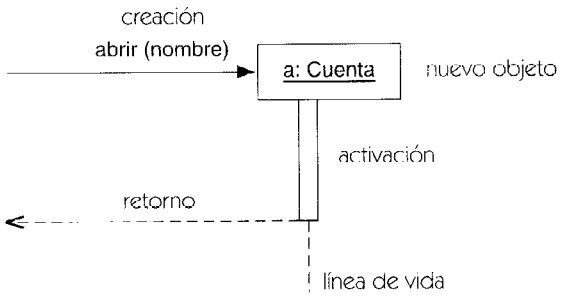


Figura 13.64 Diagramas de secuencia de la creación

Véase también diagrama de colaboración y de secuencia para entender la notación para representar la creación dentro de la implementación de un procedimiento.

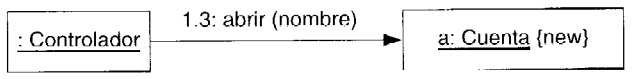


Figura 13.65 La creación en un diagrama de colaboración

delegación

La capacidad de un objeto de emitir un mensaje a otro objeto en respuesta a un mensaje. La delegación puede ser utilizada como alternativa a la herencia. En algunos lenguajes (tales como *Self*), está apoyado por mecanismos de herencia en el propio lenguaje. En la mayoría de los otros lenguajes, tales como C++ y Smalltalk, puede implementarse con una asociación o agregación a otro objeto.

Una operación en el primer objeto invoca una operación en el segundo objeto para lograr realizar su trabajo. Contrastar con: herencia.

Véase también asociación.

dependencia

Una relación entre dos elementos, en los cuales un cambio en un elemento (el proveedor) puede afectar o proveer la información necesaria para el otro elemento (el cliente). Éste es un término de conveniencia que agrupa varios tipos de modelados de relaciones.

Véase relación: Tabla 13-2 para un diagrama completo de las relaciones de UML.

Semántica

La dependencia es una declaración de relación entre dos elementos en un modelo o en modelos diferentes. El término, de forma un poco arbitraria, agrupa varios tipos de relaciones diferentes,

al igual que el termino biológico *invertebrado* agrupa a todos los animales excepto a los *vertebrados*.

En un caso en el cual la relación represente una asimetría del conocimiento, los elementos independientes se llaman proveedores y los elementos dependientes se llaman clientes.

Una dependencia puede tener un nombre para indicar su rol en el modelo. Generalmente, sin embargo, la presencia de la dependencia por sí misma es suficiente para que el significado quede totalmente claro, y el nombre sea redundante. Una dependencia puede tener un estereotipo para establecer la naturaleza precisa de la dependencia, y puede tener un texto descriptivo para explicarlo con más detalle, aunque de manera informal.

Una dependencia entre dos paquetes indica la presencia de, al menos, una dependencia del tipo indicado entre un elemento de cada paquete (excepto para acceso e importación, las cuales están relacionadas con los paquetes directamente). Por ejemplo, la dependencia de uso entre dos clases, se puede representar como una dependencia de uso entre los paquetes que las contienen. Una dependencia entre paquetes no significa que los elementos en el paquete tengan dicha dependencia, de hecho esa situación es poco común.

Véase paquete.

Una dependencia puede contener un conjunto de referencias a dependencias subordinadas. Por ejemplo, una dependencia entre dos paquetes puede hacer referencia a dependencias subyacentes entre clases.

Las dependencias no son necesariamente transitivas.

Observe que la asociación y la generalización encajan dentro de la definición general de dependencia, pero tienen su propia representación y notación en el modelo y no están consideradas generalmente como dependencias. Una relación de realización tiene su propia notación especial, pero se considera una dependencia.

La dependencia tiene diferentes variantes que representen diversos tipos de relaciones: abstracción, ligadura, combinación, permiso, y uso.

Abstracción. Una dependencia de abstracción representa un cambio en el nivel de abstracción de un concepto. Ambos elementos representan el mismo concepto de diversas maneras. Generalmente uno de los elementos es más abstracto, y el otro es más concreto, aunque las situaciones son posibles cuando ambos elementos son representaciones alternativas en el mismo nivel de abstracción. De lo menos específico a las relaciones más específicas, la abstracción incluye los estereotipos traza, refinamiento (la palabra clave **refine**), realización (que tiene su propia notación especial), y derivación (la palabra clave **derive**).

Ligadura. Una dependencia de ligadura relaciona un elemento sujeto a una plantilla con la propia plantilla. Los argumentos para los parámetros de la plantilla se añaden a la dependencia de ligadura como una lista.

Permiso. Una dependencia de permiso (representada siempre como un estereotipo específico) relaciona un paquete o una clase con un paquete o una clase a la cual se concede una cierta categoría de permiso para utilizar su contenido. Los estereotipos de la dependencia de permiso son **access**, **friend**, e **import**.

Uso. Una dependencia de uso (palabra clave «**use**») conecta un elemento del cliente con un elemento del proveedor, cuya modificación puede requerir cambios al elemento cliente. El uso representa a menudo una dependencia de implementación, en la cual un elemento hace uso de los servicios de otro elemento para implementar su comportamiento. Los estereotipos de uso incluyen llamada, instanciación (palabra clave **instantiate**), parámetro, y envío. Esto es una lista abierta. En varios lenguajes de programación pueden existir otros tipos de dependencia de uso.

Notación

Una dependencia se representa mediante una flecha con línea discontinua entre dos elementos del modelo. El elemento de modelo en la cola de la flecha (el cliente) depende del elemento modelo en la punta de flecha (el proveedor). La flecha se puede etiquetar con una palabra clave opcional, para indicar el tipo de la dependencia, y un nombre opcional (Figura 13.66).

Muchos otros tipos de relaciones usan una flecha de líneas discontinuas con una palabra clave, aunque no entran la definición de dependencia. Éstos incluyen el flujo (conversión y copia), la combinación (extensión e inclusión), y la anexión de una nota o de una restricción al elemento modelo que describe. Si uno de los elementos es una nota o una restricción, la flecha se suprime porque la nota o la restricción es siempre el origen de la flecha.

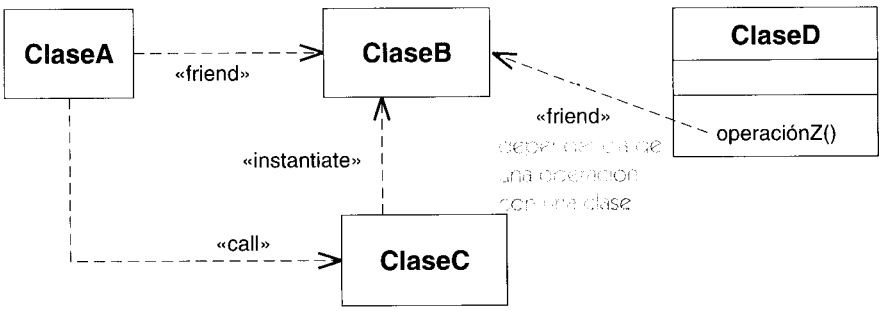


Figura 13.66 Algunas dependencias entre clases

Elementos estándar

become, bind, call, copy, create, derive, extend, friend, import, include, instanceOf, instantiate, powertipe, send, trace, use.

derivación

Una relación entre un elemento y otro elemento que se puede calcular a partir de aquél. La derivación se modela como un estereotipo de una dependencia de abstracción con la palabra clave **derive**.

Véase elemento derivado.

desarrollo incremental

Denota el desarrollo de un modelo y de otros artefactos de un sistema como una serie de versiones, cada una de las cuales está completa hasta cierto punto en lo tocante a precisión y funcionalidad, pero de tal modo que cada una va añadiendo más detalles a la versión anterior. La ventaja es que cada versión del modelo se puede evaluar y depurar tomando como base los cambios relativamente pequeños efectuados respecto a la versión anterior, lo cual hace más sencillo efectuar correctamente esos cambios. El término está íntimamente relacionado con el concepto de desarrollo iterativo.

Véase proceso de desarrollo.

desarrollo iterativo

Dícese del desarrollo de un sistema mediante un proceso fragmentado en una serie de pasos o iteraciones, cada uno de los cuales proporciona una aproximación mejor que la anterior del sistema deseado. El resultado de cada paso tiene que ser un sistema ejecutable que se pueda ejecutar, comprobar y depurar. El desarrollo iterativo está ligado fuertemente con el concepto de desarrollo incremental. En el desarrollo iterativo e incremental, cada iteración agrega una funcionalidad incremental a la iteración anterior. El orden en que se añade la funcionalidad se selecciona para equilibrar el tamaño de las iteraciones y para atacar desde un principio las fuentes potenciales de riesgo, antes de que se incremente demasiado el coste de resolver los errores.

Véase proceso de desarrollo.

descendiente

Un hijo o un elemento encontrado mediante una cadena de relaciones filiales; el fin transitivo de la relación de especialización. Antónimo: antecesor.

Véase generalización.

descriptor

Un elemento del modelo que describe las propiedades comunes de un conjunto de instancias, incluyendo su estructura, relaciones, comportamiento, restricciones, propósito, etcétera.

Contrastar con: instancia.

Semántica

La palabra *descriptor* caracteriza los elementos del modelo que describen conjuntos de instancias. La mayoría de los elementos en un modelo son descriptores. La palabra incluye todos los elementos del modelo —clases, asociaciones, estados, casos de uso, colaboraciones, etcétera—. A veces, la palabra *tipo* se utiliza con este significado, pero esa palabra se utiliza a menudo en un sentido más restringido para designar cosas como clases. La palabra *descriptor*

pretende incluir cualquier clase de elementos descriptivos. Un descriptor tiene un objetivo y una extensión. La descripción de la estructura y otras reglas generales son el objetivo. Cada descriptor caracteriza un sistema de casos, que son su extensión. No se asume que la extensión sea accesible físicamente en tiempo de ejecución. La dicotomía principal de un modelo es la distinción descriptor-instancia.

Notación

La relación entre un descriptor y sus instancias se refleja usando el mismo símbolo geométrico para ambos, pero subrayando la cadena del nombre de una instancia. Un descriptor tiene un nombre, mientras que una instancia tiene un nombre individual y un nombre de descriptor, separado por dos puntos, y se subraya la cadena del nombre.

descriptor completo

La descripción implícita completa de una instancia directa. El descriptor completo se ensambla implícitamente mediante herencia a partir de todos los antecesores.

Véase también clase directa, herencia, clasificación múltiple.

Semántica

Una declaración de una clase o de otro elemento modelo es, de hecho, solamente una descripción parcial de sus instancias; llámese el *segmento de la clase*. En general, un objeto contiene más estructura que la descrita por el segmento de la clase de su clase directa. El resto de la estructura es obtenido por herencia de las clases de los antecesores. La descripción completa de todos sus atributos, operaciones, y asociaciones se llama descriptor completo. El descriptor completo no es generalmente manifiesto en un modelo o un programa. El propósito de las reglas de la herencia es proporcionar una manera de construir automáticamente el descriptor completo de los segmentos. En principio, hay varias maneras de hacer esto, a menudo llamado *protocolo de metaobjetos*. UML define un conjunto de reglas para la herencia que cubren la mayoría de los lenguajes de programación más populares y son también útiles para modelado conceptual. Sepa, sin embargo, que existen otras posibilidades —por ejemplo el lenguaje CLOS.

despliegue[2]

Etapa del desarrollo que describe la configuración del sistema para su ejecución en un ambiente del mundo real. Para el despliegue, se deben tomar decisiones sobre los parámetros de la configuración, funcionamiento, asignación de recursos, distribución, y concurrencia. Los resultados de

[2] El término inglés "deployment" ha recibido diferentes traducciones dependiendo del contexto en que aparece en el conjunto de libros sobre UML. Se traduce por despliegue cuando se trata de diagramas de despliegue aunque a veces se ha utilizado el término "implantación" cuando se trata de la etapa final del desarrollo.

esta fase se capturan en archivos de configuración y también en la vista del despliegue. Ponga en contraste el análisis, el diseño, la implementación, y el despliegue (implantación).

Véase proceso de desarrollo, etapas de modelado.

destrucción

La eliminación de un objeto y la liberación de sus recursos. La destrucción de un objeto compuesto conduce a la destrucción de sus partes compuestas. La destrucción de un objeto no destruye automáticamente los objetos relacionados por la asociación ordinaria o incluso por la agregación, pero cualquier enlace que involucre al objeto se destruye con él.

Véase también composición, estado final, instanciación.

Notación

Véase diagrama de colaboración y de secuencia (Figura 13.70) donde se muestra la notación de destrucción dentro de la implementación de un procedimiento. En un diagrama de secuencia, la destrucción de un objeto es representada por una X grande en la línea de vida del objeto (Figura 13.67). Se coloca en el mensaje que hace que el objeto sea destruido o en el punto donde el objeto finaliza por sí mismo. Un mensaje que destruye un objeto se puede representar con el estereotipo «**destroy**». En un diagrama de colaboración, la destrucción de un objeto durante una interacción se representa por la restricción {**destroyed**} en el objeto. Si el objeto se crea y se destruye en la interacción, se debe usar en su lugar la restricción {**transient**}.

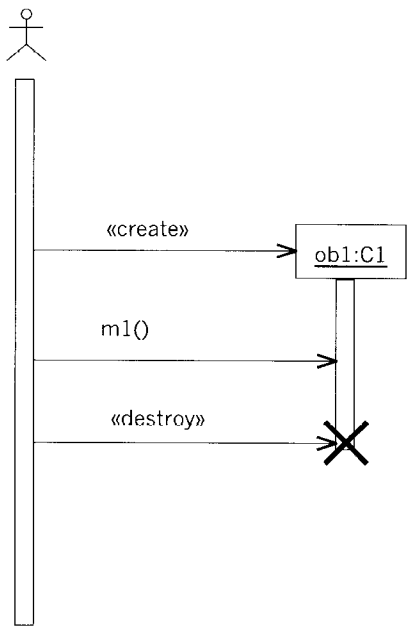


Figura 13.67 Creación y destrucción

destruir

Para eliminar un objeto y liberar sus recursos. Esto es generalmente una acción explícita, aunque puede ser la consecuencia de otra acción o restricción o de la recolección de basura.

Véase destrucción.

diagrama

Una representación gráfica de una colección de elementos del modelo, construida a menudo como un gráfico conexo de arcos (relaciones) y de vértices (otros elementos modelo). UML ofrece diagramas de clases, diagramas de objetos, diagramas de casos de uso, diagramas de secuencia, diagramas de colaboración, diagramas de estados, diagramas de actividad, diagramas de componentes, y diagramas de despliegue.

Véase también información de fondo, uso de la tipografía, hiperenlace, palabra clave, etiqueta, paquete, ruta, elemento de representación, lista de propiedades.

Semántica

Un diagrama no es un elemento semántico. Un diagrama muestra representaciones de elementos semánticos de modelo, pero su significado no se ve afectado por la forma en que son representados.

Un diagrama está contenido dentro de un paquete.

Notación

La mayoría de los diagramas de UML y algunos símbolos complejos son grafos que contienen formas conectadas por rutas. La información está sobre todo en la topología, no en el tamaño o la colocación de los símbolos (hay algunas excepciones, tales como un diagrama de secuencia con un eje métrico de tiempo). Hay tres clases importantes de relaciones visuales: conexión (generalmente de líneas a formas de 2 dimensiones), contención (de símbolos por formas cerradas de 2 dimensiones), y adhesión visual (un símbolo que está "cerca" de otro en un diagrama). Esas relaciones geométricas se reasignan a conexiones entre nodos en un gráfico en la forma analizada de la notación.

La notación de UML está pensada para ser dibujada en superficies bidimensionales. Algunas formas bidimensionales son proyecciones de formas tridimensionales, tales como cubos, pero todavía se representan como iconos en una superficie bidimensional. En un futuro cercano, la disposición y la navegación tridimensional real será posible en máquinas de sobremesa pero no es común en la actualidad.

Hay cuatro clases de construcciones gráficas que se usan en la notación de UML: iconos, símbolos bidimensionales, rutas, y cadenas.

Un icono es una figura gráfica con un tamaño y forma fijos. No se amplía para contener a su contenido. Los iconos pueden aparecer dentro de símbolos de área, como terminadores en las rutas, o como símbolos independientes que puedan o no conectar con las rutas. Por ejemplo, los símbolos para la agregación (un diamante), la dirección de navegación (una punta de flecha), el estado final (un ojo de buey), y la destrucción del objeto (una X grande) son iconos.

Los símbolos de dos dimensiones tienen altura y anchura variables, y pueden ampliarse para permitir otras cosas, tales como listas de cadenas o de otros símbolos. Muchos de ellos están divididos en compartimentos similares o de tipos diferentes. Las rutas se conectan con los símbolos bidimensionales terminando en el límite del símbolo. El arrastrar o suprimir un símbolo bidimensional afecta a su contenido y a cualquier ruta conectada con él. Por ejemplo, los símbolos para la clase (un rectángulo), el estado (un rectángulo redondeado), y la nota (un rectángulo espigado) son símbolos bidimensionales.

Una ruta es una secuencia de segmentos de recta o de curva que se unen en sus puntos finales. Conceptualmente, una ruta es una sola entidad topológica, aunque sus segmentos se pueden manipular gráficamente. Un segmento no debería existir separado de su ruta. Las rutas se unen siempre a otros símbolos gráficos en ambos extremos (no hay rutas colgantes). Las rutas pueden tener terminadores —es decir, iconos que aparecen en una secuencia en el extremo de la ruta y que califican el significado de la ruta—. Por ejemplo, los símbolos para la asociación (línea continua), la generalización (línea continua con un icono de triángulo), y la dependencia (línea discontinua) son rutas.

Las cadenas presentan varias clases de información en una forma "no analizada". UML asume que cada uso de una cadena en la notación tiene una sintaxis por la cual pueda ser analizada la información del modelo subyacente. Por ejemplo, hay sintaxis para los atributos, las operaciones, y las transiciones. Esta sintaxis es conforme con la extensión mediante herramientas como una opción de representación. Las cadenas pueden existir como el contenido de un compartimiento, como elementos en las listas (en cuyo caso la posición en la lista transporta información), como etiquetas unidas a los símbolos o a las rutas, o como elementos independientes en un diagrama. Por ejemplo, los nombres de clases, las etiquetas de transición, las indicaciones de multiplicidad y las restricciones, son cadenas.

diagrama de actividades

Diagrama que muestra un grafo de actividades.

Véase grafo de actividades.

diagrama de caso de uso

Diagrama que muestra las relaciones existentes entre actores y casos de uso dentro de un sistema.

Véase actor, caso de uso.

Notación

Un diagrama de caso de uso es un grafo de actores, un conjunto de casos de uso encerrados por los límites de un sistema (un rectángulo), asociaciones entre los actores y los casos de uso y generalización entre los actores. Los diagramas de casos de uso muestran elementos procedentes del modelo de casos de uso (casos de uso, actores).

diagrama de clases

Un diagrama de clases es una presentación gráfica de la vista estática, que muestra una colección de elementos declarativos (estáticos) del modelo, como clases, tipos, y sus contenidos y relaciones. Un diagrama de clases puede mostrar una vista de un paquete y contener símbolos de paquetes anidados. Un diagrama de clases contiene ciertos elementos materializados de comportamiento, como operaciones, pero cuya dinámica está representada en otros diagramas, como diagramas de estados o diagramas de colaboración.

Véase también clasificador, diagrama de objetos.

Notación

Un diagrama de clases muestra una presentación gráfica de la vista estática. Normalmente son necesarios varios diagramas de clases para mostrar una vista estática completa. Los diagramas de clases individuales no indican necesariamente divisiones del modelo subyacente, aunque las divisiones lógicas, como los paquetes, son fronteras naturales a la hora de formar diagramas.

diagrama de colaboración

Diagrama que muestra interacciones organizadas alrededor de los roles —esto es, ranuras para instancias y sus enlaces dentro de una colaboración—. A diferencia de los diagramas de secuencia, los diagramas de colaboración muestran explícitamente las relaciones entre roles. Por otra parte, un diagrama de colaboración no muestra el tiempo como una dimensión aparte, por lo que resulta necesario etiquetar con números de secuencia tanto la secuencia de mensajes como los hilos concurrentes. Los diagramas de secuencia y los diagramas de colaboración expresan información similar pero de forma diferente.

Véase colaboración, diagrama de secuencia, patrón.

diagrama de componentes

Es un diagrama que muestra las organizaciones y las dependencias entre tipos de componentes.

Semántica

Un diagrama de componentes representa las dependencias entre componentes software, incluyendo componentes de código fuente, componentes del código binario, y componentes ejecutables (Figura 13.53). Un módulo software se puede representar como componente. Algunos componentes existen en tiempo de compilación, algunos existen en tiempo de enlace, y algunos existen en tiempo de ejecución; otros componentes existen en varias de estas ocasiones. Un componente de sólo compilación es aquel que es significativo únicamente en

tiempo de compilación. Un componente de ejecución, en este caso, sería un programa ejecutable.

Un diagrama de componentes tiene solamente una versión con descriptores, no tiene versión con instancias. Para mostrar las instancias de los componentes, se debe usar un diagrama de despliegue.

Notación

El diagrama de componentes muestra clasificadores de componentes, las clases definidas en ellos, y las relaciones entre ellas. Los clasificadores de componentes también se pueden anidar dentro de otros clasificadores de componentes para mostrar relaciones de definición.

Una clase definida dentro de un componente se puede representar dentro de él, aunque para los sistemas de cualquier tamaño, es posible que resulte más conveniente proporcionar una lista de las clases definidas dentro de un componente, en vez de mostrar sus símbolos.

Un diagrama que contiene clasificadores de componentes y clasificadores de nodo se puede utilizar para mostrar las dependencias del compilador, que se representan como flechas con líneas discontinuas (dependencias) de un componente cliente a un componente proveedor del que depende de cierta manera. Los tipos de dependencias son específicos del lenguaje y se pueden representar como estereotipos de las dependencias.

El diagrama también se puede utilizar para mostrar interfaces y las dependencias de llamada entre componentes, usando flechas con líneas discontinuas desde los componentes a las interfaces de otros componentes.

Véase componente para los ejemplos de diagramas de componentes.

diagrama de despliegue

Un diagrama que muestra la configuración de los nodos de proceso y las instancias de componentes y objetos que residen en ellos. Los componentes representan unidades de código en ejecución. Los componentes que no existen como entidades de ejecución (porque se han compilado aparte) no aparecen en estos diagramas; deben aparecer en diagramas de componentes. Un diagrama de despliegue muestra instancias mientras que un diagrama de componentes muestra la definición de los tipos de los componentes por sí mismos.

Véase también componente, interfaz, nodo.

Semántica

La vista de despliegue contiene las instancias de los nodos conectados por los enlaces de comunicaciones. Las instancias de los nodos pueden contener instancias de ejecución, como instancias de componentes y objetos. Las instancias de componentes y objetos también pueden contener otros objetos. El modelo puede mostrar dependencias entre las instancias y sus interfaces, y también puede modelar la migración de entidades entre nodos u otros contenedores.

La vista de despliegue tiene una forma de descriptor y otra de instancia. La forma de instancia (descrita arriba) muestra la localización de las instancias de los componentes específicos en instancias específicas del nodo como parte de una configuración del sistema. Éste es el tipo más común de vista del despliegue. La forma de descriptor muestra qué tipo de componentes pueden subsistir en qué tipo de nodos y qué tipo de nodos se pueden conectar, de forma similar a un diagrama de clases.

Notación

El diagrama del despliegue es una red de símbolos de nodo conectados por líneas que muestran las asociaciones de comunicación (Figura 13.68). Los símbolos de nodo pueden contener instancias de componentes, indicando que el componente vive o se ejecuta en el nodo. Los símbolos de componente pueden contener objetos, indicando que el objeto es parte del componente. Los componentes se conectan con otros componentes con flechas discontinuas de dependencia (posiblemente a través de interfaces). Esto indica que un componente utiliza los servicios de otro componente. Se puede utilizar un estereotipo para indicar la dependencia exacta, si es necesario.

Los diagramas de despliegue son, en gran medida, semejantes a los diagramas de objetos. Generalmente, muestran las instancias individuales del nodo implicadas en un sistema. Es menos común mostrar un diagrama de despliegue que defina los tipos de nodos que existen y sus posibles relaciones con otros tipos de nodos, como en un diagrama de clases.

La migración de componentes de un nodo a otro nodo u objetos de un componente a otro componente se puede representar usando la palabra clave «**become**» en una flecha de línea discontinua. En este caso, el componente o el objeto reside en su nodo o componente sólo parte del tiempo. La Figura 13.146 muestra un diagrama de despliegue en el que un objeto se mueve entre los nodos.

Véase convertirse.

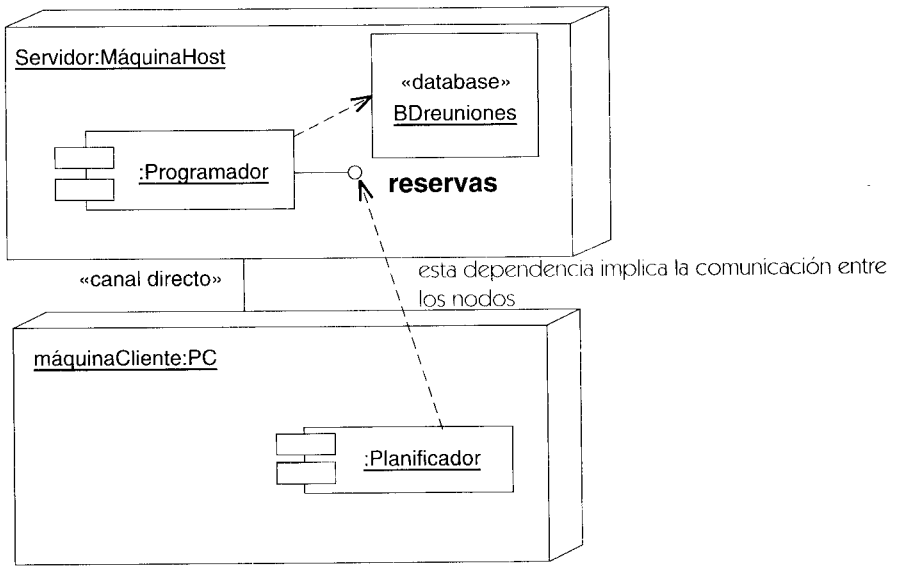


Figura 13.68 Diagrama de despliegue de un sistema cliente-servidor

diagrama de estados

Diagrama que muestra una máquina de estados, incluyendo estados simples, transiciones y estados compuestos anidados. El concepto original fue inventado por David Harel.

Véase máquina de estados.

diagrama de interacción

Se trata de un término genérico que se aplica a varios tipos de diagramas que hacen hincapié en las interacciones entre objetos. Los diagramas de actividades están íntimamente relacionados.

Véase también colaboración, interacción.

Notación

El patrón de interacción entre objetos se muestra en un diagrama de interacción. Los diagramas de interacción tienen diferentes formas, basadas todas ellas en una misma información subyacente pero resaltando cada una un punto de vista de la misma: diagramas de secuencia, diagramas de colaboración y diagramas de actividades.

Un diagrama de secuencia muestra una interacción que está organizada como una secuencia temporal. En particular, muestra los objetos que participan en la interacción mediante sus líneas de vida y mediante los mensajes que intercambian, organizados en forma de una secuencia temporal. Un diagrama de secuencia no muestra los enlaces existentes entre objetos. Los diagramas de secuencia tienen distintos formatos, adecuados para propósitos diferentes.

Un diagrama de secuencia puede existir en forma genérica (describiendo todas las posibles secuencias) y en forma de instancia (describiendo una secuencia de ejecución consistente con la forma genérica). En los casos que carecen de bucles o bifurcaciones, las dos formas son isomorfas: el descriptor es un prototipo de sus instancias.

Un diagrama de colaboración muestra una interacción organizada en torno a los objetos que efectúan operaciones. Es parecido a un diagrama de objetos que muestra los objetos y los enlaces existentes entre ellos que se necesitan para implementar una operación de nivel más elevado.

La secuencia temporal de mensajes se indica mediante los números de secuencia que aparecen en las flechas de flujo de mensajes. Se pueden mostrar secuencias tanto secuenciales como concurrentes, empleando la sintaxis adecuada. Los diagramas de secuencia muestran secuencias temporales empleando el orden geométrico de las flechas que constan en el diagrama. Por tanto, no se necesitan números de secuencia aunque se pueden incluir estos números por comodidad o para permitir el paso a un diagrama de colaboración.

Los diagramas de secuencia y de colaboración expresan una información similar, pero la muestran de maneras diferentes. Los diagramas de secuencia muestran la secuencia explícita de mensajes y son mejores para especificaciones de tiempo real y para escenarios complejos. Los

diagramas de colaboración muestran las relaciones entre objetos y son mejores para comprender todos los efectos que tiene un objeto y para el diseño de procedimientos.

Discusión

Un diagrama de actividades muestra los pasos procedimentales implicados en la realización de una operación de alto nivel. No es un diagrama de interacción, porque muestra el flujo de objeto entre pasos procedimentales y no el flujo de objeto entre objetos. Un diagrama de actividades se centra sobre todo en los pasos del procedimiento. No muestra la asignación de operaciones a clases destino. Un grafo de actividades es una forma de máquina de estados; modela el estado de ejecución de un procedimiento. Existe un cierto número de iconos especiales que se usan en los diagramas de actividades y son equivalentes a las estructuras básicas de UML con ciertas restricciones adicionales, pero se ofrecen por comodidad.

diagrama de objetos

Término que denota los diagramas que muestran los objetos y sus relaciones en un determinado instante de tiempo. Un diagrama de objetos se puede considerar como un caso especial de diagrama de clases en el que se pueden mostrar tanto las clases como las instancias. También están relacionados los diagramas de colaboración, que muestran objetos prototípicos (roles de clasificador) dentro de un contexto.

Véase también diagrama.

Notación

No es necesario que las herramientas admitan un formato separado para los diagramas de objetos. Los diagramas de clases pueden contener objetos, así que un diagrama de clases con objetos pero sin clases es un "diagrama de objetos". La frase resulta útil, sin embargo, para caracterizar una utilización particular que se puede realizar de distintas formas.

Discusión

Los diagramas de objetos muestran un conjunto de objetos y enlaces que representan el estado de un sistema en un determinado instante de tiempo. Contienen objetos con valores, no descriptores, aunque por supuesto pueden ser prototípicos en muchos casos. Para mostrar un patrón general de objeto y relaciones que se puedan instanciar muchas veces, utilice un diagrama de colaboración, que contiene descriptores (roles de clasificador y roles de asociación) de objetos y enlaces. Si se instancia un diagrama de colaboración, producirá un diagrama de objetos.

El diagrama de objetos no muestra la evolución del sistema con el tiempo. Para este propósito, haga uso de un diagrama de colaboración con mensajes, o bien emplee un diagrama de secuencia para representar una interacción.

diagrama de secuencia

Diagrama que muestra las interacciones entre objetos organizadas en una secuencia temporal. En particular, muestra los objetos participantes en la interacción y la secuencia de mensajes intercambiados.

Véase también activación, colaboración, línea de vida, mensaje.

Semántica

Un diagrama de secuencia representa una interacción, un conjunto de comunicaciones entre objetos organizadas visualmente por orden temporal. A diferencia de los diagramas de colaboración, los diagramas de secuencia incluyen secuencias temporales pero no incluyen las relaciones entre objetos. Pueden existir en forma de descriptor (describiendo todos los posibles escenarios) y en forma de instancia (describiendo un escenario real). Los diagramas de secuencia y los diagramas de colaboración expresan información similar, pero la muestran de maneras distintas.

Notación

Un diálogo de secuencia posee dos dimensiones: la dimensión vertical representa el tiempo; la dimensión horizontal representa los objetos que participan en la interacción (Figura 13.69 y Figura 13.70). En general, el tiempo avanza hacia abajo dentro de la página (se pueden

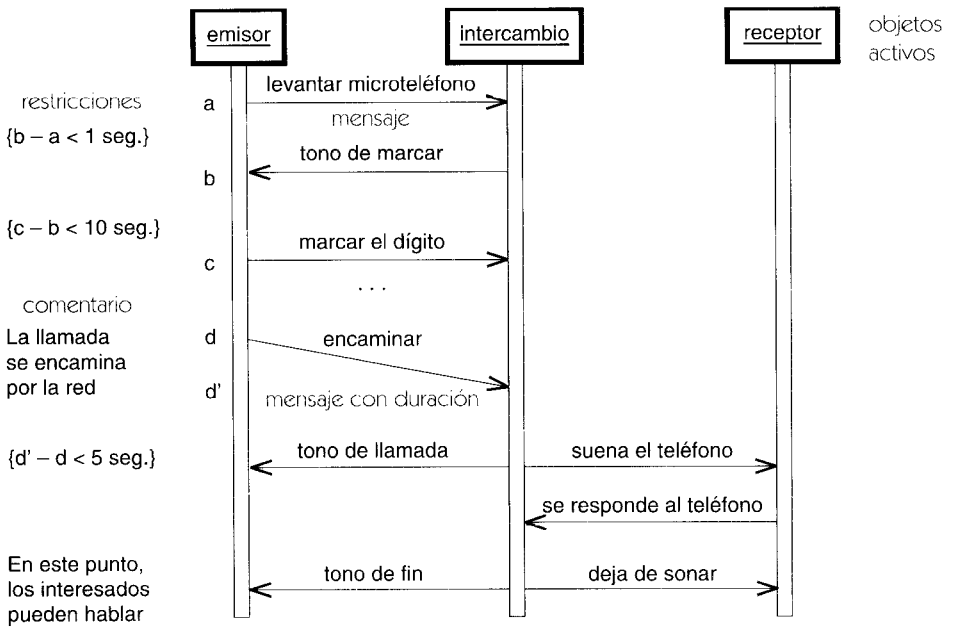


Figura 13.69 Diagrama de secuencia con control asíncrono

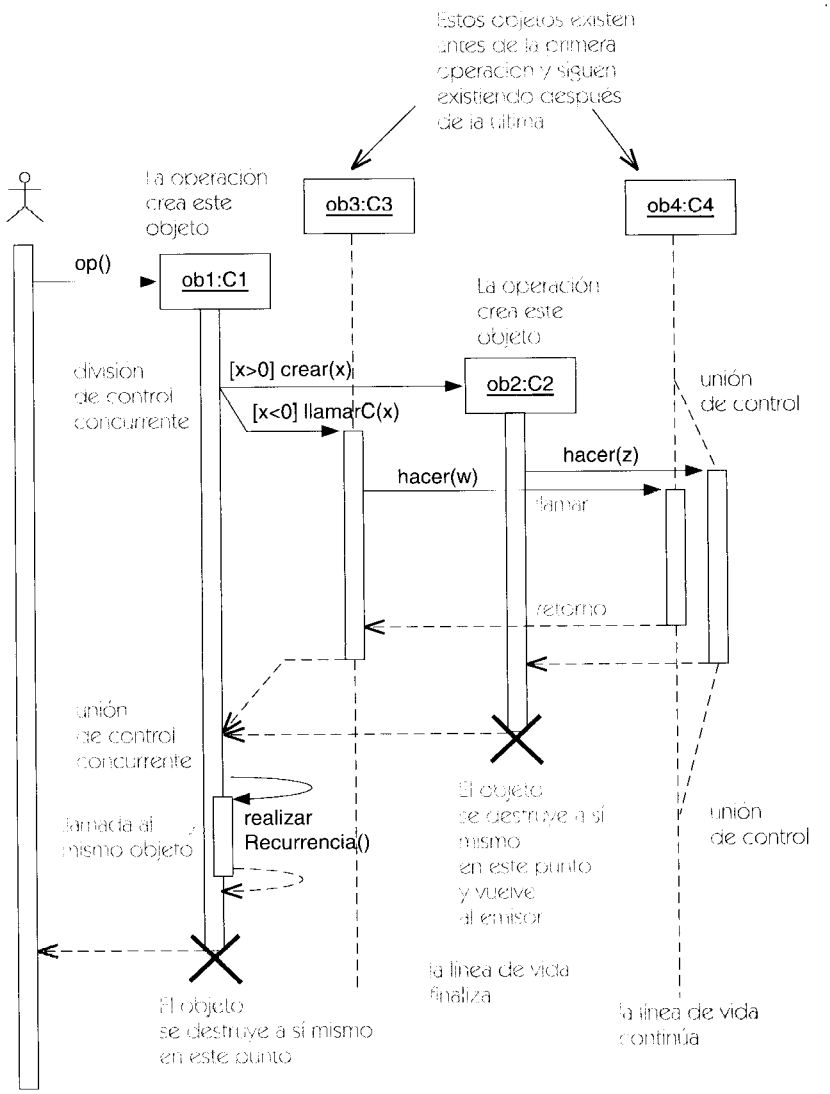


Figura 13.70 Un diagrama de secuencia con flujo de control procedural

invertir los ejes si se desea). Con frecuencia sólo son importantes las secuencias de mensajes pero en aplicaciones de tiempo real el eje temporal puede ser una métrica. La ordenación horizontal de los objetos no tiene ningún significado.

Cada objeto se representa en una columna distinta. Se pone un símbolo de objeto (un rectángulo con el nombre del objeto subrayado) al final de una flecha que representa el mensaje que ha creado el objeto; está situada en el punto vertical que denota el instante en que se crea el objeto. Si un objeto existe desde antes de la primera operación del diagrama, se dibuja el símbolo del objeto en la parte superior del diagrama, antes de todo mensaje. Se dibuja una línea discontinua desde el símbolo de objeto hasta el punto en que se destruye el objeto (si es que esto sucede durante el período de tiempo que muestra el diagrama). Esto se llama línea de vida del objeto. Se pone una *X* grande en el punto en que deja de existir el objeto o en el punto en que el objeto se destruye a sí mismo. Para cualquier período durante el cual esté activo el

objeto, la línea de vida se amplía para ser una doble línea continua. Esto incluye toda la vida de un objeto activo o las activaciones de un objeto pasivo, un período durante el cual se está ejecutando una operación del objeto, incluyendo el tiempo durante el cual la operación espera al retorno de alguna operación que ella haya invocado. Si el objeto se llama a sí mismo recursivamente, bien sea de manera directa o indirecta, entonces se superpone otra copia de la doble línea continua para mostrar la doble activación (potencialmente, podrían ser más de dos copias). El orden relativo de los objetos no tiene significado, aun cuando resulta útil organizarlos de tal modo que se minimice la distancia que tienen que recorrer las flechas de mensajes. Se puede poner cerca de ella un comentario relativo a la activación.

Cada mensaje se representa mediante una flecha horizontal que va desde la línea de vida del objeto que envió el mensaje hasta la línea de vida del objeto que ha recibido el mensaje. Se puede poner una etiqueta en el margen del lado opuesto a la flecha para denotar el momento en que se envía el mensaje. En muchos modelos se supone que el mensaje es instantáneo, o al menos atómico. Si un mensaje requiere un cierto tiempo para llegar a su destino, entonces la flecha del mensaje se dibuja diagonalmente hacia abajo, de tal modo que el instante de recepción sea posterior al instante de emisión. Los dos extremos pueden tener etiquetas para indicar el momento en que se ha enviado o recibido el mensaje.

Para un flujo de objeto asíncrono entre objetos activos, los objetos se representan mediante líneas dobles continuas y los mensajes se representan como flechas. Se pueden enviar simultáneamente dos mensajes pero no se pueden recibir simultáneamente porque no se puede garantizar una recepción simultánea. La Figura 13.69 muestra un diagrama de secuencia asíncrono.

La Figura 13.70 muestra un flujo de objeto procedimental en un diagrama de secuencia. Cuando se está modelando un flujo de objeto procedimental, un objeto cede el control al producirse una llamada, hasta el subsiguiente retorno. Las llamadas se muestran mediante una punta de flecha continua y rellena. La punta de una flecha de llamada puede hacer que comience una activación o bien un nuevo objeto. Los retornos se muestran con una línea discontinua. El origen de una flecha de retorno puede hacer que finalice una activación o un objeto.

Las bifurcaciones se muestran partiendo la línea de vida del objeto. Cada bifurcación puede enviar y recibir mensajes. Normalmente, cada rama envía mensajes diferentes. Eventualmente, las líneas de vida del objeto tienen que fusionarse de nuevo.

La Figura 13.71 muestra los estados que aparecen durante la vida de una entrada para algún espectáculo. Las líneas de vida pueden verse interrumpidas por un símbolo de estado para mostrar un cambio de estado. Esto corresponde a una transición de conversión en un diagrama de colaboración. Se puede dibujar una flecha hasta el símbolo de estado para indicar el mensaje que ha dado lugar al cambio de estado.

Gran parte de esta notación se ha extraído directamente de la notación Object Message Sequence Chart (Diagrama de Secuencias de Mensajes de Objetos) propuesta por Buschmann, Meunier, Rohnert, Sommerlad y Stal [Buschmann-96], que a su vez se deriva de la notación Message Sequence Chart (diagrama de secuencias de mensajes).

Un diagrama de secuencia también se puede mostrar en forma de descriptor, en el cual los constituyentes son roles en lugar de objetos. Este diagrama muestra el caso general, no una sola ejecución del mismo. Los diagramas del nivel de descriptores se dibujan sin subrayados, porque los símbolos denotan roles y no objetos individuales.

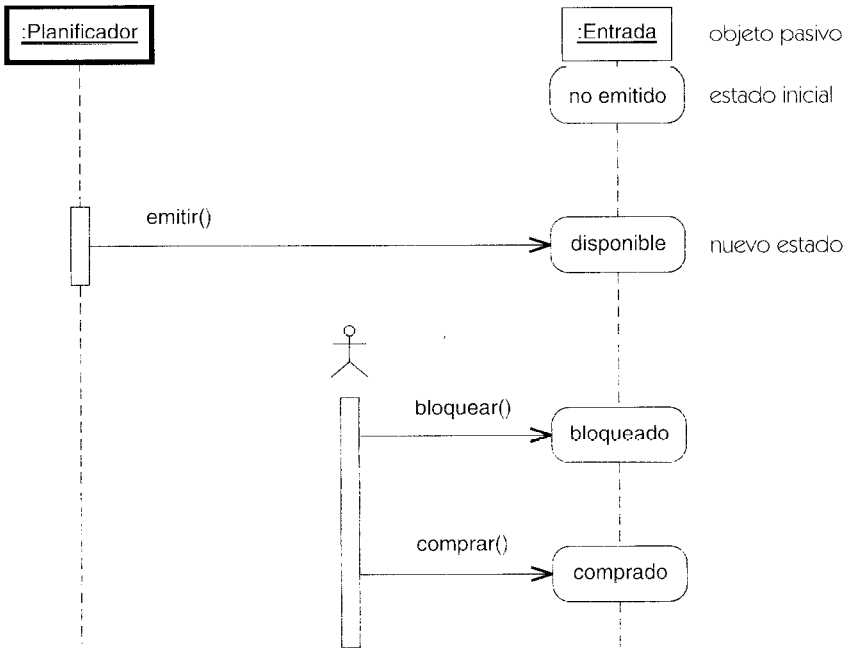


Figura 13.71 Estados de objetos en un diagrama de secuencia

discriminador

Un pseudoatributo que selecciona un elemento hijo de entre un conjunto de hijos en una relación de generalización. Todos los hijos representan una cualidad dada por la que especializar al padre, en contraste a otras cualidades potenciales por las que especializar al mismo padre; representa una dimensión de la especialización.

Véase también generalización, supratipo, pseudoatributo.

Semántica

A veces, un elemento del modelo se puede especializar según diversas cualidades. Cada cualidad representa una dimensión ortogonal independiente de la especialización. Por ejemplo, un vehículo se puede especializar por la propulsión (motor de gasolina, motor espacial, viento, animal, humano), así como por el lugar donde se mueve (tierra, agua, aire, espacio exterior). Un discriminador es el nombre de una dimensión de la especialización. Un elemento se puede especializar en múltiples dimensiones, que deben estar presentes en una instancia concreta.

Una relación de generalización puede tener un discriminador: una cadena que represente una dimensión para clasificar a los hijos del padre. Todas las relaciones de especialización de un solo padre con el mismo nombre de discriminador forman un grupo; cada grupo es una dimensión separada de la especialización. El conjunto completo de nombres del discriminador representa el conjunto completo de dimensiones para especializar al padre. Una instancia debe ser simultáneamente una instancia de un hijo de cada grupo del discriminador. Por ejemplo, un vehículo debe tener una propulsión y un lugar.

Cada discriminador representa una cualidad abstracta del padre, una cualidad que es especializada por los hijos que llevan esa relación del discriminador al padre. Pero un padre con discriminadores múltiples tiene dimensiones múltiples, que se deben especializar para producir un elemento concreto. Por lo tanto, los hijos dentro de un grupo del discriminador son intrínsecamente abstractos. Cada uno de ellos es solamente una descripción parcial del padre, una descripción que acentúe una cualidad e ignore el resto.

Por ejemplo, una subclase de vehículo que se centra en la propulsión omite el lugar. Un elemento concreto requiere la especialización de todas las dimensiones simultáneamente. Esto puede ocurrir por la herencia múltiple del elemento concreto del modelo de un hijo en cada una de las dimensiones, o por la clasificación múltiple de una instancia de un hijo en cada una de las dimensiones. Hasta que se combinan todos los discriminadores, la descripción sigue siendo abstracta.

Por ejemplo, un vehículo real debe tener un medio de propulsión y un lugar. Un vehículo con propulsión por viento y que se mueve por el agua es un barco de vela. No hay nombre particular para un vehículo de tracción animal que se mueva por el aire, pero las instancias de la combinación existen en la fantasía y en la mitología.

La ausencia de una etiqueta de discriminador indica el discriminador «empty» (vacío), que también se considera un discriminador válido (el discriminador por defecto). Esta convención permite que el caso usual no discriminado sea tratado uniformemente. En efecto, si ninguna de las líneas de generalización tiene discriminadores, entonces todos los hijos están en el mismo discriminador. Es decir, hay un discriminador al cual pertenecen todas las especializaciones, y produce la misma semántica que el caso sin discriminador.

Estructura

Cada arco de especialización (generalización) tiene una cadena de discriminador, que puede ser la cadena vacía.

El discriminador es un pseudoatributo del padre. Debe ser único entre los atributos y los roles de la asociación del padre. El dominio para el pseudoatributo es el conjunto de clases hijas. Las múltiples incidencias del mismo nombre de discriminador se permiten entre hijos diferentes y el padre e indican que los hijos pertenecen a la misma partición.

Notación

Un discriminador se muestra como una etiqueta de texto en una flecha de generalización (Figura 13.72). Si dos arcos de generalización con el mismo discriminador comparten una punta de flecha, el discriminador se puede colocar en la punta de flecha.

Ejemplo

La Figura 13.72 muestra una especialización de **Empleado** en dos dimensiones: **estado** del empleado y **localidad**. Cada dimensión tiene el rango de los valores representados por las subclases. Pero son necesarias ambas dimensiones para producir una subclase instanciable. El **Enlace**, por ejemplo, es una clase que es ambos, **Supervisor** y **Expatriado**.

Cualquier descendiente de una sola dimensión es abstracto, hasta que las dos dimensiones se recombinan en un descendiente con herencia múltiple.

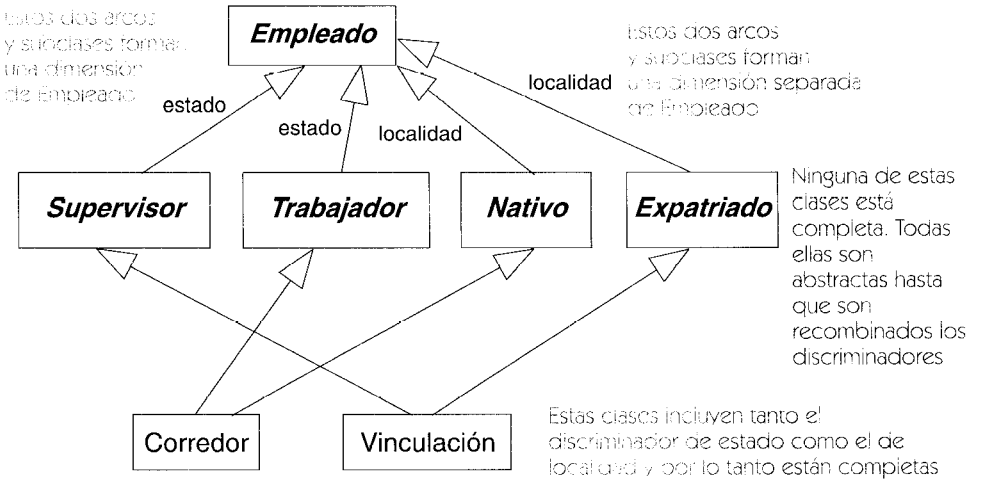


Figura 13.72 Discriminadores

diseño

Esa etapa de un sistema que describe cómo se implementará el sistema, en un nivel lógico sobre código real. En el diseño, las decisiones estratégicas y tácticas se toman para resolver los requisitos funcionales y de calidad requeridos de un sistema. Los resultados de esta etapa son representados por los modelos a nivel de diseño, especialmente la vista estática, vista de la máquina de estados, y vista de interacción. Contrastar con: análisis, diseño, implementación, y despliegue.

Véase fases de modelado, proceso de desarrollo.

disparador

Evento cuya aparición hace que una transición pueda dispararse. Se puede emplear la aplicación como nombre (para denotar el efecto en sí) o como verbo (para denotar la aparición del evento[3]).

Véase también transición de finalización, transición.

Semántica

Toda transición (salvo una transición de finalización, que se dispara al finalizar alguna actividad interna) hace referencia a un evento que forma parte de su estructura. Si se produce el evento

[3] En inglés, la palabra "trigger" significa tanto "disparador" como "disparar". *N. del T.*

cuando el objeto se encuentra en un estado que contenga una transición saliente cuyo disparador sea ese evento o un antecesor del evento, entonces se comprueba la condición de guarda de la transición. Si se cumple la condición, entonces se permite que se dispare la transición. Si hay más de una transición disponible para ser disparada, sólo llega a dispararse una. La selección puede no ser determinista. (Si el objeto posee más de un estado concurrente, se puede disparar una transición desde cada estado, pero como máximo se puede disparar una transición desde cada estado.)

Observe que la condición de guarda sólo se comprueba una vez, en el momento en que se produce el evento disparador (incluyendo los eventos implícitos de finalización). Si no se habilita para el disparo ninguna transición al producirse un evento, entonces se ignora el evento. Esto no es un error.

Los parámetros del evento disparador se pueden utilizar en la condición de guarda o en una acción asociada a la transición, así como en una acción de entrada en el estado destino.

A lo largo de la ejecución de un paso que se ejecuta hasta finalizar realizado después de una transición, el evento disparador sigue estando disponible como evento actual para las acciones de los subpasos de la transición. El tipo exacto de este evento puede ser desconocido en una acción de entrada o en algún segmento posterior de una transición de múltiples segmentos. Por tanto, el tipo de evento se puede discriminar en la acción empleando una operación polimorfa o una sentencia de selección (como *case* o *switch*). Una vez conocido el tipo exacto del evento, se pueden utilizar sus parámetros.

Notación

El nombre y signatura del evento disparador forman parte de la etiqueta de la transición.

Véase transición.

Se puede acceder al evento disparador desde una expresión mediante la palabra clave **currentEvent**. Esta palabra clave se refiere al evento disparador del primer segmento en las transiciones de múltiples segmentos.

disparar

Ejecutar una transición.

Véase también atómico/a, disparador.

Semántica

Cuando ocurre un evento requerido por una transición, y la condición de guarda en la transición está satisfecha, la transición realiza su acción y cambia el estado activo.

Cuando un objeto recibe un evento, se guarda el evento si la máquina de estados está ejecutando una etapa atómica. Cuando se termina la etapa, la máquina de estados gestiona el evento que ha ocurrido. Una transición se activa si su evento se maneja mientras el propio

objeto está en el estado que contiene la transición o en un subestado anidado dentro del estado que contiene la transición. Un evento satisface un evento de disparador cuyo tipo sea un antecesor del tipo del evento que ocurre. Si una transición compleja tiene múltiples estados de origen, todos deben estar activos para que la transición esté disponible. Una transición de terminación se activa cuando su estado origen termina su actividad. Si es un estado compuesto, está disponible cuando todos sus subestados directos han terminado o han alcanzado sus estados finales.

Cuando se trata el evento, se evalúa la condición de guarda (si existe). Si la expresión booleana en la condición de guarda se evalúa como verdadera, entonces la transición se dispara. La acción en la transición se ejecuta, y el estado del objeto se convierte en el estado destino de la transición (sin embargo, no ocurre ningún cambio de estado en una transición interna). Durante el cambio de estado, cualquiera que sean acciones de salida y acciones de entrada en la ruta mínima, desde el estado original del objeto al estado destino, se ejecutan las transiciones. Observe que el estado original puede ser un subestado anidado del estado origen de la transición.

Si la condición de guarda no está satisfecha, nada sucede como resultado de esta transición, aunque alguna otra transición podría dispararse si sus condiciones están satisfechas.

Aunque más de una transición sea elegible para ser disparada, sólo una de ellas se disparará. Una transición en un estado anidado tiene prioridad sobre una transición en un estado que incluye. Si no, la elección de transición es indefinida y puede ser no determinista. Esto es a menudo una situación del mundo real.

Como cuestión práctica, una implementación puede proporcionar una ordenación de las transiciones para dispararlas. Esto no cambia la semántica, pues el mismo efecto podría ser alcanzado organizando las condiciones de guarda de modo que no se solapen. Pero es a menudo más simple poder decir "Esta transición solamente se dispara si ninguna otra transición se dispara".

Eventos diferidos. Si el evento o uno de sus antecesores está marcado como diferido en el estado o en un estado lo que incluye, y el evento no acciona una transición, el evento es un evento diferido hasta que el objeto pasa a un estado en el cual el evento no se difiera. Cuando el objeto pasa a un nuevo estado, cualquier evento previamente diferido pendiente que ya no sea diferido, pasa a estar pendiente y ocurren en un orden indeterminado. Si el primer evento pendiente no causa que se dispare una transición, se ignora y otro ocurre evento pendiente. Si un evento previamente diferido está marcado para el aplazamiento en el nuevo estado, puede accionar una transición, pero sigue diferido si no puede accionar una transición. Si la ocurrencia de un evento causa una transición a un nuevo estado, cualquier evento pendiente restante o evento diferido son evaluados de acuerdo con el estado de aplazamientos del nuevo estado y se establece un nuevo conjunto de eventos pendientes.

Una implementación puede imponer reglas más terminantes sobre el orden en el que los eventos diferidos se procesan u ofrecer operaciones para manipular dicho orden.

división

Una transición compleja en la cual un estado origen es sustituido por dos o más estados destino, dando por resultado un aumento en el número de estados activos. Antónimo: unión.

Véase también transición compleja, estado compuesto, unión.

Semántica

Una división es una transición con un estado origen y dos o más estados de destino. Si el estado origen está activo y ocurre el evento disparador, se ejecuta la acción de la transición y todos los estados destino pasan a estar activos. Los estados destino deben estar en diversas regiones de un estado compuesto concurrente.

Notación

Una división se representa como una línea gruesa a la que llega una flecha de la transición y salen dos o más flechas de transición. Puede tener una etiqueta de transición (condición de guarda, evento disparador, y acción). La Figura 13.73 muestra una división explícita a un estado compuesto concurrente.

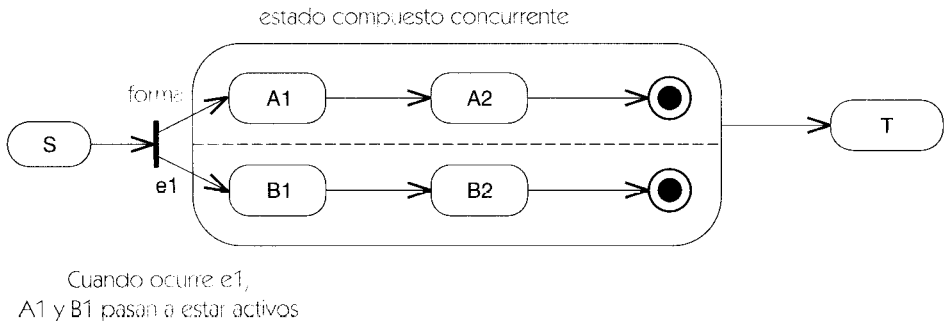


Figura 13.73 División

eficiencia de navegación

Término que indica si es o no posible recorrer eficientemente una asociación binaria partiendo de un objeto para obtener el objeto o conjunto de objetos asociados con él. El concepto no es aplicable a las asociaciones *n*-arias. La eficiencia de navegación está relacionada con la navegabilidad, pero no es una propiedad definitoria.

Véase también navegabilidad.

Semántica

Se puede definir la eficiencia de navegación en forma general, para que sea aplicable al diseño abstracto y también a distintos lenguajes de programación. Diremos que una asociación binaria es navegable eficientemente si el coste medio necesario para obtener el conjunto de objetos asociados es proporcional al número de objetos del conjunto (no al límite superior de la multiplicidad, que podría ser infinito) más una constante determinada. En términos de complejidad computacional, el coste es $O(n)$. Si la multiplicidad es uno o cero-uno, entonces el coste de acceso tiene que ser constante, lo cual impide efectuar búsquedas en una lista de longitud variable. Una definición ligeramente más imprecisa de la eficiencia de navegación permitiría un coste mínimo de $log(n)$.

Aun cuando las asociaciones navegables de multiplicidad uno suelen implementarse empleando un puntero incrustado en el bloque que contiene los atributos del objeto, es posible una implementación externa empleando tablas hash, que tienen un coste medio de acceso constante. Por tanto, se puede implementar una asociación como un objeto de tabla de búsqueda externo con respecto a las clases participantes y se puede seguir considerando navegable. (En algunas situaciones de tiempo real, es preciso limitar el tiempo en el caso peor y no el tiempo medio. Esto no requiere un cambio en la definición básica salvo cambiar el tiempo medio por el tiempo en el caso peor, pero pueden ser descartados los algoritmos probabilísticos como las tablas hash.)

Si una asociación no es navegable en una dirección dada, esto no significa que no se pueda recorrer en absoluto, sino que el coste del recorrido puede ser significativo —por ejemplo, quizá requiera efectuar búsquedas en una larga lista—. Si el acceso en una dirección es infrecuente, una búsqueda puede ser una opción razonable. La eficiencia de navegación es un concepto de diseño que permite al diseñador diseñar las rutas de acceso a objetos teniendo una comprensión de los costes de complejidad computacional. Normalmente, la navegabilidad implica la eficiencia de navegación.

Resulta posible (aun siendo relativamente raro) tener una asociación que no sea navegable eficientemente en ninguna dirección. Esta asociación podría estar implementada como una lista de enlaces que es preciso estudiar para realizar un recorrido en cualquier dirección. Sería posible recorrerla, pero sería poco eficiente. Sin embargo, la utilidad de una de estas asociaciones es reducida.

Discusión

La eficiencia de navegación indica la eficiencia que se tiene al obtener el conjunto de objetos que están relacionados con un objeto dado. Cuando la multiplicidad es 0..1 ó 1, entonces la implementación evidente es un puntero en el objeto origen. Cuando la multiplicidad es muchos, entonces la implementación normal es una clase contenedora que alberga un conjunto de punteros. La clase contenedora, a su vez, puede residir o no en el registro de datos de un objeto de la clase, dependiendo de si se puede o no obtener con coste constante (que es lo que suele suceder para el acceso a través de punteros). La clase contenedora debe tener la propiedad de eficiencia de navegación. Por ejemplo, una lista simple de todos los enlaces de una asociación no sería eficiente, porque los enlaces de un objeto estarían mezclados con muchos otros enlaces carentes de interés y sería preciso hacer una búsqueda. Una lista de enlaces almacenada en cada objeto sí sería eficiente, porque no se requeriría una búsqueda innecesaria.

En una asociación cualificada, una indicación de navegabilidad en la dirección que sale del calificador suele indicar que es eficiente obtener el objeto o conjunto de objetos seleccionado por un objeto origen y un valor del calificador. Esto es consistente con una implementación que haga uso de tablas hash o quizá con un árbol binario indexado por el valor del calificador (que es precisamente la razón por la que se incluyen los calificadores como concepto de modelado).

ejecutar hasta finalizar

Transición o serie de acciones que tiene que acabar por completo.

Véase también acción, atómico/a, máquina de estados, transición.

Semántica

En una máquina de estados, ciertas acciones o series de acciones son atómicas: no se pueden hacer acabar, ni abortar, ni pueden ser interrumpidas por otras acciones. Cuando se dispara una transición, todas las acciones asociadas a ella o que hayan sido invocadas por ella deben terminarse como un grupo, incluyendo las acciones de entrada y las acciones de salida de aquellos estados en los que entre o de los que salga. Se dice que la ejecución de una transición se ejecuta hasta finalizar porque no espera para admitir otros eventos.

La semántica de ejecutar-hasta-finalizar puede contrastarse con la semántica de los estados normales. Cuando está activo un estado, un evento puede producir una transición a otro estado. Toda actividad de ese estado será abortada por la transición.

Una transición puede estar compuesta por múltiples segmentos organizados en forma de cadena y separados mediante pseudoestados. Un grupo de varias cadenas se puede separar o fusionar, así que el modelo global puede contener un grafo de segmentos separados por pseudoestados. Sólo el primer segmento de la cadena puede tener un evento disparador. La transición se dispara cuando la máquina de estados maneja el evento disparado. Si se satisfacen las condiciones de guarda de todos los segmentos, entonces la transición está habilitada y se dispara, siempre y cuando no se dispare otra transición. Se ejecutan las acciones de los sucesivos segmentos. Una vez comenzada la ejecución, es preciso ejecutar las acciones de todos los segmentos antes de que haya finalizado el paso de ejecución hasta finalizar.

Durante la ejecución de un paso que se ejecuta hasta finalizar, el evento disparador que haya iniciado la transición está disponible para las acciones como evento en curso. Las acciones de entrada y salida pueden por tanto obtener argumentos del evento disparador. Los distintos eventos pueden dar lugar a una acción de entrada o de salida, pero una acción puede discriminar el tipo de suceso en curso en una sentencia de caso.

Como consecuencia de la semántica que poseen las acciones en el caso de una ejecución hasta finalización, debería utilizarse para modelar asignaciones, indicadores, aritmética sencilla y otros tipos de operaciones de mantenimiento doméstico. Los cálculos largos deberían modelarse como actividades interrumpibles.

elaboración

La segunda fase de un proceso de desarrollo de software, durante el cual el diseño del sistema ha empezado y la arquitectura es desarrollada y probada. Durante esta fase, se completa la mayor parte de la vista del análisis, junto con las partes arquitectónicas de la vista del diseño. Si se construye un prototipo ejecutable, se hace algo de la vista de la implementación.

Véase proceso de desarrollo.

elemento

Un componente atómico de un modelo. Este libro describe los elementos que pueden utilizarse en los modelos de UML —elementos del modelo, que expresan información semántica, y

elementos de presentación, que proporcionan representaciones gráficas de la información del elemento modelo.

Véase también diagrama, elemento modelo.

Semántica

El término *elemento* es tan amplio que tiene poca semántica específica.

Todos los elementos pueden tener la siguiente propiedad.

valor etiquetado — Se pueden unir cero o más pares etiqueta-valor a cualquier elemento. La etiqueta es un nombre que identifica el significado del valor. Las etiquetas no son fijas en UML sino que se pueden extender para denotar varios tipos de información significativa para el modelador o para una herramienta de edición.

Elementos estándar

documentation.

elemento modelo

Dícese de un elemento que es una abstracción extraída del sistema que se está modelando. Compárese con elemento de presentación, que es una presentación (generalmente visual) de uno o más elementos de modelado para la interacción humana.

Semántica

Todos los elementos que tienen semántica son elementos del modelo, incluyendo los conceptos del mundo real y los conceptos de implementación de un sistema de computadores. Los elementos gráficos que tienen como propósito visualizar un modelo son elementos de representación. No son elementos del modelo, por cuanto no añaden semántica al mismo.

Los elementos modelo pueden poseer nombres, pero el uso y restricciones relativos a los nombres varían según la clase de elemento modelo y se discuten para cada una de estas clases. Todo elemento modelo pertenece a un espacio de nombres adecuado para la clase del elemento. Todos los elementos modelo pueden tener asociadas las propiedades siguientes:

valor etiquetado — Es posible asociar cero o más parejas valor-etiqueta a cualquier elemento modelo o de representación. La etiqueta es un nombre que identifica el significado del valor. Las etiquetas no están fijadas en UML pero se pueden extender para denotar distintas clases de información significativas para quien crea el modelo o para alguna herramienta de edición.

restricción	Es posible asociar cero o más restricciones a los elementos modelo. Las restricciones son limitaciones que se expresan como cadenas lingüísticas en un lenguaje de restricción.
estereotipo	Se pueden asociar cero o más estereotipos a un elemento modelo, siempre y cuando el estereotipo sea aplicable al elemento base del modelo. El estereotipo no altera la estructura de la clase base, pero puede agregar restricciones y valores etiquetados que sean aplicables a los elementos del modelo que lleven ese estereotipo.

Además, los elementos del modelo pueden participar en relaciones de dependencia.

Véase el Capítulo 14, Elementos Estándar, para estudiar una lista de etiquetas, restricciones y estereotipos predefinidos.

elemento de representación

Dícese de la proyección textual o gráfica de uno o más elementos del modelo. *Véase también* diagrama.

Semántica

Los elementos de representación (que a veces se llaman elementos de visualización, aunque incluyen formas de representación que no son gráficas) presentan la información de un menú desplegable de forma adecuada para su percepción por seres humanos. Son la notación. Muestran toda la información semántica relativa a un elemento del modelo, o parte de ella. También pueden agregar información estética útil para las personas, por ejemplo agrupando aquellos elementos que están relacionados conceptualmente. Pero la información añadida no posee contenido semántico. Lo que se espera es que los elementos de representación sean responsables de mantenerse correctamente a sí mismos a pesar de los cambios experimentados por el modelo subyacente; sin embargo, los elementos del modelo no tienen por qué ser conscientes de los elementos de presentación para funcionar correctamente.

Las descripciones de la notación UML de este libro definen la correspondencia entre elementos del modelo y representaciones gráficas en una pantalla. La implementación de los elementos de representación como objetos es responsabilidad de la implementación de las herramientas.

elemento derivado

Un elemento que se puede calcular a partir de otros elementos y es incluido por claridad o por propósitos del diseño aunque no añade ninguna información semántica.

Véase también restricción, dependencia.

Semántica

Un elemento derivado es lógicamente redundante dentro de un modelo porque puede ser calculado a partir de uno o más elementos diferentes. La fórmula para calcular un elemento derivado se puede dar como restricción.

Un elemento derivado puede ser incluido en un modelo por varias razones. En el análisis, un elemento derivado es semánticamente innecesario, pero puede ser utilizado para proporcionar un nombre o una definición para un concepto significativo, como un tipo de macro. Es importante recordar que un elemento derivado no agrega nada a la semántica de un modelo.

En un modelo de diseño, un elemento derivado representa una optimización —un elemento que puede ser calculado a partir de otros elementos pero que está físicamente presente en el modelo para evitar el coste o el problema del recálculo—. Los ejemplos son un valor intermedio de un cálculo o un índice a un conjunto de valores. La presencia de un elemento derivado implica la responsabilidad de actualizarlo si los valores de los que depende cambian.

Notación

Un elemento derivado se representa mediante una barra inclinada (/) delante del nombre del elemento derivado, tal como una atributo, un nombre de rol, o de un nombre de la asociación (Figura 13.74).

Los detalles sobre cómo calcular un elemento derivado se pueden especificar por una dependencia con el estereotipo «**derive**». Generalmente, es conveniente suprimir en la notación la flecha de la dependencia que va desde la restricción al elemento y colocar simplemente una cadena de restricción cerca del elemento derivado, aunque la flecha se puede incluir cuando sea provechoso.

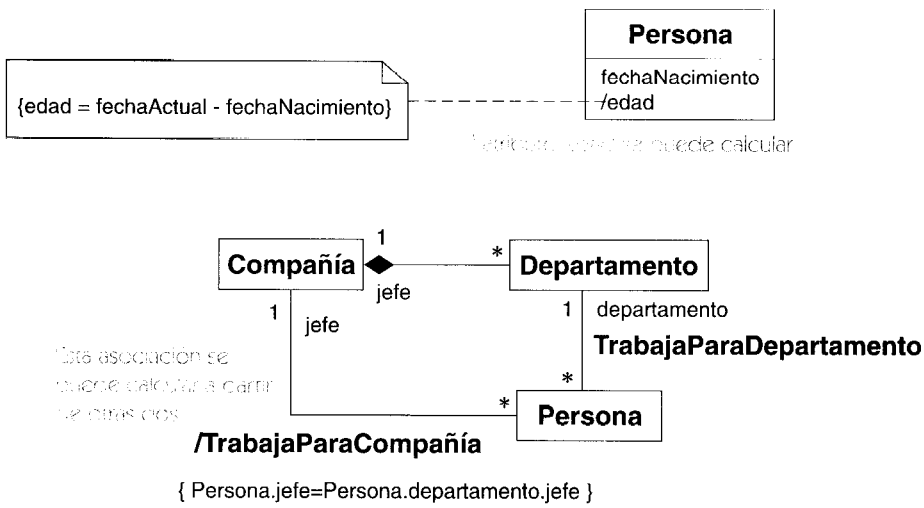


Figura 13.74 Atributo derivado y asociación derivada

Discusión

Las asociaciones derivadas son probablemente el tipo más común de elemento derivado. Representan asociaciones virtuales que se pueden calcular a partir asociaciones de dos o más asociaciones fundamentales. En la Figura 13.74, por ejemplo, la asociación derivada **TrabajaParaCompañía** puede ser calculada componiendo **TrabajaParaDepartamento** con la composición **jefe**.

Una implementación puede incluir explícitamente **TrabajaParaCompañía** para evitar el recálculo, pero no representa ninguna información adicional.

Hay una diferencia con la generalización de la asociación (Figura 13.112), que representa dos niveles de detalle para una asociación. Normalmente no serán implementados ambos niveles. Generalmente solamente las asociaciones del hijo serán implementadas. A veces solamente se implementará la asociación del padre, con las asociaciones del hijo limitando los tipos de objetos que pueden relacionarse.

elemento generalizable

Un elemento del modelo que puede participar en una relación de generalización.

Véase también generalización, herencia.

Semántica

Un elemento generalizable puede tener padres e hijos. Una variable que se clasifica con un elemento puede sostener una instancia de un descendiente del elemento.

Los elementos generalizables incluyen clases, casos de uso, otros clasificadores, asociaciones, estados, eventos, y colaboraciones. Un elemento generalizable hereda las características de sus antecesores. La definición de qué partes de cada tipo de elemento generalizable se heredan, depende del tipo del elemento. Las clases, por ejemplo, heredan atributos, operaciones, métodos, participación en asociaciones, y restricciones. Las asociaciones heredan las clases participantes (éstas pueden especializarse por sí mismas) y las propiedades del extremo de la asociación. Los casos de uso heredan los atributos y operaciones, asociaciones a actores, relaciones de inclusión y extensión a otros casos de uso, y secuencias de comportamiento. Los estados heredan transiciones.

Véase generalización, generalización de asociaciones, generalización de casos de uso.

Estructura

Un elemento generalizable tiene propiedades que declaran dónde puede aparecer dentro de una jerarquía de generalización.

abstraction — Especifica si el elemento generalizable describe instancias directas o es un elemento abstracto que debe ser especializado antes de que pueda ser instanciado. **True** indica que el elemento es abstracto (no puede tener instancias directas); **false** indica que es concreto (puede tener instancias

directas). Para ser utilizable, un elemento abstracto debe tener descendientes concretos. Una clase con una operación que carece de métodos es abstracta por necesidad.

leaf — Especifica si el elemento generalizable puede ser especializado. **True** indica que no puede tener descendientes (es decir, debe ser una hoja); **false** indica que puede tener descendientes (tanto si los tiene actualmente como si no). Una clase abstracta que es una hoja, es inútil para todo menos para agrupar atributos y operaciones globales.

root — Especifica si el elemento debe ser una raíz sin antecesores. **True** indica que debe ser una raíz y no puede tener antecesores; **false** indica que no necesita ser una raíz y puede tener antecesores (tanto si tiene antecesores en ese momento como si no).

Obsérvese que el declarar hojas y clases raíz no afecta a la semántica, pero tales declaraciones pueden proporcionar una declaración de las intenciones del diseñador. Pueden también permitir una compilación más eficiente de paquetes separados evitando la necesidad de un análisis global o de asunciones excedentemente conservadoras sobre operaciones polimórficas.

Elementos estándar

leaf.

elemento ligado

Elemento del modelo que se crea enlazando los parámetros de una plantilla con unos valores de los argumentos.

Véase también ligadura, plantilla.

Semántica

Una plantilla es una descripción parametrizada de un grupo potencial de elementos. Para obtener un elemento real, hay que *ligar* los parámetros de la plantilla con los valores reales. El valor real de cada parámetro es una expresión proporcionada por el alcance en el que sucede la ligadura. La mayoría de los argumentos son o clases o enteros.

Si el alcance es en sí mismo una plantilla, los parámetros de la plantilla externa pueden ser utilizados como argumentos para ligar la plantilla original, volviendo a parametrizarla. Pero los nombres de los parámetros de una plantilla no tienen sentido fuera del cuerpo de la misma. No se puede suponer que los parámetros de dos plantillas se corresponden sólo porque tienen los mismos nombres, al igual que no se asume que los parámetros de las subrutinas se corresponden con sus argumentos sólo basándose en los nombres.

Un elemento ligado está completamente especificado por su plantilla, por lo que su contenido no puede ampliarse (extenderse). No se permite la declaración de nuevos atributos u operaciones para las clases, pero por ejemplo, de una clase ligada podrían derivarse subclases y las subclases extenderse de la manera habitual.

Ejemplo

La Figura 13.75 muestra cómo ligar más de una vez una plantilla. **ArrayPuntos** es una plantilla con un parámetro, el tamaño **n**. Queremos construirla a partir de una plantilla existente, **FArray**, que tiene dos parámetros, el tipo de los elementos **T** y el tamaño **k**. Para hacerlo, el parámetro **k** de la plantilla **FArray** se liga con el parámetro **n** de la plantilla **ArrayPuntos**, y el parámetro **T** de la plantilla **FArray** se liga con la clase **Punto**. Al hacer esto, se produce el efecto de borrado de un parámetro de la plantilla original. Si ahora queremos utilizar la plantilla **ArrayPuntos** para construir una clase **Triángulo**, el parámetro que indica el tamaño **n**, se liga con el valor 3. Para construir una clase **Cuadrilátero**, se liga con el valor 4, y así sucesivamente.

La Figura 13.75 también muestra una plantilla **Polígono**, hija de la clase **Figura**. Esto significa que cada clase construida a partir de la plantilla **FArray** es una subclase de **Figura**: las clases **Triángulo** y **Cuadrilátero** son ambas subclases de **Figura**.

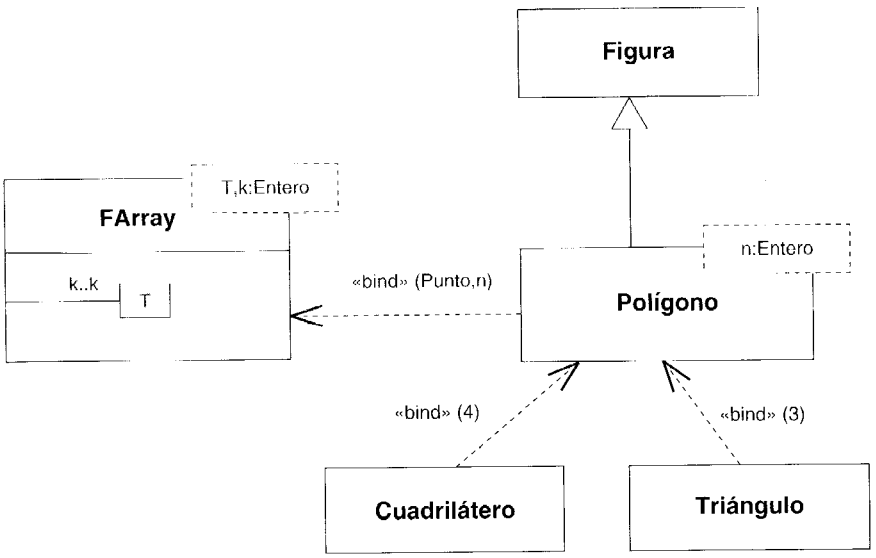


Figura 13.75 Ligado doble de una plantilla

Notación

Un elemento ligado se representa utilizando una flecha discontinua con la palabra clave «**bind**» que parte del elemento ligado y que apunta a la plantilla. Una forma de representación alternativa es utilizar la sintaxis textual NombrePlantilla<argumento$_{1..n}$>, empleando la correspondencia de nombres para identificar la plantilla. La forma textual evita tener que mostrar la plantilla y dibujar la flecha, por lo que es particularmente útil cuando el elemento ligado se utiliza como clasificador para un atributo o parámetro de una operación.

Véase ligadura para más detalles. La Figura 13.130 muestra un ejemplo.

En una clase ligada, los apartados de atributos y operaciones normalmente se suprimen ya que no pueden modificarse en los elementos ligados.

El nombre de un elemento ligado, bien en la forma "anónima" entre símbolos menor y mayor que, o bien en la forma explícita de "ligadura con flecha", se puede utilizar en cualquier parte donde se pueda utilizar el nombre de un elemento del tipo parametrizado. Por ejemplo, un nombre de clase ligada podría utilizarse como tipo de un atributo o en la signatura de una operación dentro de un símbolo de clase en un diagrama. La Figura 13.76 muestra un ejemplo de todo esto.

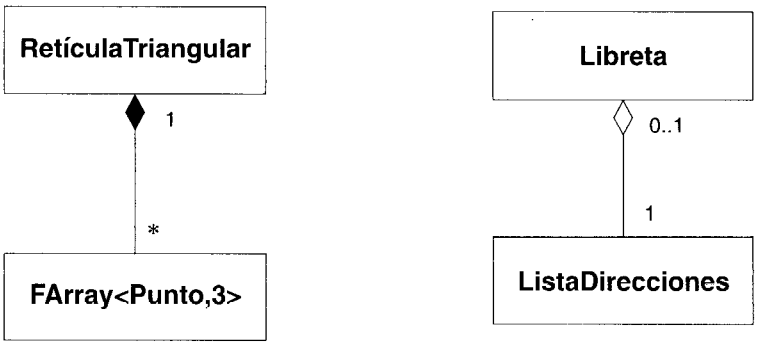


Figura 13.76 Uso de plantillas ligadas en asociaciones

Discusión

Los clasificadores son candidatos obvios a la parametrización. Los tipos de sus atributos, operaciones o clasificadores asociados son parámetros muy comunes en las plantillas. Las colaboraciones parametrizadas se denominan patrones. Las operaciones, en cierto sentido, son inherentemente parametrizadas. La utilidad de la parametrización de otros elementos no está tan clara, pero posiblemente se le encontrarán otros usos en el futuro.

elemento parametrizado

Véase plantilla.

emisor

Objeto que pasa una instancia de mensaje a un objeto receptor.

Véase llamada, enviar.

enlace

Una tupla de referencias de objetos que es una instancia de una asociación o un rol de asociación.

Semántica

Un enlace es una conexión individual entre dos o más objetos. Se trata de una tupla (lista ordenada) de referencias de objetos. Los objetos deben ser instancias directas o indirectas de las clases situadas en las posiciones correspondientes de la asociación. Una asociación no puede contener enlaces duplicados procedentes de la misma asociación —esto es, dos tuplas idénticas de referencias de objetos.

Un enlace que sea una instancia de una clase de asociación puede tener una lista de valores de atributos además de la tupla de referencias de objetos. No se permiten los enlaces duplicados con la misma tupla de referencias de objetos, aun cuando los valores de sus atributos sean distintos. La identidad de un enlace proviene de su tupla de referencias de objetos, que debe ser única.

Se puede emplear un lenguaje para la navegación. En otras palabras, un objeto que aparece en una posición en un enlace puede obtener el conjunto de objetos que aparecen en otra posición. Entonces puede enviarles mensajes (esto se denomina "enviar un mensaje a través de una asociación"). El proceso es eficiente si la asociación tiene la propiedad de navegabilidad en la dirección del destino. El acceso puede o no ser posible si la asociación no es navegable, pero es probable que sea ineficiente. La navegabilidad en direcciones opuestas se especifica de manera independiente.

Dentro de una colaboración, un rol de asociación es una relación contextual, frecuentemente transitoria, entre clasificadores. Una instancia de rol de asociación también es un enlace, pero típicamente se trata de uno cuya vida está limitada a la duración de la colaboración.

Notación

Los enlaces binarios se representan mediante una ruta entre dos objetos —esto es, uno o más segmento de línea o arcos—. En el caso de una asociación reflexiva, la ruta es un bucle cuyos dos extremos llegan a un único objeto.

Véase asociación para más detalles acerca de las rutas.

Puede mostrarse un nombre de rol en el extremo de cada enlace. Se puede mostrar un nombre de asociación junto a la ruta. Si está presente, se subraya el nombre para indicar que es una instancia. Los enlaces poseen nombres de instancia. Toman su identidad de los objetos que relacionan. No se muestra multiplicidad para los enlaces porque las instancias *no* poseen multiplicidad; la multiplicidad es una propiedad del descriptor que limita el número de instancias que pueden existir. Es posible mostrar otros adornos de asociación (agregación, composición y navegación) en los roles de enlace.

Se puede mostrar un calificador en un enlace. El valor de calificador se puede mostrar en su cuadro. La Figura 13.77 muestra tanto enlaces ordinarios como enlaces cualificados.

Otros adornos de los enlaces pueden mostrar propiedades de sus asociaciones, incluyendo el sentido de navegación, la agregación o composición, los estereotipos de implementación y la visibilidad.

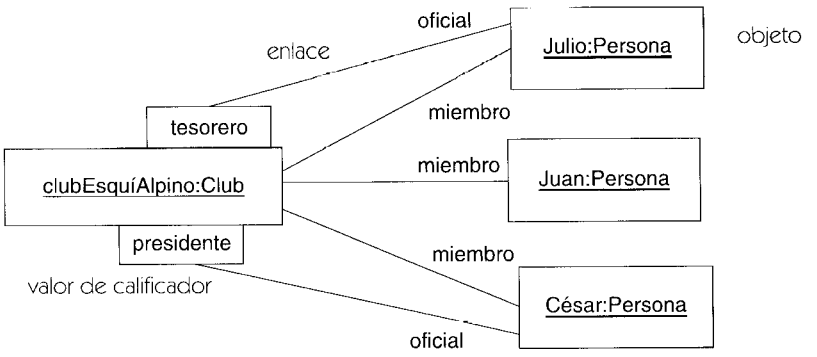


Figura 13.77 Enlaces

***Enlace* n-*ario*.** Un enlace *n*-ario se representa mediante un rombo con una ruta que va hacia cada uno de los objetos participantes. Los demás adornos de la asociación y los adornos de los roles tienen las mismas posibilidades que un enlace binario.

Discusión

¿Cómo debería mostrarse una dependencia en un diagrama de objetos? En general, una dependencia representa una relación entre clases, no entre objetos, y pertenece a un diagrama de clases, no a un diagrama de objetos. ¿Qué sucede con los argumentos de procedimiento, las variables locales de los procedimientos y quien haga la llamada a una operación? Todos ellos deberían existir como estructuras de datos reales y no sólo como simples dependencias. Por tanto, se pueden mostrar como enlaces. El que haga una llamada a un procedimiento necesitará una referencia del objeto destino —esto es, un enlace—. Algunos enlaces pueden ser instancias de roles de asociación en las colaboraciones, tal como la mayoría de los parámetros y variables locales. Las dependencias restantes son relevantes para la clase en sí y no para sus objetos individuales.

enlace transitorio

Enlace que existe durante un período limitado, tal como la ejecución de una operación.

Véase también asociación, colaboración, uso.

Semántica

Durante la ejecución, algunos enlaces existen durante un período limitado. Por supuesto, casi todos los objetos o enlaces poseen una duración limitada, si el período de tiempo considerado es suficientemente grande. Algunos enlaces, sin embargo, existen únicamente en ciertos contextos limitados, tales como la ejecución de un método. Los argumentos de procedimientos y las variables locales se pueden representar como enlaces transitorios. Se pueden modelar

todos estos enlaces como asociaciones, pero entonces hay que enunciar de forma muy imprecisa las condiciones de las asociaciones y pierden gran parte de su precisión para restringir las combinaciones de objetos. En lugar de hacer esto, esas situaciones se pueden modelar empleando colaboraciones, que son configuraciones de objetos y enlaces que existen en contextos especiales.

Se puede considerar que un rol de asociación de una colaboración es un enlace transitorio que existe únicamente durante la ejecución de una entidad de comportamiento, tal como un procedimiento. Aparece dentro del modelo de clases como una dependencia de uso. Para disponer de todos los detalles es preciso consultar el modelo de comportamiento.

Notación

Los enlaces transitorios se representan mediante una asociación con un estereotipo junto al rol de enlace para indicar distintos tipos de implementación. Se pueden utilizar los estereotipos siguientes:

«**parameter**»	Parámetro de procedimiento.
«**local**»	Variable local de un procedimiento.
«**global**»	Variable global (algo visible dentro de todo un modelo o paquete); evítese, si es posible, por violar el espíritu de la orientación a objetos.
«**self**»	Autoenlace (capacidad de un objeto para enviarse mensajes a sí mismo, implícita en los objetos y sólo resulta útil mostrarlo en situaciones dinámicas con flujos de mensajes).
«**association**»	Asociación (por defecto, es innecesario especificarlo salvo para hacer hincapié); no es un enlace transitorio pero se enumera por compleción.

enumeración

Un tipo de dato, cuyas instancias forman una lista de valores literales con nombre. Generalmente, se declaran el nombre de la enumeración y sus valores literales.

Véase también clasificador, tipo de dato.

Semántica

Una enumeración es un tipo de dato definible por el usuario. Tiene un nombre y una lista ordenada de los nombres literales de la enumeración, cada uno de los cuales es un valor en el rango del tipo de dato —es decir, es una instancia predefinida del tipo de dato—. Por ejemplo, **RGBColor** = {**rojo**, **verde**, **azul**}. El tipo de dato **Booleano** es una enumeración predefinida con los literales **false** y **true**.

Notación

Una enumeración se representa mediante un rectángulo con la palabra clave «**enum**» sobre el nombre de la enumeración en el compartimento superior (Figura 13.78). El segundo compar-

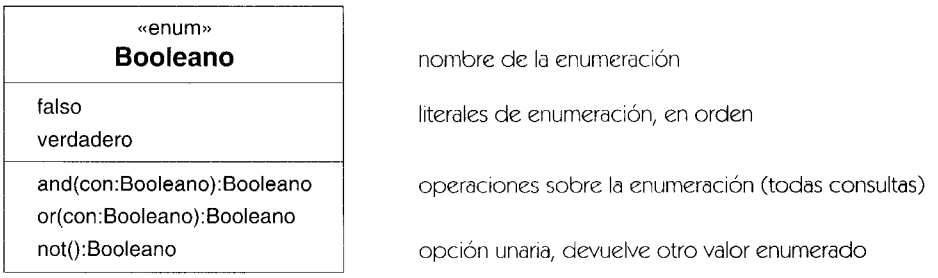


Figura 13.78 Declaración de una enumeración

timento contiene una lista de nombres literales de la enumeración. El tercer compartimento (si existe) contiene un conjunto de operaciones sobre el tipo. Todas deben ser consultas (las cuales, por lo tanto, no necesitan ser declaradas explícitamente como tales).

enviar

Crear una instancia de señal en un objeto emisor y transferirla a un objeto receptor para aportar información.

Una dependencia de uso de envío relaciona una operación o método que envía una señal o a una clase que contiene esa operación o método con la clase que recibe la señal.

Véase también señal.

Semántica

Un envío es una operación especial que puede realizar un objeto. Especifica la señal que se envía, una lista de argumentos de la señal y un conjunto de objetos destino que deben recibir la señal.

Los objetos envían señales a un conjunto de objetos; con frecuencia, se trata de un conjunto que contiene un único objeto. Se puede considerar una "emisión" como el envío de un mensaje a todos los objetos, aunque en la práctica podría ser implementada como un caso especial, para mayor eficiencia. Si el conjunto de objetos destino contiene más de un objeto, se envía una copia de la señal a todos los objetos del conjunto concurrentemente. Si el conjunto está vacío, no se envía la señal. Esto no es un error.

La creación de un nuevo objeto se puede considerar como enviar un mensaje a un objeto factoría, tal como una clase, que crea la nueva instancia y después le pasa el mensaje como "evento de nacimiento". Esto proporciona un mecanismo para que el creador se comunique con su creación; el evento de nacimiento se puede considerar como algo que va del creador al nuevo objeto, con el efecto secundario consistente en instanciar el nuevo objeto de pasada. La Figura 13.79 muestra la creación de un objeto empleando tanto la sintaxis de texto como su versión gráfica.

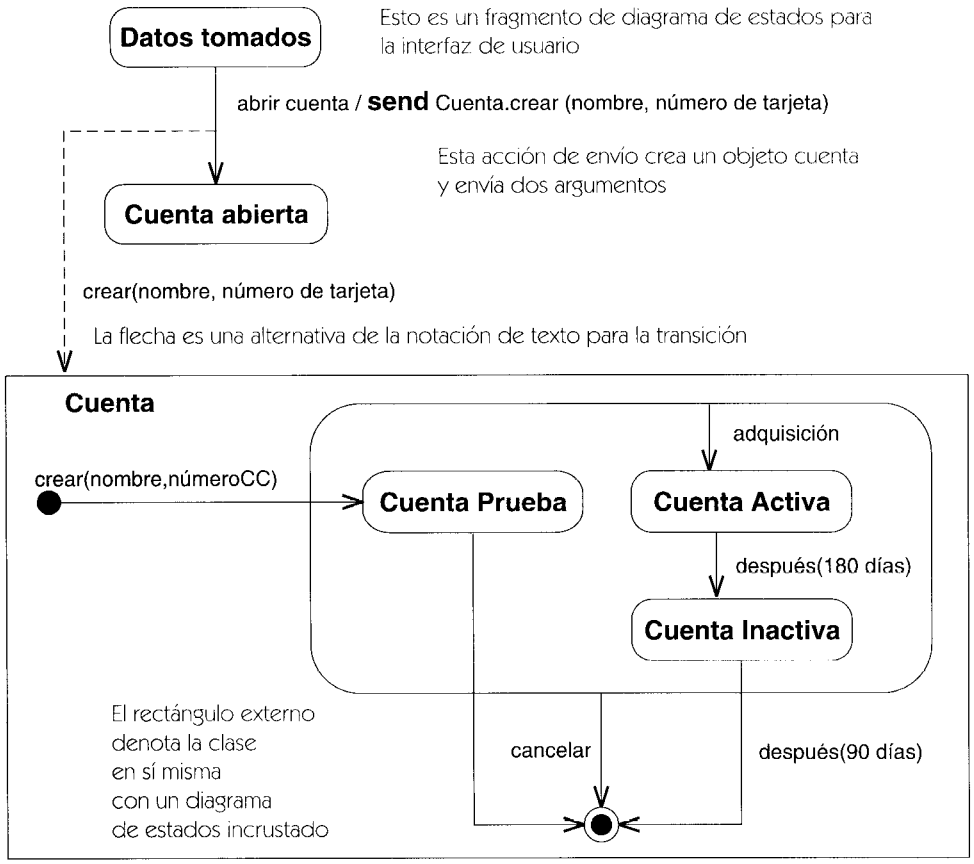


Figura 13.79 Creación de un nuevo objeto enviando un mensaje

Se puede usar este modelo aun cuando el lenguaje destino, como C++, no admita las clases como objetos en tiempo de ejecución. En tal caso, la acción de creación se compila (lo cual impone ciertas restricciones sobre su generalidad, por ejemplo, el nombre de la clase tiene que ser un valor literal) pero la intención subyacente es la misma.

Una dependencia de envío es un estereotipo de una dependencia de uso del emisor de la señal con respecto a la clase que recibe la señal.

Notación de texto

Dentro de una transición, enviar una señal posee su propia sintaxis, aunque en realidad no es más que un caso especial de una acción. Dentro de la secuencia de acción, la expresión de envío tiene la sintaxis

send expresión-destino . nombre-mensaje-destino (argumentos$_{lista}$)

La palabra reservada **send** es opcional. Se puede distinguir una llamada de un envío por la declaración del nombre del mensaje. En algunas ocasiones, sin embargo, resulta útil una distinción explícita.

La expresión-destino tiene que producir, al evaluarse, un conjunto de objetos. Es válido un conjunto que contenga un único objeto. El mensaje se enviará a todos los objetos del conjunto.

El nombre-mensaje-destino es el nombre de una señal o de una operación que admitan los objetos destino. Los argumentos son expresiones que al evaluarse producen valores que deben ser compatibles con los parámetros declarados del evento u operación. La diferencia entre una señal y una operación está basada en las declaraciones de señales en el paquete y de operaciones en la clase destino. No hay ambigüedad en el modelo interno.

Ejemplo

Esta transición interna selecciona un objeto situado dentro de una ventana empleando la localización del cursor y después le envía una señal para resaltarlo.

```
clic-botón-derecho (localización) [localización in ventana]

/objeto:= seleccionar-objeto (localización); send objeto.resaltar()
```

Notación de diagramas

El envío de un mensaje se puede mostrar también mediante símbolos en un diagrama.

El envío de un mensaje entre máquinas de estados se puede mostrar dibujando una flecha discontinua que va del emisor al receptor. La flecha está etiquetada con las expresiones de nombre y argumentos del mensaje. El diagrama de estados tiene que estar contenido en un rectángulo que representa un objeto o clase dentro del sistema. Gráficamente, los diagramas de estados pueden estar anidados físicamente dentro del símbolo de un objeto o bien pueden ser implícitos y ser mostrados en algún otro lugar. Los diagramas de estados representan el control de los objetos que colaboran. Los que interactúan pueden ser roles de una colaboración o bien clases que indican la forma general en que se comunican los objetos de las clases. La Figura 13.80 contiene diagramas de estado que muestran el envío de señales entre tres objetos.

Obsérvese que esta notación también se puede utilizar en otras clases de diagramas para mostrar el envío de eventos entre clases u objetos.

El símbolo del emisor (que se encuentra en el origen de la flecha) puede ser una:

clase — El mensaje es enviado por un objeto de la clase en algún instante de su vida, pero no se especifican detalles.

transición — El mensaje se envía formado parte de la acción de disparo de la transición (Figura 13.79 y Figura 13.80). Se trata de una presentación alternativa de la sintaxis de texto para enviar mensajes.

El símbolo del receptor (situado en la cabeza de flecha) puede ser una:

clase — El mensaje es recibido por el objeto y puede desencadenar una transición dentro del objeto. El símbolo de clase puede contener un diagrama de estados (Figura 13.80). El objeto receptor puede poseer múltiples transiciones que utilicen como disparador el mismo evento. Esta notación no se puede emplear cuando el objeto destino se calcula dinámicamente. En tal caso, es preciso utilizar una expresión de texto.

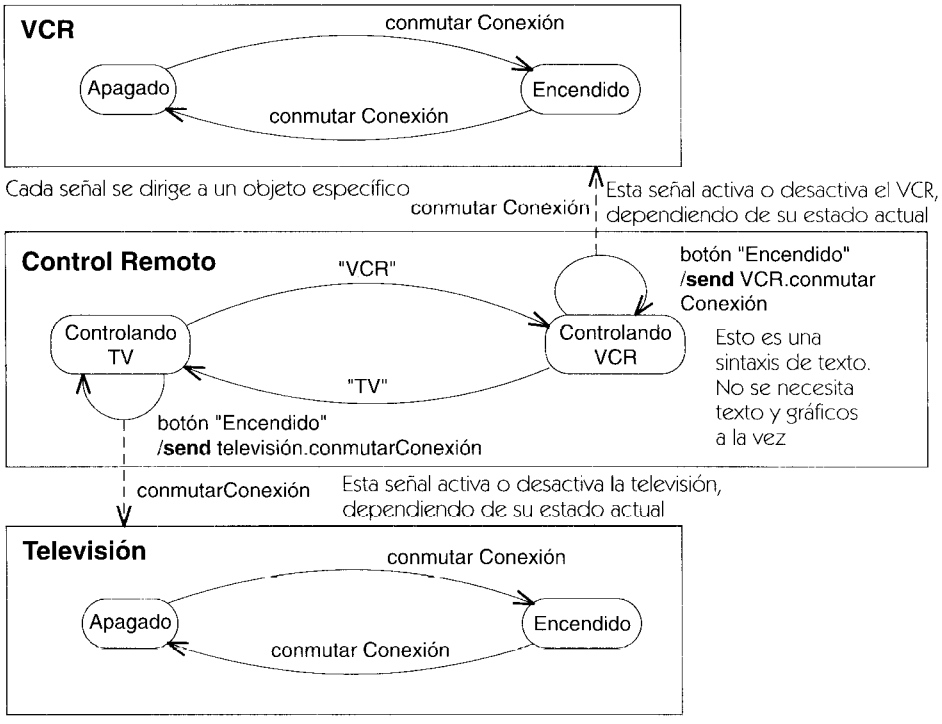


Figura 13.80 Envío de señales entre objetos

metaclase — Esta notación se emplearía para modelar la invocación de operaciones con alcance de clase, tales como la creación de una nueva instancia. La recepción de tales mensajes da lugar a la instanciación de un nuevo objeto en su estado inicial por defecto. El evento que ve el receptor puede utilizarse para disparar una transición de su estado inicial por defecto y por tanto representa una forma de pasar información procedente del creador al nuevo objeto.

transición — La transición tiene que ser la única transición de la clase que haga uso del evento, o al menos la única transición que pudiera ser disparada por el envío de ese mensaje en particular (Figura 13.79). Esta notación no es posible cuando la transición disparada depende del estado del objeto receptor. En tal caso, la flecha tiene que dibujarse para que llegue hasta la clase.

Dependencia de envío. Una dependencia de envío se muestra como una flecha discontinua que va desde la clase u operación que envía la señal hasta la clase que recibe al señal; se asocia el estereotipo «**send**» a la flecha.

escenario

Secuencia de acciones que ilustra un comportamiento. Un escenario se puede utilizar para ilustrar la interacción o ejecución de una instancia de caso de uso.

espacio de nombres

Cierta parte del modelo en la cual se pueden definir y usar los nombres. Dentro de un espacio de nombres, cada nombre posee un significado exclusivo.

Semántica

Todos los elementos con nombre se declaran en un espacio de nombres y sus nombres están dentro del alcance así definido. Los espacios de nombres del más alto nivel son los paquetes (incluyendo los subsistemas), contenedores cuyo propósito es agrupar elementos primordialmente para el acceso y comprensión por parte de personas, así como organizar los modelos para su almacenamiento y manipulación por computador durante el desarrollo. Los elementos primarios de los modelos, como las clases, asociaciones, máquinas de estados y colaboraciones, actúan como espacios de nombres para sus contenidos, tales como atributos, extremos de asociación, estados y roles de colaboración. El alcance de cada elemento del modelo se discute dentro de su propia descripción. Todos estos elementos del modelo poseen su propio espacio de nombres distintivo.

Los nombres que se definen dentro de un espacio de nombres tienen que ser únicos (después de todo, ése es su propósito). Dado un espacio de nombres y un nombre, es posible hallar un elemento particular dentro del espacio de nombres (si es que tiene nombre —algunos elementos son anónimos y deben hallarse mediante alguna relación con elementos que tengan nombre—). Es posible buscar hacia el interior a lo largo de una lista de espacios de nombres anidados si se dan sus nombres.

Para poder acceder a otros espacios de nombres, un paquete puede acceder a otro paquete o importarlo.

El propio sistema define el espacio de nombres más externo, que proporciona la base para todos los nombres absolutos. Se trata de un paquete, que normalmente posee paquetes anidados en varios niveles hasta que finalmente se llega a los elementos primitivos.

Notación

La notación de un nombre de ruta, que es una ruta o camino que recorrer varios espacios de nombres anidados, se obtiene por concatenación de los nombres de los espacios de nombres (tales como paquetes o clases) separándolos mediante parejas de signos de dos puntos (::).

InterfazUsuario :: **SistemaAyuda** :: **PantallaAyuda**

especialización

Producir una descripción más específica del modelo añadiéndole hijos. La relación opuesta es la generalización, que también se utiliza como nombre de la relación existente entre el elemento más específico y el más general, por cuanto no existe un término adecuado para la relación que carezca de dirección. Un elemento hijo es la especialización de un elemento padre. A la inversa, el padre es la generalización del hijo.

Véase generalización.

especificación

Descripción declarativa de lo que es o hace algo. Por ejemplo, un caso de uso o una interfaz es una especificación. Por contraste: implementación.

especificador de interfaz

Especificación del comportamiento requerido por parte de una clase asociada para satisfacer la intención de la asociación. Consta de la referencia de una interfaz, clase u otro clasificador que especifique el comportamiento requerido.

Véase también rol de asociación, nombre de rol, tipo.

Semántica

En muchos casos, una clase asociada puede tener más funcionalidad de la necesaria para admitir una cierta asociación. Por ejemplo, la clase puede participar en otras asociaciones y su comportamiento global es el comportamiento necesario para admitirlas todas a la vez. Puede ser conveniente especificar con más precisión la funcionalidad que necesita una clase para admitir una cierta asociación. Un especificador de interfaz es un clasificador conectado al final de una asociación, que indica la funcionalidad necesaria para admitir la asociación, sin tener en cuenta la utilización de la clase destino por parte de otras asociaciones.

No se necesita un especificador de interfaz. En muchos casos, incluso en la mayoría, una clase que participa en una asociación posee tan sólo la funcionalidad necesaria y no es preciso decir más. Si se omite un especificador de interfaz, se puede emplear la asociación para acceder a la funcionalidad de la clase asociada.

El especificador puede ser un conjunto de clasificaciones, cada uno de los cuales enuncia el comportamiento que tiene que admitir la clase destino. La clase destino tiene que admitirlos todos, pero puede hacer más que lo que requieran los especificadores.

Notación

Un especificador de interfaz se representan mediante la sintaxis

nombre_de_rol: $\text{nombrei}_{\text{lista,}}$

en lugar de un nombre de rol normal, en donde **nombrei** es el nombre de una interfaz o de algún otro clasificador. Si hay más de un especificador, sus nombres se darán en una lista separada por comas.

Ejemplo

En la Figura 13.81, la clase **Servidor** almacena las solicitudes en una clase **Matriz**. Para este propósito, sin embargo, se necesita únicamente la funcionalidad de una clase **Cola**. No se

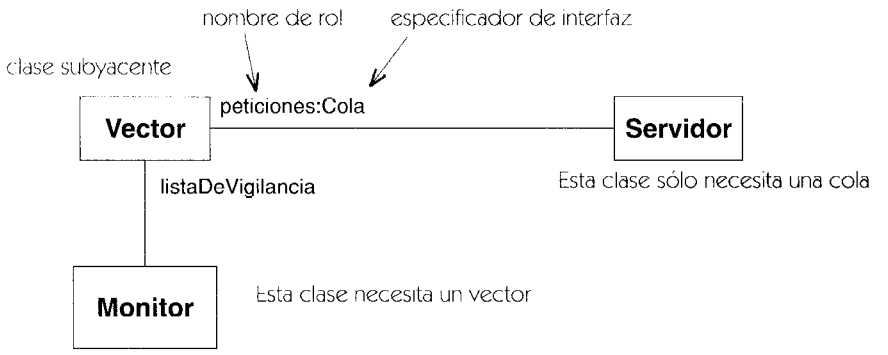


Figura 13.81 Especificador de interfaz

efectúa un acceso aleatorio a la información, por ejemplo. La clase **Matriz** como tal satisface las necesidades del especificador de interfaz **Cola** —una matriz incluye la funcionalidad de una cola—. El **Monitor**, sin embargo, posee una **Matriz** que emplea para visualizar el estado de las solicitudes. El **Monitor** hace uso de la funcionalidad plena de una **Matriz**.

Discusión

El uso de un nombre de rol y de un especificador de interfaz son equivalentes a la creación de una pequeña colaboración que incluya únicamente un rol de asociación y dos roles clasificadores, cuya estructura está definida por el nombre de rol y el rol de clasificador de la asociación original. La asociación y las clases originales son, por tanto, un uso de la colaboración. La clase original tiene que ser compatible con el especificador de interfaz (que puede ser una interfaz o un tipo).

estado

Condición o situación, durante la vida de un objeto, a lo largo de la cual éste satisface alguna condición, realiza una cierta actividad o espera algún evento.

Véase también actividad, grafo de actividades, estado compuesto, acción de entrada, acción de salida, estado final, transición interna, pseudoestado, máquina de estados, submáquina, estado de sincronización, transición.

Semántica

Un objeto almacena una serie de estados a lo largo de su vida. El objeto permanece en un cierto estado durante una cantidad de tiempo finita (no instantánea). Se pueden introducir estados neutros por comodidad, que ejecutan acciones triviales y salen inmediatamente. Pero no es éste el propósito principal de los estados y los estados neutros se pueden eliminar, aunque resultan útiles para evitar duplicaciones.

Los estados están contenidos en una máquina de estados que describe la forma en que evoluciona la historia de un objeto con el paso del tiempo y como respuesta a los eventos. Cada máquina de estados describe el comportamiento de los objetos de una clase. Cada clase puede poseer una máquina de estados. Una transición describe la respuesta de un objeto en un cierto estado frente a la recepción de un evento: el objeto ejecuta una acción opcional asociada a la transición y pasa a un nuevo estado. Cada estado posee su propio conjunto de transiciones.

Las acciones son atómicas y no interrumpibles. Toda acción está asociada a una transición, un cambio de estado, que también es atómico y no interrumpible. La actividad en curso puede estar asociada a un estado. Esta actividad se enuncia como una máquina de estados anidada o una expresión **do**. Alternativamente, la actividad en curso puede estar representada mediante una pareja de acciones, una acción de entrada que da comienzo a la actividad cuando se entra en el estado y una acción de salida que concluye la actividad al salir del estado.

Los estados se pueden agrupar para formar estados compuestos. Toda transición que llega a un estado compuesto afecta a todos los estados que contiene, así que aquellos eventos que afecten de igual modo a muchos subestados pueden ser manejados con una sola transición. Un estado compuesto puede ser secuencial o concurrente. En un estado compuesto secuencial, sólo está activo un subestado en cada instante. En un estado compuesto concurrente, todos los subestados están activos simultáneamente.

Para impulsar el encapsulamiento, un estado puede poseer estados iniciales y estados finales. Se trata de pseudoestados, cuyo propósito es ayuda a estructurar la máquina de estados. Una transición a un estado compuesto representa una transición a su estado inicial. Es equivalente a hacer una transición directa al estado inicial, pero el estado se puede utilizar externamente sin tener conocimiento de su estructura interna.

Una transición a un estado final dentro de un estado compuesto representa la terminación de la actividad en el estado que lo encierra. La terminación de la actividad en el estado que lo encierra dispara un suceso de terminación de actividad y hace que se produzca una transición de finalización en ese estado. Una transición de finalización es la transición que carece de un evento disparador explícito (o más exactamente, una transición en que el evento de finalización es el disparador implícito aunque no se haya modelado explícitamente). La finalización del estado más externo de un objeto corresponde a su muerte.

Si un estado es un estado compuesto concurrente, entonces todas las subregiones concurrentes tienen que terminar antes de que se produzca el evento de finalización en el estado compuesto. En otras palabras, una transición de finalización procedente de un estado compuesto concurrente representa una unión de control de todos sus subhilos concurrentes. Espera a que hayan finalizado todos ellos antes de seguir adelante.

Estructura

Los estados poseen las partes siguientes.

Nombre. El nombre del estado, que tiene que ser único dentro del estado que lo contiene. Se puede omitir el nombre, dando lugar a un estado anónimo. El número de estados anónimos que pueden coexistir no está limitado. Un estado anidado puede identificarse mediante su nombre de ruta (si tienen nombre todos los estados en que esté incluido).

Subestados. Si una máquina de estados posee una subestructura anidada, se denomina estado compuesto. Un estado compuesto es o bien una red de subestados disjuntos (esto es, subestados que se activan secuencialmente) o bien un conjunto de subestados concurrentes (esto es, subestados que están todos activados concurrentemente). Un estado que no posee subestructura (salvo las posibles acciones internas) recibe el nombre de estado simple.

Acciones de entrada y salida. Un estado puede tener una acción de entrada y una acción de salida. El propósito de estas acciones es encapsular el estado de tal modo que se pueda utilizar externamente sin tener conocimiento de su estructura interna. La acción de entrada se ejecuta cuando se entra en el estado, después de cualquier posible acción asociada a la transición entrante y antes de cualquier otra actividad interna. La acción de salida se ejecuta al salir del estado, una vez finalizada toda actividad interna y antes de ejecutar cualquier acción que pueda estar asociada a la transición saliente. En una transición que cruce los límites de varios estados, se pueden ejecutar varias acciones de entrada y salida de forma anidada. En primer lugar, se ejecutan las acciones de salida, comenzando por el estado más interno y avanzando hacia el más externo; a continuación se ejecuta la acción asociada a la transición y después las acciones de entrada, comenzando por la más externa y terminando en la más interna. La Figura 13.82 muestra el resultado de disparar una transición que cruza los límites de un estado. Las acciones de entrada y de salida no se pueden evitar en modo alguno, incluyendo la aparición de excepciones. Proporcionan un mecanismo de encapsulamiento para la especificación del comportamiento de la máquina de estados, con la garantía de que se ejecutarán las acciones necesarias en cualquier circunstancia.

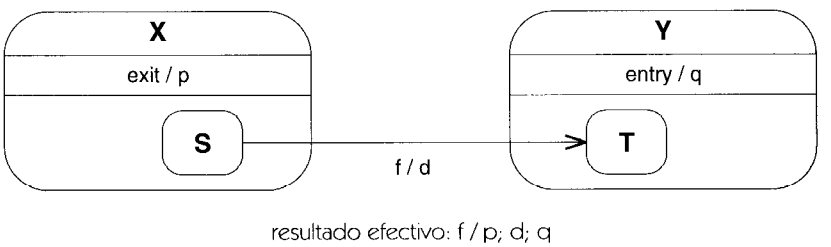


Figura 13.82 Transición a través de los límites de un estado, con acciones de entrada y salida

Actividad interna. Un estado puede contener una actividad interna descrita en forma de una expresión. Cuando se entra en ese estado, la actividad comienza una vez finalizada la acción de entrada. Si termina la actividad, el estado ha finalizado. Entonces se dispara una transición de finalización que sale del estado. En caso contrario, el estado espera al disparo de una transición que dé lugar a un cambio de estado. Si se dispara una transición mientras se está ejecutando la actividad, entonces finaliza la actividad y se ejecuta la acción de salida del estado.

Transiciones internas. Un estado puede poseer una lista de transiciones internas, que son como transiciones normales salvo que no tienen estado destino y no dan lugar a un cambio de estado. Si su evento se produce mientras un objeto se encuentra en el estado que posee la transición, entonces se ejecuta la acción de la transición interna pero no se produce un cambio de estado ni se ejecutan las acciones de entrada ni de salida, aun cuando la transición interna esté *declarada* en un estado que contiene al actual (porque el estado actual no ha cambiado). Esto es

lo que las distingue de una autotransición, en la cual se produce una transición externa de un estado a ese mismo estado, dando lugar a la ejecución de las acciones de salida de todos los estados anidados dentro del estado que tiene la autotransición, a la ejecución de su propia acción de salida y a la ejecución de su acción de entrada. La acciones se ejecutan, incluso, en una autotransición al estado actual, del cual se sale y en el que se vuelve a entrar. Si se dispara una autotransición en un estado que contiene al actual, entonces el estado final es el estado contenedor, no el estado actual. En otras palabras, una autotransición puede forzar una salida de un estado anidado, pero no una transición interna.

Submáquina. El cuerpo de un estado puede representar una copia de otra máquina de estados distinta, a la cual se hace referencia por su nombre. La máquina de estados a la que se hace referencia se denomina submáquina porque está anidada dentro de la máquina de estados mayor y el estado que hace la referencia recibe el nombre de estado de submáquina. Una submáquina puede asociarse a una clase que le proporcione el contexto para las acciones que contiene, tales como los atributos que se puedan leer o escribir. Las submáquinas están destinadas a ser reutilizadas en cualquier máquina de estados para evitar la repetición de un mismo fragmento de máquina de estados. Una submáquina es algo parecido a una subrutina de una máquina de estados.

Dentro del estado de referencia a la submáquina, se hace referencia a la submáquina mediante un nombre con una posible lista de argumentos. El nombre tiene que ser el nombre de una máquina de estados que posea un estado inicial y un estado final. Si la submáquina tiene parámetros en su transición inicial, entonces la lista de argumentos tiene que tener argumentos coincidentes con éstos. Cuando se entra en el estado de submáquina, se ejecuta primero su acción de entrada y después comienza la ejecución de la submáquina en su estado inicial. Cuando la submáquina alcanza su estado final, se ejecutan las posibles acciones de salida del estado de submáquina. Entonces se considera que el estado de submáquina está terminado y se puede efectuar una transición basada en la finalización implícita de la actividad.

Una transición a un estado de referencia de una submáquina activa el estado inicial de la submáquina destino. Pero en algunas ocasiones se desea hacer una transición a un estado diferente de la submáquina. Un estado abreviado externo es un pseudoestado que se encuentra dentro de un estado de referencia a la submáquina y que identifica a un estado contenido en la submáquina. Las transiciones se pueden conectar al estado abreviado externo desde otros estados de la máquina de estados principal. Si se dispara una transición a un estado abreviado externo, el estado al que se hace referencia en la copia de la submáquina se activa.

Una submáquina representa una actividad anidada e interrumpible dentro de un estado. Es equivalente a reemplazar el estado de la submáquina por una copia única de la submáquina. En lugar de proporcionar una máquina de estados, se puede asociar a la submáquina una expresión procedural (que es una actividad). Se puede considerar que una actividad define una serie de estados, uno por cada expresión primitiva, que se puede interrumpir entre dos pasos cualesquiera. No es lo mismo que una acción, que es atómica y no se puede interrumpir.

Concurrencia dinámica. Un estado de actividad o un estado de submáquina puede tener una multiplicidad y una expresión de concurrencia. La multiplicidad especifica cuántas copias del estado se pueden ejecutar concurrentemente. El caso normal es una multiplicidad de exactamente uno, lo cual significa que el estado representa un hilo normal de control. Si no se determina el valor de la multiplicidad, el número de unidades de ejecución se determinará

dinámicamente en el tiempo de ejecución. Por ejemplo, el valor 1..5 significa que se ejecutan concurrentemente entre una y cinco copias de esa actividad. Si existe la expresión de concurrencia (que es obligatoria si la multiplicidad no es exactamente uno), entonces en el tiempo de ejecución tendrá que evaluarse como un conjunto de listas de argumentos. La cardinalidad del conjunto indica el número de activaciones concurrentes del estado. Cada estado recibe una lista distinta de valores de argumentos como valor del evento implícito actual. Cuando han finalizado todas las ejecuciones, el estado dinámicamente concurrente se considera finalizado y la ejecución pasa al estado siguiente.

Esta capacidad está destinada a los diagramas de actividades.

Eventos diferibles. Lista de eventos cuya aparición en el estado queda pospuesta, si no disparan una transición, hasta que disparen una transición o el sistema haga una transición a un estado en el que no se puedan retrasar, momento en el cual serán tratados. La implementación de tales eventos diferibles implicaría una cola interna de eventos.

Notación

Un estado se representa mediante un rectángulo con esquinas redondeadas. Puede tener uno o más compartimentos. Los compartimentos son opcionales. Se pueden incluir los siguientes compartimentos:

Compartimento de nombre. Contiene el nombre (opcional) del estado en forma de cadena. Los estados sin nombre son anónimos y son todos diferentes. No es conveniente repetir dos veces el mismo símbolo de estado con nombre en el mismo diagrama, porque puede inducir a confusión.

Estado anidado. Muestra un diagrama de estados de un estado compuesto, formado a su vez por estados subordinados anidados. El diagrama de estados se dibuja dentro de los límites del estado exterior. Las transiciones se pueden conectar directamente a los estados anidados, así como al límite del estado exterior. En una región disjunta, los subestados se dibujan directamente dentro del estado compuesto. En una región concurrente, el símbolo de estado se divide en subregiones mediante líneas discontinuas (esto es, se apilan).

Véase estado compuesto para más detalles y ejemplos.

Compartimento de transición interna. Contiene una lista de acciones o actividades internas que se ejecutan como respuesta a eventos recibidos mientras el objeto se encuentra en ese estado, sin cambiar de estado. Una transición interna tiene el formato

$$\text{nombre-evento}_{opt}\overline{(\text{argumentos}_{lista,opt})}\ \overline{[\text{condición-guarda}]_{opt}}\ \overline{/\text{expresión-acción}_{opt}}$$

Las expresiones de acción pueden emplear atributos y enlaces del objeto poseedor y parámetros de transiciones entrantes (si aparecen en todas las transiciones entrantes).

La lista de argumentos (incluyendo los paréntesis) se puede omitir si no hay parámetros. La condición de guarda (incluyendo los corchetes) y la expresión de acción (incluyendo la barra) son opcionales.

Las acciones de entrada y de salida tienen la misma forma pero emplean las palabras reservadas **entry** y **exit** que no se pueden emplear para nombres de eventos.

entry / expresión-acción

exit / expresión-acción

Las acciones de entrada y de salida pueden no tener argumentos o condiciones de guarda (porque son invocadas implícitamente y no explícitamente). Para obtener los parámetros de una acción de entrada, se puede acceder al evento actual desde la acción. Esto resulta especialmente útil para obtener los parámetros de creación desde un objeto nuevo.

El nombre de acción reservado **defer** indica un evento que se puede retrasar en un cierto estado y en sus subestados. La transición interna no debe tener condición de guarda ni acciones.

nombre-evento / defer

La palabra reservada **do** representa una expresión para una actividad no atómica.

do / expresión-actividad

Estado de referencia a submáquina. La invocación de una submáquina anidada se muestra mediante una cadena de la forma siguiente situada en el cuerpo del símbolo de estado:

include nombre-maquina(argumentos$_{lista,}$)$_{opt}$

Tanto las actividades **do** como las submáquinas describen cómputos no atómicos que normalmente se ejecutan hasta haber finalizado, pero que pueden ser absorbidos por un evento que dispare una transición.

Ejemplo

La Figura 13.83 muestra un estado con transiciones internas. La Figura 13.84 muestra la declaración y utilización de una submáquina.

Concurrencia dinámica. La concurrencia dinámica con un valor distinto de uno exactamente se muestra mediante una cadena de multiplicidad situada en la parte superior derecha del símbolo de estado. La cadena no debería incluirse para la ejecución secuencial normal. Esta notación está destinada principalmente a los diagramas de actividades y debe evitarse, en general, en los diagramas de estados.

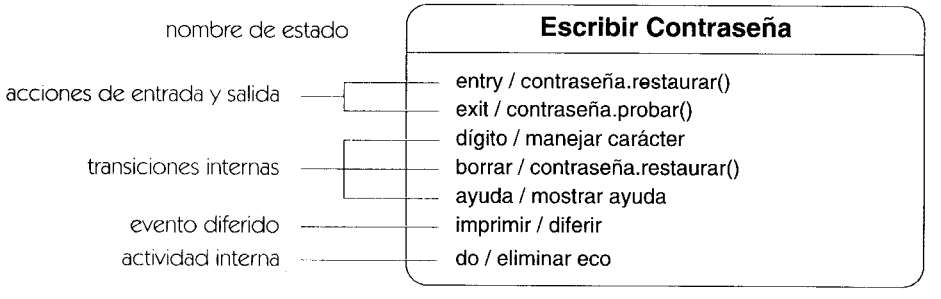


Figura 13.83 Transiciones internas, con acciones de entrada y salida y un evento diferido

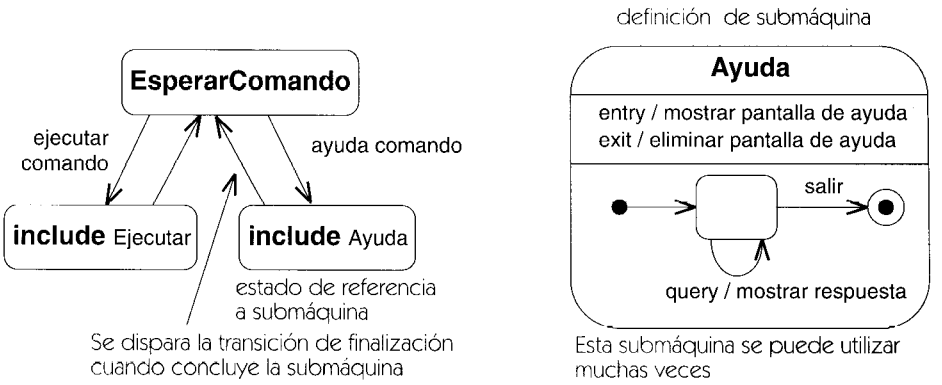


Figura 13.84 Submáquina

estado abreviado externo

Pseudoestado dentro de un estado de referencia a submáquina que identifica a un estado de la submáquina a la que se hace referencia.

Véase también transición abreviada, submáquina, estado de referencia a submáquina.

Semántica

Una transición a un estado de referencia a submáquina activa el estado inicial de la submáquina. Pero en algunas ocasiones se desea una transición a un estado diferente de la submáquina. Se sitúa el estado externo dentro de un estado de referencia a submáquina y mediante éste se identifica a un estado perteneciente a la submáquina. Se pueden conectar transiciones entre el estado abreviado externo y otros estados pertenecientes a la máquina de estados que lo contiene. Si se dispara una transición a un estado abreviado externo, el estado identificado dentro de la submáquina se activa. Si un estado de la submáquina está activado, entonces una transición procedente de un estado abreviado externo que lo identifique es candidata a ser disparada. No están permitidas las conexiones entre estados abreviados externos del mismo estado de referencia a submáquina.

Notación

Una transición desde o hacia un estado abreviado externo se dibuja como una transición abreviada hacia o desde el estado de referencia a submáquina —esto es, como una flecha que acaba o empieza en una barra corta dentro del símbolo de estado correspondiente al estado de referencia a submáquina—. La barra está etiquetada con un nombre, que tiene que coincidir con el estado de la submáquina a que se hace referencia.

La Figura 13.91 muestra una abreviatura dentro de un estado de referencia a subrutina. La Figura 13.92 muestra la definición de submáquina correspondiente.

estado destino

El estado de la máquina de estados al que se llega al dispararse la transición. Una vez que un objeto maneja un evento que da lugar a que se dispare una transición, el objeto queda en el

estado destino de la transición (o estados destino, si se trata de una transición compleja con múltiples estados destino). No es aplicable a una transición interna, que no da lugar a cambios de estado.

Véase transición.

estado compuesto

Un estado que consiste en subestados concurrentes (ortogonales) o subestados secuenciales (disjuntos).

Véase también transición compleja, estado simple, estado.

Semántica

Un estado compuesto puede descomponerse en subestados concurrentes usando relaciones AND, y en estados mutuamente disjuntos usando relaciones OR. Un estado se puede refinar solamente de una de estas dos maneras. Sus subestados se pueden refinar de cualquier manera. Si un estado compuesto secuencial está activo, uno de sus subestados disjuntos es activo. Si un estado compuesto concurrente está activo, todos sus subestados ortogonales están activos. El efecto final es un árbol Y-O. Cada máquina de estados tiene un estado de nivel superior, que es un estado compuesto.

Un sistema puede estar en múltiples estados simultáneamente. El conjunto de estados activos se llama configuración del estado activo. Si un estado anidado está activo, entonces todos los estados compuestos que lo contienen están activos. Si el objeto permite concurrencia, entonces puede estar activo más de un subestado concurrente.

Véase transición compleja para una discusión de la ejecución concurrente. La Figura 13.4 muestra un árbol Y-O.

Un objeto creado por primera vez comienza en su estado inicial, que el estado compuesto exterior debe tener. El evento que crea el objeto se puede utilizar para accionar una transición del estado inicial. Los argumentos del evento de creación están disponibles para esta transición inicial.

Un objeto que cambia a su estado final exterior es destruido y deja de existir.

Estructura

Un estado compuesto contiene un conjunto de subestados. Un estado compuesto es concurrente o secuencial.

Un estado compuesto secuencial debe tener, en la mayoría de los casos, un estado inicial y un estado final. También suele tener, en la mayoría de los casos, un estado superficial de historia y un estado profundo de historia.

Un estado compuesto concurrente no puede tener un estado inicial, un estado final, o estados de historia, pero cualquier estado compuesto secuencial anidado dentro de ellos puede tener tales pseudoestados.

Notación

Un estado compuesto es un estado con detalle subordinado. Tiene un compartimento para el nombre, un compartimento de transición interna, y un compartimento gráfico que contiene un diagrama anidado que muestra el detalle subordinado. Todos los compartimentos son opcionales. Por conveniencia y el aspecto, los compartimentos de texto (nombre y transiciones internas) se pueden contraer como lengüetas (tabs) dentro de la región gráfica, en vez de atravesarla horizontalmente.

Una expansión de un estado concurrente compuesto, en subestados concurrentes se muestra dividiendo el compartimento gráfico del estado usando líneas discontinuas para separarlo en subregiones. Cada subregión es un subestado concurrente, el cual puede tener un nombre opcional y debe contener un diagrama de estados anidado con subestados disjuntos. Los compartimentos de texto del estado completo se separan de los subestados concurrentes por una línea continua.

Una expansión de un estado en subestados disjuntos se muestra mediante un diagrama de estados anidado dentro de la región gráfica.

Un estado inicial se representa con un círculo negro pequeño. En una máquina de estados de nivel superior, la transición de un estado inicial se puede etiquetar con el evento que crea el objeto. Si no, debe estar sin etiqueta. Si está sin etiqueta, representa cualquier transición al estado que lo incluye. La transición inicial puede tener una acción. El estado inicial es un dispositivo notacional. Un objeto no puede estar en tal estado, pero debe pasar desde él a un estado real.

Un estado final se representa como una circunferencia que rodea un círculo sólido pequeño (un ojo de buey). Representa la terminación de la actividad en el estado que lo incluye, y acciona una transición en el estado que lo incluye, etiquetada por el evento implícito de terminación de actividad (representado generalmente como una transición sin etiqueta).

Ejemplo

La Figura 13.85 muestra un estado compuesto secuencial que contiene dos subestados disjuntos, un estado inicial, y un estado final. Cuando el estado compuesto llega a ser activo, el subestado **Comienzo** (destino del estado inicial) se activa primero.

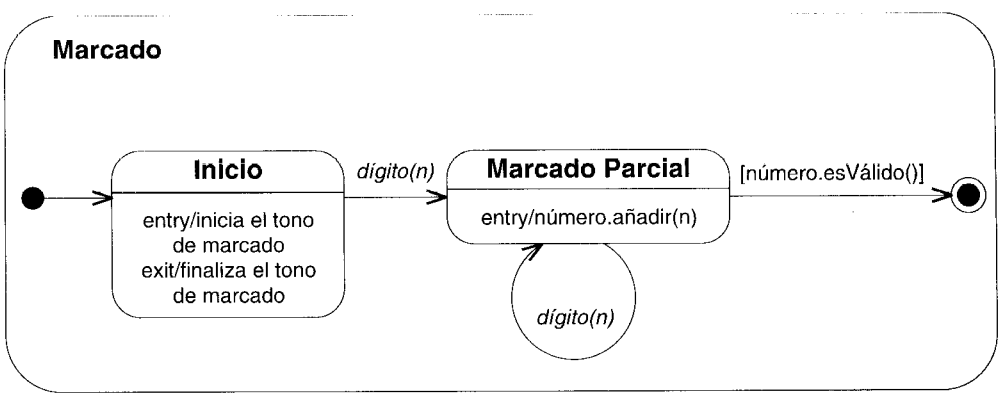


Figura 13.85 Estado compuesto secuencial

La Figura 13.86 muestra un estado compuesto concurrente que contiene tres subestados ortogonales. Cada subestado concurrente se descompone más a fondo en subestados secuenciales. Cuando el estado compuesto **Incompleto** llega a ser activo, los destinos de los estados iniciales se activan. Cuando las tres subregiones alcanzan el estado final, entonces se dispara la transición de la terminación del estado compuesto externo **Incompleto** y el estado **Aprobado** pasa a estar activo. Si ocurre el evento de **Suspenso** mientras el estado **Incompleto** está activo, entonces las tres subregiones concurrentes terminan y el estado **Suspenso** se convierte en activo.

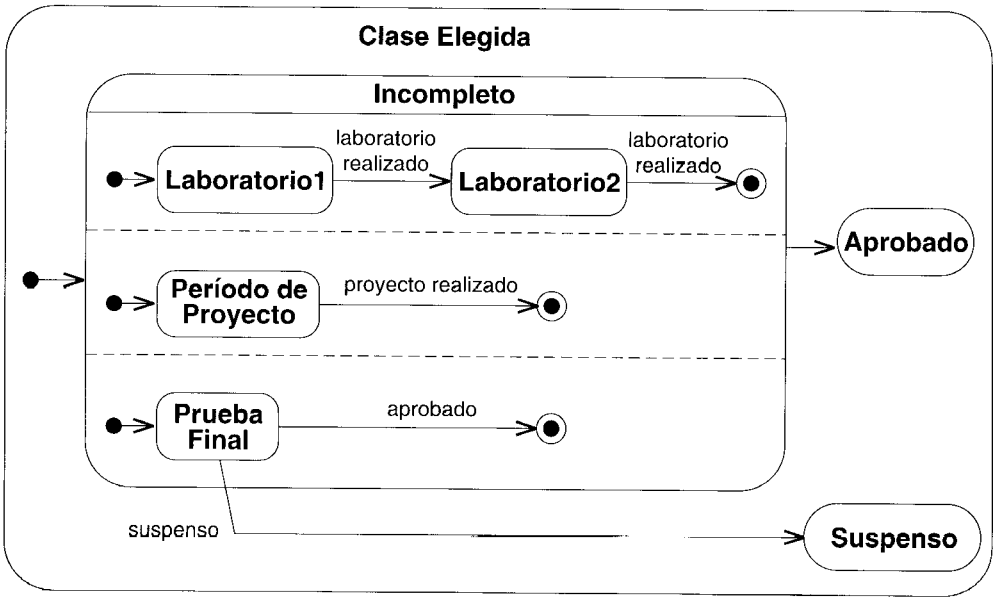


Figura 13.86 Estado concurrente compuesto

estado de acción

Estado cuyo propósito es ejecutar una acción y luego realizar una transición para cambiar a otro estado.

Véase también estado de actividad, transición de finalización.

Semántica

Un estado de acción es un estado cuyo propósito es ejecutar una acción de entrada, después de la cual realiza una transición de finalización hacia otro estado. El estado de acción es atómico, lo que significa que no puede ser terminado por una transición de un evento externo. Conceptualmente representa una computación que se completa de modo casi instantáneo y que no interactúa con otras acciones que se producen al mismo tiempo. En la práctica, puede requerir tiempo para completarse, pero será siempre menor que el tiempo de respuesta requerido por cualquier evento que pudiera ocurrir. No puede tener transiciones disparadas por eventos. Un estado de acción no tiene subestructura, acciones internas ni transiciones internas. Es una

especie de estado tonto útil para organizar máquinas de estados en estructuras lógicas. Normalmente cuenta con una transición de finalización saliente, aunque puede tener múltiples transiciones de finalización salientes si tienen condiciones de guarda (y por tanto, cada una de ellas representa una bifurcación).

Notación

No hay una notación especial para los estados de acción, por lo que se representan como un estado ordinario con una acción de entrada, aunque también pueden representarse mediante un estado de actividad.

estado de actividad

Estado que representa la ejecución de un cómputo puro que tiene una subestructura, normalmente la invocación a una operación, una sentencia interna a ella, o la realización de un procedimiento del mundo real. Un estado de actividad puede ser terminado externamente por un evento que fuerce una transición de salida del estado. Un estado de actividad no tiene por qué terminar por sí mismo, pues no existe ningún límite sobre cuánto tiempo puede estar activo.

Véase también actividad, transición de finalización.

Semántica

Un estado de actividad es un estado con un cómputo interno, y normalmente, al menos una transición de finalización que se dispare a la finalización de la actividad del estado (puede haber varias transiciones de este tipo si tienen condiciones de guarda). Los estados de actividad no deberían tener transiciones internas ni transiciones de salida basadas en eventos explícitos: utilice estados normales para modelar estas situaciones. El uso habitual de un estado de actividad es modelar un paso en la ejecución de un algoritmo (un procedimiento). Si todos los estados de un modelo son estados de actividad y las actividades concurrentes no acceden a los mismos valores, el cómputo es determinista aunque implique ejecución concurrente.

Un estado de actividad puede hacer referencia a una submáquina, normalmente a otro grafo de actividades. Esto es equivalente a expandir una copia de la red de la submáquina referenciada en el lugar que ocupa el estado de actividad. Es una subrutina de máquina de estados.

Un estado de actividad es un estado del proceso de ejecución de un procedimiento en lugar de un estado de un objeto normal.

Un estado de acción es un estado de actividad atómico, es decir, no puede ser interrumpido por una transición mientras permanece activo. Puede modelarse como un estado de actividad con una única acción de entrada.

Los estados de actividad se pueden utilizar en las máquinas de estados ordinarias, pero es más común utilizarlos en los grafos de actividades.

Las transiciones que salen de un estado de actividad normalmente no deberían incluir un evento disparador, ya que dichas transiciones se disparan implícitamente cuando termina la actividad del estado. Las transiciones pueden incluir condiciones de guarda y acciones. Asegúrese de que todas las posibles condiciones están cubiertas en las transiciones que salen de una actividad, pues de lo contrario el control podría quedar colgado. Si se evalúa a verdadero más de una condición de guarda, no estará definida cuál de ellas debe escogerse. Puede ser una elección no determinista, o imponerse una regla para cambiar la semántica.

En otras situaciones utilice un estado normal.

Notación

Un estado de actividad se representa como se muestra en la Figura 13.87, con una figura en forma de píldora (un símbolo con líneas horizontales arriba y abajo y lados convexos), dentro de la cual se escribe la expresión de actividad. La expresión de actividad no tiene por qué ser única en el diagrama.

Figura 13.87 Actividades

Discusión

Los estados de acción están pensados para operaciones cortas de contabilidad y los estados de actividad para cómputos de cualquier duración o complejidad. Una acción podría bloquear el sistema, por lo que debe ser breve, mientras que una actividad puede ser finalizada externamente, por lo que el sistema no necesita esperar a que finalice si ocurre algo urgente. La semántica de UML no impide acciones prolongadas, pero un generador de código debería asumir que una acción tiene que completarse de una sola vez, mientras que una actividad puede ser interrumpida por otras acciones.

estado de flujo de objeto

Se trata de un estado que representa la existencia de un objeto o de una clase particular en cierto punto de una computación, tal como una vista de interacción o un grafo de actividades.

Véase también flujo de objeto, clase-en-un- estado.

Semántica

Tanto los grafos de actividades como las vistas de interacción representan el flujo de objeto entre operaciones de objetos destino por medio de mensajes, pero los mensajes no muestran el

flujo de los objetos que son los argumentos de esas operaciones. Este tipo de flujo de información se puede representar en modelos de comportamiento empleando estados de flujo de control.

Un estado de flujo de objeto representa un objeto de una clase que existe en un cierto punto dentro de una computación, tal como un grafo de actividades o una vista de interacción. El objeto puede ser la salida de una actividad y la entrada de otras muchas actividades. En un grafo de actividades, puede ser el destino de una transición (con frecuencia se trata de una bifurcación y la otra rama es la ruta principal de control); puede ser el origen de una transición de finalización a una actividad. Cuando se dispara la transición precedente, el estado de flujo de objeto se activa. Esto representa la creación de un objeto de esa clase. Para mostrar el paso de un objeto a un estado, mejor que la creación de un nuevo objeto, se puede declarar un estado de flujo de objeto como una clase en cierto estado, una clase-en-un-estado.

El estado de flujo de objeto tiene que coincidir con el tipo del resultado o parámetro que representa. Si es la salida de una operación, tiene que satisfacer el tipo del resultado. Si es la entrada de una operación, tiene que coincidir con el tipo de un parámetro.

Si el estado de flujo de objeto va seguido por una transición de finalización a una actividad, entonces se puede realizar la actividad en cuanto esté disponible el valor del objeto. No se necesita ninguna entrada de control adicional. En otras palabras, la creación de datos en la forma correcta es el disparador para llevar a cabo la actividad.

Para mostrar que una actividad requiere tanto una ruta de control como la presencia de un valor, la acción anterior de la ruta de control y un estado de flujo de objeto para ese valor pueden llevar a una transición compleja. Se efectúa la actividad cuando están preparadas todas las transiciones de entrada. Las rutas múltiples hacia una transición indican sincronización.

Los estados de flujo de objeto suelen ser útiles para documentar relaciones de entrada-salida a efectos de comprensión humana, más que para especificar con precisión una computación. La información mostrada por los estados de flujo de objeto ya está disponible.

La producción de un evento por parte de una actividad en un grafo de actividades se puede modelar como un estado de flujo de objeto cuyo clasificador es una señal. Se puede emplear el estereotipo «signal». El estado de flujo de objeto es una salida de la actividad. Si la actividad produce múltiples eventos, entonces los estados de control de flujo son los destinos de una bifurcación.

Notación

Un objeto de una clase en un cierto estado se muestra en un diagrama de actividades mediante un rectángulo que contiene el nombre de la clase subrayado seguido por el nombre del estado entre corchetes cuadrados

nombre de la clase [nombre del estado]

Un ejemplo podría ser

Pedido [Efectuado]

Un símbolo de flujo de objetos representa la existencia del objeto en el estado del procedimiento en sí y no simplemente el propio objeto, como datos. El símbolo de flujo de objetos (que representa un estado) puede aparecer como destino de una flecha de transición y como origen de múltiples flechas de transición. Para distinguirlas de las transiciones ordinarias en un diagrama de actividades, se dibujan como flechas discontinuas en lugar de emplear flechas continuas. Representan el flujo de objetos.

Ejemplo

La Figura 13.88 muestra estados de control de flujo en un diagrama de actividades. Se crea un estado de flujo de objeto cuando finaliza una operación. Por ejemplo, **Pedido[Efectuado]** se crea cuando termina **SolicitarServicio**. Como esta actividad va seguida por otra actividad, el estado

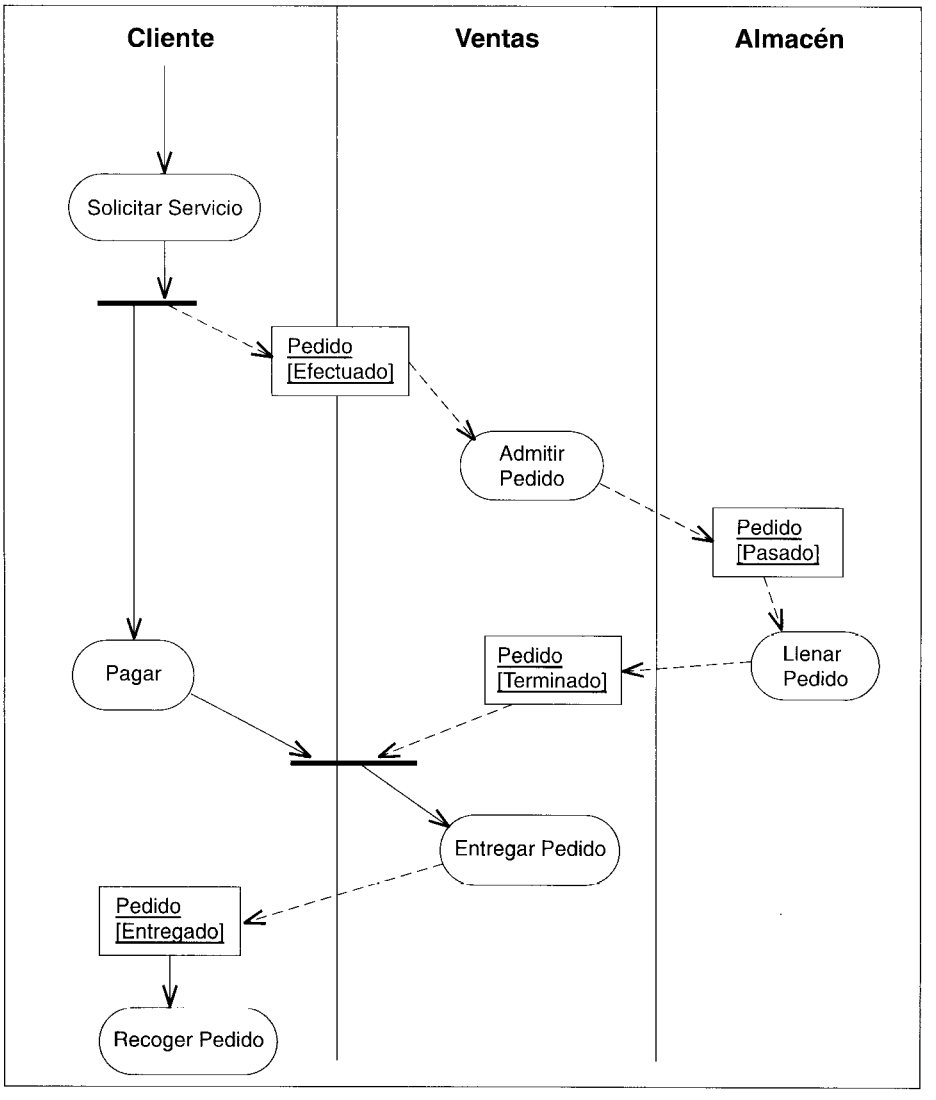


Figura 13.88 Estados de flujo de objetos en un diagrama de actividades

de flujo de objeto **Pedido[Efectuado]** es una salida de un símbolo de bifurcación. El estado **Pedido[Pasado]**, por otra parte, es el resultado de finalizar la actividad **AdmitirPedido**, que no posee otras actividades descendientes.

La Figura 13.89 muestra una parte de un diagrama de actividades dedicado a la construcción de una casa. Una vez construida la estructura, el carpintero ya puede trabajar en el tejado y la casa está preparada para instalar la fontanería. Estos eventos se modelan como estados de control de flujo de señales —**Carpintero libre** y **Estructura**—. Como resultado de estos eventos, se puede construir el tejado y se pueden instalar las tuberías. Por tanto, los estados de control de flujo se muestran como entradas de las actividades. En el modelo, la producción de un suceso por haber terminado una actividad y su utilización como disparador de la siguiente actividad están implícitos en la conexión de las actividades. La necesidad de un evento manifiesto se ha omitido. Por tanto, la aparición de las señales como estados de control de flujo es por estructura de información, más que de implementación.

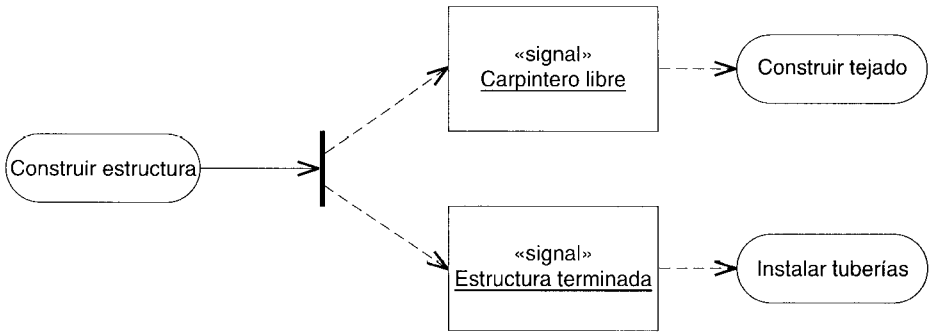


Figura 13.89 Producción de señales en un diagrama de actividades

Discusión

Un estado de flujo de objeto representa el punto de vista de flujo de datos de una computación. A diferencia del flujo de datos tradicional, sin embargo, existe en un punto definido dentro de un modelo de control de flujo (una máquina de estados o un grafo de actividades) y no dentro de un modelo de flujo de datos. Esto lo pone directamente en un marco de trabajo orientado a objetos. La orientación a objetos une los puntos de vista de la estructura de datos, el flujo de objeto y el flujo de datos en un único modelo.

estado de historia

Un pseudoestado que indica que el estado compuesto que lo contiene recuerda su subestado activo previo una vez que concluye.

Véase también estado compuesto, pseudoestado, máquina de estados, transición.

Semántica

El estado de historia permite que un estado compuesto secuencial recuerde el último subestado que haya estado activado en él antes de una transición procedente del estado compuesto. Una

transición al estado de historia da lugar a que vuelva a activarse de nuevo el subestado anteriormente activo. Se llevan a cabo todas las acciones de entrada necesarias. Un estado de historia puede tener transiciones entrantes que procedan de fuera del estado compuesto o del estado inicial. Un estado de historia puede tener una sola transición saliente sin rotular. Esta transición indica el estado de historia almacenado inicialmente. Se hace uso de él si una transición llega al estado de historia cuando no está presente ningún estado almacenado. El estado de historia no puede tener transiciones entrantes que procedan de otros estados pertenecientes al estado compuesto porque ya está activo.

El estado de historia puede recordar una *historia próxima* o *una historia profunda*. Un estado de historia próxima recuerda y reactiva un estado con el mismo nivel de anidamiento que el estado de historia en sí. Si una transición procedente de un subestado ha salido directamente del estado compuesto, se activa el subestado que lo contiene y que ocupa el nivel más elevado dentro del estado compuesto. Un estado de historia profunda recuerda un estado que puede haber estado anidado a alguna profundidad dentro del estado compuesto. Para recordar un estado profundo, la transición debe haber sacado directamente del estado profundo al estado compuesto. Si una transición pasa de un estado profundo a otro más superficial, que después hace una transición y sale del estado compuesto, entonces el estado que se recordará será el menos profundo. Una transición a un estado de historia profunda restaura el estado anteriormente activo sea cual fuere la profundidad. En el proceso, se ejecutan las acciones de entrada si están presentes en estados internos que contengan el estado recordado. Un estado compuesto puede tener a la vez un estado de historia próximo y un estado de historia profundo. Tiene que haber una transición entrante que esté conectada a uno o al otro.

Si un estado compuesto alcanza su estado final, entonces pierde la historia que pudiera tener almacenada y se comporta como si no se hubiera entrado en él por primera vez.

Notación

Un estado de historia próxima se representa mediante un pequeño círculo que contiene la letra **H**, tal como se muestra en la Figura 13.90. Un estado de historia profunda se representa mediante un círculo que contiene **H***.

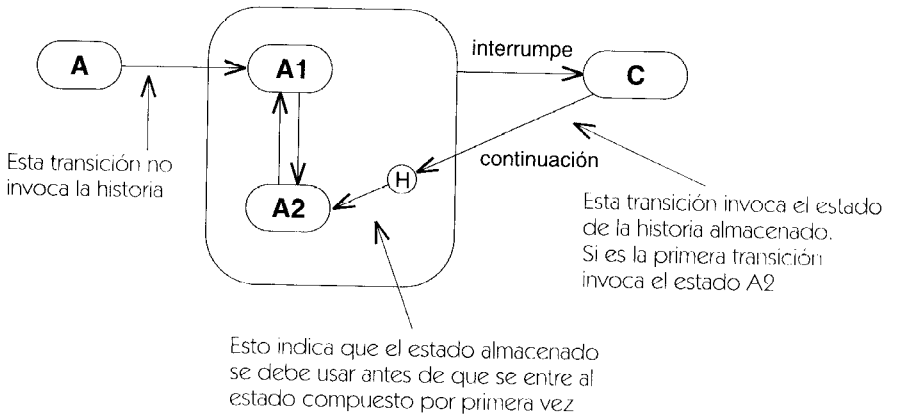


Figura 13.90 Estado de historia

estado de origen

Estado del que parte una transición dentro de la máquina de estados. La transición se le aplica al estado de origen. Si un cierto objeto está en el estado o un estado está anidado en su interior, entonces la transición es un candidato para el disparo.

estado de referencia a submáquina

Estado que hace referencia a una submáquina de estados. Una copia de esta submáquina forma parte implícita de la máquina de estados que la encierra, en el lugar del estado de referencia a submáquina. Puede contener estados abreviados externos, que identifican a estados de la submáquina.

Véase también estado, máquina de estados, estado abreviado externo.

Semántica

Los estados de referencia a submáquina equivalen a insertar una copia de la submáquina en lugar del estado de referencia.

Notación

Los estados de referencia a submáquina se dibujan como un símbolo de estado con una etiqueta de la forma

include nombre-submáquina

Se pueden dibujar flechas de transición hacia estados externos situados dentro del estado de referencia a submáquina. Se dibujan como transiciones abreviadas —esto es, flechas que terminan en una barra transversal—. La barra transversal se etiqueta con el nombre del estado dentro de la submáquina a que se hace referencia.

Ejemplo

La Figura 13.91 muestra parte de una máquina de estados que contiene un estado de referencia a submáquina. La máquina de estados que lo contiene vende entradas a aquellos clientes que tienen cuentas. Tiene que identificar al cliente como parte de su tarea. La identificación del cliente es un requisito de otra máquina de estados, así que se ha definido como otra máquina de estados por separado. La Figura 13.92 muestra la definición de la máquina de estados **Identificar**, que es empleada como submáquina por otras máquinas de estados. La entrada normal en la submáquina se encarga de leer la tarjeta del cliente, pero existe un estado de entrada específico que ofrece la introducción manual del nombre del cliente por parte del encargado de la taquilla. Si tiene éxito el proceso de identificación, la submáquina concluye en su estado final. En caso contrario, pasa al estado **Incorrecto**.

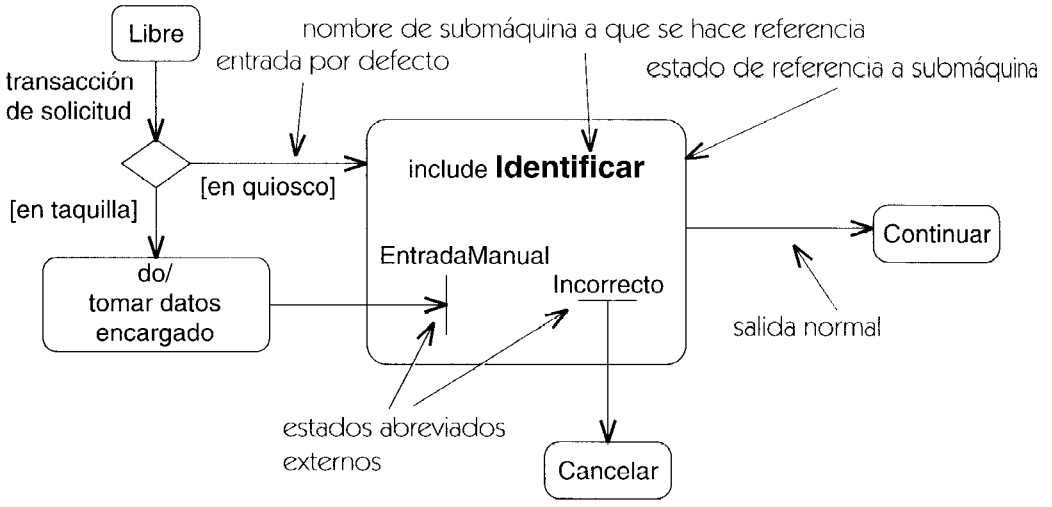


Figura 13.91 Estado de referencia a submáquina

En la Figura 13.91, la referencia a la submáquina se muestra mediante un icono de estado con la palabra reservada **include** y el nombre de la submáquina. La entrada normal a la submáquina se muestra mediante una flecha que llega a su periferia. Esta transición activa el estado inicial de la submáquina. La salida normal se muestra mediante una transición de finalización que sale de la periferia. Esta transición se dispara si la submáquina finaliza normalmente.

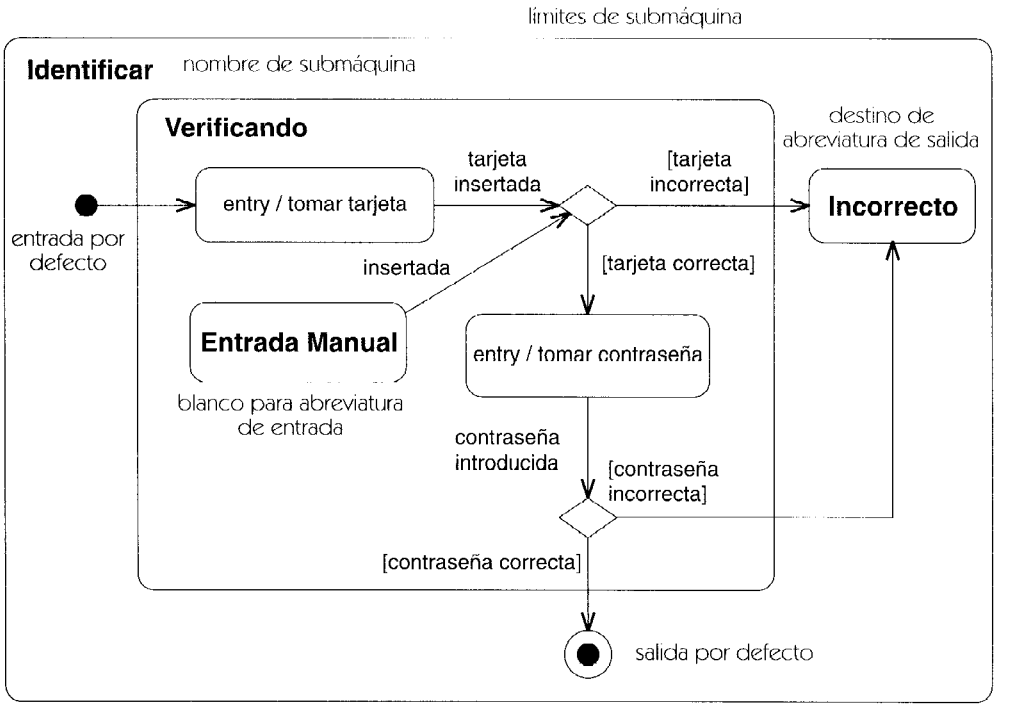


Figura 13.92 Definición de submáquina

La entrada al estado explícito **EntradaManual** se muestra mediante una transición a una abreviatura situada dentro del símbolo de referencia a la submáquina. La abreviatura está etiquetada con el nombre del estado destino de la submáquina. De forma análoga, la salida del estado explícito **Incorrecto** se muestra mediante una transición de finalización procedente de una abreviatura. Las transiciones a abreviaturas pueden tener o no un evento disparador.

estado de sincronización

Estado especial que permite la sincronización del control entre dos regiones concurrentes de una máquina de estados.

Véase también transición compleja, estado compuesto, bifurcación, unión, máquina de estados, transición.

Semántica

Un estado compuesto puede constar de varias regiones concurrentes, cada una de las cuales posee su propio conjunto secuencial de estados. Cuando se entra en un estado compuesto concurrente, todas las regiones concurrentes pasan a estar activas. En cada región concurrente hay un hilo de control y cada uno de ellos se ejecuta independientemente de los demás (esto es lo que significa la concurrencia). En algunas ocasiones, sin embargo, es preciso sincronizar el control entre regiones concurrentes. Un posible enfoque consiste en que una transición de una región tenga una condición de guarda que dependa del estado de otra región. Esto puede ser útil para la exclusión mutua, incluyendo los recursos compartidos, pero no captura aquellas situaciones en que la actividad de una región tenga consecuencias subsiguientes en otra (aun cuando la primera región pueda seguir adelante después). Para capturar esta última situación pueden utilizarse estados de sincronización.

Un estado de sincronización es un estado especial que conecta dos regiones concurrentes. Las regiones pueden ser pares, esto es, dos regiones concurrentes que pertenezcan a un mismo estado compuesto, o bien pueden estar anidadas entre pares, a cualquier profundidad. Estas regiones no pueden estar relacionadas secuencialmente.

Hay una transición que conecta la salida de una bifurcación situada en una región con la entrada del estado de sincronización y otra transición que conecta la salida del estado de sincronización con una unión de la otra región. En otras palabras, un estado de sincronización es una zona intermedia que conecta indirectamente una bifurcación de una región con una unión de otra región. La bifurcación y la unión tienen que tener ambas un estado de entrada y un estado de salida dentro de sus propias regiones.

El estado de sincronización recuerda el disparo de una transición en la primera región hasta que se dispara la transición de unión en la segunda región. Si las condiciones de la transición de unión se satisfacen antes de que esté activado el estado de sincronización, entonces la unión deberá esperar a que se dispare la transición de la primera región. En otras palabras, representa una transición de la segunda región que no puede dispararse mientras no se haya disparado una transición en la primera región. En caso contrario, se bloquea hasta que se dispara la primera transición. Obsérvese que dado que toda bifurcación y toda unión tienen que tener una entrada y una salida dentro de su propia región, los estados de sincronización no cambian

el comportamiento secuencial fundamental de cada una de las regiones concurrentes, ni tampoco alteran las reglas de anidamiento para formar estados compuestos (salvo que el estado de sincronización y sus arcos no pertenecen a ninguna de las regiones concurrentes sino más bien al estado compuesto por ambas regiones).

Las situaciones del tipo productor-consumidor son ejemplos típicos de los estados de sincronización. El productor dispara una transición que activa el estado de sincronización; el consumidor posee una transición que requiere el estado de sincronización para poder dispararse.

Si la transición de entrada al estado de sincronización forma parte de un bucle, es posible que la primera región puede adelantarse a la segunda. En otras palabras, el estado de sincronización puede tener más de un *símbolo* (para usar el término de las redes de Petri). Por tanto, un estado de sincronización, a diferencia de un estado normal, representa un contador o cola (esto último si la información fluye entre las regiones, si el estado de sincronización es un estado de flujo de objeto, por ejemplo). Por defecto, los estados de sincronización pueden contener un número ilimitado de símbolos, pero el creador del modelo puede especificar un límite superior para el número que puede contener el estado de sincronización. Si se excede la capacidad del estado de sincronización, se producirá un error en tiempo de ejecución. Los más frecuente es que el límite sea ilimitado o uno, entonces esto último representa un simple interruptor. En general, un límite de uno sólo se utilizaría si fuera posible garantizar que no se va a producir un desbordamiento. Esto es responsabilidad del creador del modelo.

Si existen símbolos en un estado de sincronización cuando finaliza el estado compuesto que lo contiene, se destruyen. El estado de sincronización está vacío siempre que se entra en la región compuesta.

Puede haber más de un arco de entrada que penetre en un estado de sincronización, pero todos ellos tienen que provenir de bifurcaciones pertenecientes a la misma región secuencial. Análogamente, puede haber más de un arco de salida que salga de un estado de sincronización para llegar a uniones de la sección secuencial. Como todas las regiones son secuenciales, no hay peligro de conflictos entre múltiples arcos.

Un estado de sincronización puede ser un estado de flujo de objeto. En tal caso, representa una cola de valores que van pasando de una región a otra.

Notación

Los estados de sincronización se muestran como un circulito con un único límite superior en su interior, bien sea un entero o un asterisco (*) que denota un valor ilimitado. Hay un flecha de transición que parte desde un símbolo de barra de sincronización (una barra gruesa) situado en el estado de sincronización y otra flecha de transición que parte del estado de sincronización y llega hasta un símbolo de barra de sincronización que se encuentra en otra región (Figura 13.93).

Preferiblemente, el estado de sincronización se dibujará en el límite entre dos regiones, pero esto no siempre es posible (las regiones pueden no ser adyacentes) y en todo caso la topología de la conexión no es ambigua.

Dentro de un diagrama de actividades, cada arco de transición representa implícitamente un estado. Por tanto, se puede trazar un flecha desde la salida de una bifurcación hasta la entrada de una unión sin mostrar explícitamente el estado de sincronización (pero es necesario que el estado de sincronización muestre un límite explícito).

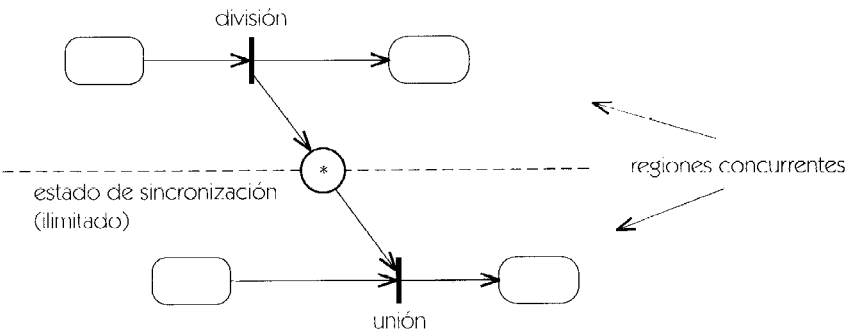


Figura 13.93 Configuración de un estado de sincronización

Ejemplo

La Figura 13.94 muestra un diagrama de estados correspondiente a una situación de compra de entradas. La emisión de entradas y el cobro de las mismas se producen concurrentemente, salvo que sea preciso seleccionar los asientos antes de que se puedan calcular y enviar los costes. Esta sincronización se muestra mediante la inserción de un estado de sincronización entre

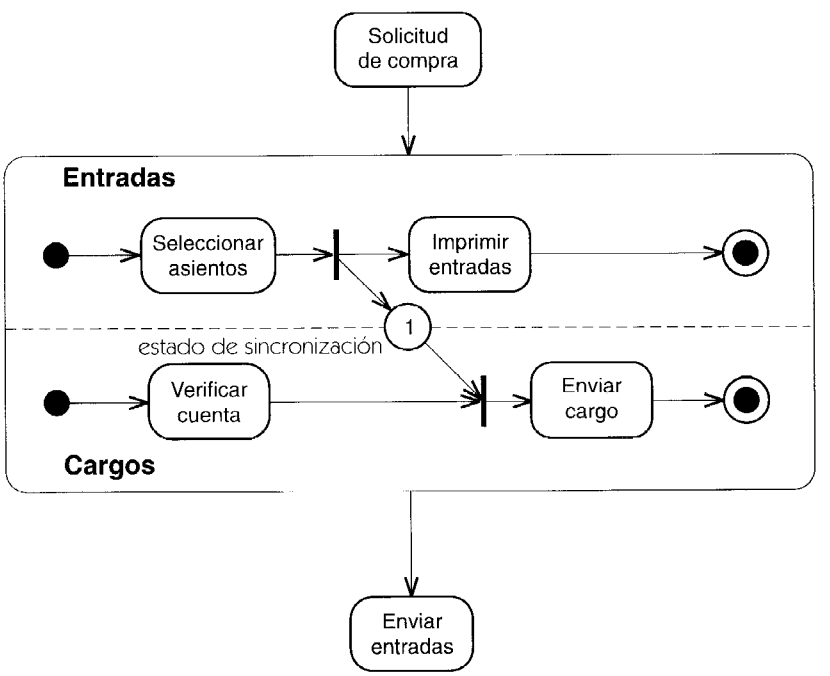


Figura 13.94 Estado de sincronización para pedido único

Seleccionar asientos y **Enviar cargo**. Existe una bifurcación después de **Seleccionar asientos**, porque va seguida tanto por **Imprimir entradas** como por el estado de sincronización. **Imprimir cargo** no tiene que esperar a la sincronización. Existe una unión antes de **Enviar cargo**, porque tiene que esperar tanto a **Verificar cuenta** como al estado de sincronización. Cuando han concluido tanto **Imprimir entradas** como **Enviar cargo**, el estado compuesto concluye y se ejecuta **Enviar entradas**.

El estado de sincronización tiene un límite superior de uno. No es necesario tener un límite mayor porque sólo hay una sincronización por cada ejecución del estado compuesto.

La Figura 13.95 muestra una versión de procesamiento por lotes de este proceso de llenado de pedidos. En esta variación, se pueden hacer pedidos sin estar conectado al sistema. Los pedidos los hace un servidor y los cargos los hace otro. No se puede hacer un cargo antes de haber llenado el pedido, pero se pueden hacer pedidos antes de hacerse los cargos así que el estado de sincronización tiene un límite superior ilimitado. Se trata de la situación clásica productor-consumidor.

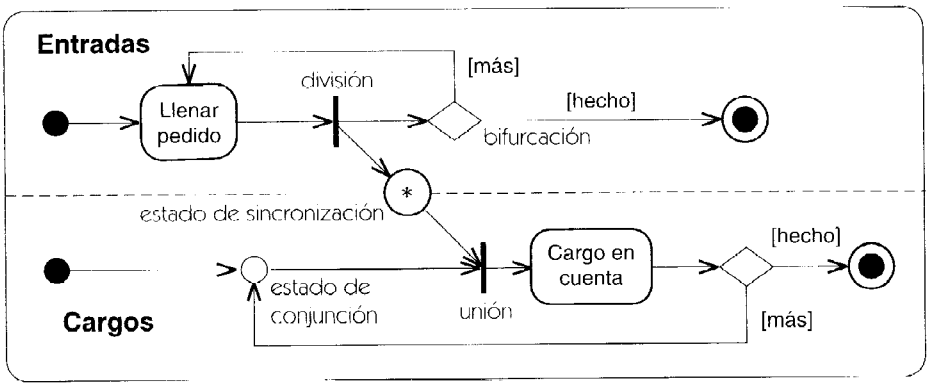


Figura 13.95 Situación productor-consumidor con estado de sincronización ilimitado

Discusión

Los estados de sincronización proporcionan la capacidad de modelar situaciones productor-consumidor con un coste mínimo y con mayor seguridad que otras estructuras de concurrencia más generales, porque cada región concurrente mantiene un único hilo de control en todo momento.

No hay peligro de desbordamiento (si el estado de sincronización es ilimitado) porque el estado de sincronización se vacía siempre que se sale del supraestado. Si se utilizan estados de sincronización dentro de bucles que contengan ramificaciones, existe el peligro de un bloqueo: esto sucederá si una región ha terminado pero la otra espera aun símbolo de sincronización que no va a llegar nunca. También es posible un interbloqueo si ambas regiones se encuentran en una ramificación que espera un símbolo procedente de otra región. No hay forma de evitar por completo estas situaciones en sistemas concurrentes que permitan la toma de decisiones. Aunque no haya estados de sincronización, no se puede garantizar la finalización como consecuencia del *problema de la detención de la ejecución*.

estado de unión o conjunción

Se trata de un pseudoestado que forma parte de una sola transición global en una máquina de estados. No interrumpe un paso simple del tipo "ejecutar hasta finalizar", en la ejecución de una transición.

Véase también bifurcación, reunificación.

Semántica

Una transición de una máquina de estados puede cruzar los límites de varios estados compuestos desde el estado origen hasta el estado destino. Al ejecutar una de estas transiciones, pueden invocarse una o más acciones de entrada o de salida. En algunas ocasiones es necesario entrelazar una o más acciones de la transición con las acciones de entrada y de salida que están asociadas a los estados anidados. Eso no puede hacerse con una sola transición, que tiene asociada una sola acción.

También es conveniente permitir que varios disparadores posean un mismo resultado o permitir que un único disparador tenga varios resultados posibles con diferentes condiciones de guarda.

En este sentido, un estado de unión es un pseudoestado que hace posible construir una única transición global a partir de una serie de fragmentos de transición.

Un estado de unión puede poseer uno o más segmentos de transición entrantes y uno o más segmentos de transición salientes. No puede poseer una actividad interna, ni una submáquina, ni ninguna transición saliente con eventos disparadores. Se trata de un estado neutro que sirve para estructurar las transiciones y no de un estado que pueda estar activo durante un tiempo finito.

Los estados de unión se emplean para estructurar una transición procedente de varios segmentos. Sólo el primer segmento de una cadena de estados de unión podrá tener un evento disparador, pero todos ellos pueden tener condiciones de guarda. Los segmentos subsiguientes deben carecer de disparadores. La condición de guarda efectiva es la conjunción de todas las condiciones de guarda originales. No se dispara la transición mientras no se cumpla todo el conjunto completo de condiciones. En otras palabras, la máquina de estados no puede permanecer en el estado de unión.

Si hay múltiples transiciones que entran en un único estado de unión, cada una puede tener un disparador diferente, o bien puede carecer de él. Cada una de las rutas que atraviesan un conjunto de estados de unión representa una transición diferente.

Una transición saliente puede tener una condición de guarda. Si existen múltiples transiciones salientes, cada una de ellas debe poseer una condición de guarda diferente. Esto es una bifurcación.

Una transición saliente puede tener asociada una acción. (El estado de unión puede poseer una acción interna, pero esto equivale a asociar una acción a la transición saliente, que es el enfoque preferible.) La acción se ejecuta siempre y cuando se cumplan todas las condiciones de

guarda, incluyendo aquellas que se encuentren en los segmentos subsiguientes. Una transición no puede efectuar un "disparo parcial" de tal modo que se detenga el estado de unión. Tiene que alcanzar un estado normal.

Cuando se dispara una transición entrante, la transición saliente se dispara inmediatamente. Entonces se ejecuta cualquier acción que pudiera estar asociada. La ejecución de la transición entrante y de la transición saliente forma parte de un único paso atómico (un paso de los que se ejecutan hasta finalizar), esto es, no puede ser interrumpida por un evento ni por otras acciones.

Notación

Los estados de unión se muestran en la máquina de estados mediante un circulito. No tienen nombre. Pueden tener flechas de transición entrantes y salientes.

Ejemplo

La Figura 13.96 muestra dos transiciones completas del estado **S** al estado **T** —una transición de un solo segmento disparada por el evento **f** y una transición de múltiples segmentos disparada por el evento **e**, que se ha estructurado empleando dos estados de unión—. Las anotaciones muestran el entrelazado de las acciones de transición con las acciones de entrada y salida.

Observe que la situación del rótulo de acción en la línea de transición no posee significado propio. Si la acción **d** hubiese estado situada dentro del estado **X**, se habría ejecutado en todo caso después de salir del estado **X** y antes de entrar en el estado **Y**. Por tanto, debe dibujarse en la posición más externa de la transición.

Para otros ejemplos, véanse la Figura 13.95 y la Figura 13.180.

Véanse también los iconos de control para otros símbolos abreviados que pueden incluirse en los diagramas de estados y en los diagramas de actividades

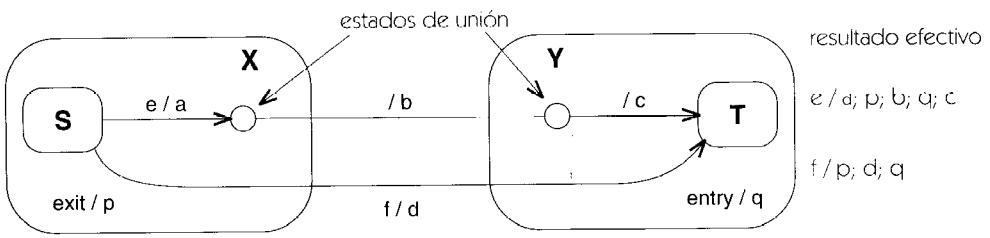


Figura 13.96 Estados de unión

estado final

Un estado especial dentro de un estado compuesto que, cuando está activo, indica que la ejecución del estado compuesto ha terminado. Cuando un estado compuesto alcanza su estado fi-

nal, se dispara una transición de terminación, que sale del estado compuesto y se puede activar si se satisface su condición de guarda.

Véase también actividad, transición de finalización, destrucción.

Semántica

Para promover la encapsulación, es deseable separar la vista exterior de un estado compuesto de los detalles internos tanto como sea posible. Desde el exterior, el estado se ve como entidad opaca con una estructura interna que se oculta. Desde el punto de vista exterior, las transiciones van a y desde el propio estado. Desde un punto de vista interior, conectan con los subestados dentro del estado. Los estados inicial y final son mecanismos para permitir la encapsulación de estados.

Un estado final es un estado especial que indica que la actividad del estado compuesto está completa y que una transición de finalización que sale del estado compuesto está activada. Un estado final no es un pseudoestado. Un estado final puede ser activado por un período de tiempo, a diferencia de un estado inicial que transita inmediatamente a su sucesor. El control puede permanecer dentro de un estado final mientras se espera la terminación de otros subestados concurrentes del estado compuesto —es decir, mientras espera para sincronizarse con múltiples hilos de control para unirse—. No obstante, las transiciones salientes disparadas por eventos no están permitidas en un estado final (de lo contrario, sería justo un estado normal). Un estado final puede tener cualquier número de transiciones entrantes dentro del estado compuesto que incluye, pero ninguna transición hacia afuera del estado que incluye. Las transiciones entrantes son transiciones normales y pueden tener el conjunto completo de disparadores, condiciones de guarda, y acciones.

Si un objeto alcanza su estado final de nivel superior, la máquina de estados termina y se destruye el objeto.

Notación

Un estado final se representa como un icono de un ojo de buey —es decir, un pequeño disco negro relleno rodeado por una pequeña circunferencia—. Se coloca dentro del estado compuesto que lo contiene, cuya terminación representa (Figura 13.97). Solamente puede ocurrir un estado final (directamente) dentro de un estado compuesto. Los estados finales adicionales pueden

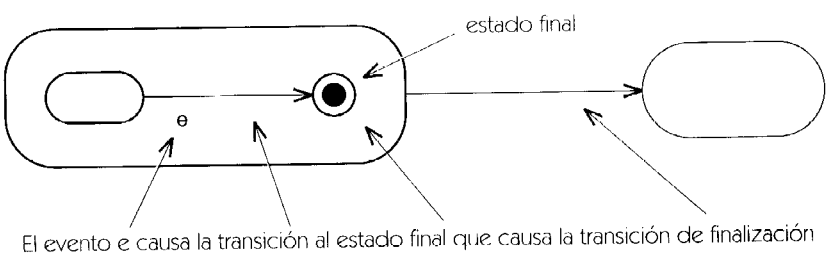


Figura 13.97 Estado final

ocurrir, sin embargo, dentro de estados compuestos anidados. Por conveniencia, el símbolo final del estado se puede repetir dentro de un estado, pero cada copia representa el mismo estado final.

estado inicial

Se trata de un pseudoestado que indica el punto de partida por defecto para una transición cuyo destino es el límite de un estado compuesto.

Véase también estado compuesto, creación, acción de entrada, iniciación, estado de unión.

Semántica

Para promover el encapsulamiento, resulta deseable separar tanto como sea posible la vista externa de un estado compuesto y los detalles internos. Desde el exterior, se ve el estado como una entidad opaca con una estructura interna oculta. Desde el punto de vista externo, las transiciones van y vienen a ese estado. Desde el punto de vista interno, están conectadas a subestados propios de ese estado.

El nombre de estado inicial alude a un estado neutro (un pseudoestado) que representa un punto de conexión para una transición entrante que llega a los límites del estado compuesto. No se trata de un estado real; el control no puede permanecer en su interior. Se trata, más bien, de una forma sintáctica de indicar dónde debería ir el control. Todo estado inicial debe tener una transición saliente sin disparador (una transición sin evento disparador, que por tanto se activará automáticamente en cuanto se entre en el estado inicial). La transición de finalización establece una conexión con un estado real del estado compuesto. La transición de finalización puede contener una acción. La acción se ejecuta cuando se entra en el estado después de ejecutar la acción de entrada (si existe) del estado compuesto. Esto permite asociar una acción a la entrada por defecto, además de la acción de entrada (que se ejecuta para todas las entradas, tanto si son por defecto como si no). Esta acción puede acceder al evento actual implícito, esto es, al evento que haya disparado el primer segmento de la transición que en última instancia ha dado lugar a la transición al estado inicial.

Un estado inicial no puede poseer una transición saliente con un evento disparador. Una transición entrante equivale a una transición entrante al estado compuesto envolvente y debe evitarse. Estas transiciones deben conectarse al compuesto.

Con gran frecuencia, la transición al estado inicial no tiene guarda. En tal caso, tiene que ser la única transición procedente del estado inicial. Se puede proporcionar un conjunto de transiciones salientes con condiciones de guarda, pero estas condiciones de guarda deben abarcar absolutamente todos los casos posibles (o bien, más sencillamente, una de ellas puede tener la condición de guarda **else**). Lo importante es que el control debe salir inmediatamente del estado inicial. No se trata de un estado real, así que debe dispararse alguna transición.

El estado inicial del estado de nivel más alto de una clase representa la creación de una nueva instancia de la clase. Cuando se sigue la transición saliente, el evento actual implícito es el evento de creación del objeto y posee los argumentos que se le pasan por parte de la operación del constructor. Estos valores están disponibles dentro de las acciones de la transición saliente.

Creación del objeto

El estado inicial del estado compuesto de nivel más elevado de una clase es ligeramente distinto. Puede poseer un disparador que tenga el estereotipo «**create**», junto con un evento disparador con un nombre y unos parámetros. Quizá haya múltiples transiciones de esta clase con distintos disparadores. La signatura de cada disparador debe coincidir con una operación de creación de la clase. Cuando se instancia un nuevo objeto de la clase, se dispara la transición que corresponde a su operación de creación; esa transición recibe los argumentos de la llamada a la operación de creación.

Notación

El estado inicial se representa mediante un circulito negro situado dentro del símbolo de su estado compuesto. Se pueden conectar a este símbolo las flechas de transición salientes. Sólo puede haber un estado inicial (directamente) dentro de cada estado compuesto. Sin embargo, puede haber estados iniciales adicionales dentro de estados compuestos anidados.

Ejemplo

En la Figura 13.98 comenzamos en el estado **X**. Cuando se produce un evento, se dispara la transición y se ejecuta una acción. La transición pasa al estado **Y**. Se ejecuta la acción de entrada **b**, y el estado inicial queda activado. La transición saliente se dispara inmediatamente, ejecutando la acción **c** y pasando al estado **Z**.

Por otra parte, si el evento **f** se produce cuando el sistema se encuentra en el estado **X**, entonces se dispara la otra transición y se ejecuta la acción d. Esta transición pasa directamente al estado **Z**. El estado inicial no está implicado. Dado que el control pasa al estado **Y**, se ejecuta la acción **b** pero la acción **c** no llega a ejecutarse en este caso.

En la Figura 13.99, el estado inicial posee una bifurcación. Suponga una vez más que el sistema comienza en el estado **X**. Cuando se produce un evento, el sistema pasa al estado **Y** y se ejecuta la acción de entrada **b**. El control pasa al estado inicial. Se comprueba atributo de tamaño del objeto propietario. Si es 0, el control pasa a **Z**; en caso contrario, el control pasa al estado **W**.

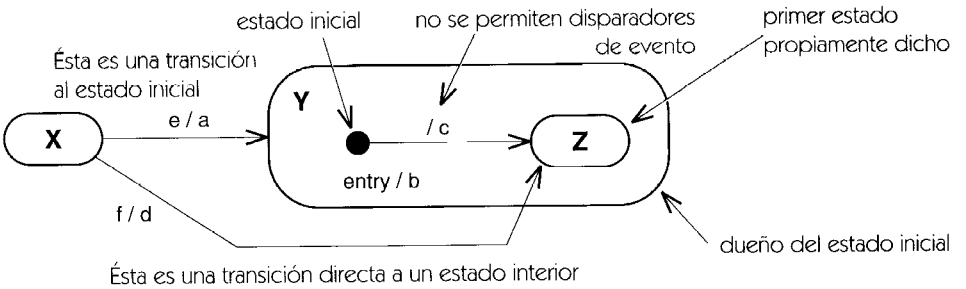


Figura 13.98 Estado inicial

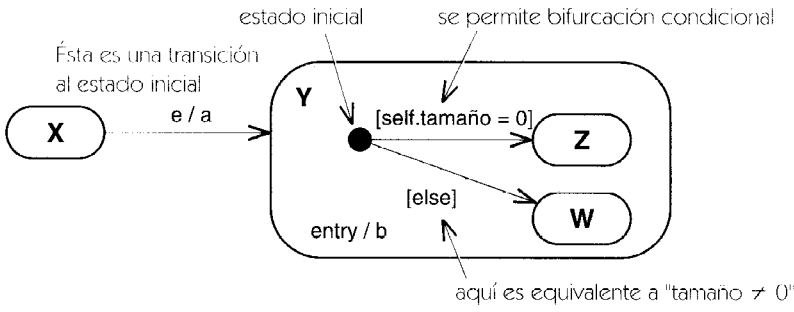


Figura 13.99 Estado inicial con ramificación

estado simple

Estado que no contiene otros estados anidados en su interior. Un conjunto de estados anidados forma un árbol y los estados simples son las hojas. Los estados simples no poseen una subestructura. Pueden tener transiciones internas, acciones de entrada y acciones de salida. Por contrastar con: estado compuesto.

estereotipo

Nueva clase de elemento del modelo definida dentro del modelo y basada en alguna clase existente de elementos del modelo. Los sistemas pueden extender la semántica pero no la estructura de clases preexistentes del metamodelo.

Véase también restricción, valor etiquetado.

Véase el Capítulo 14, Elementos Estándar, para consultar una lista de estereotipos predefinidos.

Semántica

Los estereotipos representan una variación de algún elemento existente del modelo, con la misma forma (tal como los atributos y relaciones) pero con una intención diferente. En general, los estereotipos representan una distinción de utilización. Un elemento con estereotipo puede tener restricciones adicionales, más allá de las que posea el elemento base, así como otro aspecto visual diferente. Se supone que los generadores de código y demás herramientas tratarán de forma especial a los elementos con estereotipos, generando por ejemplo un código diferente. La intención es que una herramienta genérica de modelado, como un editor o repositorio de modelos, trate (para casi todos los propósitos) a los elementos con estereotipos como elementos ordinarios con alguna información textual adicional, al mismo tiempo que se distingue ese elemento para ciertas operaciones semánticas tales como la comprobación de estar bien formados, la generación de código y la creación de informes. Los estereotipos representan uno de los mecanismos de extensión que se han incorporado a UML.

Todo estereotipo se deriva de alguna clase base de elemento del modelo. Todos los elementos que lleven el estereotipo tendrán las propiedades de la clase base de ese elemento del modelo.

Los estereotipos también se pueden especializar a partir de otros estereotipos. Las definiciones de estereotipos son elementos generalizables. El estereotipo hijo tiene las propiedades del estereotipo padre. En última instancia, todo estereotipo está basado en alguna clase de elemento del modelo.

Los estereotipos pueden poseer una lista de marcadores requeridos y algunos de ellos pueden tener un valor por defecto que se utilizará si no se proporciona un valor etiquetado explícito. El intervalo de valores permitidos para cada marcador también se puede especificar. Todo elemento que lleve el estereotipo debe tener valores etiquetados con los marcadores que se enumeran. Los marcadores que posean valores por defecto quedarán implícitos automáticamente si no son explícitos en aquellos elementos que lleven el estereotipo.

Los estereotipos pueden tener una lista de restricciones que añadirán condiciones a las implicadas por el elemento base. Todas las restricciones serán de aplicación a todo aquel elementos del modelo que lleve el estereotipo. Además, los elementos del modelo están sometidos a las restricciones aplicables al elemento base.

Un estereotipo es una clase virtual del metamodelo (esto es, no se manifiesta en el metamodelo) que se añade dentro del modelo en lugar de modificar el metamodelo predefinido de UML. Por esta razón, los nombres de los estereotipos nuevos tienen que ser distintos de los nombres de metaclases existentes en UML y de los nombres de los demás estereotipos o palabras reservadas.

Todo elemento del modelo puede tener como máximo un estereotipo. Esta regla puede no ser esencial lógicamente pero simplifica la semántica y la notación de los estereotipos sin que haya una pérdida real de potencia, porque se permite la herencia múltiple de estereotipos. Los estereotipos pueden ser hijos de otros estereotipos. Toda situación en que un elemento tenga múltiples estereotipos se puede transformar en otra en que haya un solo estereotipo que sea hijo de otros. En algunas ocasiones, esto puede obligar a quien crea el modelo a combinar otros estereotipos, pero estimamos que la mayor sencillez del caso medio compensa esta incomodidad.

Hay ciertos estereotipos que se han predefinido en UML; otros pueden ser definidos por el usuario. Los estereotipos son uno de los tres mecanismos de extensión que existen en UML.

Véase restricción, valor etiquetado.

Véase el Capítulo 13, Enciclopedia de Términos, para consultar una lista de los estereotipos predefinidos.

Notación

La notación general para el uso de estereotipos consiste en emplear el símbolo del elemento base añadiendo una cadena que contiene una palabra reservada por encima del nombre del elemento base (si existe). La palabra reservada es el nombre del estereotipo entre comillas, que son el signo de cita que se emplea en Francés y también en otros idiomas, por ejemplo, «**entre comillas**». (Observe que las comillas se parecen a dobles corchetes angulares, pero se trata de un único carácter que se halla en la mayoría de las fuentes extendidas. Casi todos los computadores tienen una utilidad de mapa de caracteres en la que se pueden buscar símbolos especiales. Quienes tengan problemas tipográficos pueden utilizar dobles corchetes angulares.) La cadena que contiene la palabra reservada suele situarse encima o delante del elemento de

modelo que se está describiendo. también se puede utilizar esa palabra reservada como elemento de una lista. En tal caso, se aplica a los elementos subsiguientes de la lista hasta que la sustituya otra cadena de estereotipo, o bien hasta que aparezca una cadena vacía de estereotipo («») que ponga fin a su alcance. Observe que el nombre de un estereotipo no debe ser idéntico al de una palabra reservada predefinida que pueda ser aplicable al mismo tipo de elemento. (Para evitar confusiones, los nombres de palabras reservadas deben evitarse en todos los estereotipos, aunque sólo se puedan aplicar a otros elementos y sean distinguibles en principio.)

Para permitir una extensión gráfica limitada de la notación de UML, se puede asociar un icono o marcador gráfico (como una textura o color) a las estructuras.

UML no especifica la forma de especificación gráfica pero existen muchos formatos de mapa de bits y de trazos que se pueden utilizar en un editor gráfico (aunque su transportabilidad es un problema difícil). Se pueden utilizar los iconos de dos maneras. En un caso, pueden emplearse en lugar de la palabra reservada de estereotipo o además de ella, dentro del símbolo del elemento base del modelo en que se basa el estereotipo. Por ejemplo, en un rectángulo de clase el icono se sitúa en la esquina superior derecha del compartimento de nombre. De esta manera, se puede ver el contenido normal del elemento en su símbolo. Alternativamente, se puede "concentrar" todo el símbolo del elemento en un icono que contiene el nombre del elemento o que tiene el nombre por encima o por debajo de su icono. Se suprime el resto de la información contenida en el símbolo del elemento base del modelo.

La Figura 13.100 muestra diferentes formas de dibujar un clase con estereotipo.

UML evita el uso de marcadores gráficos tales como el color, que supone una dificultad para ciertas personas (daltónicos) y para tipos importantes de periféricos (como impresoras, copiadoras y máquinas de fax). Ninguno de los símbolos de UML requiere el uso de estos marcadores gráficos. Los usuarios pueden emplear libremente los marcadores gráficos para sus propios propósitos (tales como resaltar elementos dentro de una herramienta) pero deben ser conscientes de sus limitaciones para el intercambio y tienen que estar dispuestos a utilizar las formas canónicas cuando sea necesario.

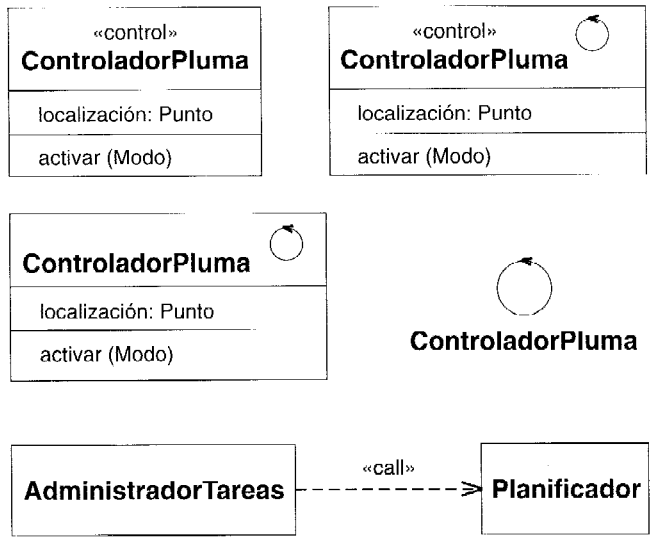


Figura 13.100 Variedades de notaciones de estereotipos

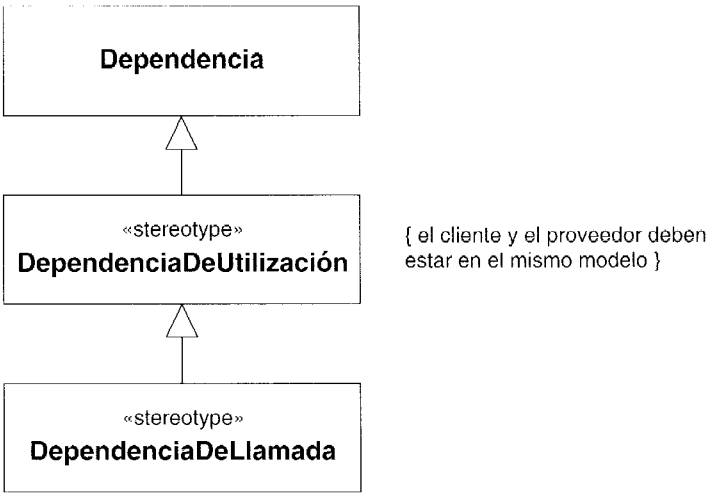


Figura 13.101 Declaración de un estereotipo

Declaración de estereotipos. La jerarquía de clasificación de los propios estereotipos se puede visualizar en un diagrama de clases. Sin embargo, se trata de un diagrama de metamodelo y debe ser distinguido (tanto por los usuarios como por las herramientas) de los diagramas ordinarios del modelo. En tales diagramas, cada estereotipo se representa mediante un símbolo de clase (un rectángulo) con la palabra reservada «**stereotype**» (estereotipo). Las relaciones de generalización pueden mostrar la jerarquía extendida del metamodelo (Figura 13.101). Dado el peligro que supone extender la jerarquía interna del metamodelo, las herramientas pueden exponer esta capacidad en los diagramas de clases, pero pueden no hacerlo. La declaración de los nombres de los estereotipos no es una capacidad que requieran los constructores de modelos ordinarios, pero se puede efectuar como tarea de apoyo.

etiqueta

El término que denota el uso de una cadena en un diagrama. Se trata de un término puramente de notación.

Véase también diagrama.

Notación

Una etiqueta es una cadena gráfica que está asociada lógicamente a otro símbolo de un diagrama. Visualmente, la asociación suele ser cuestión de situar la cadena en una región cerrada o poner la cadena cerca del símbolo. Para algunos símbolos la cadena se coloca en una posición determinada (tal como la parte inferior de una línea) pero para la mayoría de los símbolos la cadena tiene que estar "cerca" de una línea o de un icono. Una herramienta de edición puede mantener un enlace gráfico interno explícito entre la etiqueta y un símbolo gráfico para que la etiqueta quede conectada lógicamente con el símbolo aun cuando ambos estén separados visualmente. Pero el aspecto final del diagrama es una cuestión de juicio estético y

debería hacerse de tal modo que no haya confusión acerca del símbolo al que está asociada la etiqueta. Aun cuando la asociación pueda no ser evidente a partir de una inspección visual del diagrama, la asociación será clara y sin ambigüedad en el valor de estructura gráfica (y por tanto no implicará ambigüedad en la correspondencia semántica). Una herramienta puede mostrar visualmente la asociación de una etiqueta a otro símbolo empleando diferentes ayudas (tales como una línea coloreada, o el parpadeo de elementos asociados) para mayor comodidad.

evento

La especificación de una ocurrencia destacada que tiene una localización en el tiempo y en el espacio.

Véase también máquina de estados, transición, disparador.

Semántica

Dentro de una máquina de estados, una ocurrencia de un evento puede disparar una transición de estado. Un evento tiene una lista de parámetros (posiblemente vacía) que transportan la información desde el creador del evento a su receptor. El tiempo en que ocurre el evento es un parámetro implícito de cualquier evento. Otros parámetros son parte de la definición de un evento.

Una ocurrencia (instancia) de un evento tiene argumentos (valor real) que se corresponden a cada parámetro del evento. El valor de cada argumento está disponible para una acción unida a una transición accionada por el evento.

Hay cuatro clases de eventos:

evento de llamada — La recepción de una petición para invocar una operación. El resultado previsto es la ejecución de la operación accionando una transición en el receptor. Los parámetros del evento son una referencia a la operación y a los parámetros de la operación e, implícitamente, un puntero de vuelta. El llamador recupera control cuando la transición es completa (o inmediatamente si no se dispara ninguna transición).

evento de cambio — La satisfacción de una condición booleana especificada por una expresión en el evento. No hay parámetros. Este tipo de evento implica una prueba continua de la condición. El evento ocurre cuando la condición cambia de falso a verdadero. En la práctica, sin embargo, los tiempos en los cuales la condición puede ser satisfecha se pueden restringir a menudo a la ocurrencia de otros eventos, para no requerir comprobación constante.

evento de señal — La recepción de una señal, la cual es una entidad a la que se ha dado nombre explícitamente, prevista para la comunicación explícita entre objetos. Una señal tiene una lista explícita de parámetros. Es enviada explícitamente por un objeto a otro objeto o conjunto de objetos. Una difusión general de un evento se puede ver como el envío

de una señal al conjunto de todos los objetos, aunque en la práctica, se puede implementar de forma diferente, por razones de eficiencia.

El remitente especifica explícitamente los argumentos de la señal cuando se envía. Una señal enviada a un conjunto de objetos puede accionar cero o una transición en cada uno de ellos.

Las señales son los medios explícitos por los cuales los objetos pueden comunicarse entre sí de forma asíncrona. Para realizar la comunicación síncrona, se deben utilizar dos señales asincrónicas, una en cada dirección de la comunicación.

Las señales son generalizables. Una señal del hijo se deriva de una señal del padre; hereda los parámetros del padre y puede agregar parámetros adicionales de sí mismo. Una señal del hijo satisface un disparador lo que requiere a uno de sus antecesores.

evento de tiempo — La satisfacción de una expresión de tiempo, tal como la ocurrencia de un tiempo absoluto o del paso de una cantidad de tiempo dada después de que un objeto entre en un estado. Observe que el tiempo absoluto y el tiempo transcurrido se pueden definir con respecto a un reloj del mundo real o a un reloj interno virtual (en cuyo caso, puede diferir para diversos objetos).

Notación

Véase el tipo específico de evento para los detalles sobre la notación.

Elementos estándar

create, destroy.

evento actual

El evento que accionó la ejecución de una etapa atómica en la máquina de estados.

Véase también ejecución atómica, máquina de estados, transición.

Semántica

Por conveniencia, una máquina de estados puede atravesar varios segmentos de transición en respuesta a un evento. Todos, excepto el segmento final de la transición, pasan a pseudoestados —es decir, estados ficticios cuyo propósito es ayudar a estructurar la máquina de estados, pero que no esperan eventos exteriores. En principio, sería posible reunir todos los segmentos en una transición, pero la separación en múltiples segmentos usando pseudoestados permite compartir las subsecuencias comunes entre varias transiciones.

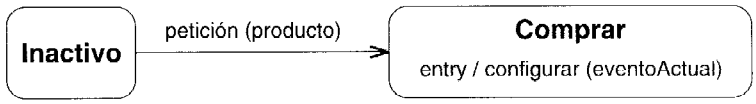


Figura 13.102 Uso del evento actual

La ejecución de una secuencia de segmentos de transición es atómica —es decir, es parte de un solo paso de ejecución que no puede interrumpir un evento exterior—. Durante la ejecución de tal secuencia de transiciones, las acciones y las condiciones de guarda unidas a los segmentos tienen acceso implícito al evento que accionó la primera transición y a los parámetros de ese evento. Este evento se conoce como evento actual durante una transición. El tipo del suceso actual se puede discriminar por una operación polimórfica o una sentencia selectiva múltiple. Después de saber el tipo exacto, será posible acceder a sus parámetros.

El evento actual es particularmente útil en la transición inicial de un nuevo objeto para obtener los parámetros de la creación. Cuando se crea un nuevo objeto, el evento que lo crea se convierte en el evento actual y sus parámetros están disponibles durante la transición inicial de la nueva máquina de estados del objeto.

Ejemplo

La Figura 13.102 muestra una transición del estado **Inactivo** al estado de **Comprar** accionado por el evento **petición**. La acción de entrada de **Comprar** llama a la operación de **configurar**, que utiliza el evento actual. El programa puede tener acceso al evento actual para obtener el evento **petición** y su parámetro, **producto**. Si hay posibles múltiples enlaces al evento actual, el programa puede necesitar que una sentencia case para obtener el evento correcto del disparador. La sintaxis es específica del lenguaje de programación.

Notación

El evento actual se puede señalar en una expresión por la palabra clave **currentEvent**. Una expresión en un lenguaje en particular puede proporcionar una sintaxis más detallada.

evento de cambio

Evento de una expresión booleana que se satisface gracias a un cambio en uno o más valores de aquellos a los que hace referencia.

Véase también condición de guarda.

Semántica

Un evento de cambio contiene una condición especificada por una expresión booleana. El evento no tiene parámetros. Ocurre cuando la condición tiene el valor verdadero tras haber tenido el valor falso, debido a un cambio en una o más variables de las que depende la condición.

Este tipo de eventos implica una continua comprobación de la condición. Sin embargo, en la práctica el desarrollador puede realizar la comprobación en tiempos discretos bien definidos analizando aquellos momentos en que las entradas de la condición cambian, por lo que no es necesaria la comprobación continua.

El evento ocurre una vez cuando el valor de la condición cambia de falso a verdadero (cambio a positivo), y no vuelve a repetirse a menos que el primer valor vuelva a valer falso.

Obsérvese la diferencia con una condición de guarda. Una condición de guarda se evalúa una vez siempre que ocurre el evento de su transición. Si la condición es falsa, no se dispara la transición y el evento se pierde (a menos que ese evento dispare alguna otra transición). Un evento de cambio se evalúa continuamente de forma implícita y ocurre una vez cuando el valor toma el valor verdadero. En ese momento, puede disparar una transición o ser ignorado. Si es ignorado, el evento de cambio no dispara una transición en ningún estado siguiente simplemente porque la condición sea aún verdadera: el evento ya ocurrió y fue ignorado. La condición debe volver a tener el valor falso y pasar de nuevo a verdadero para volver a provocar un evento de cambio.

Los valores de la expresión booleana deben ser atributos del objeto propietario de la máquina de estados que contiene la transición o valores alcanzables desde él.

Notación

A diferencia de las señales, los eventos de cambio no tienen nombre y no se declaran. Simplemente se utilizan como disparadores de las transiciones. Un evento de cambio se representa mediante la palabra clave **when** seguida de una expresión booleana entre paréntesis. Por ejemplo:

```
when (self.clientesEsperando > 6)
```

Discusión

Un evento de cambio es una prueba de satisfacción de una condición. Puede ser costoso de implementar, aunque hay muchas técnicas que permiten compilarlo para que no se produzca una comprobación continua. No obstante, es potencialmente costoso, y además oculta la relación causa-efecto directa entre el cambio de valor y los efectos disparados por dicho cambio. Esto último a veces es deseable para así encapsular los efectos, pero los eventos de cambio deben utilizarse con cuidado.

Un evento de cambio está pensado para representar un chequeo de valores visible para un objeto. Si un cambio en un atributo de un objeto dispara un cambio en otro objeto que no es consciente siquiera de la existencia del atributo, la situación debería modelarse como un evento de cambio sobre el propietario del atributo, que dispara una transición interna, la cual a su vez envía una señal al segundo objeto.

Obsérvese que un evento de cambio no se envía explícitamente a ningún sitio. Si se pretende modelar una comunicación explícita con otro objeto, debería utilizarse en su lugar una señal.

La implementación de un evento de cambio se puede llevar a cabo de varias formas, alguna de ellas haciendo pruebas dentro de la propia aplicación para calcular los tiempos de chequeo más adecuados, y otras utilizando facilidades del sistema operativo subyacente.

evento de llamada

Evento por el que se recibe una llamada para realizar una operación. Se implementa mediante acciones o transiciones de la máquina de estados.

Véase también llamada, señal.

Semántica

Un evento de llamada es una forma de implementar una operación, distinta de la ejecución de un procedimiento. Si una clase especifica la implementación de una operación como un evento de llamada, toda llamada a la operación será tratada como un evento que dispara una transición en la máquina de estados de la clase. Esto permite una implementación más dispersa, en contraposición al método monolítico de los procedimientos, que siempre hacen lo mismo (un procedimiento puede tener una sentencia de selección múltiple, por lo que no hay una diferencia real en potencia entre los dos enfoques).

Si una clase utiliza eventos para implementar una operación, su máquina de estados debe tener transiciones disparadas por el evento de llamada. La signatura de un evento de llamada es la misma que la de una operación: el nombre del evento de llamada sería el nombre de la operación, mientras que los parámetros del evento de llamada serían los parámetros de la operación.

Cuando se produce un evento de llamada, se consulta la máquina de estados del objeto destino y si el evento de llamada se corresponde con un evento disparador en una transición activa (uno que salga de un estado activo actual) se dispara una transición. Si se dispara una transición, se ejecutan su efectos, que pueden incluir cualquier secuencia de acciones, incluyendo una acción **return**(valor), cuyo propósito es devolver un valor a quien realizó la llamada.

Cuando finaliza la ejecución de la transición, quien realizó la llamada retoma el control y puede continuar la ejecución. Si la operación requiere devolver un valor de retorno y el que realizó la llamada no lo recibe, o recibe un valor inconsistente con el tipo declarado en la operación, el modelo es erróneo. Si la operación no requiere valor de retorno no hay problema. Obsérvese que si un evento de llamada no dispara ninguna transición, el control retorna inmediatamente a quien hizo la llamada; si la operación requiere devolver un valor, el modelo es erróneo; y si no, el que realiza la llamada se reanuda inmediatamente.

Si el receptor es un objeto activo, un evento de llamada se trata cuando la máquina de estados del receptor esté inactiva, es decir, cuando hayan finalizado todos los pasos de la ejecución a completar.

Notación

Un evento de llamada se representa mediante un evento disparador en un diagrama de estados que se corresponde con una operación de la clase correspondiente. La secuencia de acciones de la transición puede tener una sentencia **return**; si es así, la sentencia debe indicar el valor devuelto.

Texto del procedimiento de llamada

```
ingresar (10);
. . .
cantidad := disponer ( )
```

Sacar todo o nada dependiendo de si la cuenta está bloqueada o no

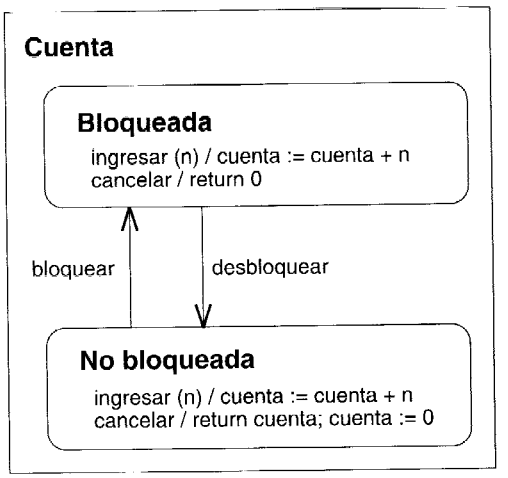


Figura 13.103 Eventos de llamada

Ejemplo

La Figura 13.103 muestra una cuenta que puede estar **Bloqueada** y **NoBloqueada**. La operación **ingresar** siempre añade dinero a la cuenta, sea cual sea su estado. La operación **disponer**, permite sacar todo el dinero de la cuenta si no está bloqueada, pero si lo está no es posible sacarlo. La operación **disponer** está implementada como un evento de llamada que dispara transiciones internas en cada uno de los estados. Cuando se produce la llamada, se ejecuta una u otra secuencia de acciones, dependiendo del estado activo. Si el sistema está bloqueado, se devuelve cero; si no lo está, se devuelve todo el dinero que tenía la cuenta y la cuenta se pone a cero.

evento de señal

Un evento que es la recepción por parte de un objeto de una señal que se le envía y que puede disparar una transición en su máquina de estados.

evento diferido

Un evento cuyo reconocimiento se retrasa mientras un objeto está en cierto estado.

Véase también máquina de estados, transición.

Semántica

Un estado puede señalar un conjunto de eventos como diferidos. Si ocurre un evento mientras un objeto está en un estado que ha diferido el evento y el evento no acciona una transición, el evento no tiene ningún efecto inmediato. Se almacena hasta que el objeto entre en un estado en el cual el evento en cuestión no esté diferido. Si ocurren otros eventos mientras el estado está

activo, se manejan de manera general. Cuando el objeto entra en un nuevo estado, los eventos guardados que ya no son diferidos ocurren uno tras otro y pueden accionar transiciones en el nuevo estado (el orden de ocurrencia de los eventos previamente diferidos es indeterminado, y es aventurado depender de un orden de ocurrencia en particular). Si el evento no acciona ninguna transición en el estado no diferido, entonces es ignorado y se pierde.

Los eventos diferidos se deben utilizar con cuidado en máquinas de estados ordinarias. Pueden ser modelados a menudo más directamente por un estado concurrente que responda a ellos mientras que el elemento computacional principal está haciendo algo más. Pueden ser útiles en los estados de actividad en los cuales permiten que los cómputos se realicen secuencialmente sin perder mensajes asincrónicos.

Si un estado tiene una transición accionada por un evento diferido, entonces la transición elimina el aplazamiento y el evento acciona la transición.

Notación

El evento diferido se indica por una transición interna en el evento con la acción reservada especial **defer** (diferir). El aplazamiento se aplica al estado y a sus subestados anidados (Figura 13.104).

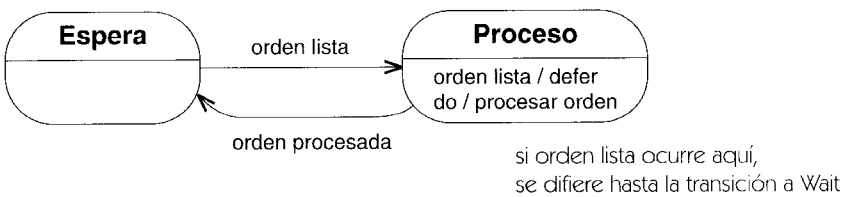


Figura 13.104 Evento diferido

evento temporal

Evento que denota el cumplimiento de alguna expresión temporal, tal como la aparición de un tiempo absoluto o el paso de un determinado período de tiempo una vez que un objeto entra en un estado.

Semántica

Un evento temporal es un evento que depende del paso del tiempo y por tanto de la existencia de un reloj. En el mundo real, el reloj es algo implícito. En un computador, se trata de una entidad física y puede haber distintos relojes en distintos computadores. El evento temporal es un mensaje del reloj al sistema. Obsérvese que se puede definir tanto el tiempo absoluto como el relativo con respecto a un reloj del mundo real o bien con respecto a un reloj interno virtual (en este último caso, puede ser distinto para diferentes objetos).

Los eventos temporales pueden estar basados en un tiempo absoluto (la hora del día o algún ajuste del reloj dentro de un sistema) o bien en tiempos relativos (el tiempo transcurrido desde la entrada en un cierto estado o bien desde que se ha producido un evento).

Notación

Los eventos temporales no se declaran como eventos con nombre tal como se hace con las señales. En lugar de hacer esto, se emplea simplemente una expresión temporal como disparador de la transición.

Discusión

En cualquier implementación real, los eventos temporales no provienen del universo; vienen de algún objeto reloj situado dentro o fuera del sistema. Como tales, son casi indistinguibles de señales, especialmente en los sistemas distribuidos y de tiempo real. En tales sistemas también hay que resolver cuál de los relojes se va a utilizar —no hay nada que sea el "tiempo real"—. (Tampoco existe en el universo real —pregúntele a Einstein—.)

excepción

Una señal levantada en respuesta a fallos del comportamiento de la ejecución de la máquina subyacente.

Véase también estado compuesto, señal.

Semántica

Una excepción se genera generalmente implícitamente, por los mecanismos subyacentes de la implementación, en respuesta a una falta durante la ejecución. Puede considerarse como una señal al objeto activo o procedimiento de activación que causó la ejecución. La máquina de estados del objeto puede tomar una acción apropiada para gestionar la excepción, incluyendo abortar el proceso actual e ir a un lugar conocido en la ejecución, ejecutar una operación sin un cambio de estado, o ignorar el evento. La capacidad de unir transiciones a los estados de alto nivel hace la gestión de excepciones flexible y potente.

Una excepción es una señal y por lo tanto tiene una lista de los parámetros que están limitados a valores cuando ocurre la excepción. Los valores de parámetros son fijados por la maquinaria de la ejecución que detecta la avería, tal como el sistema operativo. La operación que maneja la excepción puede leer los parámetros. En la mayoría de los lenguajes, las excepciones pueden ser manipuladas y retransmitidas por las operaciones que las manejan.

Notación

El estereotipo «**exception**» puede utilizarse para distinguir la declaración de una excepción. No es necesario un estereotipo para usar el nombre del evento en una máquina de estados.

exportar

En el contexto de los paquetes, hacer un elemento accesible fuera del espacio de nombres que lo contiene ajustando su visibilidad. Contrasta con el acceso y la importación, los cuales hacen a elementos exteriores accesibles dentro de un paquete.

Véase también acceder, importar, visibilidad.

Semántica

Un paquete exporta un elemento fijando su visibilidad a un nivel que permita que sea visto por otros paquetes (público para los paquetes que lo importan, protegido para sus propios hijos).

Discusión

Son necesarias dos cosas para que un elemento (tal como una clase) pueda ver un elemento en un paquete del mismo nivel. El paquete que contiene el elemento objetivo debe exportarlo dándole visibilidad pública. Además, el paquete que se refiere al elemento objetivo debe acceder o importar el paquete que contiene el elemento objetivo. Ambos pasos son necesarios.

expresión

Una cadena que codifica una sentencia que se interpretará por un lenguaje dado. Muchos tipos de expresiones proporcionan valores cuando son interpretadas; otros tipos realizan acciones específicas. Por ejemplo, la expresión "(7 + 5 * 3)" se evalúa a un valor de tipo número.

Semántica

Una expresión define una sentencia que evalúa un conjunto (posiblemente vacío) de instancias o de valores o el resultado de una acción específica cuando se ejecuta en un contexto. Una expresión no modifica el entorno en el cual se evalúa. Una expresión consiste en una cadena y el nombre del lenguaje de la evaluación.

Un elemento de la expresión contiene el nombre del lenguaje de interpretación y de la cadena que codifica la expresión en la sintaxis del lenguaje elegido. Se asume que un intérprete está disponible para el lenguaje. Proporcionar un intérprete es responsabilidad de la herramienta que modela. El lenguaje puede ser un lenguaje de especificación de restricciones, como OCL; puede ser un lenguaje de programación, como C++ o Smalltalk; o puede ser un lenguaje natural. Por supuesto, si se escribe una expresión en lenguaje natural, no puede ser evaluado automáticamente por una herramienta y deberá ser totalmente para consumo humano.

Las diferentes subclases de expresiones producen diversos tipos de valores. Éstos incluyen expresiones booleanas, expresiones de conjuntos de objetos, expresiones de tiempo, y expresiones de procedimiento.

Las expresiones aparecen en modelos de UML como acciones, restricciones, condiciones de guarda, y otros.

Notación

Se muestra una expresión como una cadena definida en un lenguaje. La sintaxis de la cadena es responsabilidad de una herramienta y de un analizador lingüístico para el lenguaje. La asunción es que el analizador puede evaluar cadenas en tiempo de ejecución para producir valores del tipo apropiado, o puede proporcionar las estructuras semánticas para capturar el significado de la expresión. Por ejemplo, una expresión de tipo evalúa una referencia del clasificador, y una expresión booleana evalúa un valor verdadero o falso. Se conoce el lenguaje para una herramienta de modelado, pero generalmente es implícito en el diagrama bajo la asunción de que la forma de la expresión hace claros sus propósitos.

Ejemplo

```
self.coste < autorización.costeMáximo

para todo (k en destinos) { k.actualizar () }
```

expresión booleana

Expresión que se evalúa a un valor booleano. Útil en condiciones de guarda.

expresión de acción

Expresión que puede estar compuesta por una acción o por una secuencia de acciones.

Discusión

UML no especifica la sintaxis de una expresión de acción, pues eso es responsabilidad de las herramientas que trabajen con UML. Se espera con ello que los diferentes usuarios puedan expresar acciones mediante lenguajes de programación, pseudocódigo o incluso mediante lenguaje natural. Para el diseño detallado se necesita una sintaxis más precisa, por supuesto, pero es ahí donde la mayoría de los usuarios utilizarán los lenguajes de programación reales.

expresión de actividad

Expresión textual de un cómputo no atómico (una actividad). Una expresión de este tipo se puede descomponer conceptualmente en fragmentos atómicos, pero es conveniente permitir una representación textual de la actividad completa. La ejecución de una expresión de actividad puede ser finalizada por una transición que desactive el estado de control.

Semántica

Una expresión de actividad es un procedimiento efectivo (algoritmo) expresado en algún lenguaje, que puede ser un lenguaje de programación u otro tipo de lenguaje formal, aunque también puede expresarse mediante lenguaje natural. Si se utiliza el lenguaje natural, no será posible ejecutarlo con herramientas ni comprobar sus errores u otras propiedades, pero puede ser suficiente en las primeras etapas de trabajo. También puede representar una operación continua del mundo real.

Notación

Una expresión de actividad se representa como un texto interpretado en algún lenguaje.

Ejemplo

do / invertirMatriz	Finito pero tarda un tiempo
do / calcularMejorMovimiento	Se ejecuta hasta que termina el tiempo
do / sonar sirena	Continuo hasta que se para

expresión de conjunto de objetos

Indica una expresión que produce un conjunto de objetos cuando se evalúa en tiempo de ejecución.

Semántica

El blanco de una acción de envío es una expresión de conjunto de objetos. Cuando se evalúa una de estas expresiones en tiempo de ejecución, produce un conjunto de objetos al cual se envía en paralelo una señal acordada. Una expresión de conjunto de objetos puede proporcionar un único elemento, en cuyo caso la acción de envío es una acción secuencial ordinaria. Puede, incluso, no producir elemento alguno (esto es, un conjunto vacío) en cuyo caso no se produce el envío.

expresión de iteración

Una expresión que proporciona un conjunto de casos de iteración. Cada caso de iteración especifica la ejecución de una acción dentro de una iteración. Un caso de iteración puede incluir la asignación de valores a una variable de iteración. La acción se ejecuta una sola vez para cada caso de iteración.

Véase también mensaje.

Semántica

La expresión de iteración representa una ejecución condicional o iterativa. Representa la ejecución de cero o más mensajes, dependiendo de las condiciones implicadas. Las opciones son:

* [cláusula de iteración]	Una iteración
[cláusula de condición]	Una bifurcación

Una iteración representa una secuencia de mensajes. La cláusula de iteración muestra los detalles de la variable y de la prueba de iteración, pero se puede omitir (en cuyo caso las condiciones de iteración quedan sin especificar). La cláusula de iteración debería expresarse en pseudocódigo o en algún lenguaje real de programación. UML no prescribe su formato. Un ejemplo podría ser:

```
* [i:= 1..n]
```

Las condiciones representan un mensaje cuya ejecución es contingente respecto a la veracidad de la cláusula de condición. La cláusula de condición debería expresarse en pseudocódigo o en algún lenguaje real de programación. UML no prescribe su formato. Un ejemplo podría ser:

```
[x > y]
```

Obsérvese que las bifurcaciones se denotan igual que las iteraciones sin un asterisco. Se puede pensar que se trata de una iteración limitada a un único caso.

La notación de iteración supone que los mensajes de la iteración serán ejecutados secuencialmente. También existe la posibilidad de ejecutarlos concurrentemente. La notación es un asterisco seguido por una doble línea vertical, para denotar paralelismo (*||). Por ejemplo:

```
* [i:= 1..n]|| q[i].CalcularResultados ()
```

Obsérvese que en una estructura de control anidada la expresión de iteración no se repite en los niveles interiores del número de secuencia. Cada nivel de estructura especifica su propia iteración dentro del contexto que lo encierra.

expresión de procedimiento

Dícese de una expresión cuya evaluación representa la ejecución de un procedimiento que puede afectar al estado del sistema en ejecución.

Semántica

Una expresión de procedimiento es una codificación de un algoritmo ejecutable. Su ejecución puede (y suele) afectar al estado del sistema, esto es, tiene efectos secundarios. En general, las expresiones de procedimiento no proporcionan valores. El propósito de su ejecución es alterar el estado del sistema.

expresión de tipo

Expresión que al evaluarse produce una referencia a uno o más tipos de datos. Por ejemplo, una declaración de tipo de atributo en un lenguaje típico de programación es una expresión de tipo.

Ejemplo

En C++, lo que sigue son expresiones de tipo:

```
Persona*

Pedido[24]

Boolean (*)(String,int)
```

expresión temporal

Expresión que al ser resuelta proporciona un valor temporal absoluto o relativo. Se utiliza para definir los eventos temporales.

Semántica

Casi todas las expresiones temporales son o bien un tiempo transcurrido desde la entrada en un estado o bien la aparición de un cierto instante absoluto. Las demás expresiones temporales deberán definirse de forma ad hoc.

Notación

Tiempo transcurrido. Un evento que denota el paso de una cierta cantidad de tiempo tras haber entrado en el estado que contiene la transición se denota mediante la palabra reservada **after** seguida por una expresión que se evalúa (durante el tiempo de modelado) para producir una cantidad de tiempo.

after (10 segundos)

after (10 segundos después de salir del estado A)

Si no se especifica un instante inicial, se entiende que es el tiempo transcurrido desde la entrada en el estado que contiene la transición.

Tiempo absoluto. Un evento que denota llegada de un instante absoluto se denota mediante la palabra reservada when seguida por una expresión booleana entre paréntesis que implica la hora.

when (fecha = 1 de enero de 2000)

extender

Una relación desde un caso de uso extensor a un caso de uso base, que especifica cómo se puede insertar el comportamiento definido para el caso de uso extensor en el comportamiento definido para el caso de uso base. El caso de uso extensor modifica incrementalmente el caso de uso base de forma modular.

Véase también punto de extensión, incluir, caso de uso, generalización de casos de uso.

Semántica

La relación de extensión conecta un caso de uso extensor a un caso de uso base. El caso de uso extensor, en esta relación, no es necesariamente un clasificador instanciable separado. Consiste en uno o más segmentos que describen las secuencias adicionales de comportamiento que modifican incrementalmente el comportamiento del caso de uso base. Cada segmento en un caso de uso extensor se puede insertar en una localización separada en el caso de uso base. La relación de extensión tiene una lista de los nombres de los puntos de extensión, igual en número al número de segmentos en el caso de uso extensor. Cada punto de extensión se debe definir en el caso de uso base. Cuando la ejecución de una instancia de un caso de uso alcanza una localización en el caso de uso base referenciada por el punto de extensión y se cumple cualquier condición en la extensión, entonces la ejecución de la instancia se puede transferir a la secuencia de comportamiento del segmento correspondiente del caso de uso extensor; cuando la ejecución del segmento de la extensión es completa, el control vuelve al caso de uso original en el punto referenciado.

Se pueden aplicar múltiples relaciones de extensión al mismo caso de uso base. Una instancia de un caso de uso puede ejecutar más de una extensión durante el curso de su vida. Si varios casos de uso extienden un caso de uso base en el mismo punto de extensión, entonces su orden relativo de ejecución no es determinista. Puede incluso haber múltiples relaciones de extensión entre los mismos casos de uso extensor y base, a condición de que la extensión se inserte en una localización diferente en la base. Las extensiones pueden incluso ampliar otras extensiones de una manera jerarquizada.

Un caso de uso extensor en una relación de extensión puede tener acceso y modificar los atributos definidos por el caso de uso base. El caso de uso base, sin embargo, no puede ver las extensiones y puede no tener acceso a sus atributos u operaciones. El caso de uso base define un marco modular en el cual puedan ser agregadas extensiones, pero la base no tiene visibilidad sobre las extensiones. Las extensiones modifican implícitamente el comportamiento del caso de uso base. Observe la diferencia con la generalización de casos de uso. Con la extensión, los efectos del caso de uso extensor se agregan a los efectos del caso de uso base en una instanciación del caso de uso base. Con la generalización, los efectos del caso de uso hijo se agregan a los efectos del caso de uso padre en una instanciación del caso de uso hijo, mientras que una instanciación del caso de uso padre no consigue los efectos del caso de uso hijo.

Un caso de uso extensor puede extender más de un caso de uso base, y un caso de uso base se puede ampliar con más de un caso de uso extensor. Esto no indica ninguna relación entre los casos de uso base.

Un caso de uso extensor puede ser él mismo la base en una relación de extensión, inclusión, o generalización.

Estructura (del caso de uso extensor)

Un caso de uso extensor contiene una lista de uno o más *segmentos de inserción*, cada uno de los cuales es una secuencia de comportamiento.

Estructura (del caso de uso base)

El caso de uso define un conjunto de puntos de extensión, cada uno de los cuales referencia una localización o un conjunto de localizaciones en el caso de uso base donde se puede insertar el comportamiento adicional.

Estructura (de la relación de extensión)

La relación de extensión tiene una lista de los nombres de los puntos de extensión, que deben estar presentes en el caso de uso base. El número de nombres debe igualar el número de segmentos en el caso de uso extensor.

La relación de extensión puede tener una condición, una expresión en términos de atributos del caso de uso base, o la ocurrencia de eventos tales como recibir de una señal. La condición se determina si el caso de uso extensor está realizado cuando la ejecución de una instancia del caso de uso alcanza una localización referida por el primer punto de extensión. Si la condición está ausente, entonces se considera siempre como verdadera. Si la condición para un caso de uso extensor está satisfecha, entonces la ejecución del caso de uso extensor procede. Si el punto de extensión se refiere a varias localizaciones en el caso de uso base, el caso de uso extensor se puede ejecutar en cualquiera de ellas.

La extensión se puede realizar más de una vez, si la condición sigue siendo verdadera. Todos los segmentos del caso de uso extendido se ejecutan el mismo número de veces. Si el número de ejecuciones debe ser restringido, se puede definir la condición acorde.

Semántica de ejecución

Cuando una instancia de un caso de uso que realiza un caso de uso base alcanza una localización en el caso de uso base, que es referenciado por una relación de extensión, entonces se evalúa la condición en la relación de extensión. Si es verdad o si está ausente, entonces se realiza el caso de uso extendido. En muchos casos, la condición incluye la ocurrencia de un evento o la disponibilidad de los valores necesarios para el propio caso de uso extendido, por ejemplo, una señal de un actor que comience el segmento de la extensión. La condición puede depender del estado de la instancia del caso de uso, incluyendo valores del atributo del caso de uso base. Si no ocurre el evento o la condición es falsa, la ejecución del caso de uso extendido no comienza. Cuando el funcionamiento de un segmento de la extensión se completa, la instancia del caso de uso continúa realizando el caso de uso base en el punto donde lo dejó.

Las inserciones adicionales del caso de uso extendido pueden ser realizadas inmediatamente si la condición se satisface. Si el punto de la extensión se refiere a localizaciones múltiples en al caso de uso base, la condición se puede satisfacer en cualesquiera de ellos. La condición puede llegar a ser verdad en cualquier localización dentro del conjunto.

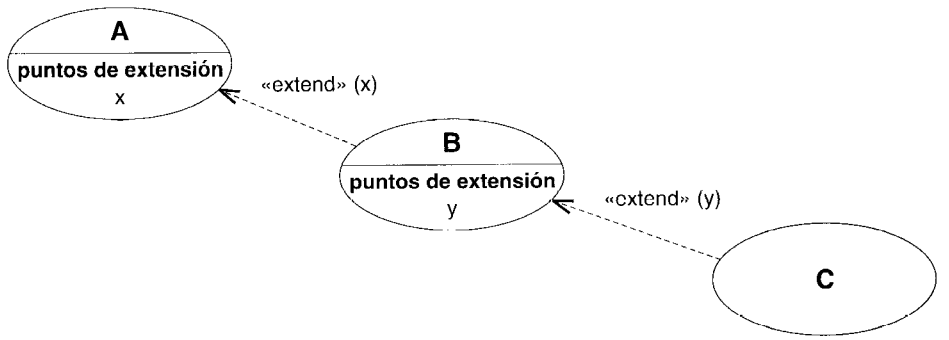


Figura 13.105 Extensiones anidadas

Si hay más de una cadena de una secuencia de inserción en un caso de uso extendido, entonces todos los segmentos de la inserción se ejecutan si la condición es verdad en el primer punto de la extensión. La condición no se vuelve a evaluar para los segmentos subsecuentes, los cuales se insertan cuando la instancia del caso de uso alcanza las localizaciones correspondientes dentro del caso de uso base. La instancia del caso de uso reasume la ejecución de la base entre las inserciones en diversos puntos de extensión. Una vez que estén comenzados, todos los segmentos deben ser realizados.

Observe que, en general, un caso de uso es una máquina de estados no determinista (como en una gramática), más que un procedimiento ejecutable. Eso es porque las condiciones pueden incluir la ocurrencia de eventos externos. Realizar un caso de uso como colaboración de clases puede requerir una transformación en mecanismos explícitos del control, así como la implementación de una gramática requiere una transformación a una forma ejecutable que sea eficiente pero más difícil de entender.

Observe que *base* y *extensión* son términos relativos. Una extensión puede servir por sí misma como base para otra extensión. Esto no presenta ninguna dificultad, y las reglas previas todavía se aplican —las inserciones se anidan—. Por ejemplo, suponga que un caso de uso B extiende un caso de uso A en el punto de extensión x, y suponga que el caso de uso C extiende

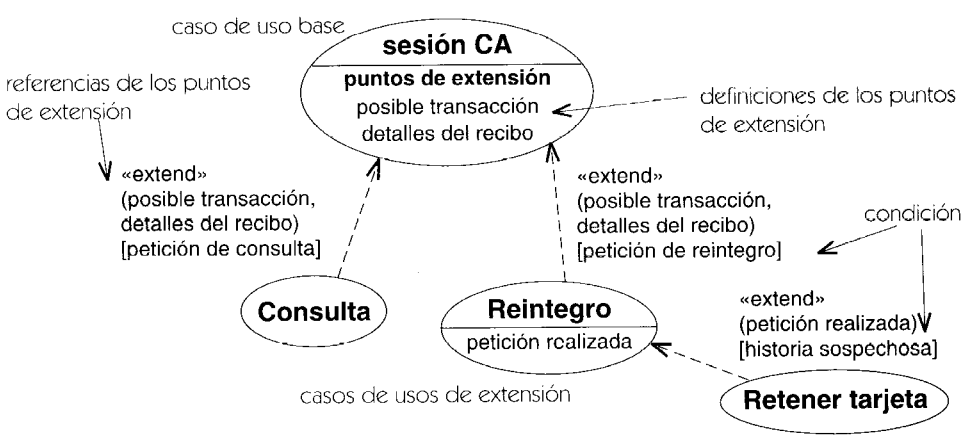


Figura 13.106 Relación de extensión

el caso de uso B en el punto de extensión y (Figura 13.105). Cuando una instancia de A llega al punto de extensión x, comienza a realizar el caso de uso B. Cuando la instancia entonces llega al punto de extensión y dentro de B, comienza a realizar el caso de uso C. Cuando se completa la ejecución de C, reasume la realización del caso de uso B. Cuando la ejecución de B está completa, reasume la ejecución de A. Es similar a llamadas anidadas a procedimientos o cualquier otra construcción anidada.

Notación

Se dibuja una flecha de línea discontinua desde el caso de uso extensor al símbolo del caso de uso base con una punta de flecha apuntando a la base. Se pone la palabra clave «**extend**» en la flecha.

Puede aparecer una lista de los nombres de los puntos de extensión entre paréntesis después de la palabra clave.

La Figura 13.106 muestra casos de uso con relaciones de extensión, y la Figura 13.107 muestra las secuencias de comportamiento de los casos de uso.

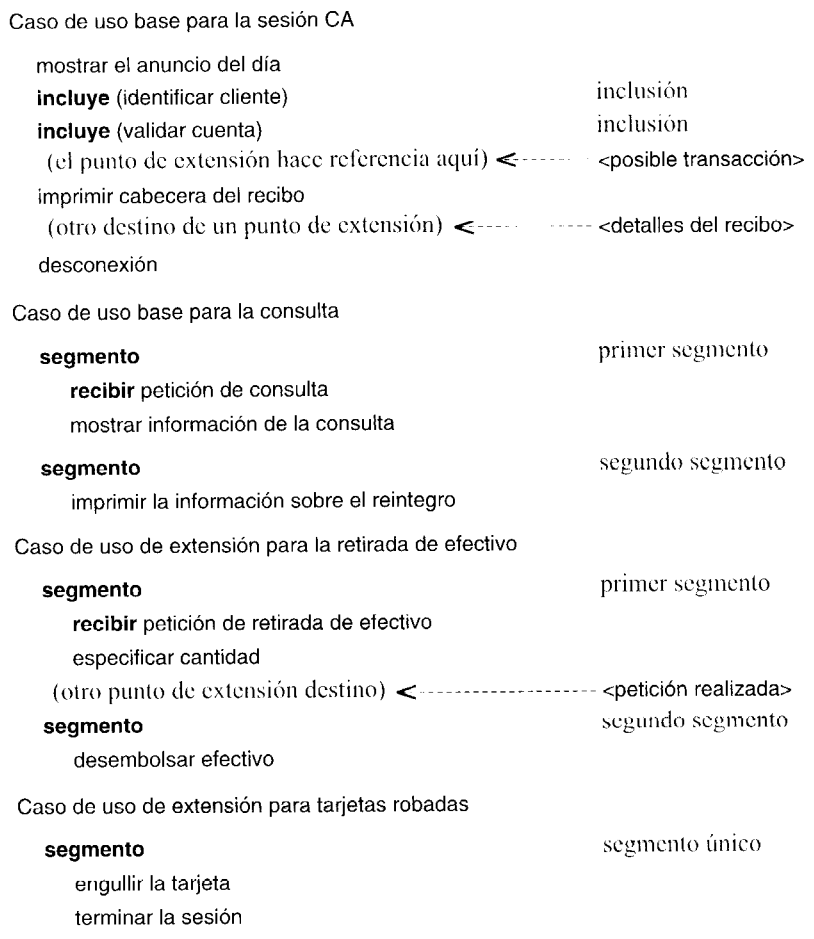


Figura 13.107 Secuencias de comportamiento para los casos de uso

Discusión

Todas las relaciones de extensión, inclusión y generalización agregan comportamiento a un caso de uso inicial. Tienen muchas semejanzas, pero en la práctica es conveniente separarlas. La tabla 13.1 compara los tres puntos de vista.

Tabla 13.1 Comparación de las relaciones de casos de uso

Propiedad	*Extensión*	*Inclusión*	*Generalización*
comportamiento base comportamiento añadido	caso de uso base caso de uso de extensión	caso de uso base caso de uso incluido	caso de uso padre caso de uso hijo
dirección de referencia.	el caso de uso extensor referencia al caso de uso base	el caso de uso base referencia al caso de uso incluido	el caso de uso hijo referencia el caso de uso padre
¿base modificada?	la extensión modifica implícitamente el comportamiento de la base. La base debe estar formada sin extensión, pero si la extensión está presente, una instancia de la base puede ejecutar la extensión	la inclusión modifica explícitamente el efecto de la base. La base puede estar bien formada o no sin la inclusión, pero una instanciación de la base ejecuta la inclusión	el efecto de ejecutar el padre no se modifica por el hijo. Para obtener efectos distintos, se debe instanciar el hijo y no el padre
¿instanciable?	la extensión no es necesariamente instanciable. Puede ser un fragmento	la inclusión no es necesariamente instanciable, puede ser un fragmento	el hijo no es necesariamente instanciable. Puede ser abstracto
¿puede acceder a los atributos de la base?	las extensiones pueden acceder y modificar el estado de la base	la inclusión puede acceder al estado de la base. La base debe proveer los atributos apropiados esperados por la inclusión	el hijo puede acceder y modificar el estado de la base (por los mecanismos usuales de herencia)
¿puede la base ver al otro?	la base no puede ver las extensiones y debe estar bien formada en su ausencia	la base ve la inclusión y puede depender de sus efectos, pero no puede acceder a sus atributos	el padre no puede ver al hijo, y debe estar bien formado en su ausencia
repetición	depende de la condición	exactamente una repetición	el hijo controla su propia ejecución

extensión

Conjunto de instancias descritas por un descriptor.

Contrastar con: intención.

Semántica

Un descriptor, tal como una clase o una asociación, tiene tanto una descripción (su intención) como un conjunto de los instancias que describe (su extensión). El propósito de la intención es especificar las características de las instancias que componen la extensión. No se asume que la extensión sea físicamente manifiesta o que pueda ser obtenida en tiempo de ejecución.

Por ejemplo, incluso en principio un modelador no debe asumir que el conjunto de los objetos que son instancias de una clase se puede obtener.

extremo de asociación

Parte estructural de una asociación que define la participación de una clase en la misma. Una clase puede estar conectada a más de un extremo en la misma asociación. Los extremos dentro de una asociación tienen distintas posiciones con diferentes nombres que, en general, no son intercambiables. Un extremo de asociación no tiene existencia independiente ni significado aparte de la asociación.

Véase también asociación.

Semántica

Estructura

Un extremo de asociación mantiene una referencia a un clasificador destino, y define la participación del clasificador en la asociación. Una instancia de la asociación (un enlace) debe contener, en una posición determinada, una instancia de la clase dada o de sus descendientes. Las clases hijas heredan la participación en una asociación.

Un extremo de asociación tiene las siguientes propiedades (véase cada entrada individual para más información):

agregación — Indica si el objeto asociado es un agregado o compuesto; es una enumeración con los valores {**none**, **aggregate**, **composite**}. Si el valor no es **none**, la asociación es una agregación o una composición. Éste es el valor por defecto. Sólo una asociación binaria puede ser agregación o composición, y sólo un extremo puede ser agregado o compuesto.

alcance destino	Indica si los enlaces relacionan objetos o clases enteras; es una enumeración con los valores {**instance**, **classifier**}. El valor por defecto es **instance** (relaciona objetos).
cualificador	Lista de atributos utilizada como selectores para encontrar objetos relacionados por una asociación.
especificador de interfaz	Restricción opcional sobre la especificación de tipo del objeto relacionado. Es un clasificador (hay quien llama a esto *rol*, aunque el término se utiliza en otros contextos).
multiplicidad	Indica el número posible de objetos que se pueden relacionar con un objeto, especificado normalmente como un rango de valores enteros.
navegabilidad	Valor lógico que indica si es posible atravesar una asociación binaria para obtener el objeto o conjunto de objetos asociados con una instancia de la clase. Por defecto es **true** (navegable).
nombre de rol	Nombre del extremo de asociación, que será un identificador de tipo cadena. Este nombre identifica el rol de la clase correspondiente dentro de la asociación. El nombre de rol debe ser único dentro de la asociación, y también entre pseudoatributos (atributos y otros nombres de rol visibles por la clase) directos y heredados de la clase origen.
ordenación	Expresa si (y potencialmente cómo) está ordenado el conjunto de objetos relacionados, es una enumeración con los valores {**ordered**, **unordered**}. También se puede utilizar el valor **sorted** con propósitos de diseño.
variabilidad	Indica si puede cambiar el conjunto de enlaces relacionados con un objeto mediante una lista con los valores {**changeable**, **frozen**, **addOnly**}. El valor por defecto es **changeable**.
visibilidad	Indica si el enlace puede acceder a otras clases además de aquellas que se encuentran en el extremo opuesto de la asociación. La visibilidad se sitúa en el extremo conectado a la clase destino. Cada dirección tiene su propio valor de visibilidad.

Notación

El extremo de la ruta de asociación se conecta con el borde del rectángulo correspondiente al símbolo de la clase. Las propiedades del extremo de asociación se representan como adornos sobre o cerca de la ruta en que se une a un símbolo de clasificador (véase Figura 13.108). La siguiente lista es un breve resumen de adornos para cada propiedad. Para más detalles, consultar individualmente cada uno de ellos.

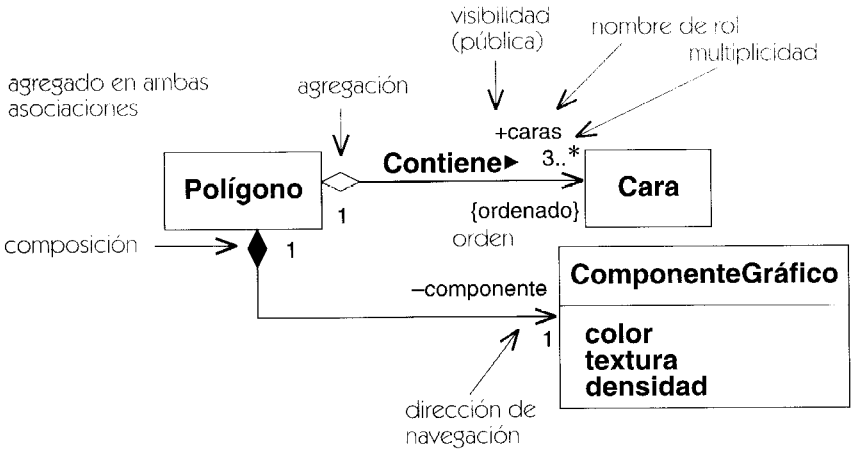


Figura 13.108 Diversos adornos en un extremo de asociación

agregación	Un pequeño rombo hueco en el extremo agregado, relleno si se trata de composición.
alcance destino	El nombre de rol de la clase alcanzada se subraya, en otro caso es alcance de instancia.
calificador	Un pequeño rectángulo entre el extremo de la ruta y la clase fuente. El rectángulo contiene uno o más atributos de la asociación: los calificadores.
especificador de interfaz	Un sufijo que se añade al nombre de rol de la forma : nombre-de-tipo.
multiplicidad	Una etiqueta de texto cerca del extremo de la ruta de la forma min..max.
nombre de rol	Una etiqueta de texto cerca del extremo destino.
ordenación	La propiedad {**ordered**} cerca del extremo destino significa que hay una lista ordenada de instancias de las clase destino.
navegabilidad	Una punta de flecha en el extremo de la ruta que muestra la dirección de navegación. Si ninguno de los dos extremos tiene flecha significa que la asociación es navegable en ambos sentidos (debido a que realmente son pocas las asociaciones que no son navegables en alguna dirección).
variabilidad	La propiedad {**frozen**} o {**addOnly**} cerca del extremo destino. Normalmente se omite cuando es {**changeable**} pero se puede poner para enfatizar la propiedad.
visibilidad	Un símbolo de visibilidad (+ # -) que precede al nombre de rol.

Si hay más de un adorno en un único rol, se representan siguiendo el siguiente orden, empezando desde el extremo de la ruta más cercano a la clase (Figura 13.16):

calificador

símbolo de agregación o composición

flecha de navegación

Los nombres de rol y la multiplicidad se deberían situar cerca del extremo de la ruta para no confundirlos con los de otra asociación. Pueden ponerse en cualquier lado de la línea. Se ha intentado que se situaran siempre en un mismo lado de la línea, pero esto muchas veces va en detrimento de la claridad. Los nombres de rol y las multiplicidades pueden ir cada una en un lado de la línea o ir juntas (por ejemplo * **empleado**).

Elementos estándar

association, global, local, parameter, self.

fases de modelado

Estados de desarrollo por los que pasa un modelo durante el proceso de diseño y construcción de un sistema.

Véase también proceso de desarrollo.

Discusión

El esfuerzo global de desarrollo se puede dividir en actividades que se centran en diferentes fines. Estas actividades no se realizan en secuencia; más exactamente, se realizan iterativamente durante las fases del proceso de desarrollo. El análisis trata de la captura de requisitos y de la comprensión de las necesidades de un sistema. El diseño trata de inventar un enfoque práctico para el problema, dentro de los límites impuestos por las restricciones de estructuras de datos, algoritmos y partes ya existentes del sistema. La implementación trata de la construcción de la solución en un lenguaje o medio ejecutable (tal como una base de datos o un cierto hardware digital). El despliegue está relacionado con poner en práctica la solución dentro de un entorno físico específico. Estas divisiones son relativamente arbitrarias y no siempre están claras, pero siguen siendo reglas útiles.

Sin embargo, estas vistas del desarrollo no deberían tomarse como fases secuenciales del proceso de desarrollo. En el Proceso en Cascada tradicional se trataban ciertamente como fases distintas. En un proceso de desarrollo iterativo, más moderno, no se trata de fases distintas. En un instante de tiempo dado pueden existir actividades de desarrollo en distintos niveles y quizá lo mejor sea comprenderlas como tareas diferentes que es preciso realizar para cada elemento del sistema, no todas al mismo tiempo.

Piense en un grupo de edificios, cada uno de los cuales posee unos cimientos, paredes y techo; todo ellos tienen que terminarse para todos los edificios, pero no todos a la vez. Normalmente, las partes de cada edificio se irán terminando más o menos por orden. Sin embargo, en algunas ocasiones el tejado puede comenzarse antes de que estén terminadas las paredes. En algunas ocasiones, la distinción entre paredes y techo se difumina: piense en una cúpula geodésica que sale del suelo.

Algunos elementos de UML están destinados a todas las fases del desarrollo. Otros están destinados a propósitos de diseño o implementación y sólo aparecerán cuando el modelo esté suficientemente avanzado. Por ejemplo, las especificaciones de visibilidad para los atributos y operaciones tienden a aparecer en la fase de diseño. En la fase de análisis se incluyen sobre todo miembros públicos.

Discusión

Un proceso de desarrollo en cascada está dividido en fases, cada una de las cuales se realiza para todo el sistema a la vez. Entre las fases tradicionales se cuentan el análisis, diseño, implementación y despliegue. Sin embargo, en un proceso iterativo el desarrollo del sistema no se realiza en formación militar. Los elementos se pueden desarrollar con diferentes velocidades. Sin embargo, cada elemento individual pasa por las mismas fases durante el desarrollo, pero los distintos elementos avanzan con distintas velocidades así que el sistema en conjunto no se encuentra en ninguna fase concreta.

Cada fase de modelado caracteriza a un cierto aspecto del problema que debe ser comprendido y modelado. Las primeras fases capturan las propiedades más lógicas y más abstractas. Las últimas fases están más centradas en la implementación y en el rendimiento. El análisis captura los requisitos y el vocabulario especializado del sistema. El diseño captura los algoritmos y estructuras de datos de la implementación abstraída en unas condiciones idealizadas pero físicamente plausibles. También se preocupa de la eficiencia, de la complejidad computacional y de las consideraciones de ingeniería del software necesarias para construir un sistema admisible. La implementación crea una descripción operativa del sistema en condiciones reales en un lenguaje real de programación, incluyendo las imperfecciones de los lenguajes reales y la descomposición de los artefactos del sistema en partes que se puedan desarrollar y almacenar por separado. El entorno de ejecución se enfrenta a los problemas de concurrencia de recursos, del entorno de computación y del rendimiento a gran escala.

UML contiene toda una gama de estructuras adecuadas para las distintas fases del desarrollo. Algunas estructuras (tales como la asociación y el estado) son significativas en todas las fases. Algunas estructuras (tales como la navegabilidad y la visibilidad) tienen sentido durante el diseño pero representan un detalle de implementación innecesario durante el análisis. Esto no se opone a su definición en una fase temprana del proyecto. Algunas estructuras (tales como la sintaxis específica de un lenguaje de programación) sólo son significativas durante la implementación y son un estorbo para el proceso de desarrollo si se introducen prematuramente.

Los modelos cambian durante el desarrollo. Un modelo de UML adopta formas distintas en cada fase de desarrollo, haciendo hincapié en distintas estructuras de UML. El modelado debería hacerse en la comprensión de que no todas las estructuras son útiles en todas las fases.

Véase proceso de desarrollo para una discusión de las fases de modelado y de las etapas de desarrollo.

fin de un enlace

Denota una instancia de un final de asociación.

flujo

Un tipo de relación que relacione dos versiones del mismo objeto en puntos sucesivos en el tiempo.

Véase también become, copia.

Semántica

Una relación de flujo relaciona dos versiones del mismo objeto en puntos sucesivos en el tiempo. Puede relacionar dos valores de un objeto en una interacción entre instancias, o puede relacionar dos roles de clasificador que describen el mismo objeto en una interacción entre descriptores.

Representa la transformación de un estado de un objeto en otro. Puede representar un cambio en valor, del estado de control, o de la localización.

Los estereotipos de la dependencia de flujo son *become* y *copy*. Otros estereotipos pueden ser agregados por los usuarios.

Notación

Una relación de flujo se representa como una flecha de líneas discontinuas con la palabra clave de estereotipo apropiada unida. No se puede utilizar una relación de flujo desnuda sin un estereotipo.

Elementos estándar

become, copy.

flujo de objeto

Una relación entre los sucesivos puntos de control dentro de una interacción, tal como un grafo de actividades o una colaboración.

Véase también acción, grafo de actividades, colaboración, transición de finalización, mensaje, estado de flujo de objetos, transición.

Semántica

Un gráfico de la vista de interacción representa el flujo de control durante un cómputo. Los elementos primitivos de una interacción son acciones y objetos simples. Un flujo de control representa la relación entre una acción y las acciones de su predecesor y de su sucesor, así como las acciones entre sus objetos de entrada y de salida. De forma abreviada, un flujo de control representa la derivación computacional de un objeto en otro o dos versiones de un objeto en un cierto período de tiempo. (Un flujo de control que implica a un objeto como entrada o salida se llama flujo de objetos.)

Notación

En un diagrama de la colaboración, el flujo de control se representa por mensajes unidos a los roles de la asociación (representando enlaces) que conectan roles de clasificador (representando objetos y otras instancias). En un diagrama de actividades, el flujo de control se muestra con flechas sólidas entre los símbolos de actividad. El flujo del objeto se representa por flechas con línea discontinua entre un símbolo de la actividad o una flecha de flujo de control y un símbolo del estado de flujo del objeto. *Véanse* esas entradas para más detalles.

flujo de objetos

Dícese de cierta variedad de flujo de control que representa la relación entre un objeto y el objeto, operación o transición que lo crea (como resultado) o que utiliza (como entrada).

Véase también flujo de control, estado de flujo de objetos.

Semántica

Un flujo de objetos es una clase de flujo de control que tiene como entrada o salida un estado de flujo de objeto.

Notación

Los flujos de objetos se representan mediante una línea discontinua que va desde la entidad de origen hasta la entidad destino. Una de las entidades (o ambas) pueden ser estados de flujo de control, que se mostrarán como símbolos de objeto. La flecha puede tener una palabra re-

servada para indicar de qué clase de flujo de objetos se trata (become o copy). Si no hay una etiqueta en la flecha, se trata de una relación de conversión.

Véase estado de flujo de objeto, become y copy para ver ejemplos.

foco de control

Un símbolo en un diagrama de secuencia que muestra el período de tiempo durante el cual un objeto está realizando una acción, directamente o a través de un procedimiento subordinado.

Véase activación.

generalización

Una relación taxonómica general entre un elemento más general y un elemento más específico. El elemento más específico es completamente consistente con el elemento más general y contiene información adicional. Una instancia de un elemento más específico puede ser utilizada donde se permita el elemento más general.

Véase también generalización de la asociación, herencia, principio de sustitución, generalización de casos de uso.

Semántica

Una relación de generalización es una relación dirigida entre dos elementos generalizables del mismo tipo, tales como clases, paquetes, u otras clases de elementos. Un elemento se llama padre, y el otro se llama hijo. Para las clases, se llama al padre superclase y al hijo subclase. El padre es la descripción de un conjunto de instancias (indirectas) con las características comunes de todos los hijos; el hijo es una descripción de un subconjunto de esas instancias que tengan las características del padre pero que también tengan propiedades adicionales peculiares al hijo.

La generalización es una relación transitiva, antisimétrica. Una dirección de la relación conduce al padre; la otra dirección conduce al hijo. Un elemento relacionado en la dirección del padre a través de una o más generalizaciones se llama antecesor; un elemento relacionado en la dirección del hijo a través de una o más generalizaciones se llama descendiente. No se permite ninguno ciclo dirigido en la generalización. Una clase no puede tenerse a sí misma por antecesor o descendiente.

En el caso más simple, una clase (o el otro elemento generalizable) tiene un solo padre. En una situación más complicada, un hijo puede tener más de un padre. El hijo hereda la estructura, el comportamiento, y restricciones de todos sus padres. Esto se llama herencia múltiple (puede ser llamado más correctamente generalización múltiple). Un elemento del hijo se refiere a su padre y debe tener visibilidad sobre él.

La generalización se puede aplicar a las asociaciones, así como a los clasificadores, a los estados, a los eventos, y a las colaboraciones.

Para el uso de la generalización a las asociaciones, *véase* la generalización de asociaciones.

Para el uso de la generalización para casos de uso, *véase* la generalización de casos de uso.

Los nodos y los componentes son, en gran medida, como las clases, y la generalización aplicada a ellos se comporta igual que lo hace para con las clases.

Restricciones

Una restricción se puede aplicar a un conjunto de relaciones de generalización y sus hijos que comparten a un padre común. Se pueden especificar las propiedades siguientes.

disjoint	**disjunto**	Ningún elemento puede tener dos hijos en el conjunto como antecesores (en una situación de herencia múltiple). Ninguna instancia puede ser una instancia directa o indirecta de dos de los hijos (en una semántica múltiple de la clasificación).
overlapping	**solapado**	Un elemento puede tener dos o más hijos en el conjunto de antecesores. Una instancia puede ser una instancia de dos o más hijos.
complete	**completo**	Todos los hijos posibles se han enumerado en el conjunto y no puede ser agregado ninguno más.
incomplete	**incompleto**	No se han enumerado todavía todos los hijos posibles en el conjunto. Se esperan más hijos o se conocen pero no se han declarado todavía.

Notación

La generalización entre clasificadores se representa como una línea continua desde el elemento hijo (tal como una subclase) al elemento de padre (tal como una superclase), con un triángulo hueco grande en el extremo de la línea donde se une el elemento más general (Figura 13.109). Las líneas al padre se pueden combinar para producir un árbol (Figura 13.110).

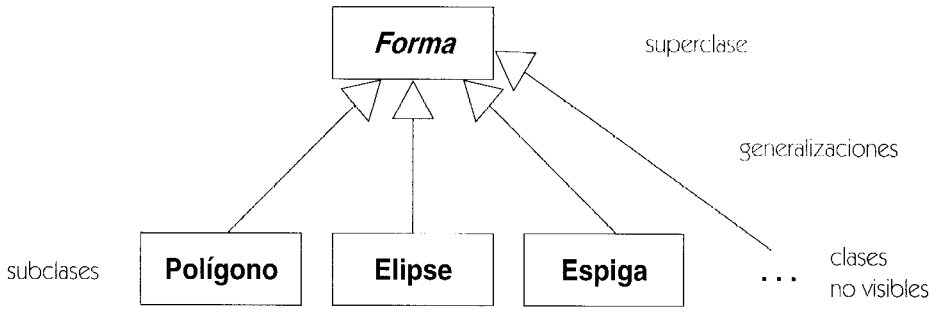


Figura 13.109 Generalización

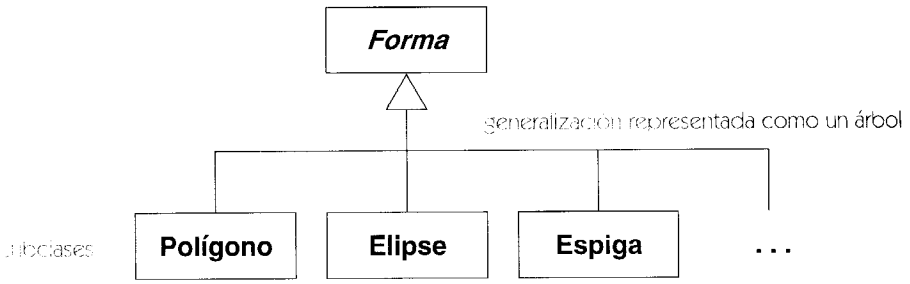


Figura 13.110 Generalización usando un estilo de árbol

La generalización se puede aplicar a las asociaciones, así como a clasificadores, aunque la notación puede ser confusa debido a las líneas múltiples. Una asociación se puede mostrar como clase de asociación con el fin de unir flechas de generalización.

La existencia de clases adicionales en el modelo que no se muestran en un diagrama se puede mostrar usando puntos suspensivos (...) en lugar de una clase. (Note que esto no es una indicación de que pudieran agregarse en el futuro clases adicionales. Indica que existen las clases adicionales ahora pero no se están mostrando. Ésta es una convención de escritura que significa que se ha suprimido la información. No es un elemento semántico.) La presencia de puntos suspensivos (...) como un nodo de subclase de una clase indica que el modelo semántico contiene por lo menos una subclase de la clase que no es visible en el diagrama actual. Los puntos suspensivos pueden tener un discriminador. Este indicador está pensado para ser mantenido automáticamente por una herramienta de edición, no para ser introducido manualmente.

Opciones de representación

Un grupo de líneas de generalización para una superclase dada se puede mostrar como árbol con un segmento compartido (triángulo incluido) a la superclase, ramificando en líneas múltiples a cada subclase. Esto es simplemente un dispositivo notacional. No indica una relación de orden *n*. En el modelo subyacente, hay una generalización para cada par subclase-superclase. No hay diferencia semántica si los arcos se dibujan por separado.

Si una etiqueta del texto se pone en un triángulo de la generalización compartido por varias líneas de la generalización a las subclases, la etiqueta se aplica a todas las líneas. Es decir, todas las subclases comparten las propiedades dadas.

Ejemplo

La Figura 13.111 muestra la declaración de restricciones en generalizaciones. También ilustra el uso del estilo árbol de la generalización, en el cual las líneas se dibujan en una rejilla ortogonal y comparten una flecha común del padre, así como el estilo binario, en el cual cada relación padre-hijo tiene su propia flecha oblicua.

Figura 13.111 Restricciones de generalización

Discusión

El elemento del padre en una relación de generalización se puede definir sin el conocimiento de los hijos, pero los hijos deben saber generalmente la estructura de sus padres para trabajar correctamente. En muchos casos, sin embargo, se diseñan para ser extendidos por los hijos y se incluye al padre cierto conocimiento de los hijos previstos. Una de las glorias de la generalización, sin embargo, es que se descubre que los nuevos hijos a menudo no habían sido anticipados por el diseñador del elemento padre, conduciendo a una expansión que es compatible en esencia con la intención original.

La relación de realización es como una generalización en la cual solamente se hereda la especificación del comportamiento en vez de la estructura o implementación. Si el elemento de la especificación es una clase abstracta sin atributos ni asociación, y con operaciones solamente abstractas, entonces la generalización y la realización son casi equivalentes, ya que no hay nada que heredar excepto la especificación del comportamiento. Sin embargo, observe que la realización no describe realmente al cliente; por lo tanto, las operaciones deben estar en el cliente o heredadas de algún otro elemento.

Elementos estándar

complete, disjoint, implementation, incomplete, overlapping.

generalización de asociaciones

Relación de generalización entre dos asociaciones.

Véase también asociación, generalización.

Semántica

Está permitida la generalización entre asociaciones, aunque es poco frecuente. Como en toda relación de generalización, el elemento hijo debe añadir algo a la declaración (reglas de definición) del padre y debe definir un subconjunto del conjunto de instancias del padre. Añadir algo a la declaración significa añadir restricciones adicionales. Una asociación hija es más restringida que su padre. Por ejemplo, en la Figura 13.112, si la asociación padre conecta las clases **Tema** y **Símbolo**, entonces una asociación hija podría conectar las clases **Orden** y **SímboloOrden**, donde **Orden** es hija de **Tema** y **SímboloOrden** hija de **Símbolo**. Ser un subconjunto del conjunto de instancias del padre significa que todos los enlaces de la asociación hija son también enlaces de la asociación padre, pero no a la inversa. El ejemplo sigue esta regla. Un enlace que conecte **Orden** y **SímboloOrden** conectará también **Tema** y **Símbolo**, pero no todos los enlaces que conecten **Tema** y **Símbolo** conectarán necesariamente **Orden** y **SímboloOrden**.

Notación

La asociación hija se conectará con la asociación padre utilizando un símbolo de generalización (una flecha con línea continua y punta triangular hueca). La punta de la flecha estará en la asociación padre. Debido a que las líneas conectan otras líneas, la notación de generalización de asociaciones puede ser confusa y debe usarse con cuidado.

Ejemplo

La Figura 13.112 muestra dos especializaciones de la asociación general **modelo-vista** entre **Tema** y **Símbolo**: la asociación entre **Cliente** y **SímboloCliente** y la asociación **Pedido** y **SímboloPedido**. Cada una de ellas conecta una clase **Tema** con una clase **Símbolo**. La asociación general **Tema-Símbolo** puede verse como una asociación abstracta en la cual las dos asociaciones hijas son concretas.

Este patrón en que dos jerarquías de clases están conectadas por asociaciones es bastante común.

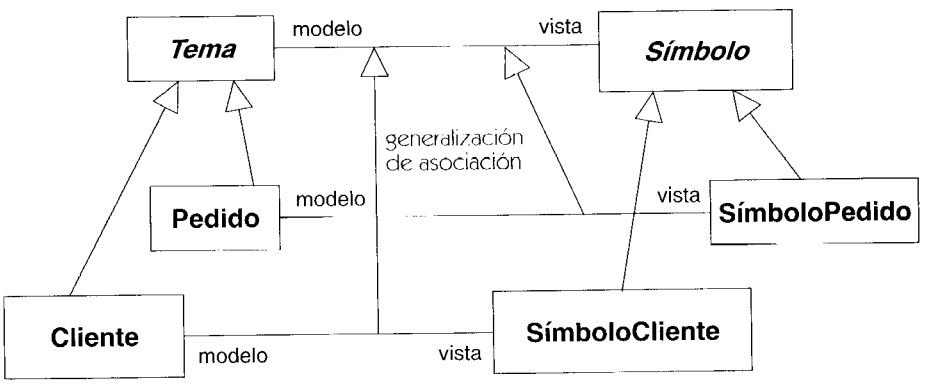


Figura 13.112 Generalización de asociaciones

Elementos estándar

destroyed.

generalización de casos de uso

Una relación taxonómica entre un caso de uso (el hijo) y el caso de uso (el padre) que escribe las características que comparte el hijo con otros casos de uso que tengan el mismo padre. Se trata de una generalización aplicable a los casos de uso.

Semántica

Un caso de uso padre puede especializarse en uno o más casos de uso hijos que representan formas más específicas del padre (Figura 13.113). Un hijo hereda todos los atributos, operaciones y relaciones de su padre, porque un caso de uso es un clasificador. La implementación de una operación heredada puede anularse en una colaboración que realice algún caso de uso hijo.

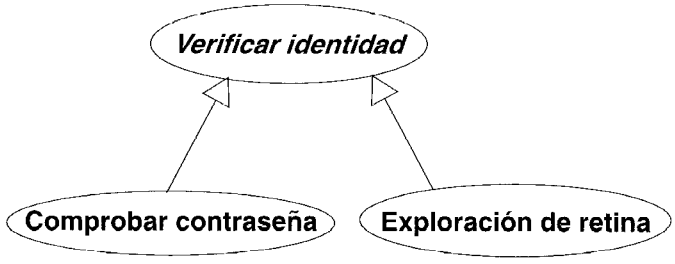


Figura 13.113 Generalización de caso de uso

El hijo hereda la secuencia de comportamiento del padre y puede insertar en ella algún comportamiento adicional (Figura 13.114). El padre y el hijo son potencialmente instanciables.

Comportamiento de caso de uso para el padre, **Verificar Identidad**:

El padre es abstracto, no hay secuencia de comportamiento.
Un descendiente concreto tiene que proporcionar el comportamiento, según se muestra más abajo.

Comportamiento de caso de uso para el hijo, **Comprobar Contraseña**:

Obtener contraseña en base de datos maestra
Pedir contraseña al usuario
El usuario proporciona la contraseña
Comparar contraseña con entrada del usuario

Comportamiento de caso de uso para el hijo, **Exploración de retina**:

Obtener signatura retinal en base de datos maestra
Explorar la retina del usuario y obtener su signatura
Comprar la signatura maestra con la signatura explorada

Figura 13.114 Secuencias de comportamiento para casos de uso hijo y padre

(si no son abstractos) y las diferentes especializaciones del mismo padre son independientes, a diferencia de una relación de extensión en la cual las extensiones múltiples modifican implícitamente un mismo caso de uso base. Es posible agregar comportamiento al caso de uso hijo añadiendo pasos a la secuencia de comportamiento heredada del padre, así como declarando relaciones de extensión y de inclusión para el hijo. Si el padre es abstracto, su secuencia de comportamiento puede tener secciones que sean explícitamente incompletas en el padre y que debe proporcionar el hijo. El hijo puede modificar pasos heredados del padre, pero tal como sucede al redefinir métodos, es preciso utilizar con cuidado esta capacidad porque debe mantenerse la intencionalidad del padre.

La relación de generalización conecta un caso de uso hijo con un caso de uso padre. Un caso de uso padre puede acceder a los atributos definidos por el caso de uso padre; puede incluso modificarlos.

La capacidad de sustitución para los casos de uso significa que la secuencia de comportamiento de un caso de uso hijo tiene que incluir la secuencia de comportamiento de su padre. Sin embargo, no tienen por qué ser contiguos los pasos en la secuencia del padre; el hijo puede intercalar pasos adicionales entre los pasos de la secuencia de comportamiento heredados del padre.

El uso de herencia múltiple en casos de uso requiere la especificación explícita de la forma en que se intercalan las secuencias de comportamiento de los padres para crear la secuencia correspondiente al hijo.

La generalización de casos de uso puede hacer uso de la herencia privada para compartir la implementación de un caso de uso base sin tener una capacidad de sustitución completa, pero esta capacidad debe emplearse con parquedad.

Notación

Se emplea el símbolo normal de generalización —una línea continua que va del hijo al padre con un cabeza de flecha triangular hueca en la línea que toca al símbolo del padre.

Ejemplo

La Figura 13.113 muestra el caso de uso abstracto **Verificar Identidad** y su especialización como dos casos de uso concretos, cuyo comportamiento se muestra en la Figura 13.114.

grafo de actividades

Caso especial de una máquina de estados en el cual todos o la mayoría de los estados son estados de actividad o estados de acción, y en el que la mayoría de las transiciones son disparadas por la finalización de una actividad en los estados origen. Un grafo de actividades muestra un procedimiento o un flujo de trabajo. Un grafo de actividades es una unidad completa en el modelo, mientras que un diagrama de actividades es un diagrama que muestra un grafo de actividades.

Véase también máquina de estados.

Semántica

Un grafo de actividades es una máquina de estados que enfatiza los pasos secuenciales y concurrentes de un procedimiento de computación. Los flujos de trabajo son ejemplos de procedimientos que a menudo se modelan utilizando grafos de actividades. Los grafos de actividades generalmente aparecen en las primeras etapas del diseño, antes de que se tomen las decisiones de implementación, y en particular, antes de que se les hayan asignado a los objetos todas las tareas a realizar. Este tipo de grafo es una variante de una máquina de estados en la cual un estado representa la realización de una actividad, como un cómputo o una operación continua del mundo real, y las transiciones se disparan como consecuencia de la finalización de operaciones. Un grafo de actividades puede asociarse a la implementación de una operación o de un caso de uso.

En un grafo de actividades los estados son principalmente estados de actividad o estados de acción. Un estado de actividad representa un estado con un cómputo interno y al menos una transición de finalización saliente que se dispara cuando finaliza la actividad del estado. Puede haber varias transiciones de salida si tienen condiciones de guarda. Los estados de actividad no deberían tener transiciones internas o transiciones de salida basadas en eventos explícitos: utilice estados normales en estas situaciones. Un estado de acción es un estado atómico, es decir, un estado que no puede ser interrumpido por transiciones, ni siquiera aquéllas de sus estados adyacentes.

El uso típico de un estado de actividad es modelar un paso en la ejecución de un procedimiento. Si todos los estados de un modelo son estados de actividad, entonces el resultado de la computación no dependerá de eventos externos. La computación es determinista si las actividades concurrentes no acceden a los mismos objetos y los tiempos relativos de finalización de las actividades concurrentes no afectan a los resultados.

Los grafos de actividades pueden incluir estados de espera ordinarios cuya existencia se dispara gracias a eventos, pero este uso frustra el propósito de centrar la atención en las actividades: utilice un modelo de estados ordinario si tiene más de unos pocos estados ordinarios.

Concurrencia Dinámica. Un estado de actividad con concurrencia dinámica representa la ejecución concurrente de múltiples cómputos independientes. La actividad se invoca con un conjunto de listas de argumentos, de forma que cada miembro del conjunto es la lista de argumentos de una invocación a la actividad. Las invocaciones son independientes unas de otras. Cuando se han completado todas las invocaciones la actividad finaliza y dispara su transición de finalización.

Flujo de objetos. A veces es útil observar las relaciones entre una operación y aquellos objetos que recibe como argumentos o que devuelve como resultado. La entrada de una operación y la salida de la misma se pueden mostrar como un estado de flujo de objetos, lo que constituye un estereotipo de un estado que representa la existencia de un objeto de una clase dada en un punto concreto del cómputo. Para mayor precisión, se puede declarar que el objeto de entrada o de salida esté en un estado determinado dentro de su clase. Por ejemplo, la salida de una operación "firmar contrato" será un estado de flujo de objetos de la clase Contrato en el estado "firmado". Este estado de flujo de objetos puede ser la entrada de muchas otras operaciones.

Calles. Las actividades de un grafo de actividades se pueden separar en particiones según diversos criterios. Cada partición representará una división significativa de las responsabilidades de las actividades, por ejemplo, las responsabilidades de negocio de la organización en un paso de un flujo de trabajo dado. A las particiones se les llama calles debido a la notación gráfica utilizada para su representación.

Notación

Un grafo de actividades se representa mediante un diagrama de actividades. Los grafos de actividades son una variante de la máquina de estados, pero hay algunos aspectos de la notación que son particularmente aplicables a los diagramas de actividades: estados de actividad, bifurcaciones, uniones, calles, estados de flujo de objetos, clases en un estado, notación de recepción y envío de señales y eventos diferidos. Véanse estas entradas para más detalles.

Véase iconos de control para más información sobre algunos símbolos opcionales que pueden ser útiles en los diagramas de actividades.

Ejemplo

La Figura 13.115 muestra un flujo de trabajo de las actividades a realizar en el procesamiento de un pedido en una taquilla de teatro. Incluye una bifurcación y la subsiguiente unión basada

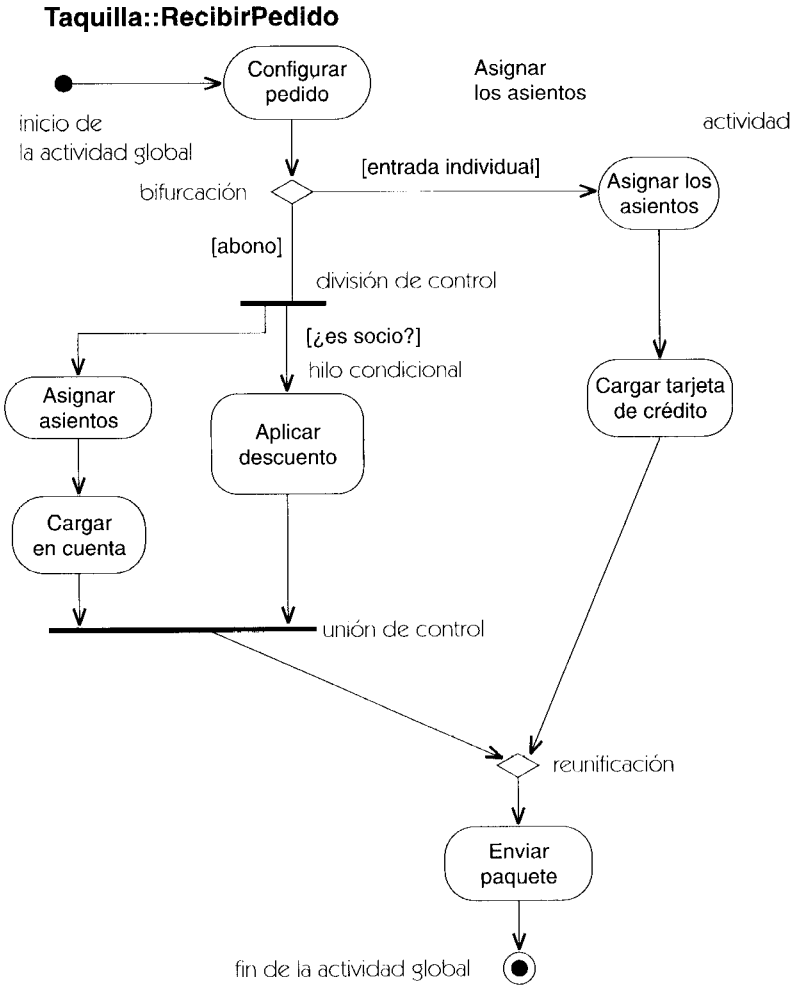


Figura 13.115 Diagrama de actividades

en si el pedido es para un abono o para entradas individuales. La división inicia las actividades concurrentes que lógicamente se producen al mismo tiempo, y cuya ejecución real puede o no solaparse en el tiempo. La concurrencia termina en una unión al mismo nivel. Si sólo hay una persona involucrada, las actividades concurrentes pueden realizarse en cualquier orden (suponiendo que no puedan realizarse a la vez, algo que permite el modelo pero que es difícil en la práctica). Por ejemplo, el encargado de la taquilla podría asignar los asientos, luego aplicar el descuento y luego cargar el importe en la cuenta; o aplicar el descuento, luego asignar los asientos y por último cargar el importe en la cuenta, pero no debería cargar el importe sin antes haber asignado los asientos.

Un segmento de salida de la división tiene una condición de guarda que comprueba si el abonado es socio. Esto es un hilo condicional que se ejecuta sólo si se satisface la condición de guarda. Si no se ejecuta el hilo, el segmento de entrada al siguiente punto de unión se considera completado. Si el abonado no es socio sólo se inicia un hilo, ya que se le asignan asientos y se le carga el importe en su cuenta pero no se le hace descuento.

Calles. Las actividades de un grafo de actividades se pueden separar en regiones a las que se llama calles por su representación gráfica, ya que se dibujan como regiones diferentes de un diagrama separadas por líneas discontinuas. Una calle sirve para organizar los contenidos de un grafo de actividades. No tiene un significado inherente, por lo que se puede utilizar como lo desee el que realiza el modelado. A menudo cada calle representa una unidad organizativa dentro de una organización del mundo real.

Ejemplo

En la Figura 13.116 las actividades están divididas en tres particiones mediante calles, cada una de las cuales corresponde a un usuario. UML no establece ningún requisito sobre si las particiones deben corresponderse con objetos, aunque en este ejemplo hay clases obvias que encajarían perfectamente en cada partición, y esas clases serían las únicas que realizarían las operaciones para implementar cada actividad en el modelo final.

La figura también muestra el uso de símbolos de flujo de objetos. Los flujos de objetos corresponden a los diferentes estados por los que pasa un objeto Pedido a través de toda la red de actividades que realiza. El símbolo **Pedido[ubicado]**, por ejemplo, significa que en ese lugar del cómputo el pedido ha avanzado hasta el estado **ubicado** en la actividad **Solicitar-Servicio** pero aún no ha sido consumido por la actividad **RecogerPedido**. Cuando se completa la actividad **RecogerPedido**, el pedido pasa al estado **introducido**, como se muestra en el diagrama mediante el símbolo de flujo de objetos situado en la salida de la actividad **RecogerPedido**. Todos los flujos de objetos de este ejemplo representan el mismo objeto en momentos diferentes de su vida, y precisamente por representar el mismo objeto, no pueden existir al mismo tiempo. Se puede dibujar entre ellos una ruta de control secuencial, como se ve en el diagrama.

Flujo de objetos. Los objetos que son salida o entrada de una acción se pueden representar mediante símbolos de objeto. El símbolo representa al objeto en el momento del cómputo en el que se produce la entrada o en el que se realiza la salida (normalmente un objeto hace las dos cosas). Se dibuja una flecha discontinua que va desde la transición de salida de un estado de actividad hasta un flujo de objetos que es una de sus salidas, y otra flecha discontinua desde el

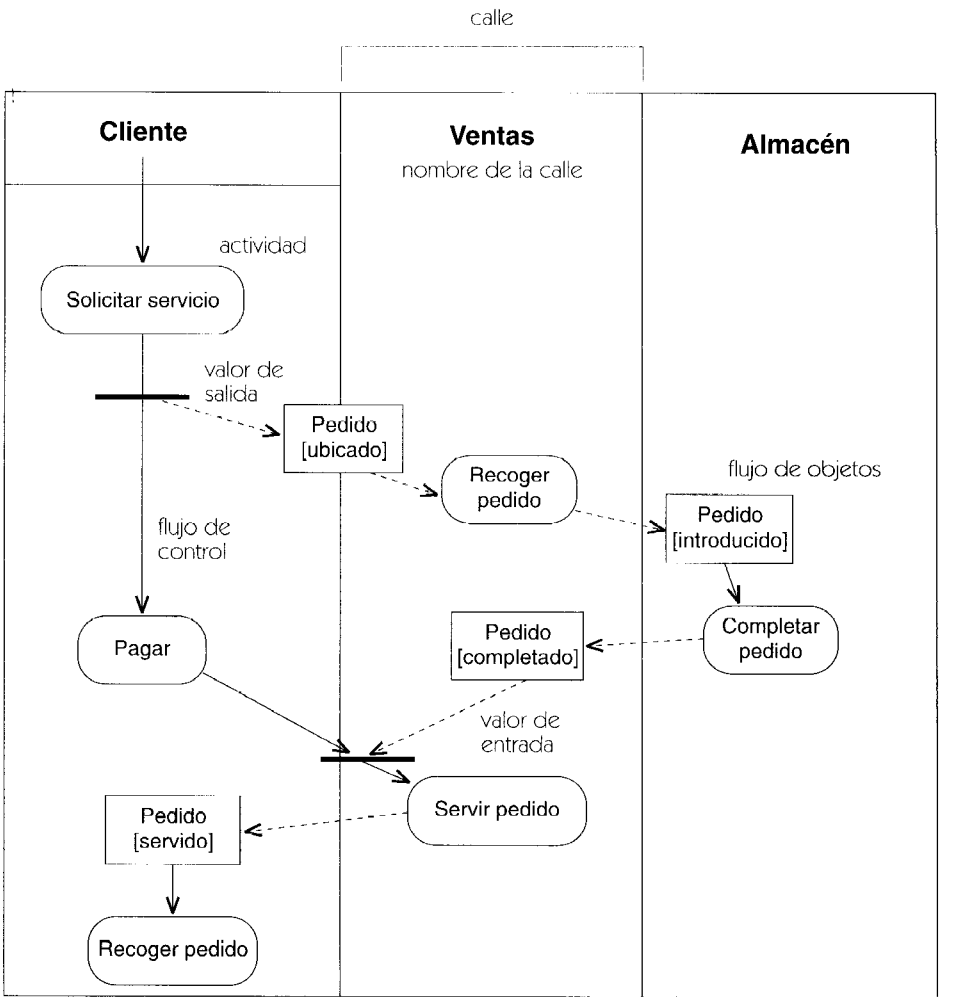


Figura 13.116 Diagrama de actividades con calles

flujo de objetos hasta la transición de entrada de un estado de actividad que utiliza el objeto como entrada. El mismo objeto puede ser, y normalmente es, salida de una actividad y entrada de una o más actividades posteriores.

Las flechas de flujo de control (continuas) pueden omitirse cuando las flechas de flujo de objetos (discontinuas) proporcionan una restricción redundante. En otras palabras: cuando una acción produce una salida que es entrada de una acción posterior, la relación de flujo de objetos implica una restricción de control.

Véase estado de flujo de objetos.

Clase en un estado. Es frecuente que unas cuantas actividades sucesivas manipulen el mismo objeto para cambiar su estado. Precisando aún más, se puede representar el objeto muchas veces en un diagrama, de forma que cada una de las apariciones del mismo objeto represente un estado diferente a lo largo de su vida. Para distinguir entre las distintas apariciones de un mismo objeto, el estado del objeto en cada punto se puede poner entre corchetes a la

derecha del nombre de la clase, por ejemplo, **PeticiónCompra[aprobado]**. Esta notación puede utilizarse también en diagramas de colaboración.

Véase también iconos de control, donde se muestran otros símbolos que se pueden utilizar en los diagramas de actividades.

Eventos diferidos. A veces existe un evento cuya existencia debe ser pospuesta para más tarde mientras se ejecuta alguna otra actividad (normalmente un evento que no se trata inmediatamente, se pierde). Un evento diferido es un evento que se sitúa en una cola interna hasta que se utiliza o hasta que es descartado. Cada estado o actividad pueden especificar el conjunto de eventos que serán diferidos si se producen durante el estado o la actividad. Los demás eventos se tratarán inmediatamente para no perderlos. Cuando la máquina de estados entra en un nuevo estado, se producen todos los eventos diferidos a menos que el nuevo estado también los marque como diferidos. Si un evento que fue diferido en el estado anterior dispara una transición del nuevo estado, la transición se ejecuta inmediatamente. Si son varias las transiciones potenciales, queda indefinido cuál de ellas se ejecutará; si se impone una regla para seleccionar una se estará introduciendo una pequeña variación de la semántica.

Si se produce un evento durante un estado para el cual está diferido y dicho evento dispara una transición, no se situaría en la cola. Si no dispara ninguna transición, se sitúa en la cola. Si cambia el estado activo, los eventos de la cola pueden disparar transiciones en el nuevo estado pero permanecerán en la cola si siguen siendo eventos diferidos en el nuevo estado. La capacidad de quitarle la marca de diferido a un evento para que dispare una transición, es útil sobre todo en un estado compuesto donde el evento debe ser diferido pero que a su vez debe permitir transiciones en uno o más subestados. A parte de eso, tendría que ser diferido en cada uno de los subestados en los que no dispara transiciones. Obsérvese que si un evento diferido debe disparar una transición pero no se cumple la condición de guarda, el evento no ha disparado la transición y por tanto no puede eliminarse de la cola.

Gráficamente, un evento que puede diferirse se lista dentro del estado, seguido por una barra inclinada y la operación especial **defer** (diferir). Si se produce el evento y no dispara ninguna transición, se almacena y vuelve a producirse cuando el objeto inicie una transición hacia otro estado, y en ese momento puede ser diferido de nuevo. Cuando el objeto llega a un estado en el que el evento no está diferido, debe ser aceptado o si no será ignorado. La ausencia de una sentencia **defer** tiene el efecto de cancelar una marca de diferido previa. La indicación **defer** se puede poner en un estado compuesto, en cuyo caso el evento es diferido a lo largo de todo el estado compuesto.

Los estados de acción son atómicos, por lo que difieren implícitamente todos los eventos que ocurren durante el tiempo en que están activos, por lo que no es necesario marcarlos como diferidos. Cualquier evento que se produzca pasa a diferido hasta que se complete la acción, momento a partir del cual el evento podría disparar una transición.

Ejemplo

La Figura 13.117 muestra los pasos necesarios para hacer café. En este ejemplo, no se muestra el objeto externo (**cafetera**) sino sólo las actividades realizadas directamente por la persona que va a hacer el café. El acto de encender la cafetera se modela como un evento que se envía a la cafetera. La actividad **Tomar Tazas** ocurre después de encender la cafetera. Después de tomar las tazas hay que esperar a que se apague la luz de la cafetera. Sin embargo, hay un problema.

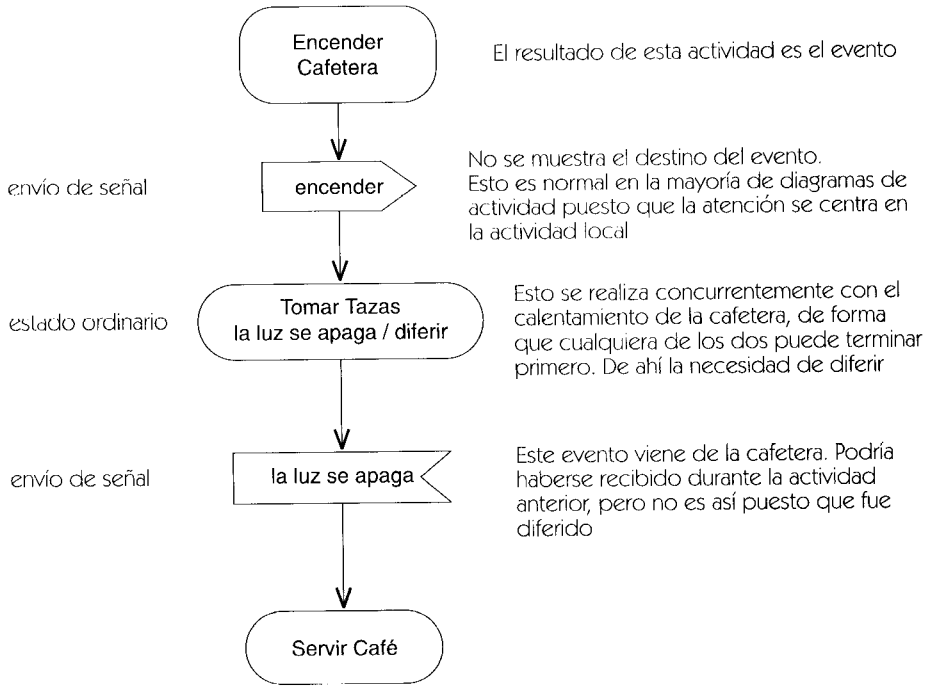


Figura 13.117 Eventos diferidos e iconos de control

Si el evento **la luz se apaga** se produce antes de que termine la actividad **Tomar Tazas**, el evento se pierde porque la máquina de estados aún no está preparada para tratarlo. Para evitar que se pierda un evento, en el estado de actividad **Tomar Tazas** se ha marcado como diferido el evento **la luz se apaga**. Si el evento se produce mientras se está ejecutando la actividad, no se perderá sino que se almacenará en una cola de espera hasta que la máquina de estados abandone el estado **Tomar Tazas**, momento en el cual se procesará y luego se disparará la transición.

Obsérvese que el evento **la luz se apaga** no es un disparador del estado **Tomar Tazas**, y sin embargo si se produce la actividad no termina. El evento es un disparador del estado de recepción de señal inmediatamente posterior a la finalización de la actividad **Tomar Tazas**.

Concurrencia dinámica. La concurrencia dinámica con un valor distinto de uno se representa mediante una cadena de multiplicidad en la parte superior derecha del símbolo de la actividad (Figura 13.118). Esto indica que las múltiples copias de la actividad se ejecutan concurrentemente. La actividad dinámica recibe un conjunto de listas de argumentos. Los detalles se deben describir mediante texto. Si se aplica la concurrencia a varias actividades, debe encerrarse en una actividad compuesta que lleve el indicador de multiplicidad.

Estructura de grafo. Una máquina de estados debe estar bien anidada, es decir, particionada recursivamente en subestados concurrentes o secuenciales. En un grafo de actividades, las bifurcaciones y divisiones deben estar bien anidadas. Cada bifurcación debe tener su correspondiente reunificación, y cada división su correspondiente unión, pero eso no siempre es lo más conveniente en un grafo de actividades. Es bastante común encontrarse con un grafo parcialmente ordenado, en el cual no hay grafos dirigidos pero las bifurcaciones y divisiones no coinciden necesariamente. Un grafo así no está bien anidado, pero puede transformarse en un

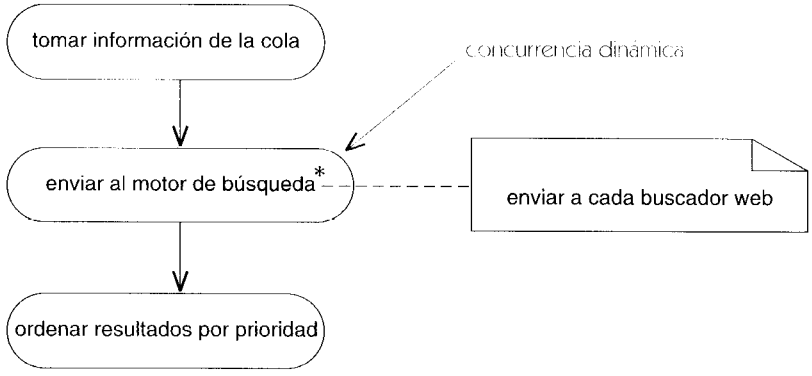


Figura 13.118 Concurrencia dinámica

grafo bien anidado asignando actividades a hilos e introduciendo estados de sincronización cuando las transiciones crucen la frontera de los hilos. La descomposición no es única, pero en un grafo parcialmente ordenado todas las descomposiciones posibles tienen la misma semántica, y por lo tanto no es necesario representar una descomposición explícita o estados de sincronización en un grafo de este tipo. En grafos más complicados que contienen condiciones y concurrencia podría ser necesario ser más explícito sobre la descomposición.

herencia

Mecanismo mediante el cual unos elementos más específicos incorporan la estructura y el comportamiento definidos por otros elementos más generales.

Véase también descriptor completo, generalización.

Semántica

La herencia permite construir una descripción completa de un elemento generalizable ensamblando fragmentos de declaraciones a partir de una jerarquía de generalización. Una jerarquía de generalización es un árbol (o más exactamente, un orden parcial) de declaraciones de elementos de un modelo, tales como clases. Sin embargo, las declaraciones no son declaraciones de un elemento completo y utilizable. Cada declaración es en realidad una declaración incremental que describe lo que añade el elemento de declaración a las declaraciones de sus antecesores dentro de la jerarquía de generalización. La herencia es el proceso (implícito) de combinación de esas declaraciones incrementales para formar descriptores completos que describen instancias reales.

Se puede pensar que todo elemento generalizable posee dos descripciones: una declaración de segmento y un descriptor completo. La declaración de segmento es la lista incremental de características que declara el elemento en el modelo: los atributos y operaciones que declara una clase, por ejemplo. La declaración de segmento es la diferencia entre el elemento y sus predecesores. El descriptor completo es la unión de los contenidos de las declaraciones de segmento que aparecen en un segmento y en todos sus antecesores.

Esto es la herencia. Se trata de la declaración incremental de un elemento. Hay otros detalles, tales como los algoritmos de búsqueda de métodos, las *vtables* y demás, que son meramente mecanismos de implementación para hacer que funcione en un determinado lenguaje: no forma parte de la definición esencial. Aunque esta descripción pueda parecer extraña a primera vista, está libre de los estorbos de implementación que se encuentran en casi cualquier otra definición, pero es compatible con ellas.

Conflictos

Si hay una misma característica que aparece más de una vez entre el conjunto de segmentos heredados, quizá se produzca un conflicto. No se puede declarar ningún atributo más de una vez en un conjunto heredado. Si esto sucediera, las declaraciones entran en conflicto y el modelo está mal formado. (Esta restricción no es esencial por razones lógicas. Se ha impuesto para evitar cierta confusión que podría producirse si fuera preciso distinguir los atributos mediante rutas de acceso.)

Se podría declarar dos veces la misma operación, siempre y cuando la declaración fuera exactamente la misma (sin embargo, los métodos pueden ser distintos) o bien una declaración descendiente fortalece una declaración heredada (por ejemplo, declarando que un descendiente es una consulta, o incrementando su estado de concurrencia). La declaración de un método en un descendiente sustituye (anula) a la declaración de un método en su antecesor. No hay conflicto. Si se heredan métodos distintos de dos antecesores diferentes que no poseen, a su vez, una relación de antecesor, entonces los métodos entran en conflicto y el modelo está mal formado.

Discusión

La generalización es una relación taxonómica entre elementos. Describe lo que es un elemento. La herencia es un mecanismo para combinar descripciones incrementales compartidas, con objeto de formar una descripción completa de un elemento. No son una misma cosa, aunque están íntimamente relacionadas. El mecanismo de herencia, aplicado a la relación de generalización, permite factorizar y compartir las descripciones y el comportamiento polimórfico. Éste es el enfoque que adopta la mayoría de los lenguajes orientados a objetos y también UML. Pero hay que tener en cuenta que existen otros enfoques que podrían haberse adoptado y que emplean algunos lenguajes de programación.

herencia de implementación

La herencia de la implementación de un elemento predecesor, esto es, de su estructura (tal como los atributos y operaciones) y de su código (tal como sus métodos). Por contraste, la herencia de interfaz implica la herencia de especificaciones de interfaz (esto es, de operaciones) pero no de métodos ni de estructuras de datos (atributos y asociaciones).

El significado normal de la generalización en UML incluye la herencia *tanto* de interfaz como de implementación. Para heredar *solamente* la implementación (herencia privada) se aplica el estereotipo «**implementation**» a una generalización. Para admitir una interfaz sin

comprometerse con una implementación, se le aplica una relación de realización a una interfaz.

Véase también generalización, herencia, herencia de interfaz, herencia privada.

herencia de interfaz

Denota la herencia de la interfaz de un elemento predecesor, pero no de su implementación ni de su estructura de datos. La intención de admitir una interfaz sin comprometerse con una implementación se modela empleando la realización.

Obsérvese que en UML la generalización implica la herencia *tanto* de la interfaz como de la implementación. Para heredar solamente la implementación sin la interfaz, utilice la herencia privada.

herencia múltiple

Denota un punto de generalización de variación semántica en el cual un elemento puede tener más de un predecesor. Se trata de la suposición por defecto en UML y se necesita para el correcto modelado de muchas situaciones, aun cuando los modeladores pueden optar por restringir su utilización para ciertas clases de elementos. Por contraste: herencia simple.

herencia privada

Denota la estructura de herencia en una situación tal que no se hereda la especificación de comportamiento.

Véase también herencia de implementación, herencia de interfaz, principio de capacidad de sustitución.

Semántica

Una generalización puede tener el estereotipo «**implementation**». Esto indica que el elemento cliente (normalmente una clase) hereda la estructura (atributos, asociaciones y operaciones) del elemento proveedor, pero no pone necesariamente esta estructura a disposición de sus propios clientes. Dado que los antecesores de esta clase (u otro elemento) no son visibles para las otras clases, no se puede utilizar una instancia de la clase como variable o parámetro declarado en la clase proveedora. En otras palabras, la clase no se puede sustituir por sus proveedores heredados de forma privada. La herencia privada no sigue el principio de capacidad de sustitución.

Notación

Se denota la herencia privada mediante la palabra reservada «**implementation**» junto a una flecha de generalización procedente del elemento que hereda (el cliente) y que llega al elemento que proporciona la estructura que hay que heredar (el proveedor).

Discusión

La herencia privada es simplemente un mecanismo de implementación y no debería pensarse que se trata de una utilización de la generalización. La generalización requiere la capacidad de sustitución. La herencia privada no es especialmente significativa en un modelo de análisis, que no implica la estructura de implementación. Incluso para la implementación, debería utilizarse con cuidado porque implica una utilización no semántica de la herencia. Suele ser una alternativa más limpia emplear una asociación con la clase proveedora. Hay muchos autores (incluyendo alguno de los presentes) que afirman que no debería emplearse nunca porque hace uso de la herencia de una manera no semántica que resulta peligrosa cuando se hacen cambios en el modelo.

herencia simple

Variación semántica de la generalización en la que un elemento sólo puede tener un padre. Admitir la herencia simple o la herencia múltiple es un punto de variación semántica.

hijo/a

Elemento más específico de una relación de generalización. Cuando se trata de clases se denomina subclase. Una cadena de una o más relaciones hijas se denomina descendiente. Antónimo: padre.

Semántica

Un elemento hijo hereda las características de su padre (e indirectamente las de sus antecesores) y puede declarar características adicionales propias. También hereda todas las asociaciones y restricciones en las que participan sus antecesores. Un elemento hijo cumple el principio de capacidad de sustitución, esto es, una instancia de un descriptor satisface cualquier declaración de variable clasificada como uno de los antecesores del descriptor. Una instancia de un hijo es una instancia indirecta del padre.

hilo

(De *hilo de control*) Una única ruta de ejecución que recorre un programa, modelo dinámico u otra representación de flujo de control. También es un estereotipo para la implementación de un objeto activo como proceso ligero.

Véase objeto activo, transición compleja, estado compuesto, máquina de estados, estado de sincronización.

hilo condicional

Una región del grafo de actividades iniciado por un segmento protegido de salida de una división que termina con un segmento de entrada a la unión correspondiente.

Véase estado compuesto, transición compleja.

hipervínculo

Una conexión invisible entre dos elementos de notación, que puede ser recorrida por algún comando.

Véase también diagrama.

Notación

Una notación escrita en un trozo de papel no contiene información oculta. Sin embargo, una notación en una pantalla de computador puede contener hipervínculos adicionales invisibles que no resultarán visibles desde un punto de vista estático pero pueden invocarse dinámicamente para acceder a alguna otra información, bien en una vista gráfica o en una tabla de texto. Estos enlaces dinámicos forman parte de una notación *dinámica* en idéntica proporción que la información visible, pero este documento no prescribe su forma. Los consideramos como una responsabilidad de la herramienta. Este documento pretende definir una notación *estática* para UML, bien entendido que es posible que una parte útil e interesante de la información no aparezca claramente (o no aparezca de ninguna forma) en una de estas vistas. Por otra parte, no se puede disponer de un conocimiento adecuado para especificar el comportamiento de todas las herramientas dinámicas y tampoco se pretende asfixiar la innovación en las presentaciones dinámicas. Eventualmente, es posible que algunas notaciones dinámicas queden tan firmemente establecidas que sea posible llegar a un estándar, pero no creemos que esto deba hacerse en estos momentos.

hoja

Término que denota un elemento generalizable carente de descendientes en la jerarquía de generalización. Tiene que ser completo (estará completamente implementado) para que pueda servir para algo.

Véase también abstracto, concreto.

Semántica

La propiedad de ser una hoja declara que un elemento *es* una hoja. El modelo estará mal formado si se declara un descendiente de uno de estos elementos. El propósito es garantizar que una clase no pueda ser modificada, por ejemplo, porque el comportamiento de la clase tienen que estar bien establecido para que sea fiable. La declaración de una hoja permite además la compilación por separado de partes de un sistema, asegurando que los métodos no se pueden redefinir y permitiendo copiar el código de esos métodos en vez de realizar llamadas a los mismos (*code inlining*, para la mejora de la velocidad de ejecución). Un elemento en que esta propiedad sea falsa puede ser ciertamente una hoja pero podría tener descendientes en el futuro si se modifica el modelo. Ser una hoja o estar obligado a ser una hoja no son propiedades semánticas fundamentales.

iconos de control

Símbolos opcionales que proporcionan una conveniente notación rápida para los diferentes patrones del control.

Véase también grafo de actividades, colaboración, máquina de estados.

Notación

Los siguientes símbolos están pensados para su uso en diagramas de actividad, pero pueden también ser utilizados en diagramas de estados, si se quiere. Estos símbolos no permiten nada que no se pudiera mostrar usando los símbolos básicos, pero pueden ser convenientes para ciertos patrones de control comunes.

Bifurcación. Una bifurcación es un conjunto de transiciones que salen de un estado simple de tal forma que siempre es satisfecha exactamente una de las condiciones de guarda de una de las trasiciones. Es decir, si ocurre el evento disparador, sólo puede dispararse una transacción. Las condiciones de guarda esencialmente representan una bifurcación del control. Si la transición es una transición de terminación, entonces una bifurcación es una decisión pura. Por conveniencia, una salida de la bifurcación se puede etiquetar con la palabra clave **else**. Se toma esta ruta cuando no es posible tomar ningún otro.

Una bifurcación se denota como un diamante con una flecha de entrada y dos o más flechas de salida. La flecha de la transición de entrada se etiqueta con el evento disparador (si lo hay). Cada salida se etiqueta con una condición de guarda (Figura 13.119).

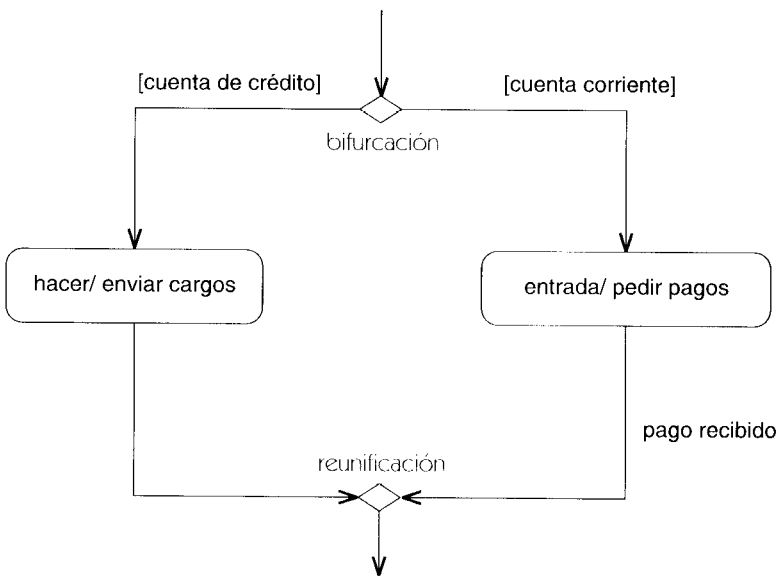


Figura 13.119 Bifurcación y reunificación

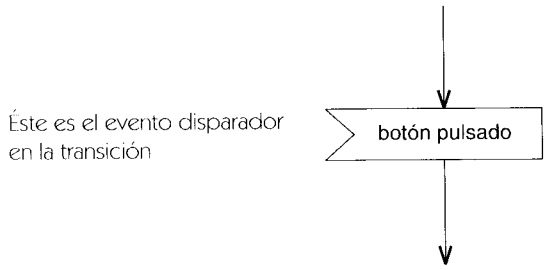


Figura 13.120 Recepción de señal

Reunificación. Una reunificación es un lugar en el cual se unen dos o más flujos de control alternativos. Es lo contrario de una bifurcación. El símbolo de una bifurcación o de una reunificación es un diamante. Es una reunificación si hay flechas múltiples de entrada; es una bifurcación si hay flechas múltiples de salida (Figura 13.119). Las reunificaciones no son estrictamente necesarias (las transiciones múltiples que pasan a un solo estado son una reunificación), sino que pueden ser visualmente útiles para mostrar cómo encajan las bifurcaciones anteriores.

Recepción de señal. La recepción de una señal se puede representar como un pentágono cóncavo que parezca un rectángulo con una muesca triangular en su lado (cualquier lado). La signatura de la señal se muestra dentro del símbolo. Se dibuja una flecha de transición sin etiqueta desde el estado anterior de la acción al pentágono, y otra flecha de transición sin etiqueta desde el pentágono al siguiente estado de acción. Este símbolo sustituye a la etiqueta del evento en la transición, que se dispara cuando la actividad anterior ha sido completada y ocurre el evento (Figura 13.120). Opcionalmente, se puede dibujar una flecha de líneas discontinuas desde un símbolo de objeto a la muesca en el pentágono para mostrar el remitente de la señal.

Envío de señal. El envío de una señal se puede representar como un pentágono convexo que parezca un rectángulo con un triángulo en un lado (cualquier lado). La signatura de la señal se muestra dentro del símbolo. Se dibuja una flecha de transición sin etiqueta desde el estado anterior de la acción al pentágono, y de otra flecha de transición sin etiqueta desde el pentágono al siguiente estado de acción.

Este símbolo sustituye la etiqueta de enviar señal en la transición (Figura 13.121). Opcionalmente, se puede dibujar una flecha con líneas discontinuas desde el extremo del pentágono a un símbolo de objeto para mostrar el receptor de la señal.

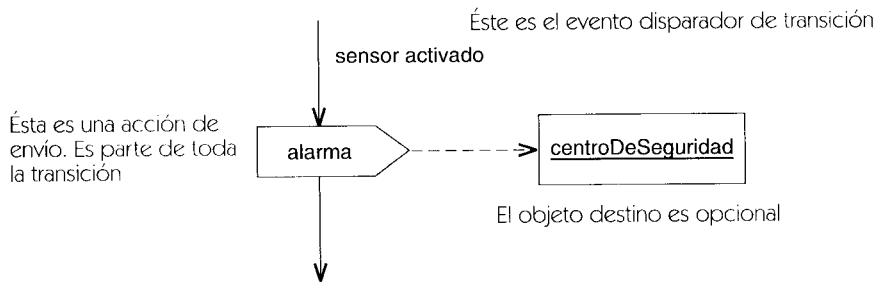


Figura 13.121 Envío de señal

Ejemplo

En la Figura 13.122, **Introducir Datos Tarjeta Crédito** y **Cargar Tarjeta** son actividades. Cuando se terminan, el proceso se mueve al paso siguiente. Después de que **Introducir Datos Tarjeta Crédito** finalice, hay una bifurcación en la cantidad de la petición; si es mayor de 25$, se debe obtener una autorización. Se envía al centro de crédito una señal de **petición**. En una máquina de estados, esto sería representado como una acción unida a la transición que sale de **Introducir Datos Tarjeta Crédito**; ambas significan la misma cosa. Sin embargo **Esperar Autorización** es en realidad un estado de espera. No es una actividad que termina internamente. En su lugar, él debe esperar una señal externa del centro de crédito (autorización).

Cuando ocurre la señal, se dispara una transición normal y el sistema pasa a la actividad **Cargar Tarjeta**. El evento disparador se habría podido mostrar como una etiqueta en la transición de **Esperar Autorización** a **Cargar Tarjeta**. Es simplemente una variante de la notación que significa la misma cosa.

La Figura 13.123 muestra el mismo ejemplo, sin los símbolos de control especiales.

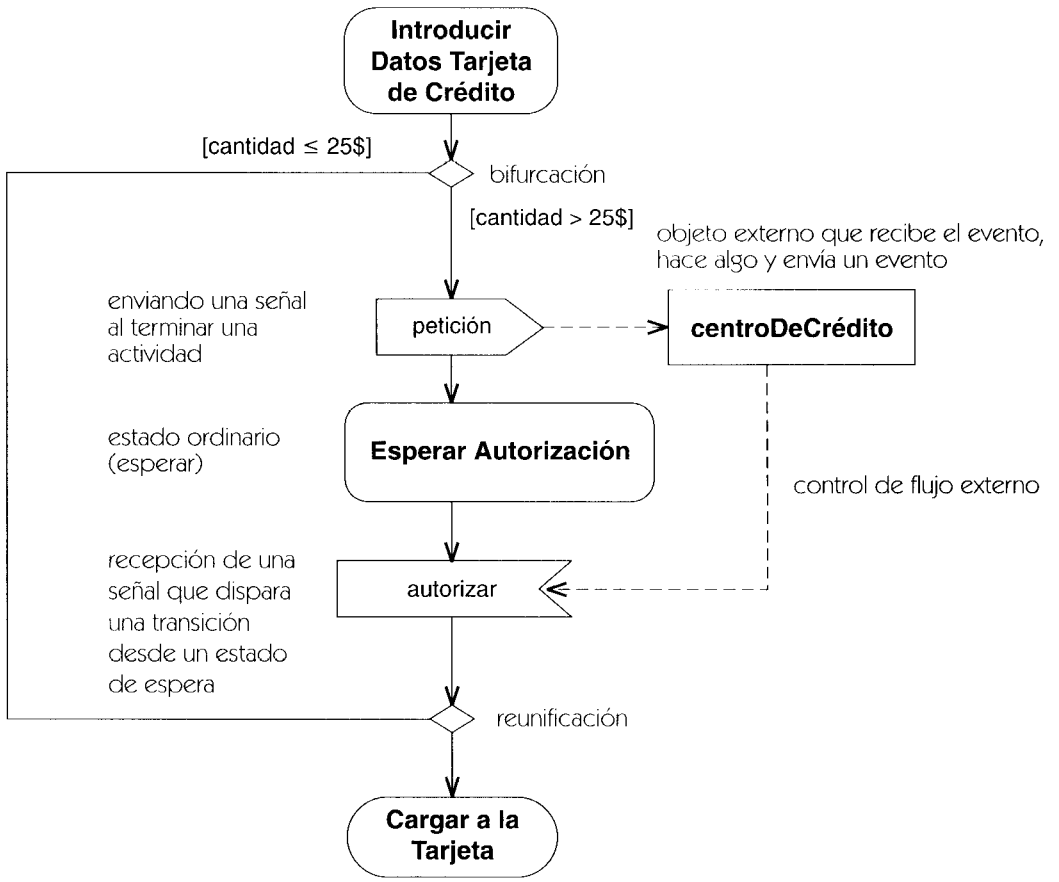


Figura 13.122 Diagrama de actividad mostrando el envío y recepción de señales

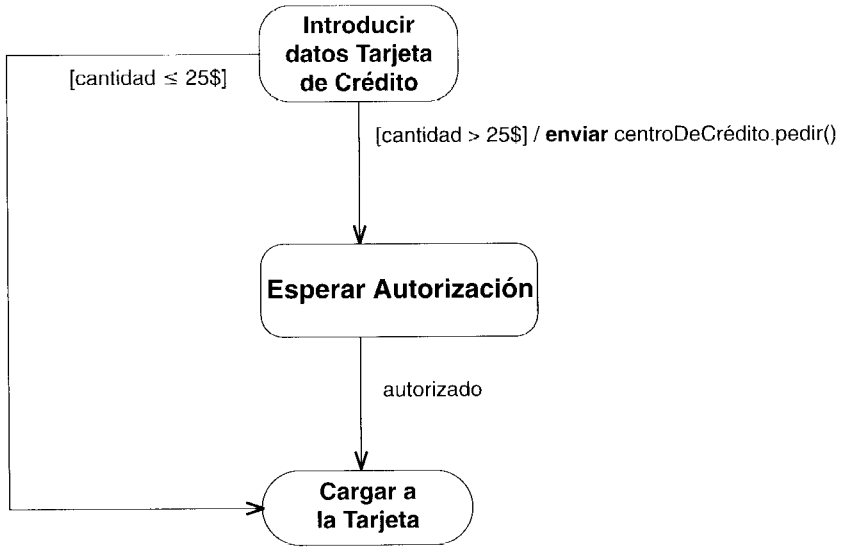


Figura 13.123 Diagrama de actividad sin símbolos especiales

identidad

Propiedad inherente de un objeto que permite distinguirlo de todos los demás objetos.

Véase también valor dato, objeto.

Semántica

Los objetos son discretos y se pueden distinguir entre sí. La identidad de un objeto es su clave conceptual, la característica inherente que permite identificar ese objeto y que otros objetos hagan referencia a él. Conceptualmente, un objeto no precisa una clave ni mecanismo adicional alguno para identificarse a sí mismo; estos mecanismos no deberían estar incluidos en los modelos. En una implementación la identidad se puede implementar mediante direcciones o claves pero éstas forman parte de la infraestructura de implementación subyacente y no es preciso incluirlos explícitamente como atributos en casi todos los modelos.

implementación

1. Una definición de la forma en que se construye o calcula algo. Por ejemplo, una clase es una implementación de un tipo; un método es una implementación de una operación. Compárese con especificación. La relación de realización es la que relaciona a una implementación con su especificación.

Véase realización.

2. Aquella fase de un sistema que describe el funcionamiento del sistema en un medio ejecutable (tal como un lenguaje de programación, una base de datos o algún hardware digital). Para la implementación es preciso hacer que las decisiones tácticas de bajo nivel adapten el diseño al medio concreto de implementación y además hay que soslayar sus limitaciones (todos los lenguajes tendrán limitaciones arbitrarias). Sin embargo, si el diseño está bien hecho entonces las decisiones de implementación serán locales y ninguna afectará a grandes segmentos del sistema. Esta fase se captura mediante modelos del nivel de implementación y especialmente en las vistas estática y de código. Compárese con análisis, diseño, implementación y despliegue.

Véase proceso de desarrollo, fases de modelado.

importar

Denota un estereotipo de la dependencia de permiso en la cual los nombres del paquete proveedor se añaden al espacio de nombres del paquete cliente.

Véase también acceso, paquete, visibilidad.

Semántica

Los nombres que se hallan en el espacio de nombres del paquete proveedor se añaden al espacio de nombres del paquete cliente, con las mismas reglas de visibilidad que se hayan especificado en el acceso. Si hay algún conflicto entre los nombres importados y los nombres que ya estén en el espacio de nombres del cliente, el modelo está mal formado.

Véase acceso para las reglas de visibilidad, tanto para el acceso como para la importación.

Notación

La dependencia de importación se representa mediante una flecha discontinua que va desde el paquete que obtiene acceso hasta el paquete que proporciona los elementos; la flecha lleva asociada la palabra clave de estereotipo «**import**».

inactivo

Un estado que no está activo; estado que no está almacenado en un objeto.

incluir

Es una relación entre un caso de uso *base* y un caso de uso *incluido*, que especifica la forma en que se puede insertar el comportamiento definido para el caso de uso incluido en el comportamiento definido para el caso de uso base. El caso de uso base puede ver la inclusión y puede

basarse en los efectos de realizar esa inclusión pero ni la base ni la inclusión pueden acceder a los atributos del otro.

Véase también extender, caso de uso, generalización de casos de uso.

Semántica

La relación de inclusión conecta un caso de uso base con un caso de uso incluido. El caso de uso incluido que figura en esta relación no es un clasificador instanciable independientemente. Lo que hace es describir explícitamente una secuencia adicional de comportamiento que se inserta en una instancia de caso de uso que está ejecutando el caso de uso base. A este mismo caso de uso base se le pueden aplicar múltiples relaciones de inclusión. El mismo caso de uso incluido se puede incluir en múltiples casos de uso base. Esto no indica ninguna relación entre las casos de uso base. Puede haber, incluso, múltiples relaciones de inclusión entre el mismo caso de uso incluido y casos de uso base, siempre y cuando cada inserción se haga en una posición diferente de la base.

El caso de uso incluido puede acceder a atributos o a operaciones del caso de uso base. La inclusión representa un comportamiento encapsulado que, potencialmente, puede reutilizarse en múltiples casos de uso base. El caso de uso base ve al caso de uso incluido, que puede dar valores a los atributos del caso de uso base. Pero el caso de uso base no debe acceder a los atributos del caso de uso incluido, porque el caso de uso incluido habrá concluido cuando el caso de uso base recupere el control.

Obsérvese que se pueden anidar las adiciones (sea cual fuere su clase). Por tanto, una inclusión puede servir como base para otra inclusión, extensión o generalización posterior.

Estructura

La relación de inclusión posee la siguiente propiedad.

localización	Es una situación, dentro del cuerpo de la secuencia de comportamiento del caso de uso base, en que se debe insertar la inclusión. Cuando una instancia de un caso de uso llega a la localización mientras está ejecutando el caso de uso base, se ejecuta el caso de uso incluido antes de reanudar el caso de uso base. La inclusión es una sentencia explícita situada dentro de la secuencia de comportamiento del caso de uso base. Por tanto la localización es implícita, a diferencia de la situación habida con la relación de extender.

La inclusión se efectúa una sola vez. Se pueden lograr otras multiplicidades mediante bucles en la secuencia de comportamiento correspondiente al caso base al que hace referencia la inclusión.

Notación

Se traza una flecha discontinua desde el símbolo del caso de uso base hasta el símbolo del caso de uso incluido, con una cabeza de flecha abierta en la inclusión. Se pone la palabra «**include**»

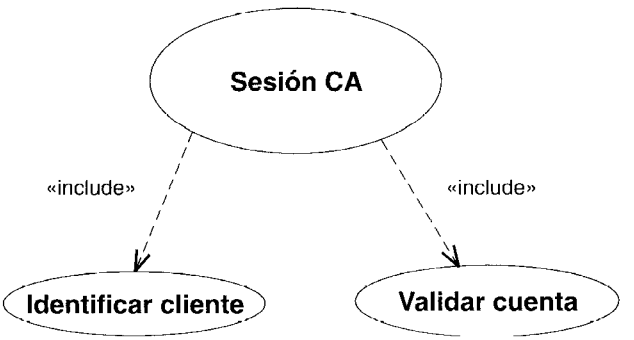


Figura 13.124 Relación de inclusión

en la flecha (Figura 13.124). Se puede asociar la localización a la flecha como una lista de propiedades entre llaves pero normalmente se hace referencia a ella formando parte del texto correspondiente al caso de uso base y no es necesario mostrarla en el diagrama. La Figura 13.125 muestra las secuencias de comportamiento para estos casos de uso.

Caso de uso base para una sesión de cajero automático:

mostrar el anuncio del día	paso de comportamiento
incluir identificación cliente	inclusión
incluir verificación de cuenta	otra inclusión
imprimir encabezado del recibo	paso de comportamiento
desconexión	paso de comportamiento

Caso de uso incluido para **Identificar cliente**:

obtener nombre cliente
incluir verificar identidad
si fracasa la verificación **entonces** abortar la sesión
obtener los números de cuenta del cliente

Caso de uso incluido para **Verificar cuenta**:

establecer conexión con la base de datos de cuentas
obtener estado de cuenta y límites

Figura 13.125 Secuencias de comportamiento para casos de uso

información de fondo

Cada aparición de un símbolo de clase en un diagrama o en diferentes diagramas puede tener distintas decisiones de presentación. Por ejemplo, el símbolo de una clase puede mostrar los atributos y operaciones y el de otra clase puede suprimirlos. Las herramientas pueden proporcionar hojas de estilo asociadas a símbolos individuales o a diagramas enteros, donde se especifiquen las decisiones de presentación, pudiéndose aplicar no sólo a las clases, sino a otros símbolos cualesquiera.

No resulta útil mostrar gráficamente toda la información de modelado. Algunas informaciones son más útiles si se representan mediante texto o en forma de tablas. Por ejemplo, la información detallada de programación a menudo está mejor presentada como listas de texto. UML no supone que toda la información de un modelo se va a expresar mediante diagramas; alguna información puede estar disponible únicamente en tablas. Este documento no intenta establecer el formato de dichas tablas ni la forma de acceder a ellas, ya que la información subyacente se describe adecuadamente en el metamodelo de UML, siendo la representación tabular responsabilidad de la herramienta. Sin embargo, se supone que pueden existir enlaces ocultos entre elementos gráficos o tabulares.

iniciación

Consiste en fijar el valor de un objeto recién creado, a saber, los valores de sus atributos, los enlaces de las asociaciones a las que pertenece y su estado de control. También se denomina inicialización.

Véase también instanciación.

Semántica

Concretamente, un objeto nuevo se crea por entero en un solo paso. Sin embargo, resulta más sencillo pensar en el proceso de instanciación como si tuviese dos pasos: creación e iniciación. En primer lugar se reserva el esqueleto vacío de un nuevo objeto, con la estructura adecuada de espacios para atributos; se le da una identidad al nuevo objeto en bruto. La identidad se puede implementar de varias maneras, tales como la dirección del bloque de memoria que contiene el objeto o bien mediante un contador entero. En todo caso, se trata de algo que resulta único en todo el sistema y se puede emplear como clave para hallar el objeto y acceder a él. En este momento, el objeto todavía no es válido: puede violar las restricciones existentes respecto a sus valores y relaciones. El paso siguiente es la iniciación. Se evalúan todas las expresiones de valores iniciales que puedan haberse declarado y se asignan los resultados a los atributos correspondientes. El método de creación puede calcular explícitamente los valores de los atributos, anulando de este modo los valores iniciales por defecto. Los valores resultantes deben satisfacer las posibles restricciones que afecten a la clase. El método de creación puede también crear enlaces que contengan al nuevo objeto. Deben satisfacer la multiplicidad declarada de todas aquellas asociaciones en que participe la clase. Una vez finalizada la iniciación, el objeto debe ser un objeto válido y tendrá que obedecer las restricciones que afecten a su clase. Una vez finalizada la iniciación, aquellos atributos o asociaciones cuya propiedad de modificabilidad sea **frozen** o **addOnly** no podrán ser alterados mientras no se destruya el objeto. Todo el proceso de iniciación es atómico y no se puede interrumpir ni entrelazar.

inicio

Es la primera fase de un proceso de desarrollo de software, durante la cual se conciben y evalúan las ideas iniciales de un sistema. En esta fase se desarrolla una parte de la vista de análisis y pequeñas porciones de otras vistas.

Véase proceso de desarrollo.

instancia

Es una entidad individual con su propio valor e identidad. Un descriptor especifica la forma y comportamiento de un conjunto de instancias cuyas propiedades son similares. Las instancias poseen identidad y unos valores que son consistentes con la especificación que consta en el descriptor. Normalmente, las instancias aparecen en los modelos como ejemplos consistentes con modelos del nivel de descriptores.

Véase también descriptor, identidad, enlace, objeto.

Semántica

Toda instancia tiene identidad. En otras palabras, dados distintos instantes de tiempo durante la ejecución de un sistema, se puede identificar una instancia con esa misma instancia en otros instantes de tiempo, aun cuando cambie el valor de la instancia. Una instancia posee, en todo momento, un valor que se puede expresar en términos de valores de datos y de referencias de otras instancias. Un valor de datos es un caso degenerado. Su identidad es lo mismo que su valor; desde un punto de vista diferente, carece de identidad.

Además de la identidad y el valor, toda instancia posee un descriptor que limita los valores que puede tener la instancia. Un descriptor es un elemento del modelo que describe instancias. Ésta es la dicotomía instancia-descriptor. Casi todos los conceptos de modelado propios de UML tienen este carácter dual. El contenido principal de muchos modelos está formado por descriptores de diferentes clases. El propósito del modelo es describir los posibles valores de un sistema en términos de las instancias y de sus valores.

Cada clase de descriptor describe una clase de instancia. Un objeto es una instancia de una clase; un enlace es una instancia de una asociación. Un caso de uso describe posibles instancias de casos de uso; un parámetro describe un posible valor de argumento y así sucesivamente. Algunas instancias no poseen nombres de familia y suelen ignorarse salvo en circunstancia muy formales, pero siguen existiendo pese a todo. Por ejemplo, un estado describe posibles apariciones del estado durante el seguimiento de una ejecución.

Un modelo describe los posibles valores de un sistema y su comportamiento al pasar de uno a otro valor durante la ejecución. El valor del sistema es el conjunto de todas las instancias que contenga y de sus valores. El valor del sistema sólo es válido si toda instancia es la instancia de algún descriptor que haya en el modelo, siempre y cuando el conjunto de instancias satisfaga todas las restricciones explícitas e implícitas que haya en el modelo.

Los elementos de comportamiento del modelo describen la forma en que progresan de un valor a otro el sistema y las instancias que contiene. El concepto de identidad de instancias es esencial para esta descripción. Todo paso de comportamiento es la descripción del cambio de valores habido en un pequeño número de instancias en términos de sus valores anteriores. El resto de las instancias del sistema mantiene sus valores sin cambios. Por ejemplo, una operación local aplicada un objeto se puede describir mediante expresiones para los nuevos valores de cada uno de los atributos del objeto, sin que haya cambios en el resto del sistema. Una función no local puede descomponerse en funciones locales de varios objetos.

Obsérvese que las instancias de un sistema en ejecución no son elementos del modelo. Normalmente, no forman parte del modelo en modo alguno. Cuando aparecen instancias en un modelo, lo hacen como ilustraciones o ejemplos de la estructura y comportamiento típicos del mismo, son instantáneas del valor del sistema o el resultado de seguir la historia de su ejecución. Resultan útiles para la comprensión humana, pero suelen ser puntos de un conjunto grande o infinito de posibles valores y no *definen* nada.

Instancia directa. Todo objeto es la instancia directa de alguna clase y la instancia indirecta de los antecesores de esa clase. Esto sucede también para las instancias de otros elementos generalizables. Un objeto es una instancia directa de una clase si la clase describe la instancia y no hay ninguna clase descendiente que también describa el objeto. En el caso de la clasificación múltiple, una instancia puede ser una instancia directa de más de un clasificador, ninguno de los cuales es un antecesor de alguno de los demás. Para una cierta semántica de ejecución, se designa a uno de los clasificadores como clase de implementación y los otros se toma como tipos o roles. El descriptor completo es la descripción completa implícita de una instancia —todos sus atributos, operaciones, asociaciones y demás propiedades— tanto si los obtiene la instancia de su clasificador director como si provienen por herencia de un clasificador que sea su antecesor. En el caso de clasificación múltiple, el descriptor completo es la unión de todas las propiedades definidas por cada uno de los clasificadores directos.

Creación. *Véase* instanciación para una descripción de la forma en que se crean las instancias.

Notación

Aun cuando los descriptores y las instancias no son una misma cosa, comparten muchas propiedades, incluyendo una misma forma (porque el descriptor debe describir la forma de las instancias). Por tanto, resulte conveniente seleccionar una notación para cada pareja descriptor-instancia tal que su correspondencia sea visualmente obvia inmediatamente. Existe un número limitado de formas de hacer esto, cada una de las cuales posee sus propias ventajas y desventajas. En UML la distinción descriptor-instancia se muestra empleando el mismo símbolo geométrico para cada pareja de elementos y subrayando la cadena que da nombre a un elemento de instancia. Esta distinción visual es en general fácil de discernir sin resultar excesiva cuando se tiene todo un diagrama que contiene elementos de instancia.

Aun cuando la Figura 13.126 muestra objetos, la convención de subrayado puede emplearse para otras clases de instancias, tales como los casos de uso, las instancias de componentes y las instancias de nodos.

Como las instancias aparecen como ejemplos en los modelos, sólo suelen incluirse los detalles relevantes para algún ejemplo particular. Por ejemplo, no es necesario incluir toda la lista completa de atributos; además se puede omitir la lista completa de valores si la atención se centra en otra cosa, tal como el flujo de mensajes entre objetos.

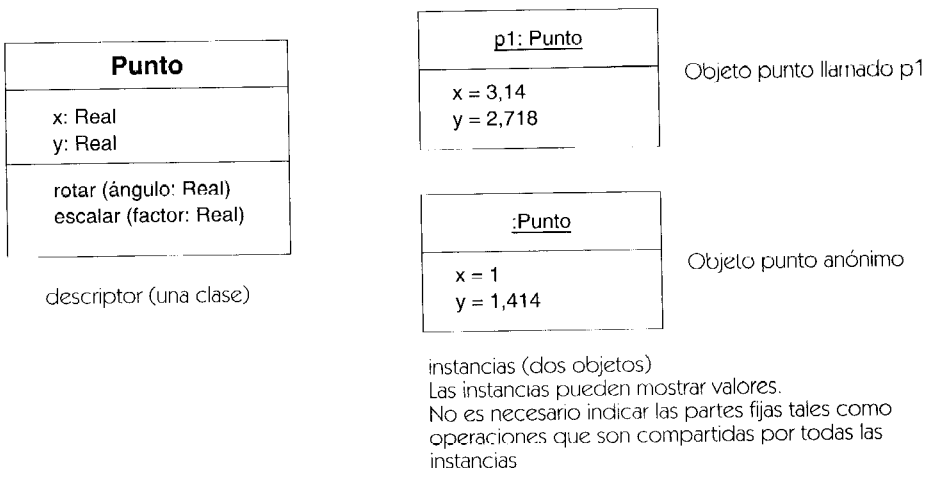


Figura 13.126 Descriptor e instancias

instanciable

Que puede poseer instancias. Es sinónimo de concreto.

Véase también abstracto, instancia directa, elemento generalizable.

Semántica

Los elementos generalizables se pueden crear como abstractos o como instanciables. Si son instanciables, entonces será posible crear instancias directas.

instanciación

Creación de nuevas instancias de elementos de un modelo.

Véase también iniciación.

Semántica

Las instancias se crean durante la ejecución como resultado de acciones primitivas de creación o de operaciones de creación. Primero se crea una identidad para la nueva instancia; después se reserva su estructura de datos según prescriba su descriptor; por último se dan valores iniciales a los valores de sus propiedades según prescriban el descriptor y el operador de creación.

La dependencia de utilización de la instanciación muestra la relación existente entre una operación que crea instancias o una clase que contiene esa operación y la clase de los objetos que se instancian.

Objetos. Cuando se instancia (se crea) un nuevo objeto, ese objeto se crea con una cierta identidad y una memoria, y luego se le dan valores iniciales. La iniciación de un objeto define los valores de sus atributos, sus asociaciones y su estado de control.

Normalmente, toda clase concreta posee una o más operaciones de constructor con alcance de clase cuyo propósito es crear nuevos objetos de la clase. Subyacente a todas las operaciones de constructor existe una operación primitiva implícita que crea una nueva instancia en blanco que después será iniciada por las operaciones de constructor. Una vez creada una instancia en blanco, ésta posee la forma prescrita por su descriptor, pero sus valores aún no han recibido valores iniciales, así que puede ser semánticamente inconsistente. Por tanto, una instancia no está disponible para el resto del sistema mientras no haya sido iniciada, lo cual sucederá inmediatamente después de la creación de la instancia en blanco.

Enlaces. Análogamente, los enlaces se crean mediante acciones u operaciones de creación, normalmente ejecutadas por operaciones de alcance de instancia asociadas a una de las clases participantes, en lugar de hacerlo mediante operaciones de constructor aplicadas al elemento de asociación en sí (aun cuanto ésta es una posible técnica de implementación en algunas circunstancias). Una vez más, existe una operación primitiva implícita que crea un nuevo enlace entre una tupla específica de objetos. Esta operación no produce efectos si ya existe un enlace de la misma asociación entre la tupla de objetos (porque la extensión de una asociación es un conjunto y no puede contener valores duplicados). En una asociación ordinaria, ya no queda nada más por hacer. Un enlace de una clase asociación, sin embargo, requiere la inicialización de los valores de sus atributos.

Instancias de casos de uso. La instanciación de un caso de uso significa que se crea una instancia de caso de uso, y la instancia de caso de uso empieza a ejecutarse al principio del caso de uso que la controla. La instancia de caso de uso puede seguir temporalmente a otro caso de uso con el que esté relacionada por relaciones de extensión o inclusión antes de seguir ejecutando el caso de uso original. Cuando la instancia de caso de uso llega al final del caso de uso que está siguiendo, la instancia de caso de uso concluye.

Otras instancias. Las instancias de otros descriptores se pueden crear en un proceso análogo de dos pasos: primero se efectúa una creación en blanco para establecer la identidad y reservar la estructura de datos; luego se inician los valores de la nueva instancia para que cumpla todas las restricciones pertinentes. Por ejemplo, se crea implícitamente una activación como consecuencia directa de una llamada a una operación.

Los mecanismos exactos para la creación de instancias serán responsabilidad del entorno de ejecución.

Notación

La dependencia de instanciación se representa mediante una flecha discontinua que va desde la operación o clase que realiza la instanciación hasta la clase que se instancia; el estereotipo «**instantiate**» se asocia a la flecha.

Discusión

En algunas ocasiones se emplea el término instanciación para denotar el enlazado de una plantilla para producir un elemento enlazado, pero para esta relación es más específico el término enlazado.

instancia de

Es la relación entre una instancia y su descriptor.

Véase instancia.

instancia de caso de uso

Ejecución de una secuencia de acciones especificada en un caso de uso.

Véase caso de uso.

instancia directa

Una instancia, tal como un objeto, cuyo descriptor más específico, tal como una clase, es una clase dada.

Utilizado en una frase como, "El Objeto O es una instancia directa de la clase C." en este caso, la clase C es la clase directa del objeto O.

Véase clase directa.

instancia indirecta

Se trata de una entidad que es una instancia de un elemento, tal como una clase; además, es una instancia de un descendiente del elemento. Esto es, se trata de una instancia pero no es una instancia directa.

instanciar

Crear una instancia de un descriptor.

Véase instanciación.

instantánea

Colección de objetos, enlaces y valores que forma la configuración de un sistema en un determinado instante de su ejecución.

intención

Dícese de la especificación formal de las propiedades estructurales y de comportamiento de un descriptor. Compárese con: extensión.

Véase también descriptor.

Semántica

Un descriptor, tal como una clase o una asociación, posee tanto una descripción (su intención) como un conjunto de instancias que describe (su extensión). El propósito de la intención es especificar las propiedades estructurales y de comportamiento de las instancias en una forma ejecutable.

interacción

Se trata de la especificación de la forma en que se envían mensajes entre objetos u otras instancias para ejecutar una tarea. La interacción se define en el contexto de una colaboración.

Véase también diagrama de interacción.

Semántica

Los objetos u otras instancias de una colaboración se comunican para lograr un propósito (tal como llevar a cabo una operación) mediante el intercambio de mensajes. Entre los mensajes pueden contarse las señales y llamadas, así como interacciones más implícitas a través de condiciones o de eventos temporales. Un patrón de intercambios de mensajes que se realizan para lograr un propósito específico es lo que se denomina una interacción.

Estructura

Una interacción es una especificación de comportamiento que abarca una secuencia de intercambios de mensajes entre un conjunto de objetos, efectuada para lograr un cierto propósito tal como la implementación de una operación. Una interacción es una colaboración más un conjunto secuenciado de flujos de mensajes que se imponen a los enlaces de la colaboración. Para especificar una interacción, primero es necesario especificar una colaboración —esto es, definir los objetos que interaccionan y sus relaciones entre sí—. Entonces se especifican las posibles secuencias de interacción, bien en una única descripción que contiene condiciones (bifurcaciones o señales condicionales) o bien como múltiples descripciones, cada una de las cuales describe una de entre las posibles rutas de ejecución. La descripción completa del comportamiento de una colaboración se puede dar como una máquina de estados, cuyos estados son los estados de ejecución de una operación o de algún otro procedimiento.

Notación

Las interacciones se muestran como diagramas de secuencia o como diagramas de colaboración. Ambos formatos de diagrama muestran la ejecución de las colaboraciones. Los diagramas de secuencia son la visualización explícita del comportamiento de las colaboraciones, incluyendo la secuenciación temporal de mensajes y una representación explícita de las activaciones de métodos. Sin embargo, los diagramas de secuencias muestran únicamente los objetos participantes y no sus relaciones con otros objetos o sus atributos. Por tanto, no muestran en su to-

talidad el punto de vista contextual de una colaboración. Los diagramas de colaboración muestran todo el contexto de una interacción, incluyendo los objetos y sus relaciones relevantes para una interacción, así que suelen ser mejores que los diagramas de secuencia a efectos de diseño.

interfaz

Un conjunto de operaciones que posee un nombre y que caracteriza el comportamiento de un elemento.

Véase también clasificador, realización.

Semántica

Una interfaz es un descriptor de las operaciones visibles externamente de una clase u otra entidad (incluyendo las unidades de compendio, tales como los paquetes) que no especifica la estructura interna. Cada interfaz suele especificar únicamente una parte limitada del comportamiento de una clase real. Una clase puede admitir muchas interfaces, cuyos efectos podrán ser disjuntos o estar solapados. Las interfaces no poseen implementación; carecen de atributos, estados y asociaciones; sólo poseen operaciones y señales que reciben. Las interfaces pueden tener relaciones de generalización. Una interfaz descendiente incluye todas las operaciones y señales de sus antecesores pero puede añadir operaciones adicionales. En esencia, una interfaz equivale a una clase abstracta sin atributos ni métodos, que poseyera únicamente operaciones abstractas. Todas las operaciones de una interfaz tienen visibilidad pública (en caso contrario, no tendría sentido incluirlas, porque una interfaz no tiene nada "interno" que pueda hacer uso de ellas).

La siguiente definición extendida indica el propósito de una interfaz.

- Una interfaz es una colección de operaciones que se emplea para especificar un servicio de una clase o de un componente.
- Una interfaz sirve para nombrar una colección de operaciones y para especificar sus signaturas y efectos. Una interfaz se centra en los efectos, no en la estructura, de un servicio dado. Una interfaz no ofrece una implementación para ninguna de sus operaciones. La lista de operaciones puede incluir también las señales que esa clase está dispuesta a manejar.
- Una interfaz se emplea para especificar un servicio que proporciona un proveedor y que pueden solicitar otros elementos. La interfaz da nombre a una colección de operaciones que funcionan en cooperación para realizar algún comportamiento de interés lógico de un sistema o como parte de un sistema.
- Una interfaz define un servicio que ofrece una clase o componente. Define un servicio que a su vez es implementado por una clase o componente. Como tal, una interfaz abarca los límites lógicos y físicos de un sistema. Una o más clases (que formarán probablemente parte de un subsistema componente) pueden proporcionar una implementación lógica de la interfaz. Uno o más componentes pueden proporcionar un empaquetamiento físico que se adapte a esa misma interfaz.

- Si una clase realiza (implementa) una interfaz, entonces debe declarar o heredar todas las operaciones de la interfaz. Si una clase realiza más de una interfaz, tiene que contener todas y cada una de las operaciones que se encuentren en cualquiera de sus interfaces. Una misma operación puede aparecer en más de una interfaz. Si coinciden sus signaturas, deben representar la misma operación o bien estarán en conflicto y el modelo estará mal formado. (Una implementación puede adoptar reglas específicas de un lenguaje para las signaturas coincidentes. Por ejemplo, en C++ los nombres de los parámetros y los tipos proporcionados se ignoran.) Una interfaz no hace afirmación alguna acerca de los atributos o asociaciones de una clase; éstos forman parte de su implementación.

Una interfaz es un elemento generalizable. Una interfaz descendiente hereda todas las operaciones de su predecesor y puede añadir algunas operaciones. La realización puede considerarse como una herencia de comportamiento; una clase hereda las operaciones de otro clasificador, pero no su estructura. Una clase puede realizar a otra clase. La clase que haga las veces de especificación actuará como interfaz en tanto en cuanto sólo sus operaciones afectan a la relación.

Las interfaces no participan en las asociaciones. Una interfaz no puede tener una asociación que sea navegable partiendo de la interfaz. Una interfaz, sin embargo, puede ser el destino de una asociación, siempre y cuando la asociación sólo sea navegable hacia la interfaz.

Notación

Una interfaz es un clasificador y se puede mostrar empleando el símbolo del rectángulo con la palabra reservada «**interface**». En el compartimento de operación se pone una lista de las operaciones que admite esa interfaz. Las señales que llevan el estereotipo «**signal**» se pueden incluir también en la lista de operaciones o bien se pueden enumerar en su propio compartimento. Se puede omitir el compartimento de atributos porque siempre está vacío.

También se puede visualizar una interfaz mediante un circulito con el nombre de la interfaz situado por debajo del símbolo. El círculo puede estar conectado mediante una línea continua con clases (u otros elementos) que lo admitan. También puede conectarse con contenedores de nivel superior, tales como los paquetes, que contengan las clases. Esto indica que la clase proporciona todas las operaciones del tipo de interfaz (y posiblemente más). La notación de círculo no muestra la lista de operaciones que admite la interfaz. Utilice el símbolo de rectángulo completo para mostrar la lista de operaciones. Una clase que utilice o requiera operaciones proporcionadas por la interfaz se puede conectar al círculo mediante una flecha discontinua que apunte al círculo. La flecha discontinua implica que la clase requiere las operaciones especificadas en la interfaz para algún propósito, pero no se exige que la clase cliente haga uso de *todas* las operaciones de la interfaz. Un servicio suele especificarse mediante una prueba de suficiencia. Si un proveedor proporciona las operaciones contenidas en un cierto conjunto de interfaces, entonces satisface los requisitos de los clientes.

La relación de realización se muestra mediante una línea discontinua con una cabeza de flecha triangular sólida (un "símbolo discontinuo de generalización") que va desde una clase a una interfaz que admita ésta. Se trata de la misma notación que se emplea para indicar la realización de un tipo por parte de una clase de implementación. De hecho, este símbolo se puede emplear entre dos símbolos de clasificador, indicando que el cliente (situado en la cola de la flecha) admite todas las operaciones definidas en el proveedor (situado en la cabeza de la

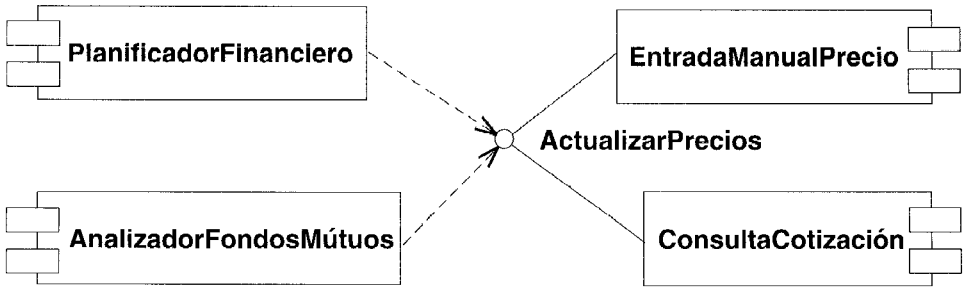


Figura 13.127 Proveedores y clientes de la interfaz

flecha), pero sin necesidad de admitir ninguna estructura de datos del proveedor (atributos y asociaciones).

Ejemplo

La Figura 13.127 muestra una vista simplificada de los componentes financieros que intervienen en los precios de las acciones de bolsa. El **PlanificadorFinanciero** es una aplicación financiera personal que administra nuestras inversiones, así como los gastos personales. El **AnalizadorFondosPensiones** examina detalladamente los fondos de pensiones. Necesita poseer la capacidad de actualizar los precios de los valores subyacentes, así como lo precios de los fondos. La capacidad de actualizar los precios de los valores se muestra mediante la interfaz **ActualizarPrecios**. Hay dos componentes que implementan esta interfaz; se muestran mediante las líneas continuas que los unen al símbolo de interfaz. El componente **EntradaManualPrecio** permite al usuario introducir manualmente los precios de los valores seleccionados. El componente **ConsultaCotización** recupera los precios de esos valores en un servidor bursátil, empleando un módem o Internet.

La Figura 13.128 muestra la notación completa de una interfaz como palabra reservada en un símbolo de clase. Se observará que esta interfaz implica dos operaciones —preguntar el precio de unas acciones y conseguir una cotización—; y además proporcionar una lista de valores de bolsa y recibir una lista de los precios que hayan cambiado. En este diagrama el componente **ConsultaCotización** está conectado a la interfaz empleando una flecha de realización, pero se trata de la misma relación que se mostraba en el diagrama anterior; sólo se está empleando una notación más explícita.

Este diagrama muestra también una nueva interfaz, **Actualizar Periódica Precios**, que es un descendiente de la interfaz original. Hereda las dos operaciones y agrega una tercera operación que envía una solicitud de actualización periódica y automática de los precios. Esta interfaz es realizada por el componente **ServidorCotización**, un servicio de suscripción. Implementa las dos mismas operaciones que **ConsultaCotización** pero de una manera diferente. No comparte la implementación de **ConsultaCotización** (en este ejemplo) y por tanto no hereda de él una implementación.

La Figura 13.128 muestra la diferencia entre herencia de interfaz y herencia completa. Esa última implica a la primera, pero una interfaz descendiente puede ser implementada de una forma distinta de la interfaz predecesora. **ServidorCotización** admite la interfaz que implementa

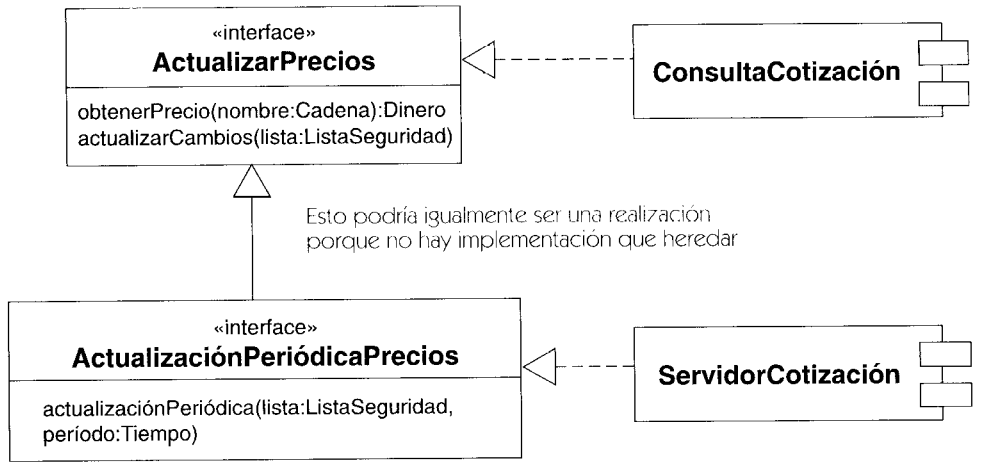


Figura 13.128 La notación de interfaz completa

ConsultaCotización, a saber, **ActualizarPrecios**, pero no hereda la implementación de **ConsultaCotización**. (En general, resulta cómodo heredar la implementación además de la interfaz, así que las dos jerarquías suelen ser idénticas.)

Una interfaz también puede contener una lista de las señales que maneja (Figura 13.129).

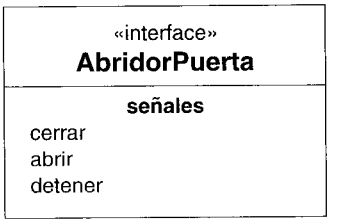


Figura 13.129 Una interfaz con señales

Las interfaces se emplean para definir el comportamiento de las clases, así como el de los componentes, sin restringir la implementación. Esto permite distinguir la herencia de interfaz, tal como se declara en Java, de la herencia de implementación.

invariante

Una restricción que forzosamente ha de ser cierta en todo momento (o por lo menos en todo momento en que no haya ninguna operación sin finalizar).

Semántica

El invariante es una expresión booleana que debe ser cierta siempre que no haya ninguna operación activa. Se trata de una aserción y no de un sentencia ejecutable. Dependiendo de la

forma exacta de la expresión, puede ser o no posible verificarla automáticamente por anticipado.

Véase también precondición, postcondición.

Estructura

Los invariantes se modelan como una restricción con el estereotipo «**invariant**» asociado al elemento.

Notación

Se puede mostrar una postcondición en una nota con la palabra reservada «**invariant**». La nota está asociada a un clasificador, atributo u otro elemento.

ligadura

Asignación de valores a parámetros para producir un elemento individual a partir de un elemento parametrizado. La relación de ligadura es un tipo de dependencia, que se utiliza para ligar plantillas y así producir nuevos elementos del modelo.

Véase también elemento ligado, plantilla.

Semántica

Una definición parametrizada, como una operación, señal o plantilla, define la forma de un elemento. Un elemento parametrizado no se puede utilizar directamente, ya que sus parámetros no tienen valores específicos. La ligadura es una dependencia que representa la asignación de valores a parámetros para producir un nuevo elemento que pueda ser utilizado. La ligadura actúa sobre las operaciones para producir llamadas, sobre las señales para producir señales de envío y sobre las plantillas para producir nuevos elementos del modelo. En las dos primeras, la ligadura se produce durante la ejecución para producir entidades en tiempo de ejecución que normalmente no figuran en los modelos excepto como ejemplos o resultados de la ejecución. Los valores de los argumentos se definen dentro del sistema de ejecución.

Sin embargo, una plantilla se liga en tiempo de modelado para producir nuevos elementos utilizables en el modelo. Los valores de los argumentos pueden ser otros elementos, como clases, además de valores dato, como cadenas o enteros. La relación de ligadura liga valores a una plantilla produciendo un elemento real del modelo que puede utilizarse directamente dentro del mismo.

Una relación de ligadura tiene un elemento proveedor (la plantilla), un cliente (el elemento recién generado), y una lista de valores para ligarlos con los parámetros de la plantilla. El

elemento ligado se define sustituyendo cada parámetro por el valor del correspondiente argumento en una copia del cuerpo de la plantilla. La clasificación de cada argumento debe ser la misma o un descendiente de la clasificación declarada de su parámetro.

Las ligaduras no afectan a la plantilla, pues se puede ligar cada plantilla muchas veces para producir cada vez un nuevo elemento ligado.

Notación

La ligadura se indica mediante la palabra clave «**bind**» asociada a una flecha discontinua que conecta el elemento generado (en el origen de la flecha) con la plantilla (en la punta de la flecha). Los valores reales de los argumentos se representan mediante una lista de expresiones de texto separadas por comas encerrada entre paréntesis a continuación de la palabra «**bind**».

Una notación alternativa y más compacta de la ligadura emplea la correspondencia de nombres para evitar la necesidad de flechas. Para representar un elemento generado mediante ligadura, se añade al nombre de la plantilla una lista de expresiones de texto separadas por comas encerrada entre los símbolos menor y mayor que (<argumento$_{lista,}$>).

En cualquier caso, cada argumento se representa mediante una cadena de texto que se evalúa estáticamente en el momento de construir el modelo. No se evalúa dinámicamente como las operaciones o los argumentos de tipo señal.

En la Figura 13.130 se declara explícitamente (utilizando flechas) una nueva clase **ListaDirecciones**, cuyo nombre puede utilizarse en modelos y expresiones. La forma implícita **Farray<Punto,3>** declara una «clase anónima» que no tiene nombre por sí misma, y que puede utilizarse en expresiones utilizando la sintaxis en línea. En ningún caso se pueden declarar atributos u operaciones adicionales: si se necesitan extensiones debe declararse una subclase.

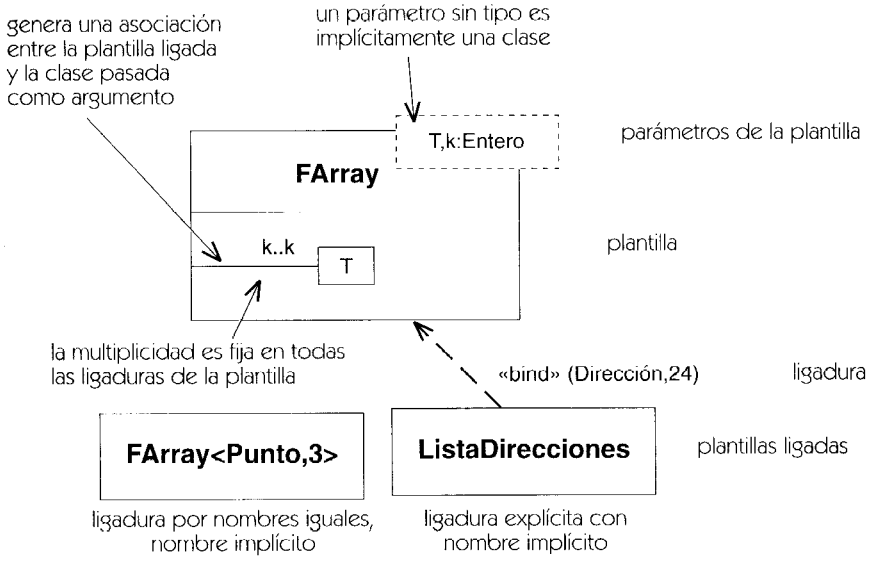


Figura 13.130 Plantillas: declaración y ligadura

Elementos estándar

bind (ligar).

línea de vida

Denota una línea discontinua que aparece en los diagramas de secuencia y muestra la existencia de un objeto a lo largo de un cierto período de tiempo. La línea es paralela al eje temporal.

Véase también diagrama de secuencia.

Semántica

La línea de vida indica el período durante el cual existe un objeto. Un objeto está activo si posee un hilo de control; esto es, durante el período de tiempo en el cual posee una llamada a procedimiento que no ha concluido. Esto último se denomina una activación. Incluye el tiempo durante el cual el procedimiento está llamando a un procedimiento de nivel inferior.

Notación

Los objetos o roles de clasificador se muestran en los diagramas de secuencia en forma de una línea vertical, llamada línea de vida. La línea de vida representa la existencia del objeto en un determinado momento.

Las flechas entre líneas de vida denotan mensajes entre objetos. Una flecha que tiene la punta en una línea de vida es un mensaje que recibe el objeto, una operación de la cual es responsable; una flecha que tiene su origen en una línea de vida es un mensaje que ha enviado el objeto, una operación invocada por él. El orden geométrico de las flechas de mensajes a lo largo de la línea de vida indica el orden temporal relativo de los mensajes.

Si el objeto es creado o destruido durante el período de tiempo que se muestra en el diagrama, entonces su línea de vida comienza o se detiene en el punto correspondiente. En caso contrario, progresa desde la parte superior del diagrama hasta la parte inferior del mismo. En la cabeza de la línea de vida se dibuja el símbolo de un objeto. Si el objeto se crea durante el período de tiempo que se muestra en el diagrama, entonces el símbolo del objeto se dibuja en la punta del mensaje que lo crea. En caso contrario, se dibuja el símbolo del objeto por encima de todas las flechas de mensajes. Si se destruye el objeto durante el período de tiempo que muestra el diagrama, entonces su destrucción se marca mediante una *X* de gran tamaño, bien en la punta de flecha del mensaje que produce la destrucción o bien (en el caso de una autodestrucción) en el mensaje final que proporciona el objeto destruido. Los objetos que existen cuando comienza una transición se muestran en la parte superior del diagrama (por encima de la primera flecha). Un objeto que existe cuando finaliza la transición posee una línea de vida que sigue adelante más allá de la flecha final.

La línea de vida puede escindirse en dos o más líneas de vida concurrentes que se mostrarán condicionalmente. Cada pista corresponde a una bifurcación condicional en el flujo de men-

sajes. Las líneas de vida pueden volver a unirse en algún punto posterior. Véase la Figura 13.70 para observar un ejemplo. Esta notación puede inducir a confusión y debería utilizarse con parquedad.

El período de tiempo durante el cual está activo temporal o permanentemente el objeto se muestra mediante una línea doble continua que se superpone para mostrar recursividad. Como un objeto activo siempre está activo, la línea doble se omite en ciertas ocasiones, porque no añade información.

Las líneas de vida pueden verse interrumpidas por un símbolo de estado, para mostrar un cambio de estado. Esto se corresponde con una transición de conversión en el seno de un diagrama de colaboración. Se puede dibujar una flecha hasta el símbolo de estado para indicar el mensaje que ha dado lugar al cambio de estado. Véase la Figura 13.71 como ejemplo.

línea de vida del objeto

Denota una línea de un diagrama de secuencia que representa la existencia de un objeto durante un cierto período de tiempo.

lista

Una colección ordenada, de longitud variable y formada por elementos del modelo, que pertenece a otro elemento del modelo, estando anidada en él.

Véase también clasificador, estado.

Semántica

Los clasificadores contienen varias listas de elementos subordinados, que incluyen atributos, operaciones y métodos. Un estado contiene una lista de transiciones internas. Hay otras clases de elementos que contienen listas de otros tipos de elementos. Cada clase de lista se describe individualmente. Esta entrada describe las propiedades de las listas empotradas, en general. Además de las listas de atributos y de operaciones, hay listas opcionales que pueden mostrar valores predefinidos o definidos por el usuario, tales como responsabilidades, reglas o historias de modificaciones. UML no define estas listas opcionales. La manipulación de las listas definidas por el usuario depende de las herramientas.

Una lista empotrada y los elementos de la lista pertenecen exclusivamente a la clase que la contiene o a algún elemento contenedor. La propiedad no se comparte entre múltiples contenedores. Las otras clases pueden tener posibilidad de acceder a los elementos de la lista —por ejemplo, por herencia o asociación— pero la propiedad de las listas contenidas para modificar el modelo pertenece al contenedor inmediato. Los elementos poseídos se almacenan, se copian y se destruyen junto con sus contenedores.

Los elementos de una lista tienen un orden que está determinado por quien haga el modelado. El orden puede resultar útil para el modelador —por ejemplo, se puede emplear en un ge-

nerador de código para generar una lista de declaraciones en algún lenguaje de programación—. Si al modelador no le preocupa el orden, quizá porque el modelo se encuentra en la fase de análisis o porque el lenguaje ignora el orden, el orden seguirá existiendo en el modelo pero puede ignorarse sencillamente por ser irrelevante.

Notación

Las listas empotradas aparecen dentro de su propio compartimento como listas de cadenas, habiendo una cadena por línea para cada elemento de la lista. Cada cadena es la representación codificada de una característica, tal como un atributo, una operación, una transición interna y así sucesivamente. La naturaleza de la codificación se describe en el artículo correspondiente a cada clase de elemento.

Orden. El orden canónico de las cadenas es el mismo que tiene la lista de elementos dentro del modelo, pero el orden interno se puede anular opcionalmente y se pueden ordenar las cadenas de acuerdo con alguna propiedad interna, tal como el nombre, visibilidad o estereotipo. Sin embargo, observe que los elementos mantienen su orden original en el modelo subyacente. La información relativa sólo se suprime de la visualización.

Puntos suspensivos. Unos puntos suspensivos (...) como elemento final de la lista o como elemento final de una sección delimitada de la lista indica que hay elementos adicionales en el modelo que satisfacen los criterios de selección pero no se muestran en la lista. En una visualización diferente de la lista, podrían aparecer estos elementos.

Estereotipo. Se puede aplicar un estereotipo a un elemento de la lista. La cadena del elemento va precedida por una palabra reservada encerrada entre comillas («»).

Cadena de propiedades. Una cadena de propiedades puede especificar una lista de propiedades de un elemento. Detrás del elemento va una lista separada por comas formada por las propiedades o restricciones, encerradas todas entre llaves ({ }).

Propiedades de grupo. Los sistemas y demás propiedades pueden aplicarse también a grupos de elementos de la lista. Si un estereotipo, una palabra reservada, una cadena de propiedades o una restricción aparece en solitario en una línea, entonces la línea no representa un elemento de lista. Lo que sucede es que las restricciones se aplican a los sucesivos elementos de la lista tal como si se hubieran situado directamente en todas las líneas. Este valor por defecto sigue siendo aplicable hasta que aparece en la lista otra propiedad de grupo. Se pueden cancelar todas las propiedades de grupo insertando una línea con una palabra reservada en blanco («») pero suele ser más claro poner todas las entradas que no estén sujetas a propiedades de grupo en la parte inicial de la lista. La Figura 13.131 muestra la aplicación de estereotipo a múltiples elementos de una lista.

Observe que las propiedades de grupo son meramente una forma cómoda de notación y que cada elemento del modelo posee su propio valor independiente para cada propiedad.

Nombre de compartimento. Los compartimentos pueden mostrar un nombre que indica de qué clase es el compartimento. El nombre se visualiza con una tipografía distintiva (tal como negrita en un tamaño más pequeño) centrado en la parte superior del compartimento. Esta capacidad resulta útil si se omiten algunos compartimentos o si se añaden compartimentos adicionales definidos por el usuario. Para una clase, los compartimentos predefinidos son los

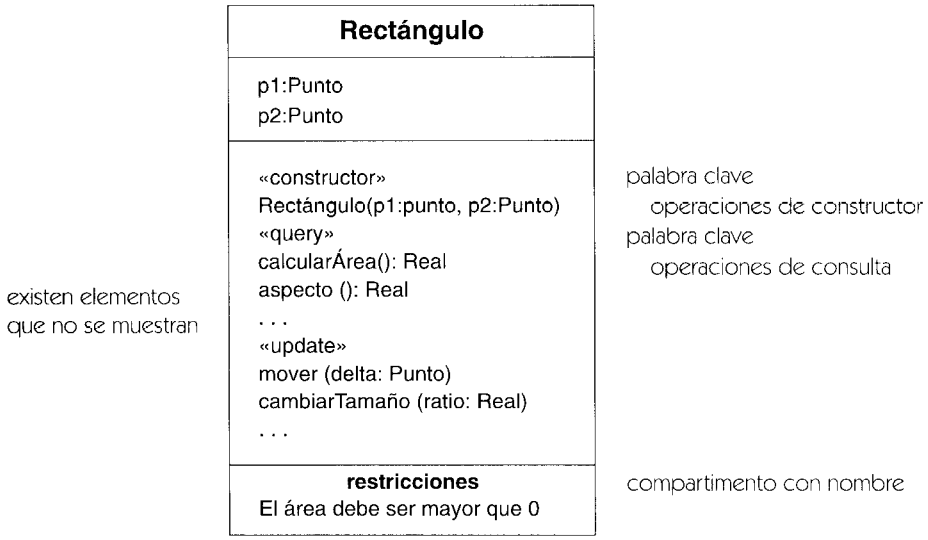


Figura 13.131 Una palabra reservada de estereotipo aplicada a grupos de elementos de una lista

atributos y **operaciones** con nombre. El compartimento de nombre de la clase debe estar siempre presente y por tanto no requiere ni admite un nombre de compartimento. La Figura 13.131 y la Figura 13.132 muestran compartimentos con nombre.

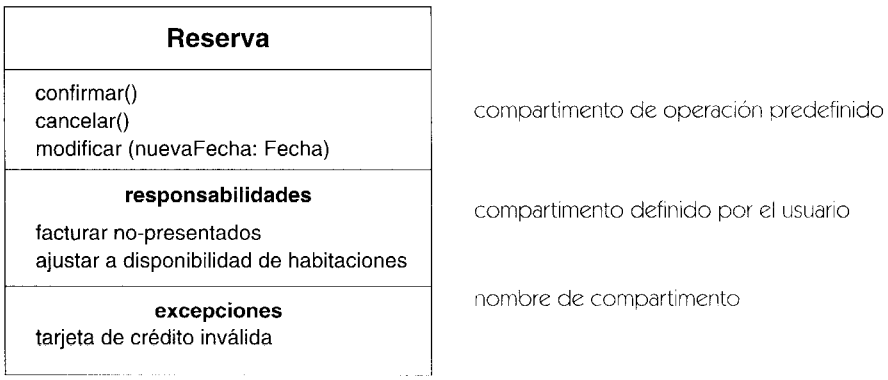


Figura 13.132 Compartimentos con nombre

Opciones de presentación

Orden. Una herramienta puede presentar los elementos de la lista colocándolos según su orden. En tal caso, el orden inherente de los elementos no es visible. Cada orden está basado en alguna propiedad interna y no indica información adicional del modelo. Entre las reglas de orden típicas se cuenta el orden alfabético, el orden por estereotipos (tales como constructores, destructores y por último métodos ordinarios), orden por visibilidad (pública, luego protegida, después privada) y así sucesivamente.

Filtrado. Los elementos de la lista pueden filtrarse empleando alguna regla de selección. La especificación de reglas de selección es una responsabilidad de la herramienta. Si una lista filtrada no muestra elementos, entonces no hay elementos que satisfagan el criterio y por tanto son invisibles. Es responsabilidad de la herramienta indicar o no de alguna forma la presencia de filtros locales o globales, aunque un diagrama independiente debería mostrar alguna indicación de ese filtrado si es que ha de resultar comprensible.

Si se suprime un compartimento, no se puede hacer deducción alguna respecto a la presencia o ausencia de sus elementos. Un compartimento vacío indica que no hay elementos que satisfagan el filtro de selección (si lo hubiere).

Observe que los atributos también se pueden mostrar por composición (véase la Figura 13.57).

lista de parámetros

Se trata de una especificación de los valores que recibe una operación o una plantilla. Una lista de parámetros es una lista ordenada de declaraciones de parámetros. La lista puede estar vacía, en cuyo caso la operación se invocará sin parámetros.

Véase parámetro.

Notación

Una lista de parámetros es una lista de declaraciones de parámetros separadas mediante comas y encerrada entre parámetros.

(**parámetros**$_{\text{lista,}}$)

Los paréntesis se muestran aun cuando la lista esté vacía.

lista de propiedades

Alude a una sintaxis de texto adecuada para mostrar una propiedad o propiedades asociadas a un elemento, especialmente valores etiquetados, pero incluyendo también los atributos incorporados de los elementos del modelo.

Notación

Se trata de una o más especificaciones de propiedades separadas mediante comas y encerradas entre llaves. Las declaraciones de propiedad tienen la forma:

```
nombre-propiedad = valor
```

o bien

```
literal-propiedad
```

en donde en literal-propiedad es un valor enumerado exclusivo cuya aparición implica un nombre de propiedad único.

Ejemplo

{abstracto, autor=José, visibilidad=privada}

Opciones de presentación

Las herramientas pueden presentar las especificaciones de propiedades en líneas distintas o sin las llaves que las encierran, siempre y cuando estén marcadas adecuadamente para distinguirlas de otras informaciones. Por ejemplo, las propiedades de una clase pueden estar enumeradas bajo el nombre de la clase y empleando algún tipo de letra distintivo, tal como cursiva o alguna otra tipografía. Éste un problema de la herramienta.

Observe que las cadenas de propiedades se pueden utilizar para mostrar atributos incorporados, así como valores etiquetados, pero debería evitarse este tipo de utilización si la forma canónica es simple.

localización

Este concepto denota la ubicación física de una entidad propia del momento de la ejecución, tal como un objeto o un componente, dentro de un entorno distribuido. En UML, la localización es discreta y las unidades de localización son los nodos.

Véase también componente, nodo.

Semántica

El concepto de localización requiere el concepto de un espacio en que puedan existir las cosas. UML no modela toda la complejidad del universo tridimensional. Lo que hace es admitir un modelo topológico de espacios conectados mediante rutas de comunicación. Un nodo es un recurso de computación en el cual puede vivir una entidad propia del momento de la ejecución. Los nodos se conectan entre sí mediante rutas de comunicaciones que se modelan mediante asociaciones. La localización de una entidad se especifica haciendo referencia a un nodo. Dentro de un nodo, algunas entidades viven dentro de otras entidades anidadas. Por ejemplo, un objeto vive dentro de un componente o bien dentro de otro objeto. La localización de estas entidades es la entidad que las contiene.

Una instancia de objeto o de componente se puede trasladar a una nueva localización. Esto se puede modelar empleando la relación de conversión, que indica que en algún momento la primera entidad es sustituida por la segunda entidad, que posee otra localización diferente.

Notación

La localización de una instancia (incluyendo los objetos, instancias de componentes e instancias de nodos) dentro de otra instancia se puede mostrar físicamente por anidamiento, según se

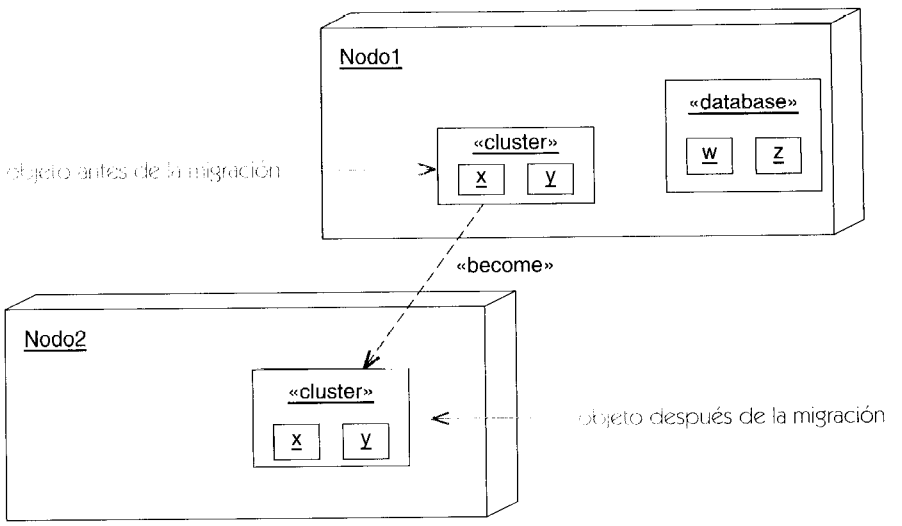


Figura 13.133 Nodos y migración de objetos

muestra en la Figura 13.133. La inclusión también se puede mostrar mediante flechas de composición. Alternativamente, una instancia puede poseer un marcador de propiedad llamado **localización** cuyo valor es el nombre de la instancia que lo contiene.

Si un objeto se traslada durante una interacción, puede aparecer como dos o más versiones con una transición de conversión **entre** versiones, tal como se muestra en la Figura 13.133. La flecha de conversión puede tener asociado un número de secuencia para mostrar el momento en que se traslada el objeto. Cada símbolo de objeto representa una versión del objeto durante una porción del tiempo total. Es preciso conectar los mensajes a la versión correcta del objeto (Figura 13.96).

llamada

Invocación a una operación.

Véase también activación, enviar, evento de llamada.

Semántica

Una llamada es la invocación de una operación en un punto determinado de la ejecución de un procedimiento. Mueve temporalmente un hilo de control desde el procedimiento que hace la llamada hacia el procedimiento llamado. La ejecución de un procedimiento que hace una llamada se bloquea durante la misma, ya que dicho procedimiento cede el control durante la ejecución de la operación y lo retoma cuando ésta termina. El procedimiento llamado recibe una lista de argumentos del procedimiento que realiza la llamada y un puntero implícito de retorno al procedimiento que realiza la llamada que apunta justo después de la instrucción de llamada. Cuando el procedimiento llamado finaliza, puede devolver una lista de valores.

A menudo la llamada se ejecuta dentro del espacio de direcciones del procedimiento que realiza la llamada, pero esto no es necesario para la semántica de la misma. Es más, resulta imposible hacerlo en un sistema distribuido, donde el receptor de la llamada puede estar separado físicamente del que llama. Mucho más importante resulta el hecho de establecer implícitamente el punto de retorno a la localización y entorno del procedimiento que realizó la llamada, pues permite restablecer el control al retornar de la misma. La localización del procedimiento que realiza una llamada se puede modelar mediante una línea de texto dentro de un procedimiento textual o mediante un estado dentro de una máquina de estados. Su entorno se modela utilizando una activación.

Una dependencia de uso de llamada modela una situación en la que una operación de la clase cliente (o la propia operación) llama a una operación de la clase proveedora (o a la propia operación). Esto se representa mediante el estereotipo «**call**».

Notación

Una llamada se representa en un diagrama de secuencia o de colaboración mediante un mensaje de texto dirigido a un objeto o clase destino.

Una dependencia de llamada se representa mediante una flecha discontinua etiquetada con el estereotipo «**call**», que sale de la clase que realiza la llamada y llega a la clase u operación llamada.

La mayoría de las llamadas se representarán como texto, formando parte del código de un procedimiento en un lenguaje de programación.

mal formado

Denota un modelo que se ha construido incorrectamente, por violar una o más de las reglas o restricciones predefinidas o especificadas por el modelo. Antónimo: bien formado.

Véase también conflicto, restricción.

Semántica

Un modelo que viola las reglas y restricciones de buena formación no es válido y por tanto tiene una semántica inconsistente. Si se intenta utilizar uno de estos modelos, se obtendrán resultados sin sentido. La herramienta de modelado tiene la responsabilidad de detectar los modelos mal formados e impedir su utilización en situaciones que pudieran dar lugar a problemas. Dado que el uso de ciertas estructuras extiende la semántica propia de UML, la verificación automática puede no ser posible en todos los casos. Además, no se puede esperar que las comprobaciones automáticas verifiquen la consistencia de las operaciones porque esto implicaría resolver el problema de la detención. Por tanto, las situaciones prácticas exigen una combinación de verificación automática y verificación humana.

Aun cuando un modelo terminado debe ser bien formado, las versiones intermedias del modelo pueden estar mal formadas en ocasiones, porque pueden ser fragmentos incompletos de al-

gún modelo final. La modificación de un modelo válido para producir otro modelo válido puede exigir ir pasando por modelos intermedios mal formados. Esto equivale a la modificación dc programas dc computador: el programa final que se le entrega al compilador tiene que ser válido, pero las copias de trabajo que residen en un editor dc texto suelen no ser válidas. Por tanto, los modelos mal formados deben poder modificarse y almacenarse empleando las herramientas de apoyo.

máquina de estados

Especificación de las secuencias dc estados por las que pasa un estado o una interacción en respuesta a los distintos eventos a lo largo de su vida, junto con las acciones de respuesta. Una máquina de estados se asocia a una clase, colaboración o método dc origen y especifica el comportamiento de las instancias del elemento origen.

Véase también grafo de actividades, estado compuesto, evento, pseudoestado, estado, transición.

Semántica

Una máquina de estados es un grafo de estados y transiciones que describe la respuesta de un instancia de un clasificador frente a la recepción de eventos. Las máquinas de estados pueden estar asociadas a clasificadores, tales como las clases y los casos de uso, así como a colaboraciones y a métodos. El elemento al que está asociada la máquina de estados se denomina maestro de la máquina de estados.

Una máquina de estados completa es un estado compuesto que se ha descomprimido recursivamente para formar subestados. Los estados simples más internos no poseen subestados.

Semántica de ejecución de una máquina de estados

La semántica de la ejecución dc una máquina de estados se discute en el resto de este artículo. Obsérvese que la sección siguiente describe los efectos semánticos de la ejecución de una máquina de estados y no debe tomarse como un enfoque de implementación. Hay muchas formas de implementar esta semántica, muchas de las cuales eliminan al compilar algunos de los pasos explícitos que se describen aquí. La mayor parte de esta semántica está descrita en otros artículos, pero se ha reunido aquí por comodidad.

En cualquier momento, uno o más estados pueden estar activados en la configuración de estado activa de la máquina de estados de un objeto o de otra instancia. Si un estado está activado, entonces se puede disparar una transición que abandone el estado, dando lugar a la ejecución de una acción y a la activación de otro estado o estados en lugar del estado original. Más de un estado hoja activo indica concurrencia interna. Existen restricciones para los estados que pueden estar activos concurrentemente, restricciones que vienen impuestas por la estructura de la máquina de estados y sus transiciones. En pocas palabras, si un estado secuencial compuesto está activado, entonces tiene que estar activo exactamente un subestado desjunto directo; si está activo un estado compuesto concurrente, todo subestado directo concurrente debe estar activo.

Disparo de transiciones y acciones

La suposición básica es que una máquina de estados procesa un evento de cada vez y termina con todas las consecuencias del evento antes de procesar otro. En otras palabras, los eventos no interaccionan con otros eventos durante el procesamiento de eventos. Esto se conoce con el nombre de procesamiento "ejecutar hasta finalizar". No significa que toda la computación sea no interrumpible. Una computación extensa ordinaria se puede descomponer en una serie de pasos atómicos y la computación puede ser interrumpida por un evento externo entre pasos consecutivos. Esto está muy cerca de la situación física que se tiene dentro de un computador, en la cual las interrupciones pueden producirse en pasos discretos pero pequeños.

Una suposición derivada es que los eventos son asíncronos. Dos eventos no pueden suceder nunca exactamente al mismo tiempo, o más precisamente, si se producen dos eventos exactamente al mismo tiempo, se trata de una coincidencia y se pueden procesar como si se hubieran producido en cualquier orden, sin pérdida de generalidad. Los resultados de los diferentes órdenes de ejecución puede ser diferentes —las situaciones de competencia son una propiedad esencial de los sistemas concurrentes— pero no se puede suponer la simultaneidad en un mundo distribuido. Toda computación que haga esta suposición es incorrecta desde el punto de vista lógico y desde el punto de vista físico. En el mundo distribuido, la ejecución concurrente exige la independencia.

Conceptualmente, las acciones son instantáneas y los eventos nunca son simultáneos. En una implementación, la ejecución de acciones requiere un cierto tiempo pero lo importante es que las acciones son (conceptualmente) atómicas y no interrumpibles. Si un objeto recibe un evento mientras está ejecutando una acción, el evento se pone en cola hasta que finalice la ejecución. Los eventos sólo se manejan cuando no se está ejecutando ninguna acción. Si una acción envía una señal a otro objeto, entonces la recepción de la señal no es síncrona. Se maneja como cualquier otro evento, una vez finalizada la acción y la transición de la que forma parte. Una llamada a una operación suspende al emisor hasta que se haya ejecutado la operación. Si el receptor lo decide así, puede ser implementada como un método o como un evento de llamada que dispare la máquina de estados del receptor. Para evitar problemas en los largos períodos en que no se puedan procesar eventos, las acciones deberían ser breves. Las acciones no están destinadas a modelar regiones protegidas o largas computaciones interrumpibles, que deberían modelarse como submáquinas o como estados de actividad anidados. Esto permite el procesamiento de eventos y hace posible interrumpir las computaciones anidadas. Si se incluyen acciones largas en sistemas reales, los eventos no podrán procesarse oportunamente. Esto es la consecuencia de un mal modelo. Las acciones tienen que ser cortas en comparación con el tiempo de respuesta requerido por los eventos que puedan producirse.

Cuando un objeto no está llevando a cabo una acción, maneja inmediatamente cualquier evento que reciba. Conceptualmente, las acciones son instantáneas pero en la práctica requieren un cierto tiempo; por tanto, es preciso almacenar los nuevos eventos en una cola del objeto. Si no hay eventos en la cola, el objeto espera hasta que recibe un evento y entonces lo procesa. Conceptualmente, los objetos manejan un único evento de cada vez. Esto no es una limitación, porque se supone que los eventos son breves y atómicos. En una implementación real, los eventos se pueden poner en cola siguiendo un determinado orden. La semántica de UML, sin embargo, no especifica un orden para procesar eventos concurrentes y el creador del modelo tampoco deberá suponer que existe. Si es preciso procesar eventos en un determinado orden, entonces debe construirse una máquina de estados para imponer ese orden. Es probable que una implementación real impusiera alguna regla simple de orden.

En el momento en que un objeto maneja un evento, su configuración del estado activo puede contener uno o más estados concurrentes. Cada estado recibe una copia independiente del evento y actúa en consecuencia de ella independientemente. Las transiciones de los estados concurrentes se disparan independientemente. Un subestado puede cambiar sin afectar a los demás, salvo en el caso de alguna transición compleja, tal como una bifurcación o una unión (que se describirán más adelante).

Para cada estado activo de un objeto, las transiciones salientes del estado son candidatos al disparo. Una transición candidata se dispara si maneja un objeto cuyo tipo sea el disparador de la transición o bien algún descendiente suyo. La transición no se dispara si recibe un predecesor del evento. Cuando se maneja un evento y se dispara una transición, se evalúa la condición de guarda de la transición. Si el valor de la condición de guarda es verdadero, entonces se habilita la transición. La expresión booleana de la condición de guarda puede contener argumentos del evento disparador, así como atributos del objeto. Observe que las condiciones de guarda no pueden producir efectos secundarios. Esto es, no pueden alterar el estado del objeto ni del resto del sistema. Por tanto, el orden en que se evalúen será irrelevante para el resultado. Una condición de guarda sólo se evalúa cuando se maneja un evento. Si es falsa, no se vuelve a evaluar si alguna de sus variables cambia posteriormente de valor.

Para estructurar condiciones complejas, se puede modelar una transición con múltiples segmentos. El primer segmento tiene un evento disparador y va seguido por un árbol bifurcado de segmentos con condiciones de guarda. Los nodos intermedios del árbol son pseudoestados, estados neutros que están presentes para estructurar las transiciones pero no pueden seguir activos al final de un paso que se ejecuta hasta finalizar. Toda posible ruta que pase a través del árbol de segmentos se considera una transición por separado y se puede seleccionar independientemente para su ejecución. Un segmento individual no se puede disparar por sí mismo. Todas las condiciones de guarda de una serie de segmentos tienen que ser ciertas, o bien la transición (incluyendo cualquiera de sus segmentos) no se disparará en modo alguno. En la práctica, las condiciones de guarda presentes en los puntos de bifurcación suelen particionar los posibles resultados. Por tanto, una implementación podría procesar la transición multisegmento paso a paso, pero no siempre.

Si no está habilitada ninguna transición, el evento se ignora sin más consideraciones. Esto no es un error. Si está habilitada exactamente una transición, se dispara. Si está habilitada más de una transición procedente de un único estado, entonces sólo se dispara una de ellas. Si no se ha especificado ninguna restricción, entonces la opción no es determinista. No se puede suponer que la opción será justa, predecible o aleatoria. Una implementación real podría proporcionar reglas para resolver conflictos, pero se advierte a los creadores de modelos que expliciten claramente su intención, en lugar de confiar en tales reglas. Tanto si se dispara alguna transición como si no, el evento se consume.

Las transiciones que abandonan un estado activo pueden ser seleccionadas para el disparo. Además, las transiciones de cualquier estado compuesto que contenga un estado activo son candidatas al disparo. Esto se puede considerar como una herencia de transiciones por parte de los estados anidados, de forma similar a la herencia de operaciones por parte de las subclases. una transición desde un estado que contiene al actual sólo puede seleccionarse para el disparo si no se dispara ninguna transición de algún estado interno, en cuyo caso queda enmascarada por la transición interna.

Cuando se dispara una transición, se ejecuta cualquier acción que pudiera estar asociada a ella. Una expresión de acción puede hacer uso de los argumentos del evento que causa el dis-

paro, así como de los atributos del objeto poseedor o de valores que resulten alcanzables desde él. Las acciones son atómicas y finalizan antes de procesar eventos adicionales. Si una transición tiene múltiples segmentos, los parámetros del evento disparador están disponibles como evento actual implícito.

Si un objeto posee estados concurrentes, entonces no deben interactuar mediante memoria compartida. Los estados concurrentes están destinados a ser independientes y deberían basarse en conjuntos de valores diferentes. Todas las interacciones deben ser explícitas, mediante el envío de señales. Si dos estados concurrentes tienen que acceder a un recurso compartido, deberán enviar explícitamente señales al recurso, que puede entonces actuar como árbitro. Una implementación puede eliminar al compilar esta comunicación explícita, pero hay que tener cuidado para asegurarse de que no surjan conflictos peligrosos o sin sentido. Si las acciones concurrentes acceden realmente a valores compartidos, el resultado no es determinista.

Si se dispara una transición que cruza los límites de un estado compuesto, se pueden ejecutar acciones de entrada o de salida. Se puede producir el cruce de un límite porque el estado de origen y el estado destino de la propia transición se encuentran en diferentes estados compuestos. También puede suceder porque la transición que se dispara se ha heredado de un estado compuesto externo, forzando entonces al objeto a salir de uno o más estados internos. Observe que una transición interna no produce un cambio de estado y por tanto ni invoca acciones de entrada ni de salida.

Para determinar cuáles son las acciones de entrada y de salida que se ejecutan, busque el estado activo actual del objeto (que puede estar anidado dentro del estado compuesto que tenga la transición) y el estado destino de la transición. Busque entonces el estado compuesto más interno que contenga tanto al estado actual como al estado destino. Llamémosle *antepasado común*. Las acciones de salida del estado actual y de todos los estados que lo contengan hasta el antepasado común (exclusive) se ejecutarán en primer lugar. Después se ejecuta la acción de transición. Después de esto, se ejecutan las acciones de entrada de los estados contenedores hasta el antepasado común (exclusive), empezando por el estado más externo. En otras palabras, se sale de los estados uno por uno hasta llegar al antepasado común y después se va entrando en un estado tras otro hasta llegar al estado destino. Las acciones de entrada y salida del antepasado común no se ejecutan porque no ha cambiado. Este procedimiento asegura que todos los estados queden fuertemente encapsulados.

La acción de la transición se ejecuta después de haber ejecutado las posibles acciones de salida y antes haber ejecutado ninguna de las posibles acciones de entrada.

Observe que el disparo de una autotransición (una transición que va de un estado a sí mismo) dará lugar a la salida de todos los estados anidados dentro del estado de origen que puedan estar activos (la transición puede haber sido heredada de un estado compuesto que lo contenga). También da lugar a la ejecución de la acción de salida del estado de origen seguida por la ejecución de su acción de entrada. En otras palabras, se sale del estado y se vuelve a entrar en él. Si no se desea este efecto, entonces debería utilizarse en su lugar una transición interna del estado. Esto no dará lugar a un cambio de estado, aun cuando el estado activo esté anidado dentro del estado que posee la transición.

Durante la ejecución de un paso que se ejecuta hasta finalizar, todas las acciones tienen acceso a un evento actual implícito, que es el evento que ha disparado la primera transición de dicho paso. Dado que puede haber más de un evento que pudiera dar lugar a la ejecución de una acción, la acción puede discriminar según el tipo de evento actual (tal como en Ada o mediante una operación polimórfica) para ejecutar bifurcaciones de código alternativas.

Una vez efectuadas todas las acciones, el estado actual original está inactivo (salvo que se trate del estado destino), el estado destino de la transición está activo y es posible procesar eventos adicionales.

Se puede estructurar una transición con varios segmentos cuyos nodos intermedios sean estados de unión. Cada segmento puede poseer su propia acción. Las acciones pueden estar entrelazadas con acciones de entrada y de salida para la transición global. Con respecto a las acciones de entrada y de salida, toda acción de un segmento de transición se produce en el lugar en que ocurriría si el segmento fuera una transición completa. Véase la Figura 13.96 para observar un ejemplo.

Transiciones internas

Una transición interna tiene un estado de origen, pero no un estado destino. Su disparo no da lugar a un cambio de estado, aun cuando la transición que se dispara se haya heredado de un estado que lo contenga. Dado que no cambia el estado, no se ejecuta ninguna acción de entrada ni de salida. El único efecto que tiene una transición interna es la ejecución de su acción. Las condiciones para el disparo de una transición interna son exactamente las mismas que para una transición externa.

Observe que el disparo de una transición interna puede enmascarar a una transición externa que haga uso del mismo evento. Por tanto, puede tener sentido definir una transición interna que no posea una acción asociada. Según se ha indicado anteriormente, sólo se dispara una transición por evento dentro de cada región secuencial y la transiciones internas tienen prioridad con respecto a las transiciones externas.

Las transiciones internas son útiles para procesar eventos sin cambiar de estado.

Estados iniciales y finales

Para el encapsulamiento de estados, suele ser deseable separar el interior del estado de su exterior. También es deseable conectar las transiciones a un estado compuesto, sin conocer la estructura interna del estado. Esto se puede lograr empleando estados iniciales y finales dentro de un estado compuesto.

Un estado puede tener un estado inicial y un estado final. Un estado inicial es un pseudoestado —un estado neutro con la conectividad de los estados normales— y un objeto no puede permanecer en un estado inicial. Un objeto puede permanecer en un estado final, pero un estado final no puede tener ninguna transición que tenga un disparador explícito; su propósito es invocar a una transición de finalización de algún estado contenedor. Un estado inicial tiene que tener alguna transición de finalización saliente. Si hay más de una transición saliente, entonces todas ellas deben carecer de disparadores y sus condiciones de guarda deben particionar los posibles valores. En otras palabras, cuando se invoca el estado inicial tiene que dispararse exactamente una transición saliente. Un objeto no puede permanecer nunca en el estado inicial, por lo que efectuará inmediatamente una transición a un estado normal.

Si un estado compuesto posee un estado inicial, entonces las transiciones se pueden conectar directamente al estado compuesto como destino. Toda transición al estado compuesto es, im-

plícitamente, una transición al estado inicial situado dentro del estado compuesto. Si un estado compuesto carece de estado inicial, entonces no se pueden hacer transiciones que tengan como destino el estado compuesto; es preciso conectarlas directamente a los subestados. Un estado que tenga un estado inicial también puede tener transiciones conectadas directamente a los estados interiores, así como al estado compuesto.

Si un estado compuesto posee un estado final, entonces puede ser el origen de una o más transiciones salientes de finalización, esto es, atributos que carecen de eventos disparadores explícitos.

Una transición de finalización es, en realidad, una transición que está habilitada implícitamente por la terminación de la actividad dentro de su estado. Por tanto una transición a un estado final es una indicación de que la ejecución del estado compuesto ha finalizado. Cuando un objeto hace una transición a un estado final, las transiciones de finalización que salen del estado compuesto que lo contiene están preparadas para dispararse si se satisfacen sus condiciones de guarda.

Un estado compuesto también puede poseer transiciones salientes etiquetadas —esto es, transiciones con eventos disparadores explícitos—. Si se produce un evento que da lugar a que se dispare una de estas transiciones, entonces toda actividad en curso dentro del estado (para cualquier nivel de anidamiento) se cancelará, se ejecutarán las acciones de salida de los estados anidados finalizados y se procesará la transición. Estas transiciones suelen utilizarse para modelar excepciones y situaciones de error.

Transiciones complejas

Una transición que llega a un estado compuesto concurrente implica una transición a todos sus subestados concurrentes. Esto puede suceder de dos formas.

Una transición puede tener múltiples estados destino, uno dentro de cada subestado concurrente. Observe que esta transición con bifurcación sigue teniendo un único evento disparador, una condición de guarda y una sola acción. Se trata de una transición explícita a un estado compuesto que especifica directamente todos los destinos. Esto representa una bifurcación explícita del control para descomponerse en subhilos concurrentes.

Alternativamente, una transición puede omitir los destinos de uno o más subestados concurrentes, o bien puede poseer como destino el estado compuesto en sí. En tal caso, cada subestado concurrente omitido debe tener dentro un estado inicial para indicar su estado de inicio por defecto. En caso contrario, la máquina de estados está mal formada. Si se dispara la transición compleja, los subestados concurrentes destino explícitos se activan, tal como sucede con los estados iniciales de los demás subestados concurrentes. En pocas palabras, toda transición a un subestado concurrente implica una transición a los estados iniciales de los demás subestados concurrentes equivalentes, que no se han mencionado explícitamente. Una transición al estado compuesto en sí implica una transición a los estados iniciales de cada una de sus regiones concurrentes. Si un estado compuesto concurrente está activo, también están activas todas sus subregiones.

De forma similar una transición desde cualquier subestado concurrente implica una transición procedente de todos ellos. Si un evento da lugar a que se dispare una de estas transiciones,

se hará que concluya la actividad de los subestados restantes, se ejecutarán sus acciones de salida, se ejecutará la acción de la transición y el estado destino pasará a activado, reduciéndose así el número de estados activos concurrentes.

La transición al estado final de un subestado concurrente no implica la terminación de los demás subestados concurrentes (no se produce una transición que salga del subestado). Cuando todos los subestados concurrentes han llegado a sus estados finales, entonces se entiende que el estado compuesto que los contiene ha finalizado su actividad y se habilitan para el disparo todas las transiciones de finalización que salgan del estado compuesto.

Una transición compleja puede tener múltiples estados de origen y múltiples estados destino. En tal caso, su comportamiento es la combinación de la bifurcación y la unión que se han descrito anteriormente.

Estado de historia

Un estado compuesto puede contener un estado de historia, que es un pseudoestado. Si una transición heredada da lugar a una salida automática del estado compuesto, el estado "recuerda" el subestado que estuviera activo cuando se produjo la salida forzada. Una transición al pseudoestado de historia contenido en el estado compuesto indica que debe reestablecerse el subestado que se recuerda. Una transición explícita a otro estado o al estado que lo contiene no habilita el mecanismo de historia y se aplican las reglas habituales de las transiciones. Sin embargo, el estado inicial del estado compuesto puede estar conectado al estado de historia. En tal caso, una transición al estado compuesto invoca (indirectamente) al mecanismo de historia. El estado de historia sólo puede tener una transición saliente de finalización sin condición de guarda; el destino de esta transición es el estado de historia por defecto. Si no se ha entrado nunca en la región del estado o si se ha efectuado una salida normal, entonces una transición al estado de historia va al estado de historia por defecto.

Existen dos clases de estado de historia: un *estado de historia próxima* y un *estado de historia profunda*. El estado de historia próxima restaura los estados contenidos directamente (con profundidad uno) en el mismo estado compuesto que el estado de historia. El estado de historia profunda restaura el estado o estados que estuvieran activos antes de la última transición explícita que haya dado lugar a una salida del estado compuesto que la contiene. Puede contener estados anidados dentro del estado compuesto hasta cualquier profundidad. Un estado compuesto puede tener, como máximo, un estado de historia de cada clase. Cada uno puede poseer su propio estado de historia por defecto.

Debería evitarse el mecanismo de historia si se puede modelar directamente la situación, porque es complicado y no tiene necesariamente un buen acuerdo con los mecanismos de implementación. El mecanismo de historia profunda resulta especialmente problemático y debería evitarse, a favor de otros mecanismos más explícitos (y más implementables).

Notación

Un diagrama de estados muestra una máquina de estados o una parte anidada de una máquina de estados. Los estados se representan mediante símbolos de estado y las transiciones se re-

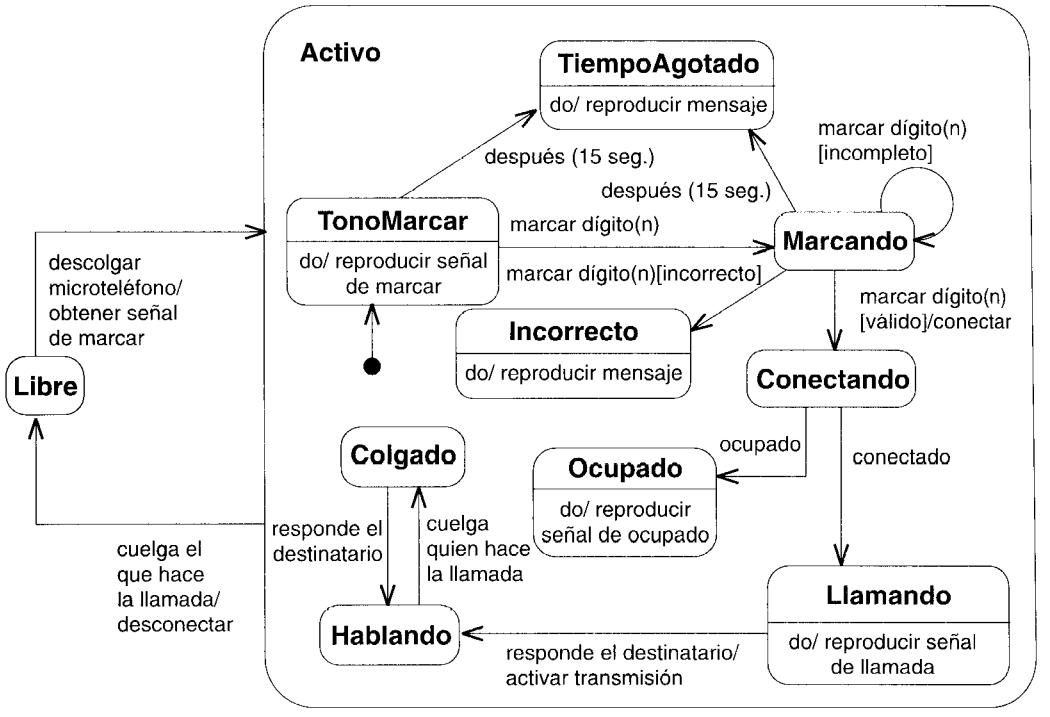


Figura 13.134 Diagrama de estados

presentan mediante flechas que conectan entre sí los símbolos de la seleccionada. Los estados también pueden contener subdiagramas, tanto físicamente como por apilamiento. La Figura 13.134 muestra un ejemplo.

La notación de diagramas de estado fue inventada por David Harel e incorpora ciertos aspectos de las máquinas de Moor (acciones de entrada) y de las máquinas de Mealy (acciones de transiciones) además de añadir los conceptos de estado anidado y de estado concurrente.

Para más detalles, *véase* estado, estado compuesto, submáquina, pseudoestado, acción de entrada, acción de salida, transición, transición interna y actividad. Para una forma variante de notación adecuada para el flujo de actividad, *véase* grafo de actividades. *Véase también* iconos de control para observar algunos símbolos opcionales destinados a ser utilizados en los diagramas de actividades, pero que pueden utilizarse en los diagramas de estado.

Discusión

Las máquinas de estados se pueden utilizar de dos maneras. Por tanto, su significado se puede comprender en cualquiera de estos dos sentidos. En un caso, la máquina de estados puede especificar el comportamiento ejecutable de su elemento maestro —típicamente, una clase—. En tal caso, la máquina de estados describe la respuesta del maestro cuando éste recibe eventos procedentes del resto del universo. La respuesta se describe mediante transiciones, cada una de las cuales indica lo que sucede cuando el maestro recibe un determinado evento mientras se encuentra en un estado dado. El efecto se expresa como una acción y un cambio de estado.

Entre las acciones se incluye enviar señales a otros objetos, que disparan las máquinas de estados de estos últimos. Las máquinas de estados proporcionan una especificación reduccionista del comportamiento de un sistema.

En el segundo caso, la máquina de estados puede utilizarse como una especificación de protocolo, que muestra el orden válido en que se pueden invocar las operaciones de una clase o de una interfaz.

En una de estas máquinas de estados, las transiciones que se disparan invocan a la operación deseada. Esto significa que se permite que el emisor invoque a la operación en ese momento. El protocolo de la máquina de estados no incluye acciones para especificar el comportamiento de la operación en sí. Muestra qué operaciones se pueden invocar en un orden particular. Esta máquina de estados especifica secuencias válidas de operaciones. Se está utilizando la máquina de estados como generador de secuencias en algún lenguaje (procedente de la teoría de lenguajes de las Ciencias de la Computación). Esta máquina hace las veces de una restricción aplicada al diseño del sistema. No es ejecutable directamente e indica lo que sucede si se produce una secuencia incorrecta, porque se supone que no pueden producirse. Es responsabilidad del diseñador del sistema asegurar que sólo se produzcan secuencias válidas. Este segundo modo de utilización es más abstracto que el primero, que especifica de forma ejecutable lo que sucede en todos los casos, pero suele ser cómodo, sobre todo en niveles altos y con codificación procedimental.

marca de tiempo

Notación que indica el instante en que se produce un evento o mensaje. Se pueden utilizar las marcas de tiempo en las restricciones.

Semántica

Una marca de tiempo se forma como una expresión que parte del nombre del mensaje. Se le puede dar nombre a un mensaje de una interacción para que sea posible formar expresiones de marca de tiempo a partir de él. En las expresiones siguientes mensaje es el nombre de un mensaje.

mensaje.horaEnvío()	Instante en que se envía el mensaje
mensaje.horaRecepción()	Instante en que se recibe el mensaje

Notación

Las expresiones de marca de tiempo se muestran en forma de texto.

Ejemplo

La restricción siguiente limita el tiempo necesario para generar una señal de marcar.

```
{tonomarcar.hora() - descolgado.hora() < 1 segundo}
```

marcador

Valor selector en una pareja de valor etiquetado. Representa el nombre de una propiedad definida en el tiempo de modelado.

marcador gráfico

Es un elemento de notación tal como la geometría, textura, trama de relleno, tipografía, color y demás.

Véase también uso de la tipografía.

Notación

Los símbolos de notación se construyen a partir de distintos marcadores gráficos. Ningún marcador gráfico tiene significado semántico por sí mismo, pero el objetivo de la notación es utilizar marcadores gráficos de forma tan consistente y ortogonal como sea posible.

Algunos marcadores gráficos se emplean para construir símbolos predefinidos de UML, mientras que otros marcadores gráficos se emplean en la notación canónica. Por ejemplo, no se ha asignado significado alguno a los colores, porque hay muchas impresoras que no lo producen y ciertas personas no distinguen todos los colores. Los marcadores gráficos que no estén reservados, como los colores, podrán emplearse dentro de las herramientas de edición para cualquier propósito propio de quien hace el modelado o de la herramienta.

UML permite una extensión gráfica limitada de su notación. Se puede asociar un icono o marcador gráfico (tal como una textura o un color) a cualquier estereotipo. UML no especifica la forma de la especificación gráfica. Sin embargo, existen muchos formatos de mapa de bits o de trazos que podrían ser empleados por un editor gráfico (aunque su transportabilidad es un problema difícil).

Se pueden concebir formas más generales de especificación y sustitución de iconos pero dejaremos estos problemas para el ingenio de los constructores de herramientas; advertimos que un uso excesivo de las capacidades de extensión puede dar lugar a dificultades para cambiar de herramientas usadas.

marco de trabajo

Una arquitectura genérica que proporciona una plantilla ampliable para su aplicación dentro de un dominio. Muy ampliamente difundido el término inglés "framework".

Véase paquete.

materialización

Acto de materializar algo. *Véase* materializar.

materializar

Tratar como objeto algo que normalmente no se considera como objeto.

Discusión

La materialización tiene un significado filosófico y literario de larga trayectoria. Se utilizaba para describir la caracterización de conceptos abstractos como cosas o personas en la mitología y en la poesía. Por ejemplo, el dios Thor era una materialización del trueno. La teoría de los ideales de Platón invertía las cosas con respecto a los conceptos vigentes. Consideraba que los conceptos puros, como Belleza, Bien y Valor eran la realidad eterna; consideraba que las instanciaciones físicas eran copias imperfectas: es la materialización llevada hasta el último extremo.

La materialización es una de las ideas más útiles para la orientación a objetos y subyace en casi todos los aspectos del modelado. En primer lugar, la construcción de un modelo requiere imponer los objetos a un mundo continuo. Los seres humanos hacemos esto de forma natural en todas las frases que pronunciamos; un sustantivo es la materialización de una cosa y un verbo es la materialización de una acción La materialización resulta especialmente útil cuando se aplica a las cosas de modelos o programas que no han comenzado siendo tratadas como objetos; tal podría ser el comportamiento dinámico. Casi todas las personas piensan en las operaciones como en objetos, pero ¿qué sucede con la ejecución (palabra que es, en sí, una materialización) de una operación? En general, las personas piensan que se trata de un proceso. Pero si se materializa y se le da un nombre (digamos activación) entonces resulta repentinamente posible darle propiedades, formar relaciones con otros objetos, manipularla y almacenarla. La materialización de un comportamiento transforma los procesos dinámicos en estructuras de datos que se pueden almacenar y manipular. Se trata de un concepto muy potente para el modelado y la programación.

mensaje

Denota el hecho de aportar información de un objeto (u otra instancia) a otro, con esperanzas de que esto dé lugar a alguna actividad. Un mensaje puede ser una señal o la llamada a una operación. La recepción de una instancia de mensaje se considera normalmente una instancia de evento.

Véase también llamada, colaboración, interacción, operación, enviar, señal.

Semántica

El término mensaje alude a enviar una señal de un objeto (el *emisor*) a otro u otros objetos (los *receptores*); también puede ser la llamada a una operación aplicada a un objeto (el *receptor*) por parte de otro objeto (el *emisor*, quien hace la llamada). La implementación de un mensaje puede adoptar distintas formas, tales como una llamada a procedimiento, una comunicación de procesos entre hilos activos, el envío explícito de un evento y así sucesivamente. En el nivel ló-

gico, enviar una señal y llamar a una operación son cosas similares. Ambas implican una comunicación procedente de un emisor y que llega a un receptor, en la cual se pasa información por valor que el receptor emplea par determinar lo que tiene que hacer. Una llamada puede considerarse una trama de señales que implica un envío con un argumento de puntero con retorno implícito que se emplea para enviar una señal de retorno al emisor. Se puede modelar una llamada como dos mensajes, un mensaje de llamada y otro mensaje posterior de retorno. En el nivel de implementación, las señales y las llamadas poseen propiedades y comportamientos detallados diferentes, por lo que se toman como elementos distintos en UML.

La recepción de una señal puede disparar una transición de la máquina de estados en el receptor. Una llamada puede manejarse de dos maneras, a elección del receptor (la elección del receptor tiene que efectuarse en el modelo). Se puede implementar un operación como cuerpo de un procedimiento (un método) que será invocado cuando llegue una llamada. Una vez finalizada la ejecución del procedimiento, el emisor recupera el control y puede recibir un valor opcional. Alternativamente, para un objeto activo, una llamada a una operación puede dar lugar a un evento de llamada, que a su vez dispara una transición de la máquina de estados. En este caso no existe el cuerpo del método, sino que la transición puede tener acciones. Cuando la transición ha finalizado, o bien inmediatamente si el evento de llamada no dispara una transición, el emisor recupera el control.

Los mensajes incluyen una expresión adecuada para un conjunto de objetos blanco. Se envía el mensaje a todos los objetos del conjunto. Salvo que se indique expresamente lo contrario (mediante una restricción), los mensajes se envían concurrentemente a todos los objetos del conjunto. Esto significa que el orden de ejecución es completamente arbitrario y podría ser paralelo. Si es preciso que se envíen los mensajes en un orden particular, deberán enviarse desde el interior de un bucle. Si se produce una llamada, el emisor recupera el control cuando han finalizado todas las llamadas.

El momento en que se envía o recibe un mensaje se puede representar mediante una expresión en el nombre del mensaje.

Véase marca de tiempo.

Estructura

El mensaje posee un emisor, un receptor y una acción.

Dentro de una interacción, el emisor es el rol de clasificador que envía el mensaje. El receptor es el rol de clasificador que recibe el mensaje. La acción es una llamada; una señal, una operación local aplicada al emisor o una acción primitiva tal como crear o destruir. La acción incluye una lista de argumentos, una expresión para un conjunto de receptores y una referencia de la operación o señal implicadas. También puede incluir una especificación de condicionalidad e iteración de la ejecución del mensaje.

Dentro de una interacción, los mensajes están relacionados mediante la relación predecesor-sucesor y la relación emisor-receptor. Esta última relación es aplicable a los métodos de procedimientos. Cada llamada añade un nivel de anidamiento a la secuencia. Dentro de una llamada, los mensajes se ordenan secuencialmente; existe la posibilidad de subsecuencias concurrentes.

La relación predecesor-sucesor (o de secuenciación) organiza los mensajes del hilo en una secuencia lineal. Si hay dos mensajes con un predecesor común y no están secuenciados de al-

guna otra forma, entonces se pueden ejecutar concurrentemente. Si un mensaje posee múltiples predecesores, tiene que esperar hasta que finalicen todos ellos. Ese mensaje es un punto de sincronización.

La relación emisor-receptor (o de activador) define una estructura de procedimientos anidada. El mensaje que llama a un procedimiento (empleando una acción de llamada) es el activador de todos los mensajes que forman el cuerpo del procedimiento invocado. Entre ellos, los mensajes invocados tienen una relación predecesor-sucesor que establece su orden relativo (y puede permitir la concurrencia).

Si un mensaje es una llamada, entonces se bloquea al emisor hasta que el procedimiento invocado finaliza y retorna. Sin embargo, si el receptor maneja la operación como evento de llamada entonces el retorno se produce cuando finaliza la transición inicial, tras lo cual el emisor recupera el control y el receptor puede reanudar su propia ejecución.

Las relaciones de secuenciación y de activador sólo relacionan a aquellos mensajes que posean la misma interacción.

Notación

La notación de los diagramas de secuencias y de los diagramas de colaboración es diferente.

Diagrama de secuencia

En un diagrama de secuencia, un mensaje se representa mediante una flecha continua que va desde línea de vida de un objeto (el emisor) hasta la línea de vida de otro objeto (el destino). Si la flecha es perpendicular a las líneas de vida, entonces se considera que la transmisión del mensaje es instantánea o al menos muy rápida en comparación con los mensajes externos. Si la flecha es oblicua, entonces se considera que la transmisión del mensaje tiene una duración, durante al cual se podrían enviar otros mensajes. En el caso de un mensaje que vaya del objeto hasta sí mismo, la flecha puede comenzar y acabar en la misma línea de vida. Las flechas de mensajes se organizan por orden secuencial verticalmente, empezando por arriba y avanzando hacia abajo. Si dos mensajes son concurrentes, su orden relativo no es significativo. Los mensajes pueden tener números de secuencia pero dado que el orden relativo de mensajes se muestra visualmente, los números de secuencia suelen omitirse.

Retardo de transmisión. Normalmente las flechas de mensajes se dibujan horizontalmente, lo cual indica que la duración necesaria para enviar el mensaje es atómica, esto es, que es breve en comparación con la granularidad que tiene la interacción y nada más puede "suceder" durante la transmisión del mensaje. Esta suposición es correcta en muchos computadores. Si el mensaje requiere un cierto tiempo para entregarlo, durante el cual puede ocurrir algo más (tal como un mensaje en dirección opuesta) entonces la flecha de mensaje se puede inclinar hacia abajo para que la punta de la flecha quede por debajo del origen de la misma.

Bifurcaciones. Las bifurcaciones se muestran mediante múltiples flechas que parten de un mismo punto y cada una de ellas está etiquetada con su propia condición de guarda. Dependiendo de si las condiciones de guarda son o no mutuamente excluyentes, esta estructura puede representar la condicionalidad o la concurrencia.

Iteración. Los conjuntos conectados de mensajes pueden estar encerrados y marcados como iteración. Los marcadores de iteración denotan que el conjunto de mensajes puede aparecer en múltiples ocasiones. Se puede especificar la condición de continuación para un procedimiento en la parte inferior de la iteración. Si hay concurrencia, entonces algunos mensajes del diagrama pueden formar parte de la iteración y otros pueden ejecutarse individualmente.

Diagramas de colaboración

En un diagrama de colaboración, los mensajes se representan mediante una flechita con etiqueta asociada a una ruta que une los objetos emisor y receptor. La ruta es la que se emplea para acceder al objeto blanco. La flecha apunta a lo largo de la ruta en la dirección del objeto destino. En el caso de que sea un mensaje que sale del objeto para volver a sí mismo, el mensaje aparece en una ruta que vuelve al mismo objeto y el extremo que llega al blanco posee la etiqueta «**self**».

Se puede asociar más de un mensaje a un enlace, tanto en la misma dirección como en direcciones distintas. El orden relativo de mensajes se muestra mediante la porción de número de secuencia que aparece en la etiqueta del mensaje.

Ambos diagramas

La flecha de mensaje tiene en su etiqueta el nombre del mensaje (el nombre de la señal o de la operación) y los valores de sus argumentos. La flecha también puede tener en su etiqueta un número de secuencia, para mostrar la posición del mensaje dentro de la interacción global. Los números de secuencia se pueden omitir en los diagramas de secuencias, en los cuales la situación física de la flecha muestra la posición relativa del mensaje, pero en los diagramas de colaboración resultan necesarios. Los números de secuencia resultan útiles en los dos tipos de diagramas para identificar los hilos de control concurrentes. También se puede poner en la etiqueta del mensaje una condición de guarda.

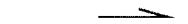
Tipo de flujo de control. Es posible emplear las siguientes variantes de puntas de flecha para mostrar las diferentes clases de flujo de control en los mensajes.

Punta de flecha con relleno 

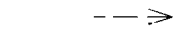
Llamada a procedimiento u otro flujo de control anidado. La secuencia anidada completa debe finalizar antes de reanudar la secuencia de nivel externo. Se puede emplear en llamadas normales a procedimiento. También se puede usar con objetos activos concurrentemente cuando uno de ellos envía una señal y espera a que finalice una secuencia de comportamiento anidada.

Punta de flecha sin relleno 

Flujo de control plano. Cada flecha muestra la progresión al próximo paso de la secuencia. En el caso de procedimientos anidados, esto se corresponde con un barrido de las hojas del árbol de acciones efectuado lateralmente desde el fondo.

Media punta de flecha 

Flujo de control asíncrono. Se emplea en lugar de la cabeza de flecha hueca para mostrar explícitamente un mensaje asíncrono entre dos objetos de una secuencia de procedimientos.

Flecha discontinua con punta sin relleno 

Retorno de una llamada a procedimiento. La flecha de retorno puede suprimirse, por cuanto queda implícita al final de la activación.

Otras variantes

Se pueden mostrar otras clases de control, tales como "rehusar" o "tiempo agotado", pero se tratan como extensiones del núcleo de UML.

Etiqueta de mensaje. La etiqueta tiene la sintaxis siguiente:

$$\text{predecesor}_{opt}\ \text{condición-de-guarda}_{opt}\ \text{expresión-secuencia}_{opt}$$

$$\overline{\text{lista-valores-retorno}} :=_{opt} \text{nombre-mensaje}\ (\text{argumentos}_{lista,})$$

La etiqueta denota el mensaje enviado, sus argumentos y valores proporcionados y la situación del mensaje dentro de la interacción más extensa, incluyendo el anidamiento de llamadas, iteración, bifurcación, concurrencia y sincronización.

Predecesor. En una colaboración, el predecesor es una lista de números de secuencia separados por comas, y seguidos por una barra (/).

$$\text{número-secuencia}_{lista,}\ /$$

La cláusula se omite si la lista está vacía.

Todo número-secuencia es una expresión-secuencia sin términos de recurrencia. Tiene que coincidir con el número-secuencia de algún otro mensaje.

El significado es que el flujo de mensajes no queda habilitado mientras no se hayan producido todos los flujos de mensaje cuyos números de secuencia estén enumerados (un hilo puede ir más allá del flujo de mensajes y la condición de guarda seguirá cumpliéndose). Por tanto, la condición de guarda representa una sincronización de hilos.

Observe que el mensaje que corresponde al número de secuencia precedente numéricamente es un predecesor implícito y no necesita ser enumerado explícitamente. Todos los números de secuencia que tienen igual prefijo forman una secuencia. El predecesor numérico es aquel en el cual el término final es uno menos. Esto es, el número 3.1.4.5 es el predecesor de 3.1.4.6.

En un diagrama de secuencia el orden visual determina la secuenciación y se muestra una sincronización por la presencia de múltiples mensajes que llegan a un mismo objeto antes de que el objeto envíe algún mensaje propio.

Expresión de secuencia. La expresión de secuencia es una lista de términos de secuencia separados por puntos y seguida por dos puntos (':'). Cada término representa un nivel de

anidamiento procedural dentro de la interacción global. Si es concurrente todo el control, entonces no se produce el anidamiento. Todos los términos de la secuencia poseen la siguiente sintaxis:

etiqueta recurrencia$_{opt}$

en donde etiqueta es entero o bien nombre

El entero representa el orden secuencial del mensaje dentro del nivel inmediatamente superior de llamadas a procedimientos. Los mensajes que difieren en un término entero están relacionados secuencialmente en ese nivel de anidamiento. Por ejemplo, el mensaje 3.1.4 sigue al mensaje 3.1.3 dentro de la activación 3.1.

El nombre representa un hilo de control concurrente. Los mensajes que difieren en el nombre final son concurrentes en ese nivel de anidamiento. Por ejemplo, el mensaje 3.1a y el mensaje 3.1b son concurrentes dentro de la activación 3.1. Todos los hilos de control son iguales dentro de cada profundidad de anidamiento.

La recurrencia representa la ejecución condicional o iterativa. Esto representa cero o más mensajes que se ejecutan, dependiendo de las condiciones. Las opciones son:

* [cláusula de iteración]	una iteración
[cláusula de condición]	una bifurcación

Una iteración representa una secuencia de mensajes en la profundidad de anidamiento dada. La cláusula de iteración se puede omitir (en cuyo caso las condiciones de iteración no estarán especificadas). La cláusula de iteración debería expresarse en pseudocódigo o en algún lenguaje de programación real; UML no prescribe su formato. Un ejemplo podría ser: *[i:= 1..n].

Una condición representa un mensaje cuya ejecución es contingente respecto a la veracidad de la cláusula de condición. La cláusula de condición debería expresarse en pseudocódigo o en algún lenguaje de programación real; UML no prescribe su formato. Un ejemplo podría ser: [x > y].

Obsérvese que las bifurcaciones se anotan igual que una iteración sin asterisco. Pueden pensarse que se trata de una iteración limitada a un solo caso.

La notación de iteración supone que los mensajes de la iteración se ejecutarán secuencialmente. También existe la posibilidad de ejecutarlos concurrentemente. La notación empleada consiste poner una doble línea vertical, indicadora de paralelismo, después del asterisco (*||).

Obsérvese que en una estructura de control anidada la recurrencia no se repite en niveles internos. Cada nivel de estructura especifica su propia iteración dentro del contexto que la contiene.

Signatura. Una signatura es una cadena que indica el nombre, argumentos y valor proporcionado por una operación, mensaje o señal. Poseen las propiedades siguientes:

lista-valores-proporcionados	Se trata de una lista de nombres separados mediante comas que denota los valores proporcionados por el mensaje en la

ejecución subsiguiente de la interacción global. Si el mensaje no proporciona ningún valor, entonces se omite el valor proporcionado y el operador de asignación.

nombre-mensaje

El nombre del evento surgido en el objeto destino (suele ser el evento que solicita la realización de una operación). Puede implementarse de varias maneras, *una* de las cuales es una llamada a una operación. Si se implementa mediante una llamada a procedimiento, entonces se trata del nombre de la operación y esa operación tiene que estar definida en la clase del receptor, o debe ser heredada por él. En otros casos puede ser el nombre de un evento que surge en el objeto receptor. En la práctica habitual, con sobrecarga de procedimientos, para identificar a una operación se necesita tanto el nombre del mensaje como la lista de tipos de los argumentos.

lista-argumentos

Se trata de una lista de argumentos separados mediante comas, encerrada entre paréntesis. Se pueden utilizar los paréntesis aun cuando la lista esté vacía. Cada argumento es una expresión en pseudocódigo o bien en un lenguaje de programación adecuado (UML no lo prescribe). La expresión puede utilizar los valores proporcionados por mensajes anteriores (del mismo alcance) y expresiones de navegación que partan del objeto original (esto es, atributos del mismo o enlaces que partan de él y rutas alcanzables desde ellos).

Ejemplo

Lo que sigue son ejemplos de la sintaxis de etiquetas de mensajes de control.

2: visualizar (x, y)	Mensaje simple
1.3.1: p:= buscar(especificaciones)	Llamada anidada que proporciona un valor
[x < 0]4: invertir (x, color)	Mensaje condicional
3.1 *: update ()	Iteración
A3, B4/ C2: copy(a,b)	Sincronización con otros hilos

Opciones de representación

En lugar de emplear expresiones de texto para los argumentos y valores proporcionados, se pueden mostrar símbolos de datos junto al mensaje (Figura 13.135). Un símbolo es un circulito etiquetado con la expresión de argumento o con el nombre del valor proporcionado. Posee una flechita que apunta a lo largo del mensaje (para un argumento) o en dirección opuesta al mensaje (para un valor proporcionado). Los símbolos representan argumentos y valores proporcionados.

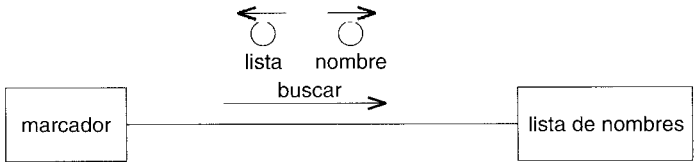


Figura 13.135 Símbolos de flujo de valores

La selección de sintaxis basada en texto o en símbolos es una opción de representación pero el texto resulta más compacto y se recomienda para la mayoría de los propósitos.

La sintaxis de los mensajes se puede expresar en la sintaxis de un lenguaje de programación tal como C++ o SmallTalk. Sin embargo, todas las expresiones de un mismo diagrama deberán utilizar una misma sintaxis.

metaclase

Una clase cuyas instancias son clases a su vez. Las metaclases se utilizan típicamente para construir metamodelos. Las metaclases se pueden modelar como un estereotipo de una clase empleando la palabra reservada «**metaclass**».

Véase también supratipo.

meta-metamodelo

Denota un modelo que define el lenguaje para expresar un metamodelo. La relación entre un meta-metamodelo y un metamodelo es análoga a la existente entre un metamodelo y un modelo. Este nivel de indirección sólo suele ser relevante para los constructores de herramientas, constructores de bases de datos y casos similares. UML está definido en términos de un meta-metamodelo, al que se denomina Servicio de Metaobjetos (MOF)[4].

metamodelo

Este término denota un modelo que define el lenguaje empleado para expresar un modelo; también puede ser una instancia de un metamodelo. El metamodelo UML define la estructura de los modelos UML.

metaobjeto

Término genérico para todas las entidades que aparecen en un lenguaje de metamodelo. Por ejemplo, los metatipos, metaclases, metaatributos y metaasociaciones.

---

[4] Meta-Object Facility en el original. *N. del T.*

metarelación

Este término agrupa aquellas relaciones que conectan a los descriptores con sus instancias. Entre ellas se cuentan la relación de instancia y la relación de supratipo.

método

Implementación de una operación. Especifica el algoritmo o procedimiento que da lugar a los resultados de una operación.

Véase también concreto, operación, realización.

Semántica

Un método es la implementación de una operación. Si una operación no es abstracta, entonces tiene que tener un método o un evento de llamada, bien sea definido en la clase que tiene esa operación o bien heredado de algún antecesor. Los métodos se especifican como expresiones de procedimientos, cadenas lingüísticas escritas en un determinado lenguaje (tal como C++, SmallTalk o algún lenguaje humano) que describen un algoritmo. El lenguaje debe ser adecuado para el propósito, por supuesto. Un lenguaje humano, por ejemplo, puede ser adecuado para un análisis inicial pero inadecuado para la generación de código.

La declaración de una operación implica la presencia de un método salvo que se declare abstracta el parámetro. En una jerarquía de generalización, cada declaración repetida de la operación implica un nuevo método que anula a cualquier método heredado de esa misma operación. Dos declaraciones representan a una misma operación si coinciden sus signaturas.

Observe que un método es un procedimiento ejecutable —un algoritmo— y no simplemente una especificación de resultados. Una especificación de antes y después no es un método. Un método es un compromiso con la implementación y con los problemas de direcciones de un algoritmo, complejidad computacional y encapsulamiento.

En ciertos sentidos, un método puede tener propiedades más estrictas que las de su operación. Un método puede ser una consulta aun cuando la operación no se declare como consulta. Pero si la operación es una consulta entonces el método tiene que ser una consulta. Análogamente, un método puede fortalecer la propiedad de concurrencia. Una operación secuencial se puede implementar con un método con guarda o concurrente. En tales casos el método es consistente con las declaraciones de su operación. Sólo fortalece las restricciones.

Notación

La presencia de un método se indica mediante una declaración de operación que carece de la propiedad abstracta (Figura 13.136). Si la operación se hereda, entonces se puede mostrar el método repitiendo la declaración de la operación en texto normal (sin cursiva) para mostrar una operación concreta. El texto del cuerpo del método se puede mostrar en forma de una nota asociada a la entrada de lista de la operación, pero normalmente los cuerpos de los métodos no se

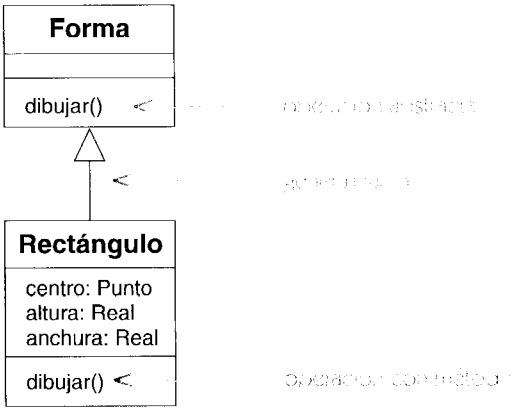


Figura 13.136 Un método de una operación no abstracta

muestran en todos los diagramas. Permanecen ocultos para que un editor de textos los muestre cuando se le pida.

miembro

Se trata del nombre de un constituyente estructural heredable y dotado de nombre que pertenece a un clasificador, bien sea un atributo, una operación o un método. Todo clasificador puede poseer una lista de cero o más de cada una de las clases de miembros. Las listas de miembros de una clase dada se denotan mediante una lista de cadenas situada dentro de un compartimento del símbolo clasificador.

Véase también lista.

modelo

Es una abstracción semánticamente completa de un sistema.

Véase también paquete, subsistema.

Semántica

Un modelo es una abstracción más o menos completa de un sistema desde un determinado punto de vista. Es completo en el sentido de que describe en su totalidad al sistema o entidad, con el nivel seleccionado de precisión y punto de vista. Los distintos modelos proporcionan, en esencia, puntos de vista independientes que se pueden manipular por separado.

Un modelo puede tener una jerarquía de contención formada por paquetes en la cual el paquete del nivel más elevado corresponde a todo el sistema. El contenido de un modelo es el cierre transitivo de sus relaciones de contención (propiedad) desde los paquetes de nivel superior hasta los elementos del modelo.

Los modelos también pueden incluir partes relevantes del entorno del sistema, representado, por ejemplo, por actores y por sus interfaces. En particular, se puede modelar la relación del entorno con los elementos del sistema. El sistema y su entorno forman un sistema más amplio y en un nivel de alcance más elevado. Por tanto, es posible relacionar elementos de diferentes niveles de forma sencilla.

Los elementos de diferentes modelos no se afectan directamente entre sí, pero suelen representar los mismos conceptos en distintos niveles de detalle o diferentes fases de desarrollo. Por tanto, las relaciones existentes entre ellos, tales como el seguimiento y el refinamiento, son importantes para el proceso de desarrollo en sí, y suelen capturar importantes decisiones de diseño.

Notación

Se puede mostrar un modelo como un paquete con el estereotipo «**model**». Sin embargo, poco es el detalle de notación que se puede mostrar acerca de los modelos. Las herramientas pueden mostrar listas de modelos pero los modelos tienen pocas relaciones entre sí. Lo más útil es la capacidad de pasar desde el nombre de un modelo hacia su paquete de más alto nivel o hasta un mapa de su contenido global.

Discusión

No hay ninguna vista de un sistema que pueda ser completa per se y lo mismo puede decirse de los sistemas. Siempre hay conexiones con el mundo exterior y los modelos nunca llegan a los límites de la realidad. Por tanto, el concepto de modelo cerrado siempre es una aproximación en la cual es preciso especificar unos límites arbitrarios para trabajar en la práctica.

Los modelos UML se representan como una jerarquía de paquetes que hace hincapié en una visualización del sistema. Cada modelo puede poseer su propia jerarquía de niveles y esa jerarquía puede ser similar o distinta de la jerarquía de niveles de otras vistas del sistema.

modelo de casos de uso

Modelo que describe los requisitos funcionales de un sistema u otro clasificador en términos de casos de uso.

Véase actor, caso de uso.

Semántica

El modelo de casos de uso representa la funcionalidad de un sistema u otro clasificador tal como lo perciben quienes interactúan con el sistema desde el exterior. Los modelos de casos de uso se muestran en un diagrama de casos de uso.

módulo

Término que denota una unidad para el almacenamiento y manipulación del software. Entre los módulos se cuentan los módulos de código fuente, los módulos de código binario y los módulos de código ejecutable. La palabra no corresponde a una única estructura de UML, sino que incluye varias estructuras.

Véase componente, paquete, subsistema.

muchos

Se trata de una abreviatura de la multiplicidad **0..*** —esto es, cero o más sin límites—. En otras palabras, un tamaño carente de restricciones.

Véase multiplicidad.

multiobjeto

Alude a un rol de clasificador que denota un conjunto de objetos y no un único objeto.

Véase también rol de clasificador, colaboración, mensaje.

Semántica

El multiobjeto es un rol de clasificador que denota un conjunto de objetos, normalmente el conjunto de objetos del extremo *muchos* de una asociación. Se utiliza un multiobjeto dentro una colaboración para mostrar operaciones que tratan a todo el conjunto de objetos como a una unidad y no a un único objeto. Por ejemplo, una operación para buscar un objeto dentro de un conjunto opera sobre todo el conjunto y no sobre un objeto individual. El modelo estático subyacente no queda afectado por este agrupamiento.

Notación

Los multiobjetos se representan mediante dos rectángulos en los cuales el rectángulo superior está desplazado vertical y horizontalmente para sugerir una pila de rectángulos (Figura 13.137).

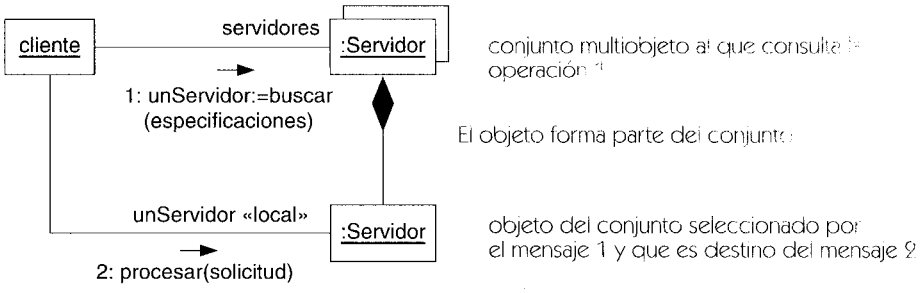


Figura 13.137 Multiobjeto

Una flecha de mensaje situada junto al símbolo del multiobjeto denota un mensaje que llega al conjunto de objetos —por ejemplo, una operación de selección para buscar un objeto individual.

Se necesitan dos mensajes para aplicar una operación a todos los objetos de un conjunto de objetos asociado: una iteración del multiobjeto para extraer enlaces de los objetos individuales y después un mensaje que se envía a cada uno de los objetos, empleando el enlace (temporal). Esto se puede omitir en un diagrama combinando los mensajes en uno que incluya una iteración y una aplicación a cada objeto. El nombre de rol destino recibe un indicador de *muchos* (*) para mostrar que están implicados muchos enlaces. Aun cuando esto puede escribirse como un único mensaje, en el modelo subyacente (y en todo código real) se necesitarán las dos capas de estructura (iteración para hallar los enlaces y un mensaje que hace uso de cada enlace) mencionadas anteriormente.

Un objeto del conjunto se representa mediante un símbolo normal de objeto pero puede estar asociado al símbolo de multiobjeto empleando un enlace de composición para indicar que forma parte del conjunto. Una flecha de mensaje que va hacia el símbolo del objeto simple denota un mensaje a un objeto individual.

Típicamente, un mensaje de selección enviado a un multiobjeto proporciona una referencia de un objeto individual al cual enviará entonces un mensaje el emisor original.

multiplicidad

Una especificación del intervalo de valores de cardinalidad admisibles —el tamaño, esto es— que puede adoptar un conjunto. Se pueden dar especificaciones de multiplicidad para extremos de asociación, partes de clases compuestas, repeticiones de mensajes y otros propósitos. En esencia una multiplicidad es un subconjunto (posiblemente infinito) de los enteros no negativos. Contrastar con: cardinalidad.

Véase también multiplicidad (de asociación), multiplicidad (de clase).

Semántica

La multiplicidad es una restricción de la cardinalidad (tamaño) de un conjunto. En principio, se trata de un subconjunto de los enteros no negativos. En la práctica, suele tratarse de un conjunto finito de intervalos enteros; en casi todas las ocasiones se trata de un único intervalo con un valor mínimo y otro máximo. Todo conjunto debe ser finito, pero el límite superior puede ser finito o ilimitado (una multiplicidad ilimitada se denomina "muchos"). El límite superior tiene que ser mayor que cero; en todo caso, una multiplicidad que sólo fuera cero no sería demasiado útil porque sólo permitiría el conjunto vacío). La multiplicidad se codifica como una cadena.

En la mayoría de los casos se puede especificar una multiplicidad como un intervalo entero —una cardinalidad máxima y mínima—pero en general puede ser un subconjunto discontinuo de los enteros no negativos. El conjunto de enteros puede ser infinito —esto es, el límite superior puede ser ilimitado (pero obsérvese que toda cardinalidad concreta del conjunto es finita).

Para casi todos los propósitos prácticos, este conjunto de enteros se puede especificar como una lista de intervalos enteros disjuntos e inconexos. Un intervalo es un conjunto de

enteros contiguos caracterizado por su valores mínimo y máximo. Algunos conjuntos infinitos no pueden especificarse de este modo —por ejemplo, el conjunto de los enteros pares— pero suele perderse poco por incluir simplemente los saltos. Para la mayoría de los propósitos de diseño, un único intervalo con un valor mínimo y máximo basta para toda la especificación de multiplicidad, porque un propósito esencial de la multiplicidad es limitar la cantidad de espacio de almacenamiento que pudiera ser necesaria.

Véase multiplicidad (de asociación) y multiplicidad (de clase) para obtener detalles específicos de la utilización de la multiplicidad con estos elementos.

Notación

La multiplicidad suele especificarse mediante una expresión de texto formada por una lista de intervalos enteros separados mediante comas, cada uno de los cuales tiene la forma

mínimo..máximo

en donde mínimo y máximo son enteros o bien máximo puede ser un "*" que denota un límite superior no acotado. Una expresión tal como **2..*** se lee en la forma "2 o más".

Los intervalos también pueden tener la forma:

número

en donde número es un entero que representa un intervalo de un único tamaño.

La expresión de multiplicidad que consta de un único asterisco:

*

equivale a la expresión **0..*** —esto es, indica que la cardinalidad es ilimitada ("cero o más, sin límites")—. Esta multiplicidad de frecuente aparición se lee en la forma "muchos".

Ejemplo

0..1

1

0..*

*

1..*

1..6

1..3,7..10,15,19..*

Reglas de estilo

- Preferiblemente, los intervalos deberían crecer de forma monótona. Por ejemplo, **1..3,7,10** es preferible a **7,10,1..3**.
- Dos intervalos contiguos deberían combinarse en un único intervalo. Por ejemplo, **0..1** es preferible a **0,1**.

Discusión

Las expresiones de multiplicidad pueden contener variables pero tiene que resolverse como valores enteros cuando el modelo esté completo, esto es, deben ser parámetros o constantes. La multiplicidad no está destinada a ser evaluada dinámicamente en el ámbito del tiempo de ejecución, como los límites de una matriz dinámica. Tiene como objeto especificar el posible intervalo de valores (en el caso peor) que podría adoptar un conjunto y la aplicación debe adaptar en consecuencia sus estructuras de datos y sus operaciones. Se trata de una constante del tiempo de modelado. Si el límite es variable en tiempo de ejecución, entonces la elección correcta de multiplicidad es *muchos* (0..*).

La multiplicidad puede suprimirse en un diagrama pero sigue existiendo en el modelo subyacente. En un modelo ya terminado, no tiene sentido hablar de una multiplicidad "no especificada". No saber la multiplicidad es lo mismo que decir que es muchos, porque en ausencia de todo conocimiento la cardinalidad podría tomar cualquier valor, que es justamente lo que significa muchos.

Véase valor no especificado.

multiplicidad (de asociación)

Expresa la multiplicidad especificada en un extremo de la asociación.

Véase multiplicidad.

Semántica

La multiplicidad asignada a un extremo de asociación declara el número de objetos que pueden satisfacer el puesto definido por ese extremo de asociación.

Para una asociación binaria, la multiplicidad del extremo destino limita el número de objetos de la clase blanco que se pueden asociar a un objeto simple dado del otro extremo (el origen). Típicamente la multiplicidad se da como un intervalo de enteros (*Véase* multiplicidad para una definición más general.) Entre las multiplicidades se cuentan exactamente uno; cero o uno, cero o más sin límites y uno o más sin límites. La frase "cero o más sin límites" suele llamarse muchos.

En una asociación *n*-aria, la multiplicidad se define con respecto los *n-1* extremos restantes. Por ejemplo, dada una asociación ternaria entre clases (A, B, C), la multiplicidad del extremo C indica el número de objetos C que pueden aparecer en asociación con una pareja particular de objetos A y B. Si la multiplicidad de esta asociación es (muchos, muchos, uno), entonces para cada posible pareja (A, B) existe un único valor de C. Sin embargo, para una pareja (B, C) dada pueden existir muchos valores de A y es posible que haya muchos valores de A, B y C que participen en la asociación.

Véase asociación *n*-aria para una discusión de la multiplicidad *n*-aria.

Notación

La multiplicidad se muestra mediante una cadena de multiplicidad que se sitúa junto al extremo de la ruta a la cual se aplica (Figura 13.138). Un intervalo de números tiene la forma `n1 ..n2`.

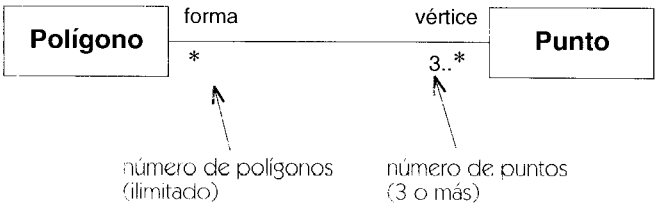


Figura 13.138 Multiplicidad de asociación

Véase multiplicidad para más detalles respecto a la sintaxis y formas más generales de especificarla (aunque posiblemente sean más generales de lo que resulta necesario en la práctica normal).

multiplicidad (de atributo)

Dícese del posible número de valores de un cierto atributo que puede haber en cada objeto.

Semántica

La multiplicidad asociada a un atributo declara el número de valores que se pueden almacenar en un objeto que tenga ese atributo.

La multiplicidad habitual es de exactamente uno (1..1), lo cual significa que todo objeto posee un único valor de ese atributo. Entre otras multiplicidades frecuentes se cuentan cero o uno (un valor opcional o "anulable"); cero o más, sin límites (un conjunto de valores) y uno o más sin límites (un conjunto no vacío de valores). La frase "cero o más sin límites" suele denominarse muchos.

Notación

La multiplicidad se representa mediante una cadena de multiplicidad entre corchetes situada junto al nombre del atributo y antes de los dos puntos (Figura 13.139). Si no hay corchetes, entonces la multiplicidad es exactamente uno (un valor escalar, el valor por defecto).

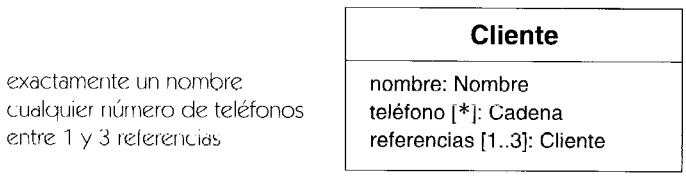


Figura 13.139 Multiplicidad de atributos

multiplicidad (de clase)

Término que expresa el rango de posibles cardinalidades de las instancias de una clase, esto es, el número de instancias que pueden existir legítimamente en un mismo instante.

Semántica

Cuando se le aplica a una clase, la multiplicidad declara el número de instancias de la clase que pueden existir. El valor normal por defecto es ilimitado, pero hay ocasiones en que es útil una multiplicidad finita, especialmente para declarar clases unitarias, esto es, clases que sólo pueden tener una instancia y que normalmente se necesitan para establecer el contexto y los parámetros de todo el sistema.

La otra aplicación de la multiplicidad a las clases es en el interior de una colaboración, en la cual puede estar asociada a un rol de clasificador para especificar el número de instancias que se pueden asociar al rol en una instancia de la colaboración.

Notación

La multiplicidad de una clase o de un rol de clasificador se muestra mediante una cadena de multiplicidad situada en la esquina superior derecha del símbolo de rectángulo (Figura 13.140). Se puede omitir la cadena si la multiplicidad es muchos (ilimitada).

Figura 13.140 Multiplicidad de clase

navegabilidad

La navegabilidad indica si es o no posible recorrer una asociación binaria situada dentro de expresiones de una clase para obtener el objeto o conjunto de objetos asociado a una instancia de la clase. El concepto no es aplicable a las asociaciones *n*-arias (*véase* el texto). La propiedades de navegabilidad es una enumeración cuyos valores son **true** (navegable) y **false** (no navegable).

Véase también eficiencia de navegación.

Semántica

La navegabilidad indica si se puede o no emplear un nombre de rol en las expresiones para recorrer una asociación de un objeto a un objeto o conjunto de objetos de la clase asociada al extremo de la asociación que lleva ese nombre de rol. Si la navegabilidad es cierta, entonces la

asociación define un pseudoatributo de la clase que está en el extremo opuesto al nombre de rol —esto es, que se puede utilizar el nombre de rol en expresiones similares a un atributo de la clase para obtener valores—. El nombre de rol se puede emplear también para expresar restricciones.

La falta de navegabilidad implica que la clase situada al otro extremo del nombre de rol no es capaz de "ver" la asociación y por tanto no puede utilizarla para formar una expresión. Una asociación sin navegabilidad no crea una dependencia de la clase origen respecto a la clase destino, pero quizá se cree una dependencia por alguna otra causa.

La falta de navegabilidad no implica que no haya forma de recorrer la asociación. Si se puede recorrer la asociación en la otra dirección, quizá sea posible investigar todas las instancias de la otra clase para hallar aquellas que llevan a un objeto, invirtiendo de este modo la asociación. Este enfoque podría, incluso, llegar a ser práctico en casos pequeños.

La navegabilidad no está definida en asociaciones *n*-arias porque requeriría especificar conjuntos de clases a los cuales o desde los cuales habría que navegar. Esto podría hacerse pero resulta demasiado complicado para ser útil como propiedad básica. Eso no significa que no se puedan recorrer las asociaciones *n*-arias, sino simplemente que la especificación de su recorrido es complicada y no resulta adecuada para una sencilla definición booleana.

La navegación suele tener la connotación de eficiencia de navegación, aun cuando esto no es un requisito estricto de las reglas de UML.

Notación

Las asociaciones navegables se muestran con una punta de flecha situada en el extremo de la ruta de asociación asociada a la clase destino. La flecha indica la dirección de recorrido (Figura 13.141). El adorno de navegabilidad se puede suprimir (normalmente, en todas las asociaciones del diagrama). Las puntas de flecha pueden no estar asociadas a ningún extremo de las asociaciones, a un extremo o a los dos.

Para mayor comodidad, se pueden omitir las puntas de flecha en las asociaciones que son navegables en ambas direcciones. En teoría, esto se puede confundir con una asociación que no sea navegable en ninguna dirección, pero este tipo de asociación es poco probable en la práctica y se puede indicar explícitamente si ocurriera.

No hay necesidad de una notación para la navegabilidad "indecisa". Si no se ha decidido la navegabilidad, entonces en el caso general será bidireccional. Toda decisión sobre la navegabilidad puede únicamente restringirla o dejar que sea completamente general.

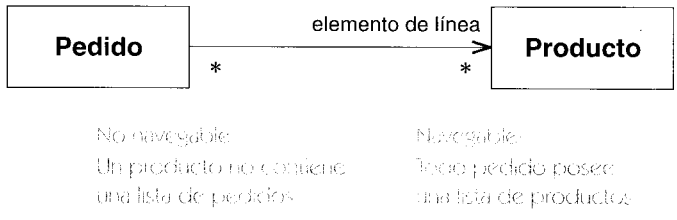


Figura 13.141 Navegabilidad

navegable

Todas aquellas asociaciones o enlaces de una expresión que se pueden recorrer. Su propiedad de navegabilidad es **verdadera**. Estos enlaces suelen implementarse como un puntero o un conjunto de punteros.

Véase navegabilidad, eficiencia de navegación.

navegación

Acción de recorrer las conexiones de un grafo y especialmente de recorrer enlaces binarios y atributos de un modelo de objetos para hacer corresponder a un objeto con un valor. En este último caso, la ruta de navegación se puede expresar como una secuencia de nombres de atributos o de nombres de rol.

Véase navegabilidad.

nodo

El término nodo describe un objeto físico en tiempo de ejecución que representa un recurso computacional; generalmente tienen al menos memoria y con cierta frecuencia poseen capacidades de procesamiento. Los objetos de tiempo de ejecución y las instancias de componentes de tiempo de ejecución también pueden residir en nodos.

Véase también localización.

Semántica

Entre los nodos se cuentan los dispositivos de computación pero también (al menos en los modelos de negocios) los recursos humanos o los recursos de procesamiento mecánico. Los nodos se pueden representar como descriptores y como instancias. Todo nodo define una localización en la cual pueden residir instancias computacionales de tiempo de ejecución, tanto instancias de objetos como instancias de componentes.

Los nodos físicos poseen muchas propiedades adicionales, tales como capacidad, rendimiento y fiabilidad. UML no prescribe estas propiedades, porque existe un elevado número de posibilidades, pero es posible modelarlas en UML empleando estereotipos y valores etiquetados.

Los nodos pueden conectarse a través de asociación para mostrar rutas de comunicación. Es posible dar estereotipos a las asociaciones para distinguir los distintos tipos de rutas de comunicación o sus diferentes implementaciones.

Los nodos forman parte, inherentemente, la vista de implementación y no de la vista de análisis. En los modelos de despliegue aparecen normalmente instancias de nodos, más que tipos de nodos. Aun cuando los tipos de nodos pueden tener un significado, los tipos de los nodos individuales suelen permanecer en el anonimato.

Todo nodo, como clasificador que es, puede poseer atributos. En casi todas las ocasiones se muestran instancias de nodos en los diagramas de despliegue. Los descriptores de nodo tienen una utilidad más reducida.

Notación

Los nodos se representan mediante una figura que parece la proyección descentrada de un cubo.

Los descriptores de nodos tienen la sintaxis:

tipo-nodo

en donde tipo-nodo es en nombre de clasificador.

Las instancias de nodo tienen nombre y nombre de tipo. El nodo puede tener una cadena de nombre subrayada en su interior o por debajo. La cadena de nombre tiene la sintaxis:

nombre : tipo-nodo

El nombre es el nombre del nodo individual (si lo hubiere). El tipo-nodo dice qué clase de nodo es. Estos elementos son opcionales individual o conjuntamente.

Se pueden emplear flechas de dependencia (flechas discontinuas con la punta de flecha en el componente) para mostrar la capacidad de un nodo para admitir un determinado tipo de componente. Se puede emplear un estereotipo para indicar con precisión el tipo de dependencia.

Es posible que existan instancias de componentes y de objetos almacenadas en el interior de símbolos de nodo. Esto indica que los elementos residen en las instancias del nodo. También se puede mostrar la contención por medio de rutas de asociación de agregación y composición.

Es posible conectar unos nodos con otros mediante símbolos de asociación. Una asociación entre dos nodos indica una ruta de comunicación entre ellos. La asociación puede tener un estereotipo para indicar la naturaleza de la ruta de comunicación (por ejemplo, la clase de canal o de red).

Ejemplo

La Figura 13.142 muestra dos nodos que contienen un objeto (cluster) que va de un componente situado en un nodo a otro componente situado en el otro.

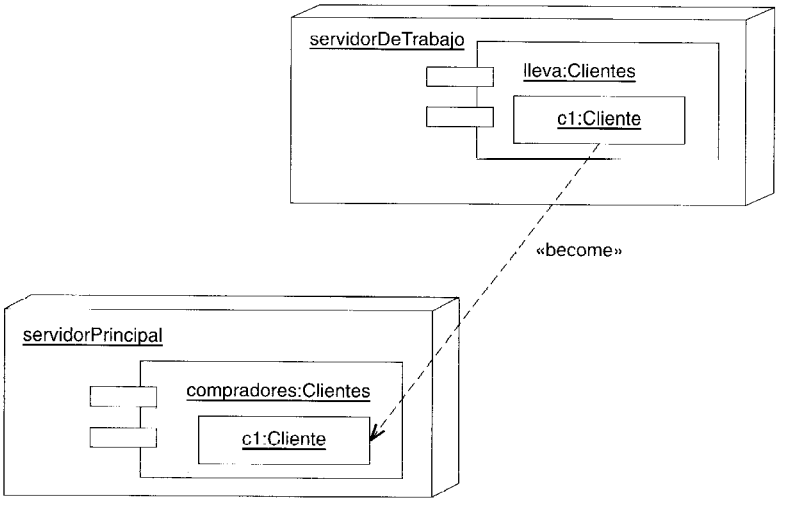


Figura 13.142 Migración entre nodos

no interpretado

Reserva de espacio para uno o más tipos cuya implementación no está especificada por el UML. Todo valor no interpretado posee una representación en forma de cadena. En cualquier implementación física, tal como un editor de modelos, la implementación tendría que ser completa, así que no habría valores no interpretados.

nombre

Término que denota la cadena que se emplea para identificar un elemento de un modelo.

Véase también espacio de nombres.

Semántica

Un nombre es un identificador —una secuencia de caracteres de algún alfabeto finito y predefinido—, en algún lenguaje definido. Una implementación puede imponer restricciones relativas a la forma de los nombres, tales como la exclusión de ciertos caracteres (por ejemplo, los signos de puntuación), restricciones relativas a los caracteres iniciales y demás. En particular, se supone que los nombres se pueden utilizar como selectores y clave de búsqueda en diferentes conjuntos de datos. Por ejemplo, los nombres del alfabeto romano suelen incluir letras mayúsculas y minúsculas; numerales y uno o más separadores tales como el guión inferior y el normal, mientras que otros signos de puntuación dependen de la implementación.

Tanto las herramientas como los lenguajes pueden imponer límites razonables sobre la longitud de las cadenas y el conjunto de caracteres que utilizarán para los nombres; es posible que sean más restrictivas que las correspondientes a las cadenas arbitrarias, tales como los comentarios.

Los nombres se definen en el interior de un espacio de nombres, tal como un paquete o una clase. Dentro del espacio de nombres, cada nombre tiene que ser único dentro de su propio grupo semántico, tal como el de los clasificadores, estados, atributos y demás; los nombres de grupos diferentes pueden coincidir (aunque esto debería evitarse para no inducir a confusión). Todo espacio de nombres, salvo el sistema completo, está contenido dentro de otro espacio de nombres. Los nombres de todos los espacios de nombres anidados y el nombre del elemento final se pueden componer en una sola cadena de nombre de ruta.

Notación

Los nombres se muestran como cadenas. Normalmente los nombres se muestran en una sola línea y contienen únicamente caracteres imprimibles. La notación canónica para los nombres incluye los caracteres, numerales y guiones inferiores. Si se admiten caracteres adicionales en una implementación particular, entonces puede que ciertos caracteres deban ser codificados al visualizarlos para evitar confusiones. Esto es una responsabilidad de implementación de las herramientas.

Los nombres individuales de la jerarquía de un espacio de nombres separados mediante dos signos de dos puntos se pueden componer para formar un nombre de ruta.

nombre de clase

Cada clase debe tener un nombre no nulo único entre los clasificadores de su contenedor (paquete o clase contenedora). El alcance de un nombre es su paquete contenedor y los demás paquetes que pueden ver dicho paquete contenedor.

Véase nombre, donde se realiza una completa discusión sobre los nombres y sus reglas de unicidad.

Notación

El nombre de clase aparece en la parte o compartimento superior del rectángulo que representa a la clase. El compartimento del nombre también puede contener un nombre de palabra clave o de estereotipo y/o un icono de estereotipo y una lista de valores etiquetados entre llaves (Figura 13.143).

Opcionalmente se puede poner también una palabra clave de estereotipo entre comillas sobre el nombre de la clase, y/o un icono de estereotipo en la parte superior derecha del compartimento. El nombre de estereotipo no debe coincidir con una palabra clave predefinida.

El nombre de la clase aparece después, centrado horizontalmente y en negrita. Si la clase es abstracta, su nombre aparece en cursiva. Observe que cualquier especificación explícita acerca de su status de generalización (como *abstract* o *concrete*) tiene preferencia sobre el tipo de letra a utilizar en el nombre.

Debajo del nombre de la clase se puede situar de forma opcional una lista de cadenas que denotan sus propiedades (atributos del metamodelo o valores etiquetados). La lista puede mostrar también atributos de clase para los cuales UML no propone notación. Se pueden utilizar algunas palabras clave sin un valor concreto para denotar una determinada combinación de propiedad y valor. Por ejemplo, una clase hoja muestra la propiedad {**leaf**}, que es equivalente a {**isLeaf=true**}.

Por defecto, una clase mostrada dentro de un paquete se asume que está definida dentro de ese paquete. Para mostrar una referencia a una clase definida en otro paquete, se utiliza la sintaxis

Nombre-Paquete::Nombre-Clase

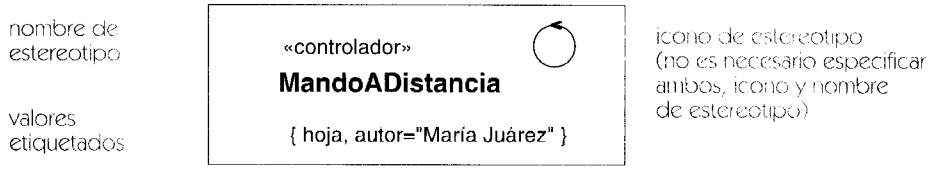


Figura 13.143 Compartimento del nombre

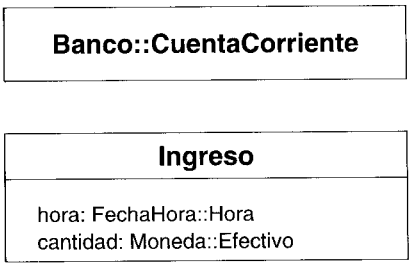


Figura 13.144 Rutas de acceso para clases de otros paquetes

como cadena de nombre en el compartimento del nombre (Figura 13.144). Una ruta de acceso completa se especifica encadenando los nombres de paquete separados por dos signos de dos puntos (::). Varias clases de diferentes paquetes pueden utilizar el mismo nombre de clase, suponiendo que sus rutas de acceso son distintas, pero esta duplicación puede dar lugar a errores, por lo que debe evitarse en lo posible.

También aparecen referencias a clases en expresiones de texto, generalmente en especificaciones de tipo para atributos y variables. En expresiones de texto, una referencia a una clase se indica simplemente utilizando el propio nombre de la clase, incluyendo un posible nombre de paquete, pero sujeto a las reglas de sintaxis de la expresión.

nombre de rol

Nombre de un extremo de asociación concreto dentro de una asociación.

Véase también pseudoatributo.

Semántica

Los nombres de rol proporcionan un nombre para identificar un extremo de asociación dentro de una asociación y también para navegar de un objeto a otro empleando la asociación. Dado que se puede utilizar un nombre de rol en estas dos formas complementarias, el nombre tiene que ser único en ambos espacios de nombres simultáneamente.

Todos los nombres de rol de una asociación tienen que ser diferentes. Dentro de una autoasociación (una asociación que implica más de una vez a la misma clase), se necesitan los nombres de rol para deshacer la ambigüedad existente en los extremos que están conectados a una misma clase. En caso contrario, los nombres de rol son opcionales porque se pueden utilizar los nombres de las clases para deshacer la ambigüedad de los extremos.

También se utilizan los nombres de rol para navegar de un objeto a los objetos vecinos relacionados. Cada clase "ve" las asociaciones que están conectadas a ella y puede utilizarlas para hallar objetos relacionados con alguna de sus instancias. Por convenio, el nombre de rol de un extremo de asociación que está conectado a una clase vecina se emplea para formar una expresión de navegación con objeto de acceder al objeto o conjuntos de objetos que están relacionados por esa asociación. En la Figura 13.145, considérese la clase **B** que

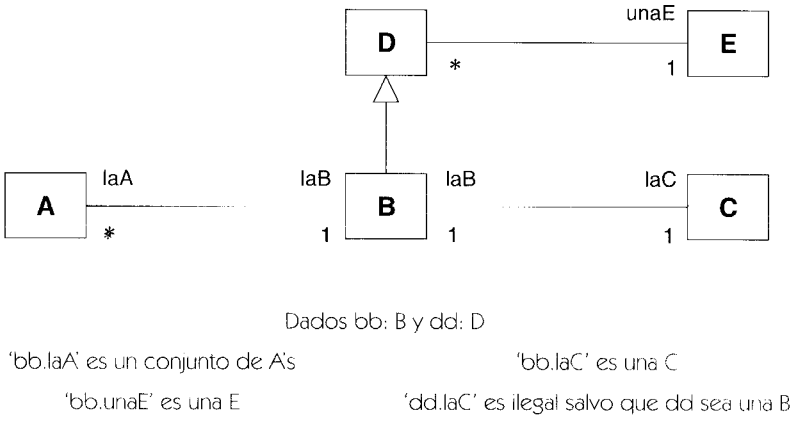


Figura 13.145 Navegación a través de asociaciones

está asociada a la clase **A** mediante una asociación uno-a-muchos y a la clase **C** mediante una asociación uno-a-uno. Dado un **bb** de clase **B**, la expresión **bb.laA** proporciona un conjunto de objetos de clase **A** y la expresión **bb.laC** proporciona un objeto de clase **C**. De hecho, el nombre de rol de lado lejano de la asociación es como un pseudoatributo de una clase, esto es, se puede utilizar como un término en una expresión de acceso para recorrer la asociación.

Dado que un nombre de rol se puede utilizar como un nombre de atributo para extraer valores, el nombre de rol entra en el espacio de nombres de la clase que se encuentra en el lado lejano de la asociación. Va al mismo espacio de nombres que los nombres de los atributos. Tanto los nombres de atributos como los nombres de rol tienen que ser únicos dentro de ese espacio de nombres. Los atributos y los nombres de roles de asociación se heredan; también los nombres de atributos y de pseudoatributos tienen que ser exclusivos entre los nombres heredados. Un nombre de rol asociado a una clase antecesora se puede utilizar para la navegación en un descendiente. En la Figura 13.145, la expresión **bb.unaE** es válida porque la clase **B** hereda el nombre de rol **unaE** de la clase **D**.

Los nombres de rol y los nombres de asociación son opcionales si se puede identificar de forma única a todas las asociaciones. Una asociación puede ser identificada bien por un nombre de asociación o bien por los nombres de rol que aparecen en sus extremos. No es necesario tenerlos todos, aunque está permitido. Si se trata de la única asociación existente entre dos clases, se puede omitir tanto el nombre de la asociación como los nombres de rol. En principio, se necesita un nombre de rol para una expresión de navegación. En la práctica, las herramientas pueden proporcionar una regla por defecto para crear nombres de rol implícitos a partir de los nombres de la clases asociadas.

Notación

Los nombres de rol se muestran mediante una cadena gráfica situada cerca del extremo de una ruta de asociación, en el lugar en que ésta llega a un cuadro de clase. Si existe, el nombre de rol no puede suprimirse.

El nombre de rol puede llevar un marcador de visibilidad —una punta de flecha— que indica si el elemento del extremo lejano de la asociación puede o no ver al elemento asociado al nombre de rol.

nombre de ruta

Cadena compuesta por la concatenación de los nombres de los espacios de nombres anidados que contienen a un elemento, partiendo del espacio de nombres implícito y anónimo que contiene todo el sistema y llegando al nombre del elemento en sí.

Semántica

Los nombres de ruta identifican de modo único a los elementos del modelo, tales como un atributo o un estado, dentro del sistema. Se pueden utilizar en una expresión para hacer referencia a un elemento. No todas las clases de elementos poseen nombre.

Notación

Los nombres de ruta se muestran como una lista de espacios de nombres anidados y de nombres de elementos separados mediante doble signos de dos puntos (::). Un espacio de nombres es un paquete o un elemento que posee declaraciones anidadas.

Contabilidad:: Personal:: Empleado:: dirección

El atributo **dirección** de la clase **Empleado** perteneciente al paquete **Personal** que forma parte de **Contabilidad**.

Los nombres de ruta son referencias de un elemento del paquete denotado por la ruta precedente.

nota

Un símbolo adecuado para mostrar un comentario u otra información textual, tal como el cuerpo de un método o una restricción.

Notación

Una nota es un rectángulo con una esquina doblada, concretamente la esquina superior derecha. Contiene un texto o un texto extendido (tal como un documento incrustado) que no será interpretado por UML. Las notas pueden presentar información de diferentes tipos respecto a los elementos del modelo; puede tratarse de un comentario, una restricción o un método. Las notas no suelen indicar explícitamente el tipo de elemento que representan pero esto suele ser evidente a partir de su forma y del modo en que se utilizan. Dentro de una herramienta de mo-

delado, el elemento subyacente quedará explícito en el modelo. Se puede asociar la nota al elemento que describe empleando una línea discontinua. Si la nota describe múltiples elementos, se lleva una línea discontinua hasta todos y cada uno de esos elementos.

Las notas pueden tener una palabra reservada entre comillas para indicar su significado. La palabra clave «**constraint**» denota una restricción.

Ejemplo

La Figura 13.146 muestra notas que se usan con distintos fines, incluyendo una restricción que afecta a una operación, una restricción que afecta a una clase y un comentario.

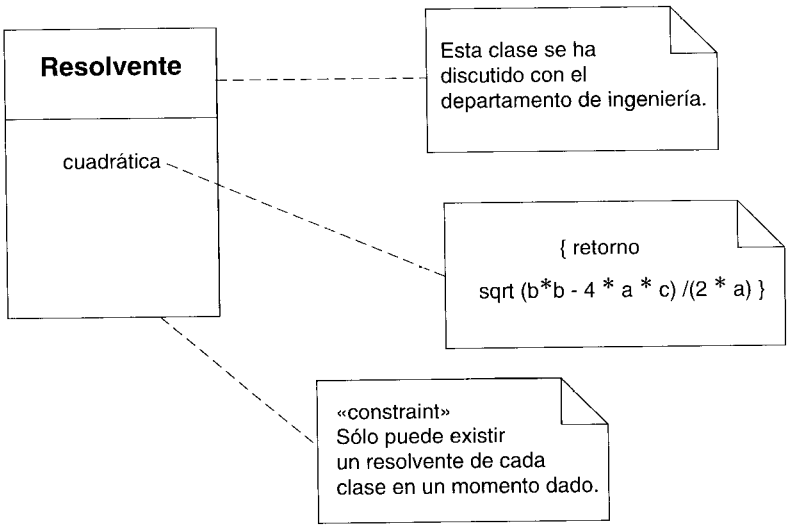


Figura 13.146 Notas

notación canónica

UML define una notación canónica que utiliza dibujos de líneas monocromáticas para visualizar todos los modelos. Éste es el "formato de publicación" estándar de los modelos en UML, aplicable también a los diagramas impresos.

Las herramientas de edición gráfica pueden extender la notación a conveniencia y proporcionar nuevas capacidades interactivas. Por ejemplo, una herramienta puede proporcionar la capacidad de resaltar en pantalla los elementos seleccionados. Otras capacidades de interacción incluyen la posibilidad de navegación dentro del modelo, y el filtrado de elementos visualizados dependiendo de unas propiedades seleccionadas. Este tipo de formato es efímero y no viene impuesto por UML. En una visualización interactiva existe un pequeño riesgo de ambigüedad, por lo que el usuario debe poder pedir explicaciones aclarativas. Por todo ello, UML hace hincapié en la forma canónica impresa, que toda herramienta debe ofrecer, a sabiendas de que una herramienta interactiva puede y debe proporcionar extensiones interactivas.

número de secuencia

Parte textual de la etiqueta de un mensaje en un diagrama de colaboración que indica el orden relativo de ejecución de los mensajes en una interacción. Un número de secuencia puede mostrar la localización del mensaje dentro de una secuencia de llamadas anidadas, el nombre de un hilo de control y una especificación de ejecución condicional e iterativa.

Véase colaboración, mensaje.

objeto

Entidades discretas, con límites bien definidos y con identidad, que encapsulan el estado y el comportamiento; se dice también de las instancias de una clase.

Véase también clase, identidad, instancia.

Semántica

Un objeto es una instancia de una clase, que describe el conjunto de posibles objetos que pueden existir. Un objeto se puede ver desde dos perspectivas relacionadas: como una entidad en un determinado instante de tiempo que posee un valor específico y como un poseedor de identidad que tiene distintos valores a lo largo del tiempo. El primer punto de vista es adecuado para una instantánea, que representa al sistema en un cierto instante de tiempo. Un objeto de una instantánea tiene una localización (en un sistema distribuido) así como valores para todos sus atributos. Los objetos están asociados a un conjunto de enlaces que los conectan con otros objetos.

Cada objeto posee su propia identidad exclusiva y se puede hacer referencia a él mediante una denominación exclusiva que lo identifica y permite acceder a él. La vista de los objetos como identidades es adecuada para una instancias de colaboración, en la cual el objeto posee relaciones de tiempo de ejecución con otros objetos y las usa para intercambiar instancias de mensajes.

Los objetos contienen una ranura de atributo para cada atributo de su descriptor completo, esto es, para todos los atributos declarados en su clase directa y en todas las clases antecesoras. Cuando ha finalizado la instanciación e iniciación de un objeto, cada ranura contiene un valor que es una instancia del clasificador declarado como tipo del atributo. Cuando se ejecuta el sistema, el valor de la ranura de atributo puede cambiar salvo que la propiedad de modificabilidad de atributos se lo impida. Los valores del objeto deben satisfacer todas las restricciones implícitas y explícitas impuestas por el modelo y deben hacerlo en todos los momentos situados entre la ejecución de operaciones. Durante la ejecución de una operación, se pueden violar temporalmente las restricciones.

Si se admite la clasificación múltiple en un entorno de ejecución, entonces un objeto puede ser instancia directa de más de una clase. El objeto contiene una ranura de atributo para cada atributo que se haya declarado en cualquiera de sus clases directas o en cualquiera de sus antecesores. El mismo atributo no podrá aparecer más de una vez, pero si dos clases directas

son descendientes de un antecesor común entonces sólo se hereda una copia de cada atributo del antecesor, independientemente de la multiplicidad de rutas que lleven a él.

Si se admite la clasificación dinámica, el objeto puede cambiar su clase directa durante la ejecución. Si se ganan atributos en el proceso, entonces es preciso especificar sus valores mediante la operación que cambia la clase directa.

Si se permiten tanto la clasificación múltiple como la clasificación dinámica, entonces el objeto puede ganar o perder clases directas durante la ejecución. Sin embargo, el número de clases directas nunca puede ser menor que uno (tiene que tener alguna estructura, aun cuando sea transitoria).

Se puede invocar a un objeto para que ejecute cualquier operación que aparezca en el descriptor completo de cualquier clase directa, esto es, que posee operaciones heredadas tanto directa como indirectamente.

Los objetos se pueden utilizar como valor de cualquier variable o parámetro cuyo tipo declarado sea de la misma clase, o un antecesor de la clase directa del objeto. En otras palabras, una instancia de cualquier descendiente de una clase puede aparecer como valor de una variable cuyo tipo declarado sea precisamente esa clase. Éste es el principio capacidad de sustitución. Este principio no es una necesidad lógica, sino que existe para simplificar la implementación de lenguajes de programación.

Notación

Todo objeto es una instancia de alguna clase. La regla general para la notación de instancias consiste en utilizar el mismo símbolo geométrico que el descriptor, subrayando el nombre de la instancia para distinguirla como individuo. En la instancia se muestran los posibles valores pero las propiedades compartidas por todas las instancias sólo se ponen de manifiesto en el descriptor.

La notación canónica de un objeto es un rectángulo con dos compartimentos. El compartimento superior contiene el nombre del objeto y el de la clase; el compartimento inferior contiene una lista de nombres y valores de atributos (Figura 13.147). No hay necesidad de mostrar operaciones porque son las mismas para todos los objetos de la clase.

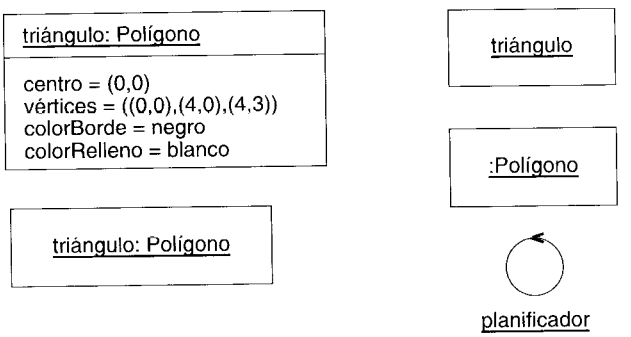


Figura 13.147 Notación de los objetos

El compartimento superior muestra el nombre del objeto y de su clase, ambos subrayados, empleando la sintaxis:

nombre-del-objeto : nombre-de-la-clase

El nombre de la clase puede incluir la ruta de acceso completa del paquete que la contiene, si es necesario. Los nombres de paquete preceden al nombre de la clase y se separan mediante grupos de dos signos de dos puntos. Por ejemplo:

ventanaVisualización:SistemaVentanas:: VentanasGraficas::Ventana

Se puede mostrar textualmente un estereotipo para la clase (entre comillas y por encima del nombre de la clase) o bien como un icono en la esquina superior derecha. El estereotipo del objeto tiene que coincidir con el estereotipo de su clase.

Para mostrar múltiples clases de las que el objeto sea una instancia, haga uso de una lista de nombres de clases separadas mediante comas. Algunas de las clases pueden ser roles transitorios que desempeña el objeto durante una colaboración. Por ejemplo:

unaPersona: Profesor, Esquiador

Para mostrar la presencia de un objeto en un estado particular de la clase, utilice la sintaxis:

nombre-de-objeto : nombre-de-clase [lista-de-nombres-de-estado]

La lista tiene que ser una lista de nombres de estado separados mediante comas y los estados tienen que poder producirse concurrentemente de forma legal.

Para mostrar un cambio de clase (clasificación dinámica) es preciso mostrar dos veces el objeto, una con cada clase. Los dos símbolos se conectan mediante una relación de conversión para mostrar que representan un mismo objeto.

El segundo compartimento muestra los atributos del objeto y sus valores en forma de lista. Cada línea de valor tiene la sintaxis:

nombre-de-atributo : tipo = valor

El tipo es redundante con la declaración del atributo en la clase y se puede omitir. El valor se especifica como una cadena que representa el valor. Los nombres de los atributos no se subrayan.

Se puede omitir el nombre del objeto. En este caso, deben mantenerse los dos puntos con el nombre de la clase. Esto representa un objeto anónimo de esa clase, al cual darán una identidad sus relaciones. Todo símbolo que contiene un objeto anónimo denota un objeto individual que se distingue por sus relaciones con otros objetos.

Se puede suprimir la clase del objeto (junto con los dos puntos) pero debería mostrarse siempre que fuera posible, para no inducir a confusión.

El compartimento de valores de atributos, en bloque, puede suprimirse.

Se pueden suprimir aquellos atributos cuyos valores no sean relevantes.

Para mostrar los cambios de valor de los atributos durante una computación, muestre dos versiones del mismo objeto con una relación de conversión entre ellos.

objeto activo

Objeto que tiene un hilo de control y que puede iniciar una actividad de control; instancia de una clase activa.

Véase también objeto pasivo, proceso, hilo.

Semántica

Un objeto activo no se ejecuta dentro de otro hilo, marco de pila, o máquina de estados. Tiene un lugar de control independiente dentro de la ejecución global de un sistema. En cierto sentido, *es* el hilo. Cada objeto activo es un lugar de ejecución distinto. Los objetos activos no son reentrantes, y además la ejecución recursiva no es posible sin la creación de objetos adicionales.

Un objeto activo es la raíz de la pila de ejecución en términos de computación convencional. La creación de un objeto activo inicia una nueva instancia de una máquina de estados. Cuando la máquina de estados realiza una transición, se crea un marco de pila de ejecución que continúa hasta que la acción de la transición se ejecuta completamente y el objeto espera una entrada externa. Por tanto, un objeto activo no se ejecuta en el ámbito de otro objeto; puede ser creado por una acción de otro objeto, pero una vez creado, tiene una existencia independiente. El creador puede ser un objeto activo o pasivo. Un objeto activo está dirigido por eventos, y las operaciones que otros objetos realicen sobre él deberían aparecer en el objeto activo como eventos de llamada.

Se puede crear un objeto pasivo como parte de una acción de otro objeto. Un objeto pasivo tiene su propio espacio de direcciones, pero no tiene ningún hilo de control. Sus operaciones se pueden llamar desde el bloque de pila de un objeto activo, y puede ser modelado como una máquina de estados para mostrar los cambios de estado provocados por las operaciones realizadas sobre él.

Un proceso de un sistema operativo convencional es muy similar a un objeto activo. Un hilo de un sistema operativo puede implementarse o no utilizando un objeto activo.

La distinción entre activo y pasivo es principalmente una decisión de diseño y no restringe la semántica de los objetos. Tanto los objetos activos como los pasivos pueden tener máquinas de estados e intercambiar eventos.

Notación

Un rol de colaboración para un objeto activo se representa en un diagrama de colaboración mediante un rectángulo con un borde más grueso. Es frecuente representar los roles de objeto activo como compuestos con partes empotradas.

Un objeto activo también se representa mediante un símbolo de objeto con el borde más grueso, y con el nombre subrayado, pero los objetos activos aparecen sólo en ejemplos de ejecución y por tanto no son muy comunes.

Para indicar que un objeto es activo, también se puede utilizar la palabra clave {**active**} en las propiedades.

Ejemplo

La Figura 13.148 muestra tres objetos activos de un sistema de automatización de una fábrica: un robot, un horno y un gestor de la fábrica, que es un objeto de control. Los tres objetos existen y se ejecutan concurrentemente. El gestor de la fábrica inicia un hilo de control en el paso 1, que más tarde se divide en dos hilos de control concurrentes, A1 y B1, ejecutados por el horno y el robot respectivamente. Cuando ambos terminan su ejecución, los hilos se juntan (paso 2) en el gestor de la fábrica. Cada objeto sigue vivo y conserva su estado hasta que le llegue el próximo evento.

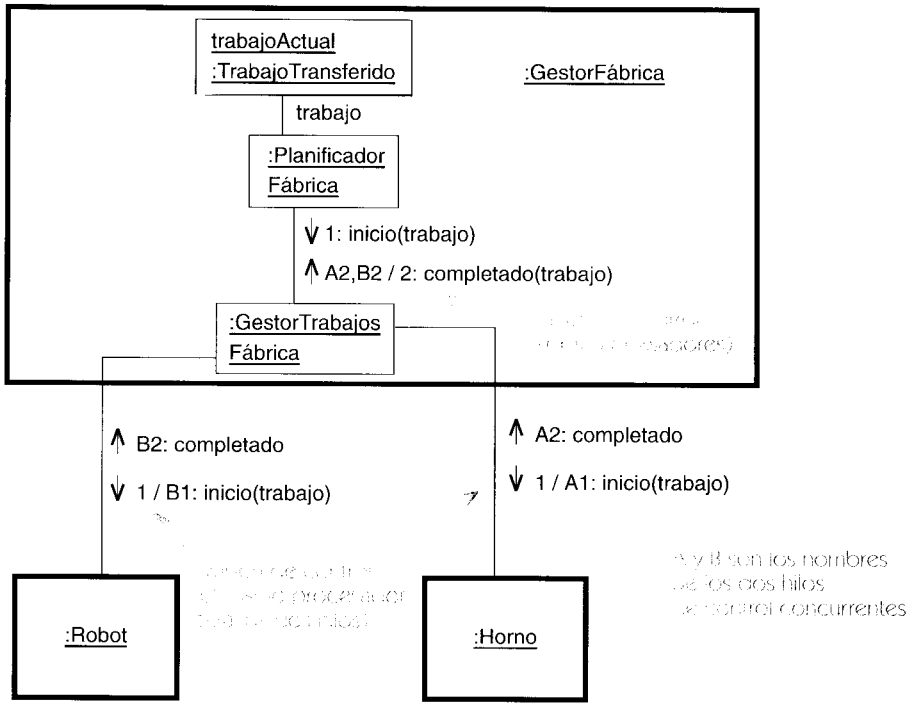


Figura 13.148 Colaboración con objetos activos y control concurrente

objeto compuesto

Un objeto compuesto representa un objeto de alto nivel construido con piezas unidas firmemente. Es una instancia de una clase compuesta, lo que implica la relación de agregación entre la clase y sus partes. Un objeto compuesto es similar a (pero más simple y más restrictivo que) una colaboración. Sin embargo, está definido por composición en un modelo estático, en lugar de por las relaciones dependientes de contexto de una colaboración.

Véase también composición.

Semántica

Un objeto compuesto tiene una relación de composición con todas sus piezas. Esto significa que es responsable de su creación y destrucción, y que no hay otro objeto que sea responsable en la misma medida. Es decir, no hay que preocuparse de la recolección de basura en lo referente a las piezas; el objeto compuesto puede y debe destruirlas cuando muere, o bien debe entregar esta responsabilidad a otro objeto.

La relación de composición se implementa a menudo por contención física dentro de la misma estructura de datos que el propio objeto compuesto (generalmente un registro). La contención física asegura que el tiempo de vida de las piezas se ajusta al tiempo de vida del objeto compuesto.

Notación

Una red de objetos y de enlaces se puede anidar con un compartimento gráfico dentro de un símbolo de objeto. El compartimento gráfico se muestra como un compartimento añadido debajo del compartimento de los atributos (el compartimento de los atributos se puede suprimir). Los objetos y los enlaces contenidos en la región gráfica son partes compuestas del objeto compuesto. Sin embargo, un enlace cuya línea rompe el límite del objeto, no es una parte compuesta de él; es un enlace entre los objetos separados.

Ejemplo

La Figura 13.149 muestra un objeto compuesto, llamada ventana de escritorio, integrada por varias partes. Contiene múltiples instancias de la clase de **Barra de Desplazamiento**. Cada

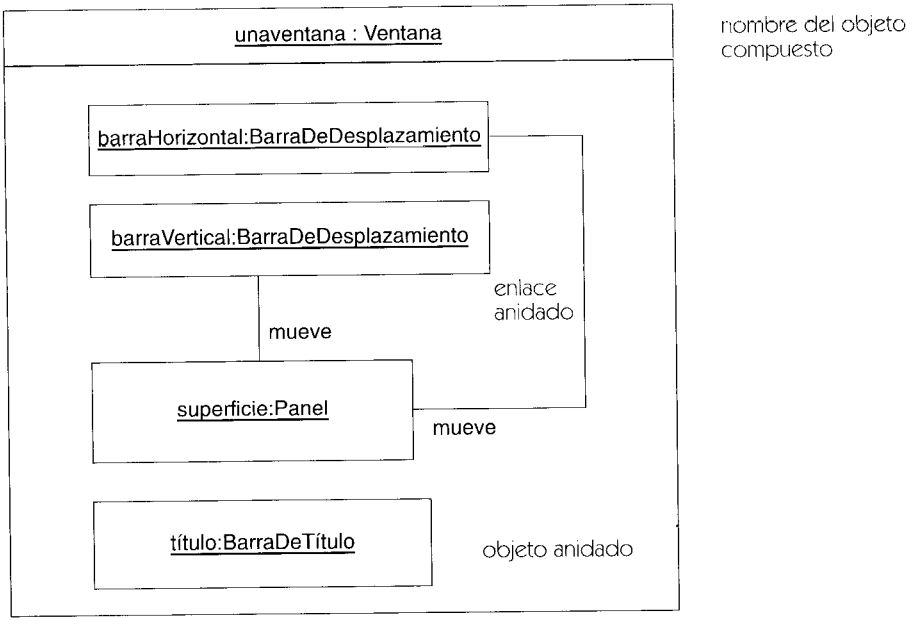


Figura 13.149 Objeto compuesto

instancia tiene su propio nombre y rol dentro del objeto compuesto. Por ejemplo, las **Barras Horizontales** y las **Barras Verticales** son ambas **Barras de Desplazamiento**, pero se comportan de forma diferentemente dentro del objeto compuesto. A este respecto, son como roles de colaboración.

objeto pasivo

Objeto que no posee su propio hilo de control. Sus operaciones se ejecutan en un hilo de control basado en un objeto activo.

Semántica

Diremos que un objeto activo es aquel que posee un hilo de control y que puede iniciar la actividad de control. Un objeto pasivo será aquel que posea valor por no puede iniciar el control. Sin embargo, una operación de un objeto pasivo puede enviar mensajes cuando procesa una solicitud recibida de un hilo ya existente.

Notación

Los objetos pasivos se representan mediante un rectángulo de clase que tiene subrayado en nombre del objeto. Las clases pasivas se representan mediante un rectángulo de clase en que el nombre de la clase no está subrayado. El rectángulo tiene un borde normal (no está acentuado en negrita). Los objetos o clases activos se dibujan con un borde acentuado.

objeto persistente

Objeto que sigue existiendo después de que ha dejado de existir el hilo que lo ha creado.

objeto transitorio

Objeto que existe únicamente durante la ejecución del hilo que lo ha creado.

OCL

Lenguaje de Restricción de Objetos (*Object Constraint Language*), un lenguaje para especificar restricciones y consultas. OCL no está destinado a escribir acciones o código ejecutable. Para una definición completa, *véase* el libro [Warmer-99].

Semántica

El Lenguaje de Restricción de Objetos (OCL) es un lenguaje de texto para escribir expresiones de navegación, expresiones booleanas y otras consultas. Se puede emplear con objeto de construir expresiones para restricciones, condiciones de guarda, acciones, precondiciones y postcondiciones, aserciones y otras clases de expresiones UML. Se hallará una descripción completa de la sintaxis y semántica de OCL en [Warmer-99]. El resumen seleccionado que se muestra a continuación contiene la sintaxis de OCL más útil para crear expresiones de navegación y condiciones booleanas. El lenguaje completo contiene un gran número de operadores predefinidos, aplicables a colecciones y a tipos predefinidos.

Notación

La sintaxis de algunas expresiones comunes de navegación se muestra más abajo. Estas formas se pueden encadenar. El elemento situado más a la izquierda debe ser una expresión de un objeto o conjunto de objetos. Las expresiones están destinadas a funcionar en conjuntos de valores cuando proceda. Para más detalles y para disponer de toda la sintaxis, consúltese la descripción de OCL.

elemento .

es el nombre de un atributo del elemento o el nombre de un rol del extremo blanco de un enlace asociado al elemento. El resultado es el valor del atributo o del objeto u objetos relacionados. El resultado es un valor o un conjunto de valores, dependiendo de las multiplicidades del elemento y de la asociación.

elemento . ( )

es el nombre de una operación que se aplica al elemento. El resultado es el valor proporcionado por la operación aplicada al elemento.

elemento . [ ]

denota una asociación calificada que califica al elemento. es un valor para el atributo calificador. El resultado es el objeto relacionado que seleccione el calificador. Obsérvese que esta sintaxis es aplicable a la indexación de matrices como una forma de calificación.

conjunto ->

es el nombre de una función de conjuntos propia de OCL. El resultado es la propiedad del conjunto. No es válido si no es un función predefinida de OCL. Se enumeran más adelante varias de las propiedades.

conjunto -> select ( )

está escrita en términos de los objetos que pertenecen al conjunto. El resultado es el subconjunto de objetos del conjunto para los cuales es verdadera la expresión booleana.

conjunto -> size	El número de elementos que hay en el conjunto.
self	Denota el objeto actual (se puede omitir si el contexto queda claro).
	Los operadores aritméticos y booleanos habituales = < > <= >= <> + - * / **not**

Ejemplo

```
vuelo.piloto.horas_entrenamiento >= vuelo.avión.mínimo_horas
-- El piloto de vuelo debe tener suficientes horas de entrenamiento

compañía.empleados ->select (cargo = "Jefazo" y self.informes ->size > 10)>
-- El conjunto de jefes con más de diez informes
```

operación

Una operación es la especificación de una transformación o consulta que puede tener que ejecutar un objeto. Posee un nombre y una lista de parámetros. Un método es un procedimiento que implementa una operación. Posee la descripción de un algoritmo o procedimiento. Una operación de una clase activa también se puede implementar utilizando un evento de llamada.

Véase también llamada, evento de llamada, método.

Semántica

La operación especifica una transformación del estado del objeto destino (y posiblemente del estado del resto del sistema que se pueda alcanzar desde el objeto destino) o bien una consulta que proporciona un valor a quien invoque esa operación. Las operaciones se pueden implementar como métodos o como un evento de llamada que da lugar a una transición en la máquina de estados del objeto activo. Las operaciones se invocan mediante una llamada, que deja en suspenso a su emisor hasta que finalice la ejecución de la operación, tras lo cual el emisor recupera el control tras el punto en que se hiciera la llamada, recibiendo un valor de retorno si es que la operación produce uno.

Las operaciones se declaran en una clase. La declaración es heredada por los descendientes de la clase. Si otra declaración tiene la misma "signatura coincidente", entonces se trata de la misma operación. Una implementación puede especificar una regla de signaturas coincidentes para detectar posibles conflictos, pero por defecto se incluye el nombre de la operación y de las clases (pero no los nombres o direcciones) de los parámetros, sin incluir los parámetros que se devuelven al retornar. Esa misma operación podría aparecer en una clase descendiente. En tal caso, será tratada como una repetición de la declaración heredada y se ignora. El propósito es permitir que una operación se puede declarar múltiples veces en clases desarrolladas en diferentes paquetes, empleando la coincidencia de nombres. La declaración de la operación que sea el antepasado común de todas las declaraciones de esa clase es lo que se denomina el origen (según Bertrand Meyer). Representa la declaración vigente de la operación, que es heradada por las demás.

Si dos declaraciones de una operación tienen igual nombre y la misma lista ordenada de tipos de parámetros (sin incluir los parámetros de retorno) pero varían otras propiedades (por ejemplo, un parámetro es de entrada en una operación pero de salida en otra) entonces las declaraciones tienen un conflicto y el modelo está mal formado.

Un método es la implementación de una operación (también puede estar implementado mediante un evento de llamada). Si se declara una operación en una clase sin la propiedad abstracta, entonces tiene una definición de método en la clase. En caso contrario, la operación puede ser abstracta (y no existe un método) o bien puede ser concreta, con un método heredado.

Estructura

Las operaciones poseen los siguientes componentes principales:

concurrencia — Es la semántica de llamadas concurrentes a una misa instancia pasiva, una enumeración. Los valores posibles son:

- **sequential** — Quienes la invoquen deben coordinarse de modo que sólo se pueda ejecutar una llamada a un objeto (para cualquier operación secuencial) al mismo tiempo. Si se producen llamadas concurrentes, entonces no se puede garantizar la semántica y la integridad del sistema.
- **guarded** — Pueden producirse múltiples llamadas simultáneas procedentes de hilos concurrentes a un mismo objeto (para cualquier operación guardada) pero sólo se permite que comience una de cada vez. Las demás quedan bloqueadas hasta que finalice la ejecución de la primera operación. Es responsabilidad del modelador asegurarse de que no se produzcan interbloqueos como consecuencia de bloqueos simultáneos. Las operaciones guardadas deben comportarse correctamente (o deben bloquearse a sí mismas) si se produce una operación secuencial simultánea bien no se puede dar por válida la semántica guardada.
- **concurrent** — Pueden producirse múltiples llamadas procedentes de hilos concurrentes a un único objeto (para operaciones concurrentes). Todas ellas pueden desarrollarse concurrentemente con una semántica correcta. Las operaciones concurrentes deben diseñarse de tal modo que actúen correctamente si se produjera una operación concurrente, secuencial o guardada en ese mismo objeto. En caso contrario, no se puede dar por válida la semántica concurrente.

polimorfismo — Indica si la implementación de la operación (el método o evento de llamada) puede o no ser anulado por las clases descendientes. Si es así, la implementación puede ser anulada por una clase descendiente que proporcione una nueva definición del método o una transición diferente de la máquina de estados. La implementación adopta distintas formas —esto es, es polimórfica—. En caso contrario, la implementación actual será heredada sin cambios por todos los descendientes. Posee una única forma.

consulta — Indica si la ejecución de la operación deja o no intacto el estado del sistema; esto es, si se trata o no de una consulta. En caso afirmativo, la operación proporciona un valor pero no posee efectos secundarios. En caso contrario, puede alterar el estado del sistema, pero no se garantiza que haya algún cambio.

nombre — Es el nombre de la operación, una cadena. El nombre, junto con la lista de tipos de parámetros (sin incluir los nombres o tipos de los parámetros de retorno) es lo que se denomina signatura coincidente de la operación. La signatura coincidente tiene que ser exclusiva dentro de la clase y de sus antecesores. Si existe un duplicado, se toma como una redeclaración de la operación, que tiene que coincidir perfectamente. Si coinciden, entonces se ignoran todas las declaraciones de la operación salvo al de su antecesor más remoto. Si no coinciden, entonces el modelo está mal formado.

lista de parámetros — Es la lista de declaraciones de los parámetros de esa operación. *Véase* lista de parámetros.

tipos de retorno — Se trata de una lista formada por los tipos de los valores proporcionados por una llamada a esa operación. Si la operación no proporciona valores, entonces esta propiedad tiene el valor null. Observe que hay muchos lenguajes que no admiten proporcionar múltiples valores, pero sigue siendo un concepto válido de modelado que se puede implementar de diferentes maneras, tales como tratar a uno o más de los parámetros como valores de salida.

alcance — Denota si la operación es aplicable a objetos individuales o a la clase en sí (alcance de propietario). Los valores posibles son:

- **instance** — La operación se puede aplicar a objetos individuales.
- **class** — La operación se puede aplicar a la clase en sí, por ejemplo una operación que crea una instancia de la clase.

especificación — Denota una expresión que describe los efectos producidos al ejecutar la operación, por ejemplo, una precondición o postcondición. El formato de la especificación no lo prescribe UML y puede adoptar distintas formas.

visibilidad — Es la visibilidad de la operación por parte de clases distintas de aquella que la define. *Véase* visibilidad.

Los métodos poseen los mismos componentes que las operaciones. Además, pueden tener uno o más de los siguientes:

comportamiento — Se trata de una máquina de estados opcional que describe la implementación del método.

cuerpo — Es una expresión que describe el procedimiento del método. Puede estar representada por una cadena o posiblemente tenga un formato analizado sintácticamente. Normalmente se expresa en un lenguaje de programación aunque se puede emplear una expresión en lenguaje

	natural en especificaciones informales. En general este valor no se proporciona si se proporciona la máquina de estados.
colaboración	Denota un conjunto de colaboraciones que describen la implementación del método como un conjunto de mensajes entre roles (una interacción).

Los eventos de llamada tienen los mismos componentes que una operación. La implementación de la operación tiene que ser especificada por una o más de las transiciones que tienen a ese evento de llamada como disparador.

Notación

Las operaciones se representan mediante una cadena de texto que se puede analizar sintácticamente descomponiéndose en propiedades de la operación. La sintaxis por defecto es

«stereotype» visibilidad nombre (lista-parámetros) : tipo-retorno
{ cadena-de-propiedades }

El estereotipo, la visibilidad, la expresión del tipo proporcionado y la cadena de propiedades son opcionales (junto con sus delimitadores). La lista de parámetros puede estar vacía. La Figura 13.150 muestra algunas operaciones típicas.

Nombre. Es una cadena que denota el nombre de la operación (sin incluir los parámetros).

Lista de parámetros. Denota una lista de declaraciones de parámetros separadas mediante comas, cada una de las cuales consta de una dirección, un nombre y un tipo. Toda la lista va encerrada entre paréntesis (incluyendo las listas vacías). *Véase* lista de parámetros y parámetros para disponer de todos los detalles.

Tipo de retorno. Se trata de una cadena que contiene una lista separada por comas de los nombres de los clasificadores (clases, tipos de datos o interfaces). La cadena de tipo va después del signo de dos puntos (:) que sigue a la lista de parámetros de la operación. Los dos puntos y la cadena de tipo proporcionado se omiten si la operación no proporciona valor alguno (por ejemplo, con el caso de **void** en C++). Algunos lenguajes de programación, aunque no todos, admiten múltiples valores de retorno.

Visibilidad. La visibilidad se representa mediante uno de los signos de puntuación '+', '#' y '-' que denota respectivamente **pública**, **protegida** o **privada**. Alternativamente, se puede mostrar la visibilidad como palabra reservada dentro de la cadena de propiedades (por ejemplo, {**visibilidad=privada**}). Es preciso utilizar esta forma para opciones definidas por el usuario o dependientes del lenguaje.

Método. Las operaciones y los métodos se declaran empleando la misma sintaxis. La aparición más externa de la signatura de una operación dentro de una jerarquía de generalización es

```
+visualizar (): Localización
+ocultar ()
«constructor» +crear ()
–asociarVentanaX(venX:ventanaX*)
```

Figura 13.150 Lista de operaciones, con una gama de operaciones

la declaración de la operación. Las signaturas idénticas en clases descendientes son declaraciones redundantes de la operación, pero pueden ser útiles para declarar operaciones cuando las clases se desarrollan por separado. Si la declaración de una operación tiene la propiedad abstracta (lo cual se denota poniendo en cursiva el nombre de la operación o la palabra reservada **abstract**) entonces no existe un método correspondiente a la declaración. En caso contrario, la declaración representa a la vez la declaración de una operación y un método que la implementa.

En las operaciones y métodos coincidentes se utiliza el nombre de la operación y la lista ordenada de tipos de parámetros, sin incluir los parámetros de retorno. Si las propiedades restantes no son consistentes (por ejemplo, si un parámetro de entrada coincide con un parámetro de salida) entonces existe un conflicto y el modelo está mal formado.

Si dos declaraciones de operación idénticas no tienen una declaración de operación en un antecesor común, pero las hereda una clase común, entonces el modelo está mal formado. En esta situación, las declaraciones producirían un conflicto en la clase que herede a ambas.

Cuerpo del método. Se puede mostrar el cuerpo de un método como una cadena situada dentro de una nota asociada a la declaración de una operación. El texto de la especificación debería ir encerrado entre llaves si se trata de una especificación formal en algún lenguaje (una restricción semántica). En caso contrario, debería ser un texto normal si es tan sólo una descripción en lenguaje natural del comportamiento (un comentario). La conexión de una declaración de método a su máquina de estados o a su colaboración carece de representación visual, pero en general se representaría mediante un hipervínculo dentro de una herramienta de edición.

Especificación. Se trata de una expresión que describe los efectos de llevar a cabo la operación. Éstos se pueden enunciar de diferentes maneras, incluyendo el texto, las precondiciones y postcondiciones y los invariantes. En todo caso, la especificación debería expresarse en términos de los efectos observables de la operación sobre el estado del sistema y no en términos del algoritmo de ejecución. El algoritmo es problema del método.

La especificación se muestra mediante una cadena de restricción situada dentro de una nota asociada a la operación de entrada.

Consulta. La opción se muestra mediante una cadena de propiedades de la forma **isQuery=true** o bien **isQuery=false**. La posibilidad true también se puede mostrar mediante la palabra reservada **query**. La ausencia de una opción explícita indica la opción **false**, esto es, que la operación puede alterar el estado del sistema (pero no se garantiza que lo altere).

Polimorfismo. La opción se muestra mediante una cadena de propiedad de la forma **isPolymorphic=true** (anulable) o bien **isPolymorphic=false** (no anulable). La ausencia de una opción explícita indica que la opción es verdadera, esto es, anulable.

Alcance. Las operaciones que tienen alcance de instancia se denotan dejando sin subrayar la cadena de operación. Una operación con alcance de clase se denota subrayando la cadena de nombre.

Concurrencia. La opción se muestra mediante una cadena de propiedad cuya forma es **concurrencia** =valor, en donde el valor puede ser **sequential**, **guarded** o **concurrent**.

Señales. Si se quiere indicar que una clase admite una señal, se antepone la palabra reservada «**signal**» delante de la declaración de la operación, dentro de la lista de operaciones. Los

parámetros son los parámetros de la señal. La declaración puede no poseer un tipo de re
La respuesta del objeto a la recepción de la señal se muestra mediante una máquina de es
Entre otras aplicaciones, esta notación puede mostrar la respuesta de los objetos de una
frente a las excepciones y situaciones de error, que deberían modelarse como señales.

Opciones de representación

La lista de argumentos y el tipo de retorno se pueden suprimir (simultáneamente,
separado).

Una herramienta puede mostrar la indicación de visibilidad de una forma diferente, ta
emplear un icono especial o bien ordenar por grupos los elementos.

La sintaxis de la cadena de signatura de la operación puede ser la de algún lengu
programación concreto, tal como C++ o SmallTalk. Se pueden incluir propiedades etiqu
específicas en la cadena.

Reglas de estilo

- Típicamente, los nombres de las operaciones comienzan por una letra minúscula.
- Los nombres de las operaciones se muestran con letra normal.
- Las operaciones abstractas se muestran en cursiva.

Elementos estándar

semantics.

operación abstracta

Operación que carece de implementación, es decir, que tiene definida la especificación
el método. Es obligatorio que alguna subclase concreta proporcione su implementació

Véase abstracto/a, elemento generalizable, herencia, polimórfico/a.

Semántica

Si una operación se declara como abstracta en una clase, su implementación no apare
dicha clase, y como consecuencia de ello la clase será necesariamente abstracta. Deb
algún descendiente concreto que proporcione una implementación para la operación
clase hereda una implementación de la operación pero declara dicha operación como a
esta declaración invalida el método heredado por la clase. Si una operación se decla
concreta en una clase, la clase debe proporcionar una implementación (un método o e
llamada) o heredarla de una clase antecesora. Si en una clase no se declara una oper
clase hereda la declaración y la implementación de la operación (o su carencia de im
tación) de sus clases antecesoras.

Una operación puede implementarse como un método o como una transición de una máquina de estados disparada por un evento de llamada. Cada clase debe declarar su propio método o evento de llamada para una operación o heredar una definición de una clase antecesora.

Notación

El nombre de una operación abstracta debe aparecer en cursiva (ver Figura 13.151). También se puede utilizar la palabra **abstract** en la lista de propiedades que aparece después de la signatura de la operación.

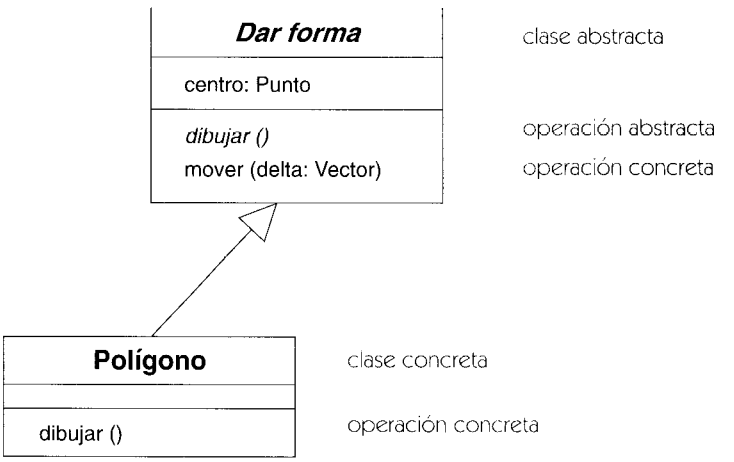


Figura 13.151 Clases y operaciones abstractas

Discusión

La utilización más importante del concepto de herencia es permitir operaciones abstractas que puedan ser implementadas de forma distinta por cada clase descendiente concreta. Una operación abstracta permite invocar una operación sin saber de forma precisa cuál es la clase del objeto destino, siempre y cuando el objeto destino tenga la operación por el hecho de ser una instancia indirecta de una clase abstracta que tiene una declaración de la operación abstracta. El significado de declaraciones polimórficas de este tipo es determinar el tipo de objeto referido por el que invoca el mecanismo de herencia. El que realiza la llamada se libera no sólo de la molestia y el coste de escribir sentencias de código, sino que además no tiene por qué ser consciente de la existencia de posibles subclases de la clase abstracta, lo que significa que se pueden añadir más tarde subclases adicionales con nuevas implementaciones de la operación. De ese modo, las operaciones abstractas, el polimorfismo y la herencia facilitan la actualización de los sistemas mediante la adición de nuevos tipos de objetos y comportamientos sin necesidad de modificar el código que invoca el comportamiento genérico. Esto reduce significativamente el tiempo necesario para actualizar un sistema, y lo que es más importante, reduce la posibilidad de inconsistencias accidentales.

ordenación

Cierta propiedad de un conjunto de valores, tal como el conjunto de valores relacionados con un objeto a través de una asociación, que indica si el conjunto está ordenado o desordenado.

Véase también asociación, extremo de asociación, multiplicidad.

Semántica

Si el límite superior de multiplicidad del extremo de una asociación es mayor que uno, entonces hay un conjunto de objetos asociado a cada uno de los objetos del otro extremo de la asociación binaria. La propiedad de ordenación declara si el conjunto está ordenado o desordenado. Si está desordenado entonces los objetos del conjunto no tienen un orden explícito; forman un conjunto ordinario. Si está ordenado, los elementos del conjunto tienen un orden explícitamente impuesto. El orden de elementos forma parte de la información que representa la asociación, esto es, se trata de una información adicional que va más allá de la información contenida en los elementos en sí. Los elementos se pueden obtener por ese orden. Cuando se añade un nuevo enlace a la asociación, es preciso especificar su posición dentro de la secuencia; esto tiene que hacerlo la operación que lo añade. La posición puede ser un argumento de la operación o bien puede ser implícito. Por ejemplo, una operación dada puede situar un nuevo enlace al final de la lista de enlaces existentes, pero la localización del nuevo enlace debe ser especificada de algún modo.

Observe que un conjunto ordenado no es lo mismo que un conjunto cuyos elementos se hayan ordenado por uno o más de los atributos de los elementos. Un orden queda totalmente determinado por los valores de los elementos del conjunto. Por tanto, no añade información, aunque ciertamente puede resultar útil a efectos de acceder a los elementos. La información de una asociación ordenada, sin embargo, es adicional con respecto a la información de los elementos en sí.

La propiedad de ordenación es aplicable a cualquier elemento que admita una multiplicidad, tal como un atributo cuya multiplicidad sea mayor que uno.

Una relación ordenada se puede implementar de varias maneras, pero la implementación suele enunciarse en forma de una propiedad de generación de código especificada en algún lenguaje. Una extensión de implementación podría reemplazar la estructura de datos que contiene los elementos por la especificación genérica ordenada.

Un conjunto ordenado requiere una especificación por separado de la regla de ordenación en sí, que se dará en forma de restricción.

Notación

La ordenación se especifica mediante una palabra clave entre llaves que se sitúa cerca del extremo de la ruta a que se aplica (Figura 13.152). La ausencia de una palabra reservada denota que no está ordenada. La palabra reservada {**ordered**} indica que se trata de un conjunto ordenado. A efectos de diseño, puede utilizarse la palabra reservada {**sorted**} para indicar un conjunto organizado mediante valores internos.

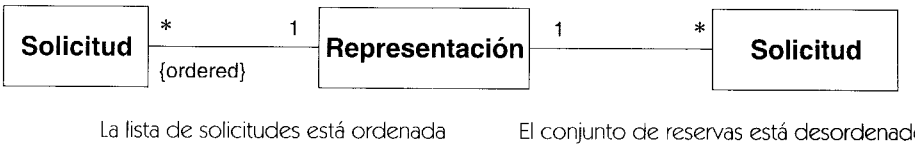


Figura 13.152 Conjuntos ordenados y desordenados

Para aquellos atributos cuya multiplicidad sea mayor que uno, se puede situar una de las palabras reservadas de ordenación después de la cadena de atributos, entre llaves, formando parte de una cadena de propiedades.

Si se omite la palabra clave de ordenación, entonces el conjunto no está ordenado.

Discusión

Un conjunto ordenado posee información en la ordenación, información que es adicional respecto a las entidades del conjunto en sí. Esto es una información real. Por tanto, no es derivable sino que se tiene que especificar cuándo se añade una entidad. En otras palabras, en toda operación que añada una entidad, es preciso especificar su posición dentro de la lista de entidades. Por supuesto, se puede implementar una operación de tal modo que la nueva entidad se inserte en alguna localización implícita, tal como el comienzo o el final de la lista. Y el hecho de que un conjunto esté ordenado no implica que se vaya a admitir cualquier ordenación de las entidades. Estas decisiones tiene que tomarlas quien haga el modelo. En general, la posición de la nueva entidad dentro de la lista es un parámetro de la operación de creación.

Observe que la ordenación de una asociación binaria tiene que ser especificada independientemente para cada dirección. La ordenación carece de sentido a no ser que la multiplicidad en una dirección sea mayor que uno. Una asociación puede ser completamente desordenada, ordenada en una dirección y no en la otra, o bien puede ser ordenada en ambas direcciones.

Consideremos una asociación entre las clases A y B que está ordenada en la dirección de B. Entonces lo normal será que los nuevos enlaces se añadan como una operación sobre un objeto A, especificando un objeto B y una posición en la lista de objetos B existentes para el nuevo enlace. Con cierta frecuencia, una operación sobre un objeto A crea un nuevo objeto B y también crea un enlace entre A y B. Es preciso añadir la lista a la lista de enlaces que mantiene A. Es posible crear un nuevo enlace desde el lado de B pero generalmente el nuevo enlace se insertará en una posición por defecto de la lista que va de A hasta B, porque la posición dentro de esa lista tiene escaso significado desde el lado de B. Por supuesto, un programador puede implementar situaciones más complicadas si fuera necesario.

Resulta poco frecuente hallar una asociación que esté ordenada en ambas direcciones, porque puede resultar engorroso especificar el punto de inserción en las dos direcciones. Pero es posible, especialmente si se añaden los nuevos enlaces en localizaciones por defecto en ambas direcciones.

Observe que un conjunto ordenado no contiene ninguna información adicional más allá de la información relativa al conjunto de entidades. La ordenación ahorra tiempo en un algoritmo pero no agrega información alguna. Se puede considerar como una optimización de diseño y no

es necesario incluirla en un modelo de análisis. Puede ser especificada como valor de la propiedad de ordenación, pero no requiere que una operación especifique una localización para la nueva entidad añadida al conjunto. La localización de la nueva entidad debe ser determinada automáticamente por el método que examina los atributos según los cuales se ordena la lista.

padre

El elemento más general dentro de una relación de generalización. Para una clase recibe el nombre de superclase. Una cadena de una o más relaciones de padre (esto es, el cierre transitivo) es un antecesor. Lo contrario es un hijo.

Véase generalización.

palabra clave

Se entiende que una palabra clave es un adorno textual que categoriza a un elemento del modelo cuando éste carece de su propia sintaxis distintiva.

Véase también marcador gráfico, estereotipo.

Notación

Las palabras reservadas se emplean para aquellos elementos incorporados del modelo que carecen de su propia notación, así como para los estereotipos definibles por el usuario. La notación general para utilizar una palabra reservada consiste en encerrarla entre comillas («») «palabra clave».

Cuando la palabra clave forma parte de un símbolo de área, tal como un rectángulo de clase, entonces la palabra reservada se pone dentro de los límites del símbolo.

En el texto de este documento se describen algunas palabras claves que se tratan como palabras claves dentro de la notación. Hay otros nombres que están a disposición de los usuarios para que los empleen como nombres de estereotipo. No se permite utilizar un nombre de estereotipo que coincida con una palabra clave predefinida.

Discusión

El número de símbolos visuales fácilmente distinguibles es limitado. Por tanto, la notación UML hace uso de palabras claves de texto para distinguir las variantes de un tema común, entre las que se encuentran las subclases de metamodelo de una clase base, los estereotipos de una clase base de metamodelo y los grupos de elementos de listas. Desde el punto de vista del usuario, la distinción de metamodelo entre subclases de metamodelo y estereotipos no suele ser importante, aunque lo sea, por supuesto, para los constructores de herramientas y para otras personas que implementen el metamodelo.

paquete

Término que denota un mecanismo de propósito general para organizar en grupos los elementos. Los paquetes se pueden anidar dentro de otros paquetes. Un sistema puede corresponder a un único paquete de alto nivel, con todo el resto del modelo contenido recursivamente en él. En un paquete pueden aparecer tanto elementos del modelo como diagramas.

Véase también acceso, dependencia, importar, modelo, espacio de nombres, subsistema.

Semántica

La palabra *paquete* alude a un agrupamiento de elementos y diagramas del modelo. Todo elemento del modelo que no forme parte de otro elemento del modelo deberá estar declarado únicamente en un espacio de nombres; el espacio de nombres que contiene la declaración de un elemento se denomina propietario del elemento. Un paquete es un espacio de nombres de propósito general que puede poseer cualquier clase de elemento de modelo que no esté limitado a una sola clase de propietario. Todo diagrama debe ser poseído exactamente por un paquete, que puede estar anidado dentro de otro paquete (y por tanto será poseído por este último). Los paquetes pueden contener paquetes subordinados y elementos ordinarios del modelo. Algunos paquetes pueden ser subsistemas o modelos. La descripción de todo el sistema puede concebirse como un único paquete de subsistema de alto nivel, que contiene absolutamente todo lo demás. Todas las clases de elementos de modelo de UML se pueden organizar en paquetes.

Los paquetes poseen elementos del modelo, subconjuntos del modelo y diagramas. Los paquetes son la base del control de configuraciones, del almacenamiento y del control de accesos. Todo elemento puede ser poseído directamente por otro elemento del modelo o por un único paquete, de tal modo que la jerarquía de posesión es un árbol estricto. Sin embargo, los elementos del modelo (incluyendo los paquetes) pueden hacer referencia a otros elementos de otros paquetes, así que la red de utilización es un grafo.

Las clases especiales de paquete son el modelo, el subsistema y el sistema. Un sistema denota un subsistema que es la raíz de la jerarquía del paquete. Se trata del único elemento del modelo que no es poseído por algún otro elemento del modelo. Indirectamente, incluye todo lo que hay en el modelo. Existen varios estereotipos predefinidos de modelo y de subsistema.

Véase el Capítulo 14, Elementos Estándar, para más detalles.

Los paquetes pueden tener relaciones de dependencia con otros paquetes. En la mayoría de los casos, estas dependencias no son otra cosa que un resumen de las dependencias que existen entre los contenidos de los paquetes. Un uso de dependencia entre dos paquetes significa que existe al menos una dependencia de uso entre elementos de los dos paquetes (esto no quiere decir que toda pareja de elementos tenga esa dependencia).

La dependencia de acceso es propia de los paquetes en sí y no es un compendio de dependencias de sus elementos. Indica que a los elementos del paquete cliente se les otorga permiso para tener relaciones con los elementos del paquete proveedor. Las relaciones también están sometidas a especificaciones de visibilidad. El acceso no significa que los nombres de los elementos que se encuentran en el paquete blanco ocupen el espacio de nombres del paquete origen: los espacios de nombres son diferentes y los elementos pueden identificarse de modo

único mediante nombres de ruta que incluyen a los paquetes anidados. La importancia de la variación de dependencia de acceso es como la sentencia **uses** de Ada. Añade los nombres del espacio de nombres del proveedor al espacio de nombres del cliente (no deben existir conflictos). Pero la dependencia de acceso no altera el espacio de nombres del cliente. Se trata, principalmente, de un mecanismo de control de acceso en los proyectos de desarrollo más extensos y no una relación semántica fundamental.

Un paquete anidado tiene acceso a todos los elementos que estén contenidos directamente en paquetes externos (hasta cualquier nivel de anidamiento) sin necesidad de dependencias de importación ni de visibilidades. Sin embargo, el paquete tiene que importar los paquetes que contenga para poder ver su interior. En general, un paquete contenido es un límite de encapsulamiento.

Los paquetes definen la visibilidad de los elementos que contienen como **privada**, **protegida** o **pública**. Los elementos privados no están disponibles de ningún modo fuera del paquete que los contiene (independientemente de las importaciones). Los elementos protegidos sólo están disponibles para aquellos paquetes que tengan generalizaciones al paquete que los contiene y los elementos públicos están disponibles para los paquetes que importan y para los descendientes del paquete.

Véase acceso para una descripción completa de las reglas de visibilidad que se aplican a los elementos de los distintos paquetes.

Notación

Los paquetes se representan mediante un rectángulo grande con un rectángulo pequeño (una "ficha") asociado a una esquina (normalmente, el lado izquierdo de la parte superior del rectángulo grande). Se trata del icono de una carpeta.

Si no se muestra el contenido del paquete, entonces el nombre del paquete se sitúa dentro del rectángulo grande. Si se muestra el contenido del paquete, entonces el nombre del paquete se puede situar dentro de la ficha.

Se puede situar la cadena de una palabra reservada por encima del nombre del paquete. Entre las palabras reservadas se cuentan subsistema, sistema y modelo. Los estereotipos definidos por el usuario se anotan también con palabras reservadas pero no deben entrar en conflicto con las palabras reservadas predefinidas.

Se puede poner entre llaves una lista de propiedades detrás o por debajo del nombre del paquete. Ejemplo: {**abstract**}.

El contenido del paquete se puede mostrar dentro del rectángulo grande.

La visibilidad de un elemento de paquete fuera del paquete se puede indicar anteponiendo al nombre del elemento un símbolo de visibilidad ('+' denota pública, '-' significa privada y '#' quiere decir protegida).

Se pueden dibujar las relaciones entre símbolos de paquete para mostrar relaciones que existan al menos entre algunos de los elementos de los paquetes. En particular, la dependencia de otros paquetes (salvo las dependencias de permiso, tales como el acceso y la importación) implican que existe una o más dependencias entre los elementos).

Opciones de representación

Las herramientas también pueden mostrar la visibilidad visualizando selectivamente aquellos elementos que satisfacen un determinado nivel de visibilidad; por ejemplo, mostrando únicamente los elementos públicos.

Las herramientas pueden mostrar la visibilidad empleando algún marcador gráfico, tal como un color o un tipo de letra.

Reglas de estilo

Se supone que los paquetes que posean contenidos extensos se mostrarán como sencillo iconos con nombre, en los cuales se puede acceder dinámicamente al contenido "haciendo zoom" para llegar a una visualización detallada.

Ejemplo

La Figura 13.153 muestra la estructura de paquete de un subsistema para el procesamiento de pedidos. El subsistema en sí se representa mediante un paquete con un estereotipo. Contiene varios paquetes ordinarios. Las dependencias entre paquetes se muestran mediante flechas discontinuas. La figura muestra también algunos paquetes externos de los cuales depende el subsistema. Éstos pueden ser componentes estandarizados o bien elementos de una biblioteca.

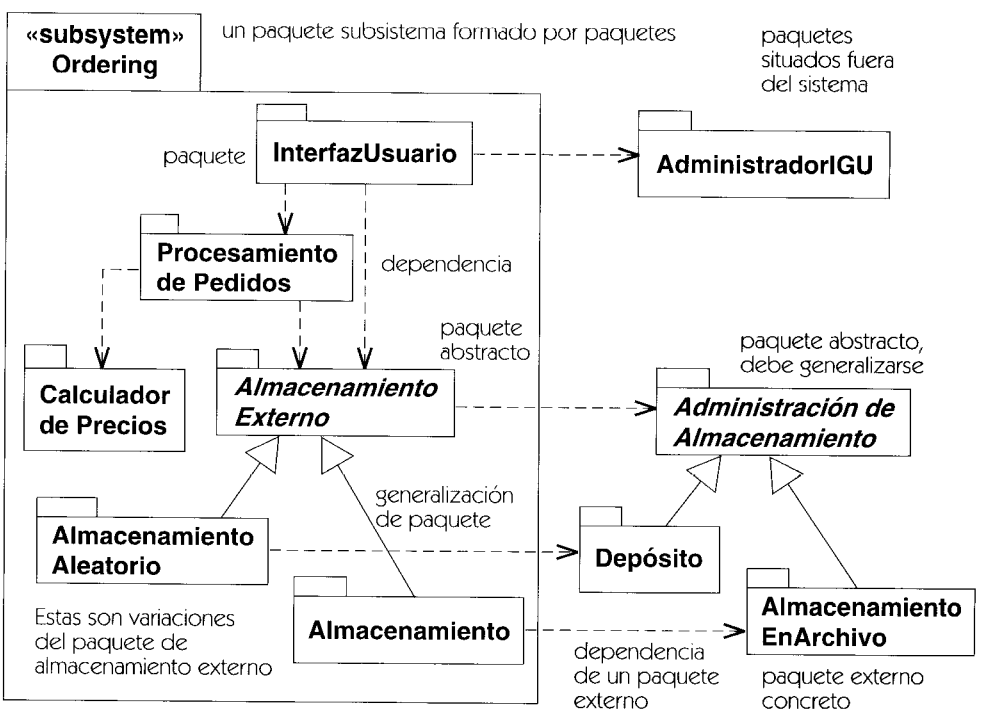


Figura 13.153 Los paquetes y sus relaciones

La generalización entre paquetes muestra las variaciones de un paquete genérico. Por ejemplo, el paquete **AlmacenamientoExterno** se puede implementar como un **Almacenamiento-Aleatorio** o un **Almacenamiento**.

Discusión

Los paquetes están destinados primordialmente a ser mecanismos de control de acceso y configuración que permitan a los desarrolladores, especialmente los que trabajan en grandes grupos, organizar grandes modelos y hacerlos evolucionar sin estorbarse unos a otros. Significan, inherentemente, lo que los desarrolladores quieran que signifiquen. Mas prácticamente, los paquetes deberían seguir algún tipo de límite semántico para que puedan ser útiles. Como están destinados a ser unidades de control de configuraciones, deberían contener elementos que tengan probabilidades de evolucionar al unísono. Los paquetes también sirven para agrupar aquellos elementos que deben ser compilados a la vez. Si un cambio de uno de los elementos impone la recompilación de otros elementos, entonces quizá convenga poner a ambos en un mismo paquete.

Todo elemento del modelo debe ser poseído exactamente por un paquete u otro elemento del modelo. En caso contrario, el mantenimiento del modelo, las versiones y el control de configuración se vuelven imposibles. El paquete que posee un elemento del modelo controla su definición. Se puede hacer referencia a él y es posible utilizarlo en otros paquetes, pero hacer un cambio en él requiere tener un permiso de acceso y el derecho de actualización para el paquete que lo posee.

Elementos estándar

access, extend, facade, framework, stub, system.

parámetro

Término que denota la especificación de una variable que se puede modificar, pasar o proporcionar. Los parámetros pueden tener nombre, tipo y dirección. Se utilizan en operaciones, mensajes, eventos y plantillas. Constrastar con: argumento.

Las dependencias de uso de parámetro relacionan a una operación que posee un parámetro o a una clase que posea esa operación con la clase del parámetro.

Véase también argumento, enlace.

Semántica

Un parámetro es una reserva de espacio para un argumento, que se enlaza con él cuando se hace uso del elemento que lo contiene. Los parámetros restringen los valores que puede admitir el argumento. Tienen las partes siguientes:

valor por defecto — Es una expresión que proporciona el valor que se utilizará si no se proporciona un argumento para el parámetro. La expresión se evalúa cuando la lista de parámetros se enlaza con los argumentos.

dirección — Es la dirección del flujo de información del parámetro; se trata de una enumeración con los valores siguientes:

- **in** — Un parámetro de entrada pasado por valor. Los cambios efectuados en el parámetro no están disponibles para el emisor.
- **out** — Un parámetro de salida. No existe valor de entrada. El valor final está disponible para el emisor.
- **inout** — Un parámetro de entrada que se puede modificar. El resultado final está disponible para el emisor.
- **return** — Valor proporcionado por una llamada. El valor está disponible para el emisor. Semánticamente, no tiene diferencias con respecto a un parámetro out, pero el resultado está disponible para su utilización en una expresión inline.

Las opciones precedentes pueden no estar disponibles directamente en todos los lenguajes de programación, pero el concepto que subyace a todas ellas tiene sentido en la mayoría de los lenguajes y se puede hacer corresponder con una implementación razonable.

nombre — Es el nombre del parámetro. Tiene que ser único dentro de su lista de parámetros.

tipo — Se trata de una referencia de un clasificador (una clase, tipo de datos o interfaz en la mayoría de los procedimientos). El argumento que esté enlazado con el parámetro tiene que ser una instancia del clasificador o de uno de sus descendientes.

Notación

Todo parámetro se representa mediante una cadena de texto que se puede analizar sintácticamente para descomponerla en las distintas propiedades del parámetro. La sintaxis por defecto es:

```
dirección nombre : tipo valor-por-defecto
```

Dirección. La dirección se representa mediante una palabra clave que precede al nombre de la operación. Si la palabra clave está ausente, entonces la dirección es in. Las opciones son **in**, **out**, **inout** y **return**. Los parámetros return suelen mostrarse en otra posición distinta en la signatura de las operaciones, lugar en que no es necesario marcar su dirección.

Nombre. El nombre se representa mediante una cadena.

Tipo. El tipo se denota como una cadena que es el nombre de una clase, una interfaz o un tipo de datos.

Valor por defecto. El valor se representa mediante una cadena de expresión. El lenguaje de la expresión debería ser conocido por la herramienta (y debe ser especificable para la herramienta) pero no se muestra en formato canónico.

Alcance. Si el alcance es de clase, entonces se subraya la cadena de operación. Si el alcance es de instancia, entonces no se subraya la cadena de la operación.

Dependencia paramétrica. Una dependencia paramétrica se representa mediante una flecha discontinua que va desde la operación que tiene el parámetro o la clase que contiene a esa operación hasta la clase del parámetro; la flecha tiene asociado el estereotipo «**parameter**».

Ejemplo

Matriz::transformación (**in** distancia: Vector, **in** ángulo: Real = 0): **return** Matriz

Se pueden omitir todas las etiquetas de dirección.

parámetro real

Véase argumento.

participantes

Denota la conexión de un elemento de un modelo con una relación o con una relación materializada. Por ejemplo, una clase participa en una asociación, un rol de clasificador participa en una colaboración.

patrón

Término que denota una colaboración parametrizada que representa un conjunto de clasificadores, relaciones y comportamientos parametrizados, que se pueden aplicar a múltiples situaciones enlazando elementos del modelo (clases normalmente) con los roles del patrón. Es una plantilla de colaboración.

Semántica

Los patrones representan una colaboración parametrizada que se puede emplear en múltiples ocasiones dentro de uno o más sistemas. Para ser un patrón, la colaboración tiene que ser utilizable en una amplia gama de situaciones, para justificar darle un nombre. Un patrón es una solución que demostradamente funciona en un cierto número de situaciones. No es necesariamente la única solución del problema, pero es una solución que ha sido efectiva en el pasado. Casi todos los patrones tienen ventajas y desventajas, que dependen de distintos aspectos del sistema global. El creador de modelos debe considerar estas ventajas y desventajas antes de tomar la decisión de utilizar un patrón.

Una colaboración parametrizada de UML representa las vistas estructural y de comportamiento de ciertas clases de patrones. Los patrones implican otros aspectos que no son modelados directamente por UML, tales como la lista de ventajas y desventajas de su utilización anterior. Hay muchos de estos aspectos que se pueden expresar en palabras. *Véase* [Gamma-95] para un tratamiento más completo de los patrones, así como un catálogo de patrones de diseño probados.

Generar colaboraciones a partir de patrones. Se puede utilizar una colaboración para especificar la implementación de estructuras de diseño. Es posible emplear la misma clase de colaboración en muchas ocasiones, parametrizando sus componentes. Un patrón es una colaboración parametrizada. En general, las clases de los roles que intervienen en la colaboración son parámetros. El patrón se instancia como una colaboración enlazando valores, normalmente clases, con sus parámetros. En el caso común de los roles parametrizados, la plantilla se enlaza especificando una clase para cada rol. Típicamente, los roles de asociación de un patrón no están parametrizados. Cuando se enlaza la plantilla, representan asociaciones implícitas entre las clases enlazadas con la colaboración, esto es, el enlazado de la plantilla para hacer una colaboración genera asociaciones adicionales.

Notación

El enlazado de un patrón para producir una colaboración se representa mediante una elipse discontinua que contiene el nombre del patrón (Figura 13.154). Se traza una línea discontinua desde el símbolo de enlazado del patrón hasta las clases (u otros elementos del modelo) que participen en la colaboración. Todas las líneas se etiquetan con el nombre del parámetro. En la mayoría de los casos el nombre de un rol de la colaboración se puede emplear como nombre de parámetro. Por tanto, un símbolo de enlazado de patrón puede mostrar la utilización de un patrón de diseño, junto con las clases que realmente aparecen en esa utilización del patrón. El

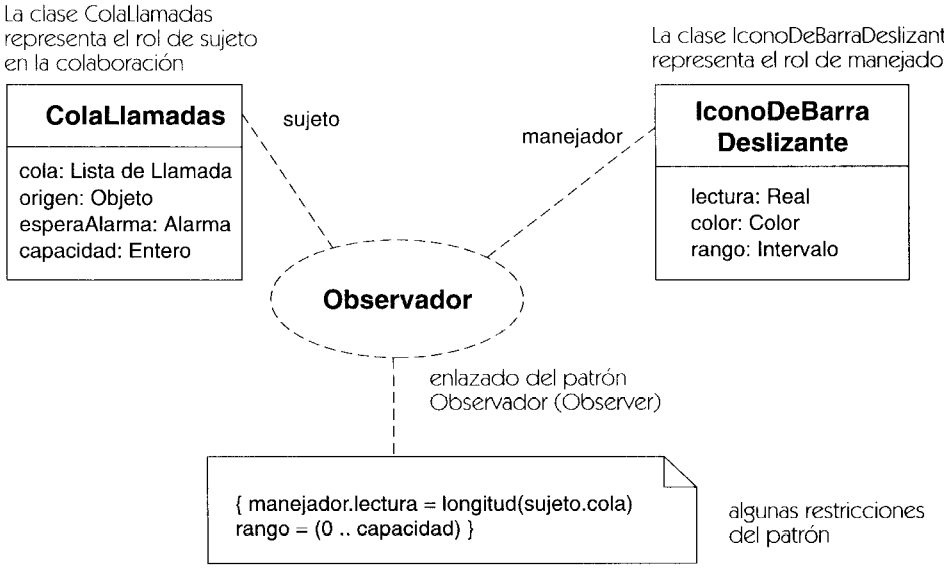


Figura 13.154 Enlazado de un patrón para formar una colaboración

enlazado del patrón no suele mostrar la estructura interna de la colaboración que se genera mediante el enlazado. Esto está implícito en el símbolo de enlazado.

permiso

Denota una clase de dependencia que concede al cliente permiso para utilizar el contenido del elemento proveedor (sometido a las declaraciones de visibilidad de los elementos del mismo).

Semántica

Los estereotipos de dependencia de permiso son access, friend e import. Nunca se utiliza un principio de dependencia al desnudo sin un estereotipo. Las dependencias de acceso y de importación se emplean en los paquetes. La dependencia de amistad se emplea con clases u operaciones como clientes y con clases como proveedores.

Notación

Las dependencias de permiso se representan mediante una flecha discontinua que va del cliente (el elemento al que se da permiso) al proveedor (el elemento que concede el permiso) con la correspondiente palabra reservada de estereotipo asociada a la flecha.

Elementos estándar

friend, import.

plantilla

Elemento parametrizado de un modelo. Para utilizarla, los parámetros tienen que estar asociados (en el tiempo de modelado) con valores reales. Sinónimo: elemento parametrizado.

Véase también enlace, elemento ligado.

Semántica

Una plantilla es el descriptor de un elemento con uno o más parámetros formales. Por tanto, define una familia de elementos potenciales, estando especificado cada uno de los elementos por el enlazado de los parámetros con valores reales. Típicamente, los parámetros son clasificadores que representan tipos de atributos, pero también pueden representar enteros o incluso operaciones. Los elementos subordinados dentro de la plantilla se definen en términos de los parámetros formales, así que también quedan enlazados cuando se enlaza la plantilla en sí como los valores reales.

Una plantilla de clase es el descriptor de una clase parametrizada. El cuerpo de una plantilla puede contener muchas apariciones de los parámetros formales, así como un elemento por defecto que representa la plantilla en sí. Las clases reales se producen enlazando los parámetros con valores. Los atributos y operaciones existentes dentro de la clase plantilla se pueden definir en términos de los parámetros formales. La clase plantilla también puede tener relaciones, tales como asociaciones y generalización, entre ella y uno de sus parámetros. Cuando se enlaza la plantilla, el resultado es una relación entre la clase de plantilla enlazada y las clases enlazadas con los parámetros de la relación.

Las clases de plantilla no son clases utilizables directamente porque poseen parámetros que no están enlazados. Es preciso enlazar los parámetros con valores reales para crear una clase real. Sólo una clase real podrá ser padre o destino de una asociación (una asociación monodireccional procedente de la plantilla y que llega a otra clase es admisible, sin embargo). Una clase de plantilla puede ser una subclase de una clase ordinaria, lo cual implica que todas las clases formadas al enlazar la plantilla son subclases de la clase dada. También puede ser hija de uno de los parámetros de plantilla; esto implica que la clase de plantilla enlazada es hija de la clase pasada como argumento.

Es posible aplicar la parametrización a otros elementos del modelo, tales como las colaboraciones e incluso paquetes completos. La descripción que se da aquí para las clases se aplica en la forma evidente a los otros elementos de modelado.

El contenido de una plantilla no está sometido directamente a las reglas de buena formación que afectan a los modelos. Esto se debe a que incluyen parámetros que no tienen una semántica completa mientras no hayan sido enlazados. Una plantilla es una especie de elemento de modelo de segundo nivel, no un elemento que modele directamente los sistemas sino uno que modela otros elementos del modelo. Por tanto, el contenido de la plantilla está fuera de la semántica del sistema. Los resultados de enlazar una plantilla son elementos ordinarios del modelo, sometidos a las reglas de buena formación y son elementos normales del sistema blanco. Se podrían derivar ciertas reglas de buena formación para plantillas a partir de las consideraciones de que los resultados enlazados tienen que ser bien formados, pero no vamos a intentar enumerarlas. En cierto sentido, cuando se enlaza una plantilla su contenido se duplica y los parámetros son sustituidos por los argumentos. El resultado pasa a formar parte del modelo efectivo como si se hubiese incluido directamente.

Hay otras clases de clasificadores, como los casos de uso y las señales, que también se pueden parametrizar. Las colaboraciones también se pueden parametrizar; entonces son patrones.

Notación

Se superpone un pequeño rectángulo discontinuo en la esquina superior derecha del rectángulo de clase u otro elemento de modelado. El rectángulo discontinuo contiene una lista de parámetros formales para la clase. Cada parámetro tiene su nombre y un clasificador. La lista no puede estar vacía (si lo estuviera, no sería una plantilla) aunque se puede suprimir en la representación. El nombre, los atributos y las operaciones de la clase parametrizada aparecen normalmente en el rectángulo de clase, pero también se pueden incluir apariciones de los parámetros formales. Las otras clases de elementos parametrizados se tratan análogamente. Las apariciones de los parámetros formales pueden producirse dentro del cuerpo de la

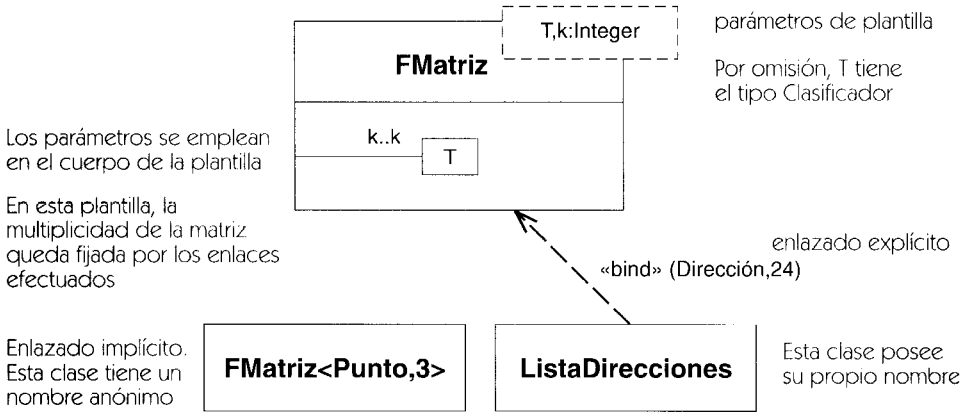


Figura 13.155 Notación de plantilla empleando un parámetro como referencia

plantilla, por ejemplo, para mostrar una clase relacionada que es identificada por uno de los parámetros.

Los parámetros tienen la sintaxis:

nombre :tipo

donde nombre es un identificador del parámetro, con alcance de plantilla; y

donde tipo es una cadena que denota una expresión de tipo para el parámetro.

Si se omite el nombre del tipo, se supone que se trata de una expresión de tipo que al resolverse producirá un clasificador, tal como un nombre de clase o un tipo de datos. Los demás tipos de parámetros (tales como **Entero**) deben mostrarse explícitamente y tienen que producir expresiones de tipo válidas cuando se resuelven.

La Figura 13.155 muestra una plantilla con un parámetro entero y un parámetro de clase. La plantilla tienen una asociación con uno de sus parámetros.

Discusión

El *modelo efectivo* es el modelo implícito que resulta de enlazar todas las plantillas; se trata del modelo implícito que describe el sistema. Los parámetros de plantilla no tienen significado dentro del modelo efectivo en sí, porque ya se habrán enlazado. Sólo se pueden utilizar dentro del alcance del cuerpo de la plantilla en sí. Esto basta para manejar los elementos constitutivos contenidos en el elemento parametrizado, por ejemplo, para atributos o operaciones contenidos en una clase parametrizada.

Los elementos que normalmente son externos respecto al elemento parametrizado plantean más dificultades. Por ejemplo, una clase puede tener asociación o generalizaciones que lleguen a otras clases. Si esas clases son parámetros de la plantilla, no puede formar parte del modelo efectivo, y sin embargo tampoco forman parte de una clase ordinaria. Por tanto, un elemento parametrizado contiene un *cuerpo* que representa un fragmento del modelo. El fragmento del modelo no forma parte de la plantilla en sí y puede incluir parámetros de plantilla tales como un

parámetro que represente una clase (que todavía no está especificada). Cuando se enlaza la plantilla, el cuerpo se copia implícitamente; los parámetros son reemplazados por los argumentos y la copia pasa a formar parte del modelo efectivo. Cada instanciación de la plantilla produce una adición al modelo efectivo. La Figura 13.156 muestra un ejemplo.

El cuerpo de la plantilla contiene implícitamente un elemento que representa el elemento de plantilla instanciado en sí; por ejemplo, la clase producida al enlazar una plantilla. Este elemento implícito se puede emplear para relaciones de construcción tales como asociaciones y generalizaciones con parámetros de plantilla. En la notación, los parámetros se dibujan dentro de los límites de la plantilla y hay una conexión al interior de los límites de la plantilla que denota una relación con el elemento instanciado implícito de la plantilla. Cuando se instancia la plantilla, éstas se transforman en relaciones del modelo efectivo entre el elemento de plantilla enlazado (recién instanciado) y los elementos (previamente existentes) que son argumentos de la plantilla.

Una plantilla puede ser hija de otro elemento. Esto significa que todo elemento enlazado generado a partir de ella es hijo del elemento dado. Por ejemplo, en la Figura 13.156, toda variable del tipo matricial (**VMatriz**) es una matriz (**Matriz**). Por tanto, **VMatriz<Punto>** es hijo de **Matriz**, **VMatriz<Dirección>** es hijo de **Matriz** y así sucesivamente.

Normalmente una plantilla no puede ser padre de otro elemento. Esto significaría que todo elemento generado al enlazar el padre sería padre del otro elemento. Aunque quizá alguien pudiera atribuir un significado a semejante situación, es poco probable.

Dos plantillas no pueden tener asociaciones entre sí, simplemente porque comparten el mismo nombre de parámetro. (Intentar hacer esto significaría que toda instanciación de la primera plantilla estaría relacionada con toda instanciación de la segunda plantilla, que no es lo que suele desearse normalmente. Esta cuestión ha sido malinterpretada por algunos autores en el pasado.) Los parámetros sólo tienen alcance dentro de su propia plantilla. Utilizar el mismo nombre de parámetro en dos plantillas no hace que sea el mismo parámetro. En general, si dos plantillas tienen elementos parametrizados que deben estar relacionados, entonces una de las plantillas debe instanciarse dentro del cuerpo de la otra. (Recuerde que una plantilla es instan-

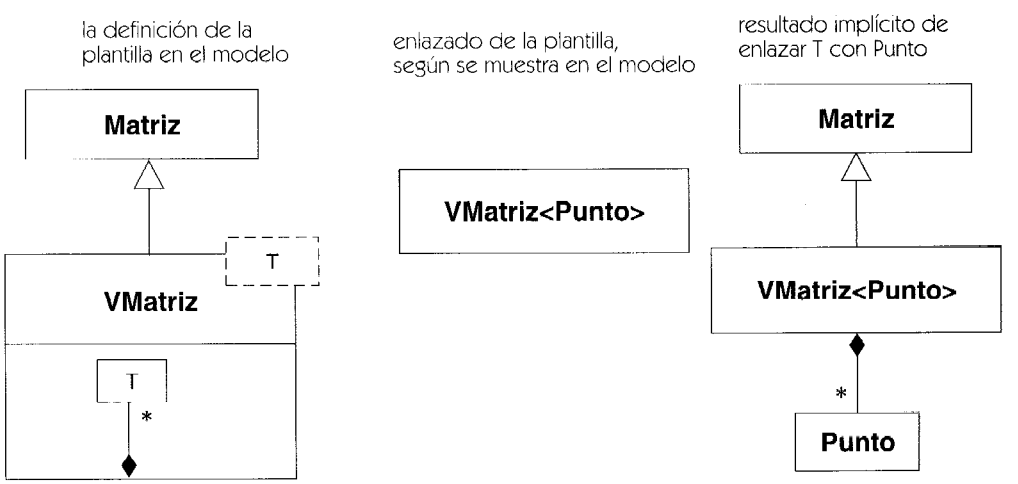


Figura 13.156 Plantilla con relación con uno de sus parámetros

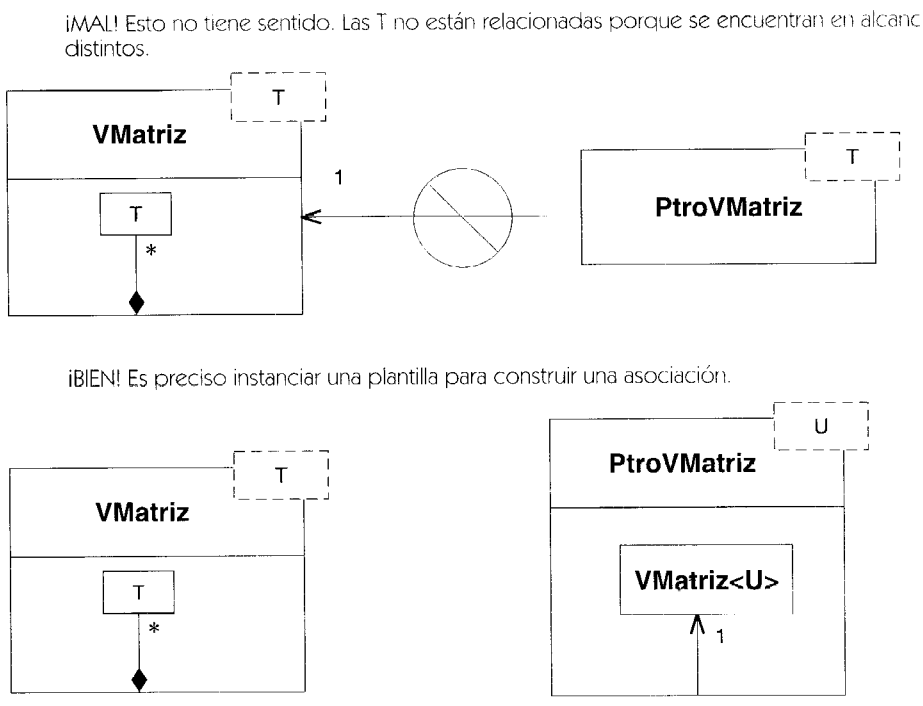


Figura 13.157 Asociaciones entre plantillas

ciada implícitamente dentro de su propio cuerpo. Por tanto, ambas plantillas son instanciadas efectivamente dentro del cuerpo y consiguientemente las relaciones son entre los elementos instanciados.) La Figura 13.157 muestra un intento incorrecto y otro correcto de definir semejante relación; en este caso, se ha empleado una clase parametrizada «**puntero**» que apunta a una matriz parametrizada de igual tipo.

Se puede utilizar un enfoque similar para declarar una clase parametrizada que sea hija de otra clase de plantilla enlazada con el mismo parámetro. El parámetro se utiliza para enlazar una copia de cada una de las otras plantillas. Entonces puede construirse una asociación entre las copias instanciadas de las plantillas. En la mayoría de los casos prácticos, no es necesario hacer esto porque se puede declarar la relación en el cuerpo de alguna de las plantillas.

polimórfico/a[5]

Indica una operación cuya implementación (un método o una máquina de estados disparada por un evento de llamada) puede ser proporcionada por una clase descendiente. Una operación que no sea polimórfica es una operación hoja.

Véase también operación abstracta, generalización, herencia, método.

[5] Utilizaremos indistintamente los términos polimorfo y polimórfico para traducir el término inglés "polymorphic". *N. del Revisor.*

Semántica

Si una operación es polimórfica, entonces se le puede proporcionar un método en una clase descendiente (tanto si ya se ha suministrado un método en la clase original como si no). En caso contrario, es preciso que esté disponible un método para esa operación en la clase en que se declara la operación y el método no podrá ser anulado en una clase descendiente. Se dice que un método está disponible cuando lo declara una clase o cuando es heredado de un antecesor. Una operación abstracta tiene que ser polimórfica (porque no tiene implementación directa). Una operación no es polimórfica si se declara como operación hoja.

Si una operación se declara polimorfa en una clase, esto es, si no se declara como hoja, entonces puede ser declarada como hoja en alguna clase descendiente. Esto evita que sea redefinida o anulada en algún descendiente posterior. Las operaciones hoja no se pueden declarar polimórficas en clases descendientes. No se podrán anular ni redefinir independientemente de la profundidad.

UML no impone unas reglas de combinación de métodos si se declara un método en una clase y se redefine este método en una clase descendiente (*véase* la discusión siguiente). Los mecanismos, como pueda ser la declaración de métodos antes, después y lateralmente, se pueden manejar tal como convenga para un lenguaje específico empleando valores etiquetados. las acciones tales como llamar explícitamente al método heredado son, por supuesto, dependientes del lenguaje de acción, en todo caso.

Notación

Se declara una operación no polimorfa empleando la palabra reservada **{leaf}** (*hoja*). En caso contrario, se supone que es polimorfa.

Discusión

Toda operación abstracta es necesariamente polimórfica. En caso contrario, no se podría implementar en modo alguno. Bertrand Meyer las llama operaciones diferidas porque su especificación se define en una clase pero su implementación se retrasa hasta las subclases. Se trata de una utilización esencial de la herencia, quizá la más esencial de todas tanto para el modelado como para la programación. Mediante el uso de la herencia, se pueden aplicar parámetros a conjuntos de objetos de clases mixtas. El emisor no necesita conocer o determinar la clase a que pertenece cada objeto. Sólo es necesario que todos los objetos se adapten a una clase antecesora que define las operaciones deseadas. La clase antecesora no necesita definir las operaciones. Basta con que se limite a definir sus signaturas. El emisor ni siquiera necesita conocer la lista de posibles subclases. Esto significa que se pueden añadir posteriormente nuevas subclases, sin perturbar las operaciones polimórficas que les son aplicables. El código fuente que invoca a las operaciones no necesita modificaciones cuando se añaden nuevas subclases. La posibilidad de añadir nuevas clases después de haber escrito el código original es uno de los pilares fundamentales de la tecnología orientada a objetos.

Una aplicación más problemática del polimorfismo es la sustitución (también la anulación puede dar problemas) de un método definido en una clase por un método distinto definido en una subclase. Suele decirse que esto es una forma de compartir, pero es peligroso. La redefinición no es incremental, así que todo lo que haya en el método original tiene que reproducirse en el método hijo, aunque sólo sea para efectuar un pequeño cambio. Esta clase de repetición es

proclive a errores. En particular, si el método original se cambia posteriormente, no hay garantía de que el método hijo sea modificado también. Hay ocasiones en que una subclase puede hacer uso de una implementación completamente distinta de cierta operación, pero hay muchos expertos que desaconsejan esta redefición por el peligro que implica. En general, los métodos deberían ser bien completamente heredados o bien diferidos; en este último caso no hay implementación en la superclase así que no hay peligro de redundancia ni de inconsistencia.

Para permitir que una subclase extienda la implementación de una operación sin perder el método heredado, la mayoría de los lenguajes de programación ofrece alguna forma de combinación de métodos que hace uso del método heredado pero permite añadir código adicional. En C++ un método heredado puede ser invocado explícitamente por el nombre de la clase y de la operación, lo cual inserta rígidamente la jerarquía de clases en el código: no es un enfoque muy orientado a objetos. En SmallTalk, un método puede invocar a una operación de **super**, lo cual da lugar a que la operación sea manejada por el método heredado. Si cambia la jerarquía de clases, entonces la herencia sigue funcionando, posiblemente con un método de otra clase diferente. Sin embargo, el método que anula debe proporcionar explícitamente una llamada a **super**. Los errores se producen, porque los programadores se olvidan de insertar las llamadas cuando se produce un cambio. Por último, CLOS ofrece unas reglas muy generales y complicadas para la combinación automática de métodos que puede invocar a varios métodos durante la ejecución de una sola operación de llamada. La operación global está implementada a partir de varios fragmentos en lugar de estar limitada a un único método. Esto es muy general pero resulta más difícil de manejar para el usuario.

UML no impone un único enfoque de combinación de métodos. La combinación de métodos es una cuestión de variación semántica. Se puede emplear cualquiera de estos enfoques. Si el lenguaje de programación no es muy potente en lo tocante a la combinación de métodos, entonces es posible que una herramienta de modelado pueda servir de ayuda para generar el código adecuado en ese lenguaje de programación, o quizá pueda emitir advertencias relativas a posibles errores causados por inacción si se emplea la anulación de métodos.

postcondición

Es una restricción que tiene que satisfacerse cuando se produzca la terminación de una operación.

Semántica

Una postcondición es una expresión booleana que tiene que ser verdadera una vez finalizada la ejecución de una operación. Se trata de una aserción[6], no es una sentencia ejecutable. Dependiendo de la forma exacta de la expresión, quizá sea posible verificarla automáticamente por anticipado. Puede ser útil verificar la postcondición después de la operación, pero esto entra dentro de los límites de la depuración de programas. Se supone que la condición es verdadera y cualquier otra cosa es un error de programación. Una postcondición es una restricción que afecta a quien implementa una operación. Si no se satisface, entonces la operación se ha implementado de forma incorrecta.

Véase también invariante, precondición.

---

[6] El término "assertion" también puede ser traducido como "aserto" o "aseveración", aunque nosotros hemos preferido el más común de "aserción". *N. del Revisor.*

Estructura

Las postcondiciones se modelan como las restricciones, con el estereotipo «**postcondition**», que está asociado a una operación.

Notación

Se puede mostrar una postcondición en una nota con la palabra reservada «**postcondition**». La nota está asociada a la operación afectada.

Ejemplo

La Figura 13.158 muestra una postcondición aplicada a una operación de clasificación de una matriz. El nuevo valor de la matriz (a') está relacionado con el valor original (a). Este ejemplo se expresa en lenguaje natural estructurado. También se podría hacer la especificación en un lenguaje más formal.

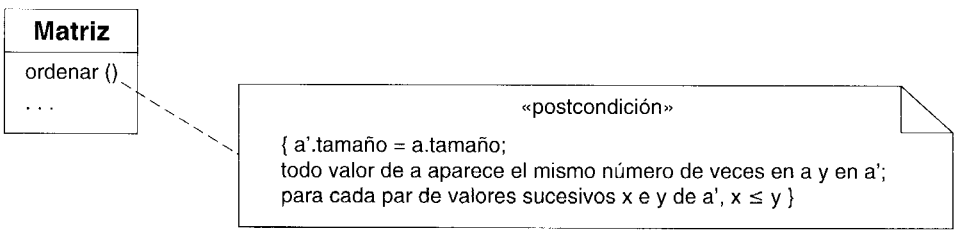


Figura 13.158 Postcondición

precondición

Una restricción que tiene que ser cierta cuando se invoca a una operación.

Semántica

Se trata de una expresión booleana que tiene que ser verdadera cuando se llama a una operación. Es responsabilidad del emisor satisfacer esa condición. No se trata de una condición que tenga que verificar el receptor. Una precondición no es una condición de guarda; es una condición que tiene que *ser* cierta y no una forma de ejecutar opcionalmente una operación. Puede resultar útil verificar la precondición al principio de la operación para más fiabilidad, pero esto entra en los límites de la depuración de programas. Se supone que la condición es verdadera y cualquier otra circunstancia es un error de programación. Si no se cumple la condición, no se puede hacer afirmación alguna con respecto a la integridad de la operación o del sistema. Tiene probabilidades de ser un fracaso absoluto. En la práctica, al verificar explícitamente las precondiciones en el receptor se pueden detectar muchos errores.

Véase también invariante, postcondición.

Estructura

Las precondiciones se modelan como restricciones con el estereotipo «**precondition**» que se asocia a una operación.

Notación

Se puede mostrar una precondición en una nota con la palabra reservada «**precondition**». La nota se asocia a la operación afectada.

Ejemplo

La Figura 13.159 muestra una precondición aplicada a un operador de producto matricial.

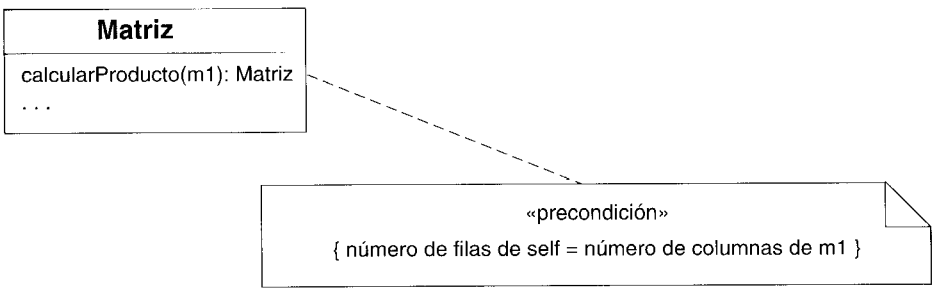


Figura 13.159 Precondición

principio de capacidad de sustitución

Principio consistente en que dada la declaración de una variable o parámetro cuyo tipo declarado es X, toda instancia de elemento que sea descendiente de X puede utilizarse en lugar del valor real sin violar la semántica de la declaración ni de su utilización. En otras palabras, se puede reemplazar una instancia de un elemento antecesor por una instancia de un elemento descendiente. (Atribuido a Barbara Liskov.)

Véase también generalización, herencia de implementación, herencia, herencia de interfaz, polimórfico/a, herencia privada.

Discusión

El propósito es asegurar que las operaciones polimórficas funcionen libremente. No se trata de un principio lógico sino más bien de una regla práctica de programación que proporciona un cierto grado de encapsulamiento. La relación de generalización apoya al principio de capacidad de sustitución.

La consecuencia del principio de capacidad de sustitución es que un hijo no puede eliminar ni redefinir propiedades de su padre. En caso contrario, no se podría utilizar al hijo como sustituto en una situación en que se hubiera declarado la utilización del padre.

privado

Se trata de un valor de visibilidad que indica que el elemento dado no es visible fuera de su propio espacio de nombres, ni siquiera para los descendientes de ese espacio de nombres.

proceso de desarrollo

Un conjunto de pautas y un conjunto de actividades de trabajo parcialmente ordenadas pensadas para producir software de una manera controlada, reproducible. El propósito de un proceso de desarrollo de software es asegurar el éxito y la calidad de un sistema acabado.

Véase también etapas de modelado.

Discusión

UML es un lenguaje de modelado, no un proceso, y su propósito es describir los modelos que se pueden producir por varios procesos de desarrollo. Para la estandarización, es más importante describir los artefactos resultantes de un desarrollo que el proceso de producirlos. Eso es porque hay muchas buenas formas de construir un modelo, y un modelo acabado puede ser utilizado sin saber cómo fue producido. Sin embargo, UML está pensado para apoyar una amplia gama de procesos.

Para más detalles del proceso iterativo, incremental, guiado por casos de uso, centrado en la arquitectura que promueven los autores de este libro. *Véase* [Jacobson-99].

La relación de las etapas de modelado y las fases de desarrollo

Las etapas de modelado encajan dentro de un proceso de desarrollo iterativo, que tiene las fases de inicio, elaboración, construcción, y transición (fase). Las fases son secuenciales dentro de una entrega de una aplicación, pero cada fase incluye una o más iteraciones. Dentro de una iteración, los elementos individuales del modelo se mueven a lo largo del camino desde el análisis hacia el despliegue, cada uno a su propio ritmo. Aunque las fases del desarrollo y las etapas de modelado no se sincronizan, hay una correlación.

En las fases anteriores del desarrollo y las iteraciones anteriores de una fase, hay más énfasis en las etapas de modelado anteriores.

La Figura 13.160 muestra el equilibrio del esfuerzo durante las fases e iteraciones sucesivas. Durante la inicio, el foco de interés está principalmente en análisis, con un esqueleto de los elementos que progresa hacia el diseño y la implementación durante la elaboración. Durante la construcción y la transición, todos los elementos deben, eventualmente, llegar a su terminación.

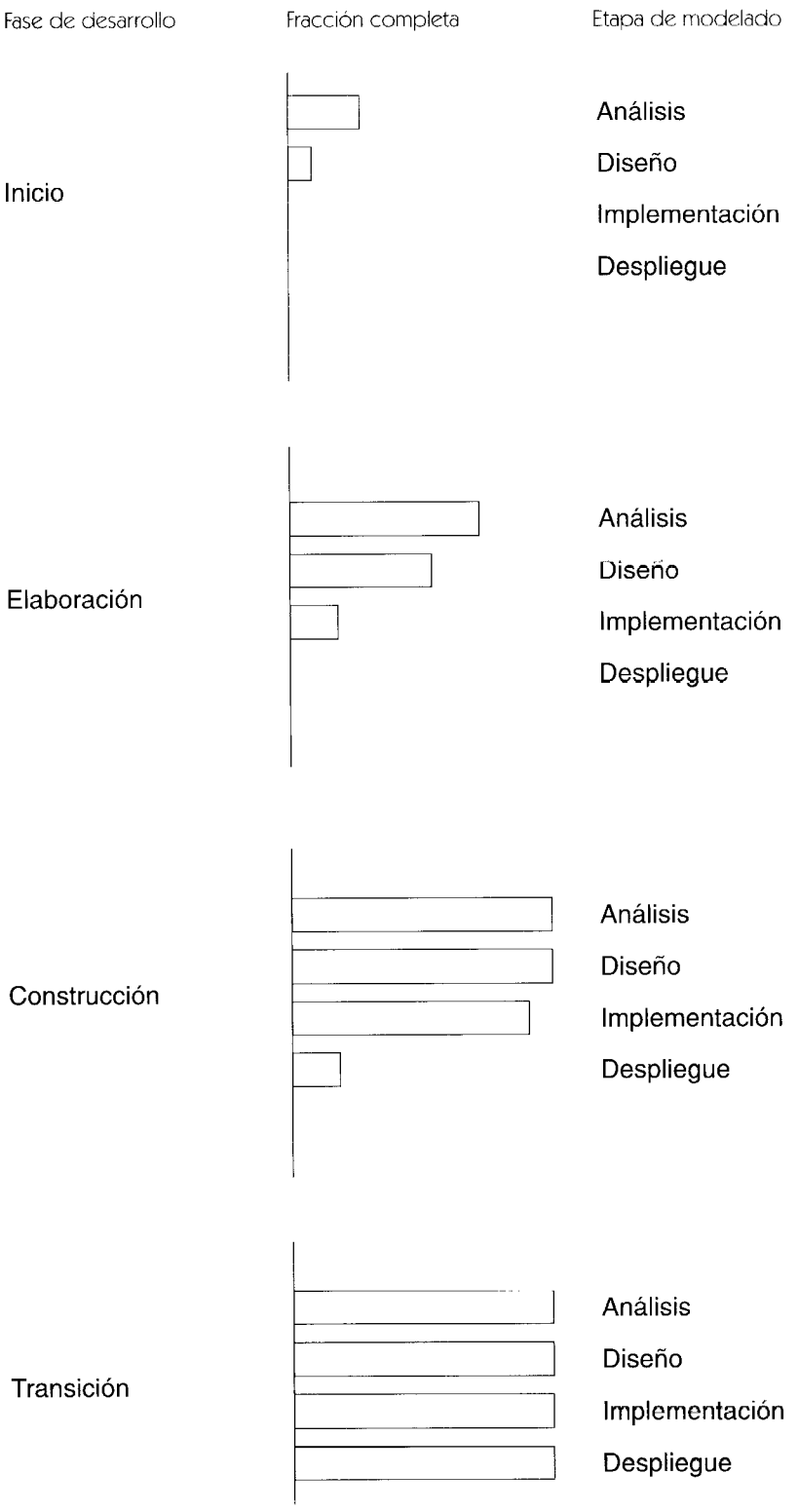


Figura 13.160 Progreso después de cada fase de desarrollo

proceso/procesar

1. Unidad compleja de concurrencia y ejecución en un sistema operativo. *Véase* hilo, que incluye los procesos complejos y ligeros. Si es necesario, se puede hacer una distinción de implementación empleando estereotipos.
2. Un proceso de desarrollo de software; son los pasos y directrices mediante los cuales se desarrolla un sistema.
3. Ejecutar un algoritmo o en general manejar algo dinámicamente.

producto

Dícese de los resultados del desarrollo, tales como modelos, código, documentación y planes de trabajo; también puede denotar un producto de trabajo.

propiedades

Término genérico que denota un valor con nombre que aporta información acerca de un elemento del modelo. Las propiedades tienen impacto semántico. Ciertas propiedades están definidas en UML, otras pueden ser definidas por el usuario.

Véase atributo, relación, valor etiquetado.

Semántica

Entre las propiedades se cuentan los atributos incorporados (definidos por UML) así como los valores etiquetados (definidos por el usuario) y las relaciones (definidas por el usuario) asociadas a un elemento. Desde el punto de vista de un usuario, suele carecer de importancia el que una propiedad sea incorporada o esté implementada como un valor etiquetado.

Discusión

Obsérvese que se utiliza la palabra *propiedad* en un sentido general, para denotar cualquier valor que esté asociado a un elemento del modelo, incluyendo los atributos, asociación y valores etiquetados. En este sentido, puede incluir los valores alcanzables indirectamente que se pueden hallar comenzando en un determinado elemento.

protegida

Dícese de aquel valor de visibilidad que indica que el elemento dado sólo es visible fuera de su propio espacio de nombres para los descendientes del espacio de nombres.

proveedor

Elemento que proporciona servicios que pueden ser invocados por otros. Contrastar con: cliente. En la notación, el proveedor aparece en la punta de flecha de una flecha discontinua de dependencia.

Véase dependencia.

proyección

Correspondencia entre un conjunto y un subconjunto del mismo. La mayoría de los modelos y de los diagramas son proyecciones del conjunto completo de información que está potencialmente disponible.

pseudoatributo

Denota un valor relacionado con una clase que se comporta como un atributo, a saber, que tiene un valor exclusivo para cada instancia.

Véase también discriminador, nombre de rol.

Semántica

Entre los pseudoatributos se cuentan los nombres de roles de asociación y los discriminadores de generalización. Un nombre de rol de asociación es un pseudoatributo de la clase que se encuentra al otro extremo de la asociación. Un discriminador de generalización es un pseudoatributo del elemento padre. En todos los elementos hijos, el valor del discriminador es el nombre del elemento hijo.

Se puede utilizar un pseudoatributo como nombre en una expresión para recuperar un valor de un objeto. Dado que los nombres de atributos y los nombres de pseudoatributos se pueden emplear en las expresiones, se encuentran en el mismo espacio de nombres y tienen que ser únicos en ese espacio de nombres. Los nombres también tienen que ser únicos respecto a los nombres de atributos y pseudoatributos heredados.

pseudoestado

Denota un vértice de una máquina de estados que posee la forma de un estado pero no se comporta como un estado completo. Véase estado abreviado externo, estado de historia, estado de unión o conjunción, estado inicial.

Semántica

Cuando un pseudoestado está activado, hay una máquina de estados que no ha finalizado su paso que se ejecuta hasta finalizar y que no procesa eventos. Entre los pseudoestados se cuentan el es-

tado inicial, el estado de unión o conjunción, el estado abreviado externo y el estado de historia. Los pseudoestados se utilizan para encadenar segmentos y una transición a uno de ellos implica otra transición automática adicional a otro estado sin necesidad de que intervenga un evento.

Los estados finales y de sincronización no son pseudoestados. Son estados especiales que pueden permanecer activos cuando una máquina de estados ya ha finalizado su paso que se ejecuta hasta finalizar, pero tienen restricciones relativas a las transiciones que pueden partir de ellos.

pública

Dícese de aquel valor de visibilidad que indica que el elemento dado es visible fuera de su propio espacio de nombres.

punto de extensión

Un marcador con nombre que se refiere a una localización o a un conjunto de localizaciones dentro de la secuencia de comportamiento para un caso de uso, en el cual se puede insertar comportamiento adicional. Una declaración de punto de extensión abre la posibilidad de extensión al caso de uso. Un segmento de inserción es una secuencia de comportamiento en un caso de uso de extensión (un caso de uso relacionado con un caso de uso base por una relación de extensión). La relación de extensión contiene una lista de nombres de puntos de extensión que indican el lugar donde los segmentos de la inserción del caso de uso extendido insertan su comportamiento.

Véase también extender, caso de uso.

Semántica

Un punto de extensión tiene un nombre y se refiere a un conjunto de una o más localizaciones dentro de una secuencia de comportamiento de un caso de uso. Un punto de extensión puede referirse a una sola localización entre dos pasos del comportamiento dentro de un caso de uso. Además, puede referirse a un conjunto de localizaciones discretas. Puede también referirse a una región dentro de una secuencia de comportamiento (éste no es nada más que el conjunto de todas las localizaciones entre los pasos de la secuencia).

Una localización es un estado dentro de la máquina de estados que describe un caso de uso, o el equivalente de una descripción diferente —entre dos sentencias de una lista de sentencias o entre dos mensajes en una interacción.

Una relación de extensión contiene una condición opcional y una lista de referencias a puntos de extensión iguales en número al número de los segmentos de inserción en el caso de uso extendido. Un segmento de inserción puede ser realizado si la condición está satisfecha mientras que una instancia del caso de uso está ejecutando el caso de uso base en cualquier localización en el punto de extensión que corresponde al segmento de la inserción.

La localización de un punto de extensión puede ser cambiada sin afectar su identidad. El uso de los puntos de extensión con nombre separa la especificación de las secuencias de los detalles internos del caso de uso base. El caso de uso base puede ser modificado o ser cambiado sin afectar las extensiones. Por otra parte, un punto de la extensión se puede mover dentro de la base sin afectar la relación o el caso de uso extendido. Como con todos los tipos de modularidad, esta independencia requiere una buena elección de los puntos de extensión y no está garantizada bajo todas las circunstancias.

Notación

Los puntos de extensión para un caso de uso se pueden enumerar en un compartimento llamado **puntos de extensión**.

Los puntos de extensión deben también referirse a localizaciones dentro de la secuencia de comportamiento del caso de uso. El beneficio neto es que los nombres del punto de extensión sirven como etiquetas para los estados, las declaraciones, y las regiones dentro de la secuencia de comportamiento. Esto no significa que deban estar presentes en el texto de la secuencia de comportamiento original.

Los puntos de extensión son nombres que permiten separar la localización de la opción para realizar una inserción. Una herramienta de edición puede poner un punto de extensión en un recubrimiento del texto de la secuencia de comportamiento sin modificar el texto original, o puede tener una tabla separada de los nombres de puntos de extensión distribuida en declaraciones (por número de declaración, etiquetas internas, o gráficos directos). En cualquier caso, el efecto neto corresponde a la localización de cada nombre de punto de extensión entre uno o más pasos en las secuencias de comportamiento. En el ejemplo de pseudocódigo en la Figura 13.161 un nombre de punto de extensión entre símbolos de < y > refiriéndose a una localización en la secuencia de comportamiento. Hay muchas sintaxis posibles para localizar puntos de extensión dentro de secuencias de comportamiento, dependiendo del lenguaje utilizado para especificar la secuencia. Esto es un ejemplo, pero no la única forma.

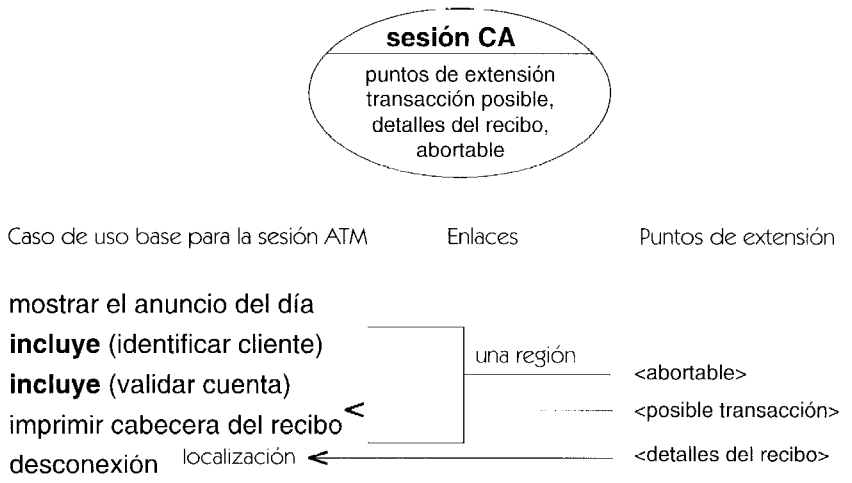


Figura 13.161 Declaración de puntos de extensión

punto de variación semántica

Punto de variación en la semántica de un metamodelo. Proporciona un grado de libertad deliberado para la interpretación de la semántica del metamodelo.

Discusión

Una misma semántica de ejecución no resulta adecuada para todas las posibles aplicaciones. Los distintos lenguajes de programación y los diferentes propósitos requieren variaciones de la semántica; algunas de ellas son sutiles y otras bastas. Un punto de variación semántica es un asunto en que distintos creadores de modelos o diferentes entornos de ejecución discrepan con respecto a la semántica específica, frecuentemente con buenas razones. Al identificar y dar nombre a los puntos de variación semántica, se pueden evitar las discusiones acerca de la semántica "correcta" de un sistema.

Por ejemplo, la opción de permitir o no la clasificación múltiple o la clasificación dinámica es un punto de variación semántica. Cada opción es una variación semántica.

Entre otros ejemplos de puntos de variación semántica se incluye si una llamada puede proporcionar más de un valor y si las clases existen durante el tiempo de ejecución como objetos reales.

realización

Relación entre una especificación y su implementación; indicación de la herencia de comportamiento sin la herencia de estructura.

Véase también interfaz.

Semántica

Las especificaciones describen el comportamiento o estructura de algo sin determinar cómo se implementará el comportamiento. Una implementación proporciona los detalles relativos a la forma de implementar el comportamiento de una forma computable efectivamente. La relación entre un elemento que especifica comportamiento y otro que proporciona una implementación se denomina realización. En general, hay muchas maneras de realizar una especificación. De forma similar, un elemento puede realizar más de una especificación. La realización es, por tanto, una relación muchos-a-muchos entre elementos.

El significado de la realización es que el elemento cliente tiene que admitir todo el comportamiento del elemento proveedor pero no tiene por qué satisfacer su estructura o su implementación. Un clasificador cliente, por ejemplo, tiene que admitir las operaciones del clasificador proveedor; además tiene que admitir todas las máquinas de estados que especifiquen el comportamiento externo del proveedor. Pero todos los atributos, asociaciones, métodos o máquinas de estados del proveedor que especifiquen la implementación son irrelevantes para el cliente. Observe que el cliente no hereda realmente las operaciones del proveedor. Tiene que declararlas él mismo o heredarlas de algún antecesor para que queden cubiertas todas las operaciones del pro-

veedor. En otras palabras, el proveedor de una realización indica qué operaciones tienen que estar presentes en el cliente, pero el cliente es quien tiene la responsabilidad de aportarlas.

Ciertas clases de elementos, tal como las interfaces y los casos de uso, están destinados a especificar el comportamiento y no contienen información de implementación. Otras clases de elementos, tales como las clases, están destinadas a implementar el comportamiento. Contienen información de implementación pero también se puede utilizar de forma más abstracta como especificadores. Normalmente, la realización relaciona un elemento de especificación, tal como un caso de uso o una interfaz, con un elemento de implementación, tal como una colaboración o una clase. Es posible utilizar un elemento de implementación, tal como una clase, para la especificación. Se puede situar en el lado de especificación de una relación de realización. En este caso, sólo las partes de especificación de la clase proveedora afectarán al cliente. Entonces, hablando con más precisión, la realización es una relación entre dos elementos en los cuales las partes de especificación de comportamiento externo de uno de ellos restringen la implementación del otro. Se podría pensar que se trata de una herencia de comportamiento sin herencia de estructura o de implementación (y con la necesidad de declarar realmente las operaciones por parte del cliente).

Si el elemento de especificación es una clase abstracta sin atributos, sin asociaciones y únicamente con operaciones abstractas, entonces toda especialización de la clase abstracta realiza la clase abstracta, porque nada hay que heredar salvo la especificación.

El elemento que hace la implementación tiene que admitir todo el comportamiento incluido en el elemento que hace la especificación. Por ejemplo, una clase tiene que contener todas las operaciones de aquellas interfaces que realiza, con una semántica que sea consistente con todas las especificaciones que requieran las interfaces. La clase puede implementar operaciones adicionales y la implementación de las operaciones puede hacer cosas adicionales, siempre y cuando no se violen las especificaciones de operación de las interfaces.

Notación

La relación de realización se muestra mediante una ruta discontinua con una punta de flecha triangular hueca adyacente al elemento que proporciona la especificación y con el origen en el elemento que proporciona la implementación (Figura 13.162).

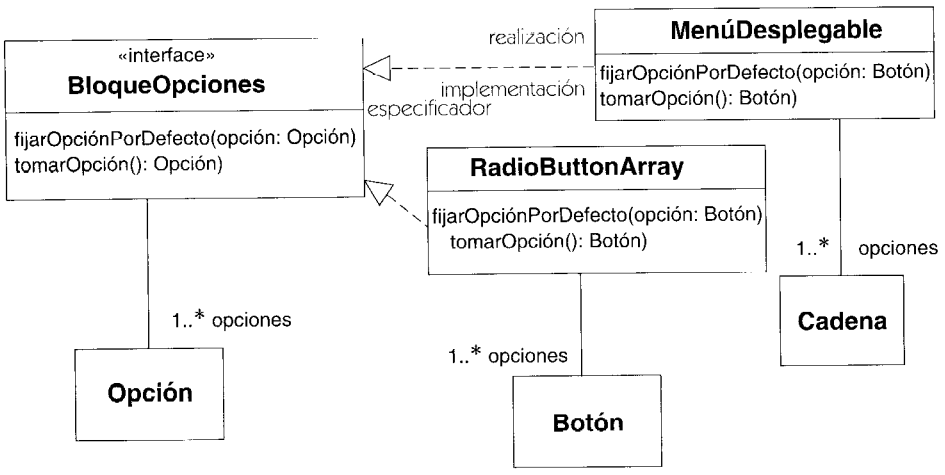


Figura 13.162 Relación de realización

Discusión

Otro caso importante es la realización de un caso de uso mediante una colaboración (Figura 13.163). Los casos de uso especifican secuencias de funcionalidad y comportamiento visibles externamente pero no proporcionan una implementación. Una colaboración describe los objetos que implementan el comportamiento del caso de uso y la forma en que interaccionan para hacerlo. Normalmente, una colaboración implementa un caso de uso, pero una colaboración se puede implementar empleando colaboraciones subordinadas, cada una de las cuales hace una parte del trabajo. Los objetos y las clases que se utilizan para implementar una colaboración suele aparecer también en otras colaboraciones.

Figura 13.163 Realización de un caso de uso por colaboración

Toda clase de la colaboración dedica una parte de su funcionalidad al caso de uso que se está implementando. Por tanto, el caso de uso será implementado eventualmente por partes en varias clases.

realizar

Proporcionar la implementación de un elemento de especificación.

Véase realización.

recepción

La declaración de que un clasificador está preparado para reaccionar cuando reciba una señal. Es un miembro de un clasificador.

Semántica

Una recepción es una declaración que indica que un clasificador está preparado para admitir una instancia de una señal y reaccionar frente a ella. Una recepción es algo parecido a una operación. Declara la signatura de un mensaje que admite el clasificador y especifica su significado.

Estructura

Una recepción posee las propiedades siguientes:

polimorfismo — Denota si la respuesta del clasificador frente a la señal es o no siempre la misma. Se codifica mediante la propiedad **isPolymorphic**, mediante los valores siguientes:

true	La respuesta es polimorfa: puede depender del estado y también puede ser anulada por un descendiente.
false	La respuesta debe ser la misma, independientemente del estado, y no puede ser anulada por un descendiente. El efecto neto es que tiene que haber una única transición de toda la máquina de estados que maneja el evento.
señal	Denota la señal a la cual responderá el clasificador.
especificación	Una expresión que enuncia los efectos que causa la recepción de la señal.

Notación

La recepción puede mostrarse en la lista de operaciones de una clase o interfaz empleando la sintaxis correspondiente a una operación con la palabra reservada «**signal**» delante del nombre de la señal.

Alternativamente, se puede situar una lista de signaturas de señal en su propio compartimento; el compartimento tiene el nombre **Señales**. Se muestran las dos formas en la Figura 13.164.

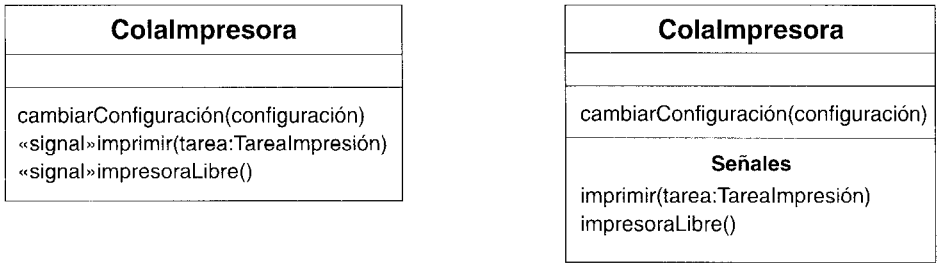


Figura 13.164 Hay dos maneras de indicar la recepción de una señal

receptor

Dícese de aquel objeto que maneja una instancia de mensaje que se le pasa desde un objeto emisor.

recibir

Manejar una instancia de mensaje que se nos pasa procedente de un objeto emisor.

Véase emisor, receptor.

referencia

Alude a una forma de denotar un elemento del modelo; suele llamarse puntero.

Semántica

Los elementos del modelo están conectados mediante dos metarrelaciones: propiedad y referencia. La propiedad es la relación que existe entre un elemento y sus partes constitutivas, las partes que están definidas en su interior y que él posee. La relación de propiedad forma un árbol estricto. Los elementos contenidos están subordinados al elemento contenedor. La propiedad, el control de configuración y el almacenamiento de modelos están basados en la jerarquía de contención.

La referencia es una relación entre elementos de un mismo nivel de detalle o bien entre diferentes contenedores. Por ejemplo, la referencia es la relación que existe entre una asociación y las clases participantes en ella, entre un atributo y la clase o tipo de dato que es una propiedad de tipo, entre una plantilla ligada y los valores de sus argumentos. Para que sea posible una referencia, el elemento del que se haga la referencia tiene que ser visible para el elemento que la hace. En general, esto significa que el paquete que contiene el origen de la referencia debe poder ver el paquete que contiene el destino de la referencia. Esto requiere una relación adecuada de acceso o de importación entre los paquetes. También requiere que el elemento al que se haga referencia tenga una configuración de visibilidad que le permita ser visto fuera de su paquete, a no ser que la referencia tenga lugar en el mismo paquete.

Obsérvese que la referencia es una relación interna de metamodelo, no una relación visible por el usuario; se emplea para construir las demás relaciones.

refinamiento

Denota una relación que representa una especificación más completa de algo que ya se ha especificado con un cierto nivel de detalle o en un nivel semántico diferente.

Véase también abstracción.

Semántica

El refinamiento es una conexión histórica o computable entre dos elementos que poseen una correspondencia (no necesariamente completa) entre sí. Con frecuencia, los dos elementos se encuentran en modelos diferentes. Por ejemplo, una clase de diseño puede ser un refinamiento de una clase de análisis; tiene los mismos atributos lógicos pero sus clases pueden provenir de una biblioteca de clases específica. Sin embargo, un elemento puede refinar a otro en un mismo modelo. Por ejemplo, una versión optimizada de una clase es un refinamiento de la versión sencilla pero ineficiente de esa misma clase. La relación de refinamiento puede contener una descripción de la correspondencia, que se puede escribir en un lenguaje formal (tal como OCL o algún lenguaje lógico o de programación). También puede ser un texto informal (lo cual, evidentemente, impide toda computación automática pero puede ser útil en las fases iniciales del desarrollo). Se puede utilizar el refinamiento para modelar la elaboración paso a paso del desarrollo, optimización, transición y el marco de trabajo.

Estructura

El refinamiento es una clase de dependencia. Relaciona a un cliente (el elemento que está más desarrollado) con un proveedor (el elemento que es la base del refinamiento).

Notación

Los refinamientos se indican mediante una flecha de dependencia (una flecha discontinua con la punta en el elemento más general y el origen en el elemento más específico) que posee la palabra reservada «**refine**». La correspondencia puede estar asociada a la ruta de dependencia mediante una línea discontinua conectada a una nota. Se han propuesto diferentes tipos de refinamientos; es posible indicarlos mediante más estereotipos. En muchos casos, el refinamiento conecta elementos de diferentes modelos y, por tanto, no será visible gráficamente. Con cierta frecuencia, sólo está implícito.

Ejemplo

La optimización es una clase típica de refinamiento. La Figura 13.165 muestra un tablero de ajedrez que posee una sencilla representación en el modelo de análisis, pero otra representación mucho más elaborada y oscura en el modelo de diseño. La clase de diseño no es una especialización de la clase de análisis, porque tiene una forma completamente diferente. La clase del modelo de análisis y la del modelo de diseño tienen el mismo nombre, porque representan un mismo concepto en dos niveles semánticos diferentes.

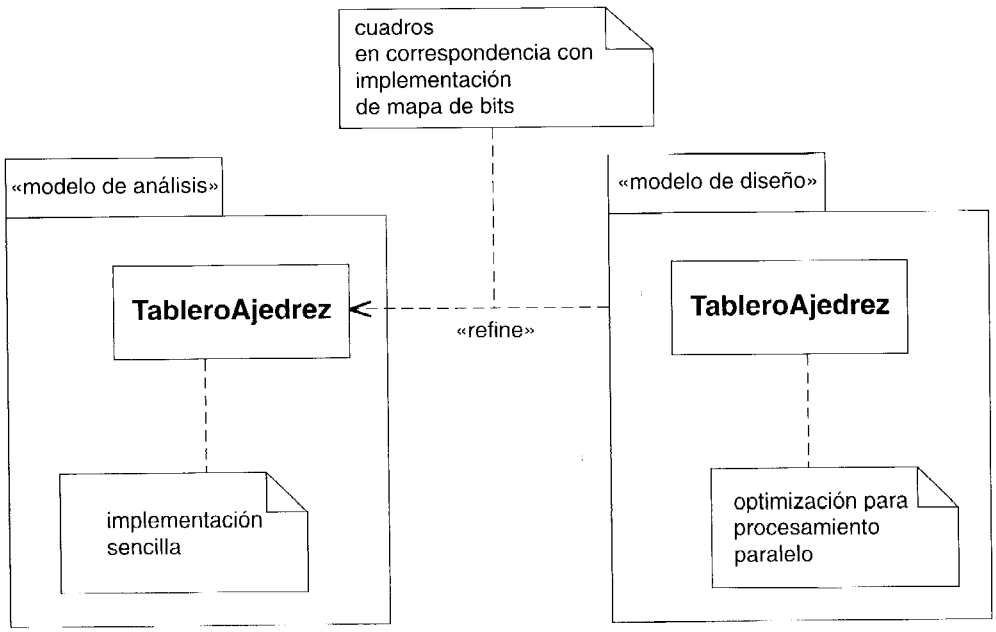


Figura 13.165 Refinamiento

refine

Palabra clave correspondiente a una dependencia de refinamiento en la notación.

relación

Dícese de una conexión semántica materializada entre elementos de un modelo. Entre las clases de relaciones se cuentan la asociación, generalización, metarelación, flujo y distintos grupos que están agrupados con el nombre de dependencia.

Semántica

La Tabla 13.2 muestra las distintas clases de relaciones que hay en UML. La primera columna (clase) denota los agrupamientos en que se organizan en el metamodelo. La segunda columna

Tabla 13.2 Relaciones en UML

Clase	*Variedad*	*Notación*	*Símbolo o palabra reservada*
abstracción	derivación	dependencia	«**derive**»
	realización	realización	- - - - ▷
	refinamiento	dependencia	«**refine**»
	traza	dependencia	«**trace**»
asociación		asociación	———
enlazado		dependencia	«**bind**» (parametros$_{list_1}$)
extensión		dependencia	«**extend**» (extensión puntos$_{lista_1}$)
flujo	conversión	dependencia	número-secuencia: «**become**»
	copia	dependencia	número-secuencia: «**copy**»
generalización		generalización	——▷
inclusión		dependencia	«**include**»
metarelación	instancia	dependencia	«**instanceOf**»
	supratipo	dependencia	«**powertype**»
permiso	acceso	dependencia	«**access**»
	amigo	dependencia	«**friend**»
	importación	dependencia	«**import**»
utilización	llamada	dependencia	«**call**»
	instanciación	dependencia	«**instantiate**»
	parámetro	dependencia	«**parameter**»
	envío	dependencia	«**send**»

(variedad) muestra las distintas clases de relaciones. La tercera columna (notación) muestra la notación base para cada relación: la asociación es una ruta continua, la dependencia es una flecha discontinua y la generalización es una ruta continua con punta de flecha triangular. La cuarta columna (palabra reservada) muestra las palabras reservadas de texto y la sintaxis adicional de aquellas relaciones que la requieren.

repositorio

Lugar de almacenamiento para modelos, interfaces e implementaciones; forma parte de un entorno para manipular artefactos de desarrollo.

requisito

Una característica, propiedad o comportamiento que se desea para el sistema.

Semántica

Un requisito de texto se puede modelar como un comando al que se adjunta el estereotipo «**requirement**».

Discusión

El término *requisito* es una palabra del lenguaje natural que se corresponde con una gama de estructuras de UML destinadas a especificar las características deseadas para un sistema. Los requisitos correspondientes a transacciones visibles por parte del usuario suelen capturarse como casos de uso. Los requisitos no funcionales, como las métricas de rendimiento y de calidad, se pueden capturar en forma de enunciados de texto que eventualmente llegarán a los elementos del diseño final. Los comentarios y restricciones de UML se pueden emplear para representar requisitos no funcionales.

responsabilidad

Contrato u obligación de una clase u otro elemento.

Semántica

Una responsabilidad puede representarse como un estereotipo en un comentario. El comentario se asocia a la clase u otro elemento que tenga la responsabilidad. Ésta se expresa en la forma de una cadena de texto.

Notación

Las responsabilidades se pueden mostrar en un compartimento con nombre situado dentro del rectángulo que es el símbolo de un clasificador (Figura 13.166).

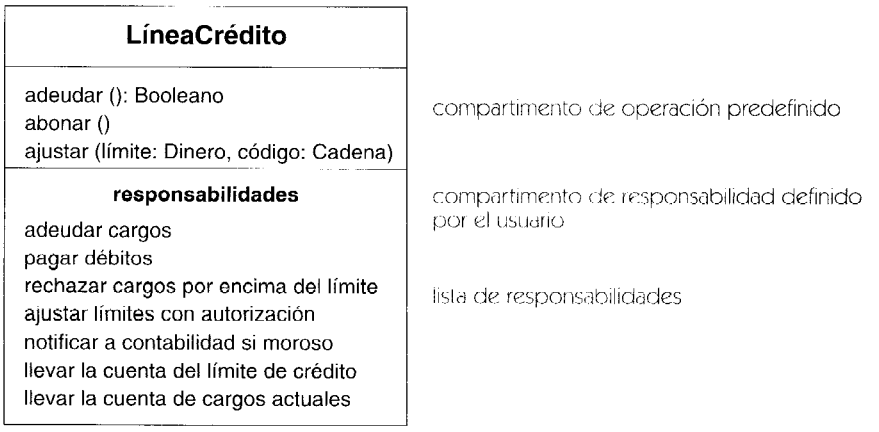


Figura 13.166 Compartimento de responsabilidades

restricción

Una condición semántica o limitación representada como expresión. Ciertas restricciones se predefinen en UML, otras se pueden definir por los modeladores. Las restricciones son uno de tres mecanismos de extensión en UML.

Véase también expresión, estereotipo, valor etiquetado.

Véase Capítulo 14, Elementos Estándar, para una lista de restricciones predefinidas.

Semántica

Una restricción es una condición o limitación semántica expresada como declaración lingüística en un determinado lenguaje textual.

En general, una restricción se puede unir a cualquier elemento o lista de elementos del modelo. Representa información semántica unida a un elemento del modelo, no sólo a una vista de él. Cada restricción tiene un cuerpo y un lenguajeje de interpretación. El cuerpo es una cadena que codifica una expresión booleana para la condición en el lenguaje de la restricción. Una restricción se aplica a una lista ordenada de uno o más elementos del modelo. Observe que el lenguaje de especificación puede ser un lenguaje formal o puede ser lenguaje natural. En el último caso, la restricción será informal y no está sujeta a la ejecución automática (que no es decir que la ejecución automática sea siempre práctica para todos los lenguajes formales). UML proporciona el lenguaje de restricciones OCL [Warmer-99], pero es posible usar otros lenguajes.

Algunas restricciones comunes tienen nombres para evitar escribir una declaración completa cada vez que son necesarias. Por ejemplo, la restricción **xor** entre dos asociaciones que compartan una clase común significa que un solo objeto de la clase compartida puede pertenecer a solamente una de las asociaciones al mismo tiempo.

Véase Capítulo 14, Elementos Estándar, para una lista de las restricciones predefinidas de UML.

Una restricción es una aserción, no un mecanismo ejecutable. Indica una condición que se debe hacer cumplir para el correcto diseño del sistema. Cómo garantizar una restricción es una decisión de diseño. Las restricciones de ejecución se hacen para ser evaluadas en los momentos en que el sistema instanciado es estable —es decir, entre la ejecución de operaciones y no en el medio de cualquier transacción atómica—. Durante la ejecución de una operación, puede haber momentos en que las restricciones se violen temporalmente.

Una restricción no se puede aplicar a sí misma.

Una restricción heredada —una restricción en un elemento antecesor del modelos o un estereotipo debe seguirse aunque se definan restricciones adicionales en los descendientes. Una restricción heredada no puede ser pasada por alto o reemplazada. Si necesita hacerlo, el modelo está mal construido y debe ser reformulado. Sin embargo una restricción heredada se puede restringir, agregando restricciones adicionales. Si están en conflicto con las restricciones heredados por un elemento, entonces el modelo está mal formado.

Notación

Una restricción se representa como una cadena de texto entre llaves ({ }). La cadena de texto es el cuerpo codificado escrito en un lenguaje de restricción.

Es de esperar que las herramientas proporcionen uno o más lenguajes en los cuales se puedan escribir las restricciones formales. El lenguaje predefinido para la escritura de restricciones es OCL. Dependiendo del modelo, un lenguaje de programación tal como C++ puede ser útil para algunas restricciones. Si no, la restricción se puede escribir en lenguaje natural, con responsabilidad humana cara a la interpretación y ejecución. El lenguaje de cada restricción es parte de la restricción en sí misma, aunque el lenguaje no se expone generalmente en el diagrama (la herramienta no lo pierde de vista).

Para una lista de los elementos representados por las cadenas de texto en un compartimento (tal como los atributos dentro de una clase): una cadena de restricción puede aparecer como una entrada en la lista (Figura 13.167). La entrada no representa un elemento del modelo. Es una *restricción viva* que se aplica a todos los elementos correctos de la lista hasta otro elemento de la lista de restricciones o el fin de la lista. La restricción viva se puede sustituir por otra restricción viva más adelante en la lista. Para eliminar una restricción de ejecución, la sustituimos por una restricción vacía. Una restricción unida a un elemento individual de la lista no sustituye la restricción viva, pero puede aumentarla con restricciones adicionales.

Para un solo símbolo gráfico (tal como una clase o una línea de asociación), la cadena de restricción se puede colocar cerca del símbolo, preferiblemente cerca del nombre del símbolo, si lo hay.

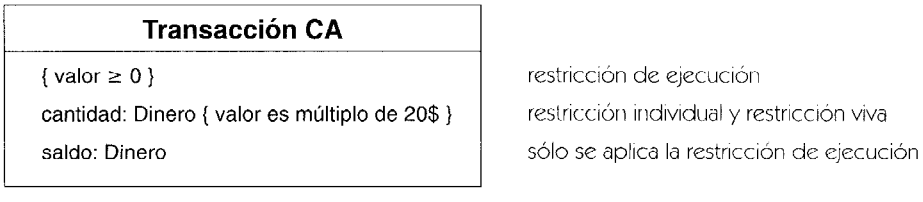


Figura 13.167 Restricciones dentro de listas

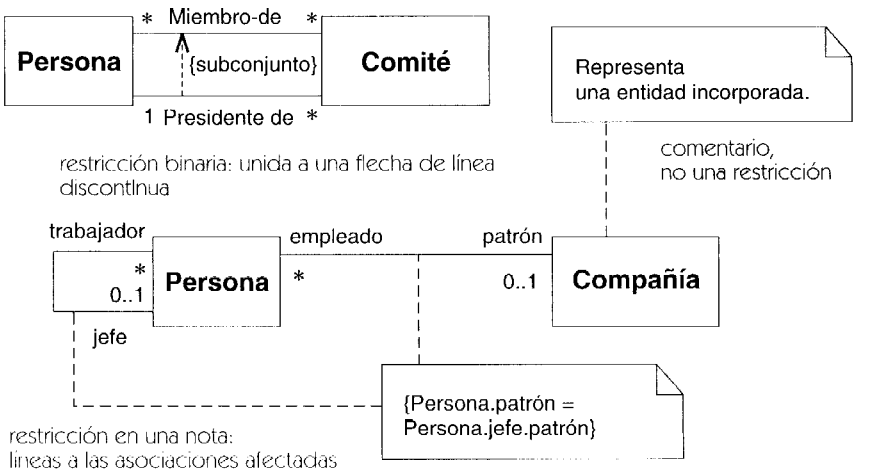


Figura 13.168 Notación de restricciones

Para dos símbolos gráficos (tales como dos clases o dos asociaciones): la restricción se representa como una flecha con línea discontinua de un elemento al otro elemento etiquetado por la cadena de la restricción (entre llaves). La dirección de la flecha es información relevante dentro de la restricción (Figura 13.168).

Para tres o más símbolos gráficos: La cadena de la restricción se coloca en un símbolo de nota y es unida a cada símbolo por una línea discontinua (Figura 13.168). Esta notación se puede utilizar para otros casos también. Para tres o más líneas del mismo tipo (tales como líneas de generalización o líneas de asociación), la restricción se puede unir a una línea discontinua que atraviesa todas las líneas. En caso de ambigüedad, las diferentes líneas se pueden numerar o etiquetar para establecer su correspondencia con la restricción.

Discusión

Una restricción hace una declaración semántica sobre el modelo en sí mismo, mientras que un comentario es un texto sin fuerza semántica y se puede unir a un elemento del modelo o a un elemento de representación. Ambos, restricciones y comentarios se pueden representar usando notas. En principio, las restricciones son ejecutables por las herramientas. En la práctica, algunas pueden ser difíciles de indicar formalmente y pueden requerir la ejecución humana. En el amplio sentido de la palabra, muchos elementos en un modelo son restricciones, pero la palabra restricción se utiliza para indicar las declaraciones semánticas que son difíciles de expresar con los elementos incorporados al modelo y que se deben indicar de manera lingüística.

Las restricciones se pueden expresar en cualquier lenguaje conveniente o aun en un lenguaje humano, aunque una restricción en lenguaje humano no se puede verificar por una herramienta.

El lenguaje OCL [Warmer-99] está diseñado para especificar restricciones de UML, pero bajo algunas circunstancias puede ser más apropiado un lenguaje de programación.

Ya que las restricciones se expresan como cadenas de texto, una herramienta de modelado genérica puede admitirlas y mantener su significado sin entenderlo. Por supuesto, una herra-

mienta o un dispositivo suplementario que verifique o haga cumplir la restricción debe entender la sintaxis y la semántica del lenguaje.

Se puede unir una lista de restricciones a la definición de un estereotipo. Esto indica que todos los elementos que llevan el estereotipo están conforme a la restricción.

Ejecución. Cuando el modelo contiene restricciones, no es necesario especificar qué hacer si son violadas. Un modelo es una declaración de lo que se supone que debe suceder. El hacer que suceda es trabajo de la implementación. Un programa puede contener aserciones y otros mecanismos de la validación, pero una falta contra una restricción se debe considerar una falta de programación. Por supuesto, si un modelo puede ayudar a producir un programa que esté listo para la construcción o se pueda verificar como correcto, entonces ha respondido a su propósito.

Elementos estándar

invariant, postcondition, precondition.

reunificación

Un lugar de una máquina de estados, diagrama de actividades o diagrama de secuencia en el cual se unen dos o mas rutas de control alternativas; una "desbifurcación". Antónimo: bifurcación.

Véase también estado de unión o conjunción.

Semántica

Una reunificación no es otra cosa que una situación en la que se juntan dos o más rutas de control. En una máquina de estados, un estado tiene más de una transición de entrada. No se requiere ni se proporciona ninguna estructura especial del modelo para indicar una reunificación. Se puede indicar mediante un estado de unión si forma parte de una única ruta que se ejecuta hasta finalizar.

Notación

Una reunificación puede aparecer en un diagrama de estados, un diagrama de actividades o un diagrama de secuencia como un rombo con dos o más transiciones de entrada y una sola transición de salida. No se necesitan condiciones. La Figura 13.169 muestra un ejemplo.

También se emplea un rombo para las bifurcaciones (que son la inversa de las fusiones), pero la bifurcación se distingue claramente porque posee una transición de entrada y múltiples transiciones de salida, cada una de las cuales posee su propia condición de guarda.

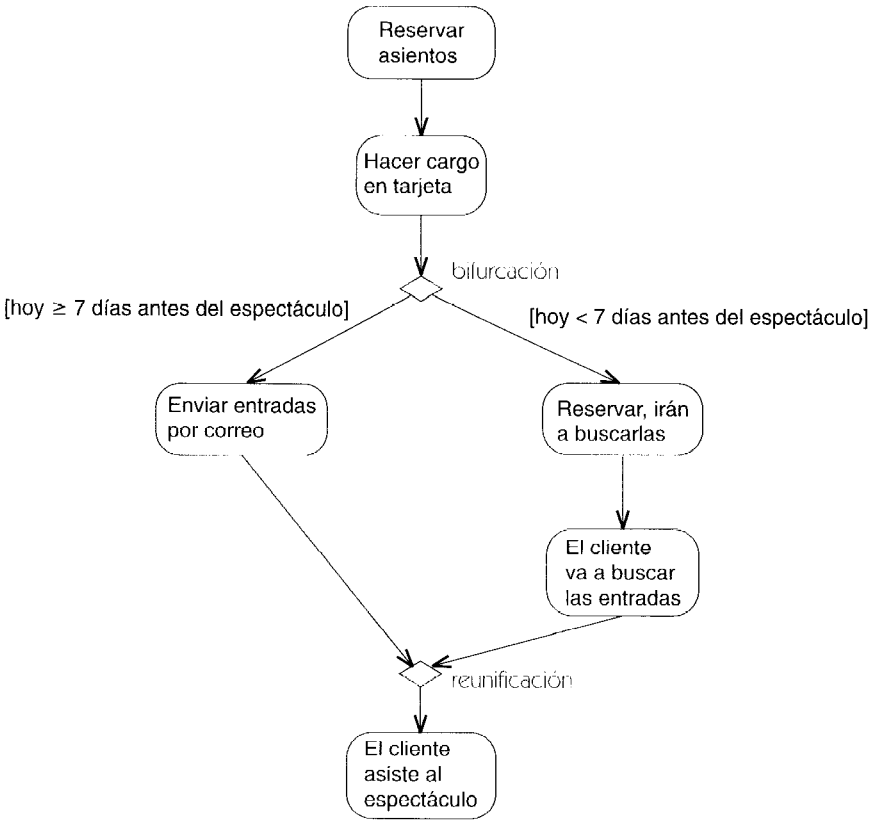


Figura 13.169 Reunificación

Una combinación de bifurcación y reunificación es válida pero tiene una utilidad limitada. Tendría múltiples transiciones de entrada y múltiples transiciones de salida con etiqueta.

Obsérvese que una reunificación es tan sólo una forma cómoda de notación y se puede omitir sin pérdida de información. Normalmente, las bifurcaciones y las fusiones se emparejan de forma anidada.

Discusión

Asegúrese de que distingue las reunificaciones de las uniones. Las reunificaciones combinan dos o más rutas alternativas de control. En toda ejecución se seguirá solamente una ruta en cada momento. No se necesita una sincronización.

La unión combina dos o más rutas concurrentes de control. En toda ejecución se seguirán todas las rutas y la unión sólo se disparará cuando todas ellas hayan llegado a los estados de origen de la unión.

reutilización

Utilización de un artefacto existente previamente.

rol

Ranura con nombre situada dentro de la estructura de un objeto que representa el comportamiento de un elemento que participa en un contexto (por oposición a las cualidades inherentes del elemento en cualquiera de sus aplicaciones). Un rol puede ser estático (como un extremo de asociación) o dinámico (como un rol de colaboración). Entre los roles de colaboración se cuentan los roles de clasificador y los roles de asociación.

Véase colaboración.

rol de asociación

Conexión de dos roles de clasificador dentro de una colaboración. Asociación entre dos clasificadores que se aplica sólo a un cierto contexto especificado por una colaboración.

Véase también asociación, colaboración.

Semántica

Un rol de asociación es una asociación que está definida y tiene significado sólo en el contexto descrito por una colaboración. Es una relación que es parte de la colaboración pero no una relación inherente en otras situaciones. Los roles de asociación son la parte estructural clave de las colaboraciones, pues permiten describir relaciones contextuales.

Dentro de una colaboración, un rol de clasificador denota una aparición individual de un clasificador, distinta de otras apariciones del clasificador y distinta también de la propia declaración del clasificador. Es un clasificador por derecho propio que representa una restricción en el uso del clasificador base basada en el contexto de una colaboración. De forma similar, un rol de asociación representa una asociación utilizada en un contexto particular, a menudo un uso restringido de una asociación normal. Un rol de asociación conecta dos roles de clasificador. Cuando se instancia una colaboración, los objetos se enlazan a roles de clasificador y los enlaces a roles de asociación. Un objeto puede desempeñar (enlazarse a) más de un rol.

Un rol de asociación conecta dos o más clasificadores o roles de clasificador dentro de una colaboración. El rol de asociación tiene una referencia a la asociación —la asociación base— y puede tener una multiplicidad que indique cuántos enlaces pueden representar el rol en una instancia de la colaboración. En algunos casos, las conexiones dentro de la colaboración pueden verse como usos de una asociación general entre las clases participantes. La colaboración muestra una forma de utilizar la asociación general para un propósito determinado dentro de la colaboración.

En otros casos, los roles de clasificador están conectados mediante asociaciones que no tienen validez fuera de la colaboración. Si un rol de asociación no tiene una asociación base explícita, entonces define una asociación implícita válida únicamente dentro de la colaboración.

Notación

Un rol de asociación se representa de la misma forma que una asociación, en concreto, mediante una línea continua entre dos símbolos de rol de clasificador (véase Figura 13.170). Es fácil distinguir que se trata de un rol de asociación pues conecta roles de clasificador.

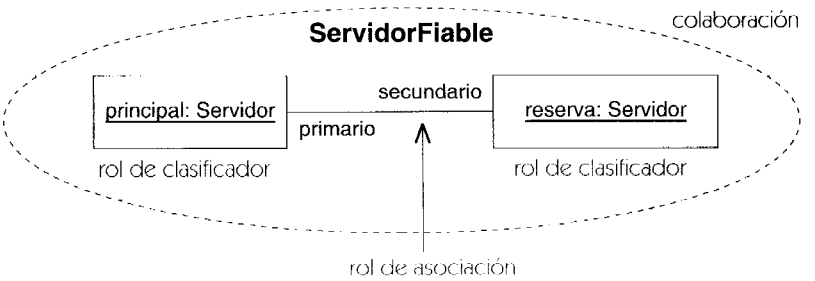


Figura 13.170 Rol de asociación

Elementos estándar

new, transient.

rol de clasificador

Ranura de una colaboración que describe el rol desempeñado por un participante en una colaboración.

Véase también colaboración.

Semántica

Una colaboración describe un patrón de interacción entre un conjunto de participantes, que son instancias de clases o de tipos de datos. Un rol de clasificador es la descripción de un participante. Cada rol es un uso distinto del clasificador en un contexto propio y único. Puede haber más de un rol para el mismo clasificador, cada uno de los cuales tendrá un conjunto distinto de relaciones con otros roles dentro de una colaboración. Sin embargo, un rol no es un objeto individual, sino una descripción de todos los objetos que pueden tomar parte en una instancia de una colaboración. En cada instancia de la colaboración, los roles serán desempeñados por diferentes objetos y enlaces.

Un rol de clasificador tiene una referencia a un clasificador (clasificador base) y una multiplicidad. El clasificador base restringe el tipo de objeto que puede desempeñar el rol de clasificador. La clase del objeto puede ser la misma o un descendiente del clasificador base. La multiplicidad indica cuántos objetos pueden desempeñar el rol a la vez en una instancia de la colaboración.

Un rol de clasificador puede tener un nombre o ser anónimo. Puede tener múltiples clasificadores base si se realiza clasificación múltiple.

Un rol de clasificador puede conectarse con otros roles de clasificador mediante roles de asociación.

Objetos. Una colaboración representa un grupo de objetos que trabajan juntos para llevar a cabo un objetivo. Un rol representa la parte de uno de los objetos (o un grupo de los objetos) que reali-

za dicho objetivo. Un objeto es una instancia directa o indirecta de la clase base de su rol. No todos los objetos de la clase base aparecen necesariamente en una colaboración de su rol, además es posible que objetos de la misma clase base desempeñen múltiples roles en la misma colaboración.

El mismo objeto puede desempeñar diferentes roles en diferentes colaboraciones. Una colaboración representa una faceta de un objeto, pudiendo combinar un mismo objeto físico diferentes facetas conectando implícitamente las colaboraciones en las que desempeña algún rol.

Notación

Un rol de clasificador se representa utilizando el símbolo de clasificador (un rectángulo) con un nombre de rol y de clasificador separados por dos puntos, esto es, nombreRol:ClaseBase. Sin embargo, un rol no es un objeto, sino un clasificador que describe muchos objetos que aparecen en diferentes instancias de la colaboración.

Se puede omitir el nombre del rol o el del clasificador, pero deben aparecer los dos puntos para distinguirlo de una clase ordinaria. Dentro de una colaboración existe un pequeño riesgo de confusión, pues todos los participantes son roles.

Un rol de clasificador puede mostrar un subconjunto de las características del clasificador, es decir, los atributos y operaciones utilizados en un contexto dado. El resto de características pueden suprimirse si no se utilizan.

La Figura 13.171 muestra diversas formas que puede tomar un rol de clasificador.

Figura 13.171 Rol de clasificador

Elementos estándar

destroyed, new, transient.

rol de colaboración

Ranura para un objeto o enlace dentro de una colaboración. Especifica el tipo de objeto o enlace que puede aparecer en una instancia de la colaboración.

Véase también colaboración, rol de asociación, rol de clasificador.

Semántica

Un rol de colaboración describe un objeto o un enlace. Sin embargo, no representa un único enlace u objeto sino un sitio que puede sustituirse por un objeto o un enlace cuando se

instancia la colaboración. Una colaboración es o bien un rol de clasificador o bien un rol de asociación. Un rol de clasificador tiene uno o más clasificadores base y puede ser instanciado como un objeto, el cual es una instancia de los clasificadores o uno de sus descendientes. Un rol de asociación puede tener una asociación base y ser instanciado como un enlace, el cual es una instancia de la asociación o uno de sus descendientes. En muchos casos, la asociación sólo está definida en la colaboración —esto es, ocurre sólo entre objetos que desempeñan los roles y no tiene significado fuera de la colaboración—. En estos casos se puede omitir la asociación base. La asociación se define implícitamente por su aparición en la colaboración, con la restricción de que no puede ser utilizada en otro sitio.

Notación

Los roles de colaboración pueden ser roles de clasificador o roles de asociación.

Rol de clasificador. Un rol de clasificador es un clasificador y se representa mediante el símbolo de las clases: un rectángulo. A menudo sólo se muestra el compartimento del nombre, que contiene la cadena:

nombreRolClasificador :NombreClasificador BaseLista

El nombre de clasificador base puede incluir la ruta de paquetes completa si es necesario (una herramienta permitirá normalmente caminos abreviados cuando no haya ambigüedad). Los nombres de paquete preceden al nombre de la clase separados por dos signos de dos puntos, por ejemplo:

mostrar_ventana:SistemaVentanas::VentanasGraficas:: Ventana

Un estereotipo para un rol de clasificador puede representarse textualmente entre comillas dobles sobre la cadena del nombre o como un icono en la esquina superior derecha. El estereotipo de un rol de clasificador debe coincidir con el estereotipo de su clasificador base.

Un rol de clasificador que representa un conjunto de objetos incluye un indicador de multiplicidad (por ejemplo un asterisco) en la esquina superior derecha del símbolo de la clase. Este indicador especifica el número de objetos que pueden ligarse al rol en una instancia de la colaboración. Cuando se omite el indicador, el valor es exactamente uno.

El nombre del rol de clasificador puede omitirse, en cuyo caso deberían mantenerse los dos puntos al lado del nombre de la clase. Este caso representa un objeto anónimo de la clase.

Si hay clasificación múltiple, el rol puede tener más de un clasificador, en ese caso, el objeto es una instancia de cada uno de ellos.

La clase del rol de clasificador puede suprimirse (junto con los dos puntos).

Un objeto o enlace que se crea durante una interacción tiene la palabra clave **new** como restricción en su rol. Un objeto o enlace destruido durante una interacción tiene la palabra clave **destroyed** como restricción en su rol. Las palabras clave pueden utilizarse incluso aunque el elemento no tenga nombre. Ambas palabras clave pueden utilizarse conjuntamente, pero resulta más útil emplear la palabra clave **transient** en lugar de la combinación **new destroyed**.

Rol de asociación. Un rol de asociación es una asociación, y se representa mediante una ruta entre dos símbolos de rol de clasificador. La ruta puede tener asociado un nombre de la forma:

nombreRolAsociación NombreAsociaciónBase

Si se omite el nombre de la asociación base, entonces no hay asociación base. En la ruta pueden mostrarse los nombres de rol y otros adornos de la asociación base.

Si un extremo de la ruta del rol de asociación está conectado a un rol de clase con una multiplicidad distinta de uno, puede situarse un indicador de multiplicidad en ese extremo de la asociación para enfatizar dicha multiplicidad.

La Figura 13.46 muestra un ejemplo de roles de colaboración.

ruta

Se trata de una serie conexa de segmentos gráficos que conectan símbolos entre sí, normalmente para mostrar una relación.

Notación

Las *rutas* son conexiones gráficas entre los símbolos de un diagrama. Las rutas se utilizan en esta notación para las relaciones de distintas clases, tales como asociaciones, generalizaciones y dependencias. Los extremos de dos segmentos conectados coinciden. Un segmento puede ser un segmento de línea recta, un arco o alguna otra forma, aun cuando muchas herramientas admiten únicamente líneas rectas y arcos (Figura 13.172). Las líneas se pueden dibujar en cualquier ángulo aunque algunos creadores de modelos prefieren limitar las líneas a ángulos rectos y posiblemente impongan una cuadrícula rectangular por razones estéticas y para facilitar el trazado. En general, el trazado de una ruta no tiene significado, aunque las rutas deberían evitar cruzar regiones cerradas, porque cruzar los límites de una región gráfica puede tener significado semántico. (Por ejemplo, una asociación entre dos clases de una colaboración debería dibujarse dentro de la región de colaboración para indicar una asociación entre objetos de la misma instancia de colaboración; sin embargo, una ruta que hiciera una salida de la región indicaría una asociación entre objetos procedentes de distintas instancias de colaboración.) Más exactamente, las rutas son topológicas. Su trazado exacto no tiene semántica, pero su conexión con otros símbolos y sus intersecciones poseen un significado. El trazado exacto de las rutas tiene gran importancia a efectos de la compresión y de la estética, por supuesto; puede connotar de forma sutil la importancia de las relaciones y otras cosas. Pero estas consideraciones son para los seres humanos y no para los computadores. Se supone que las herramientas admitirán de forma sencilla los trazados y retrazados de las rutas.

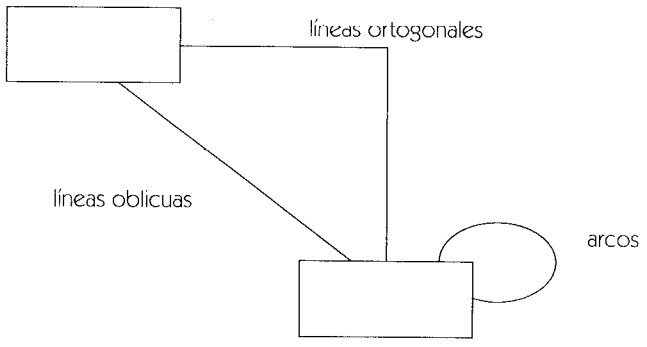


Figura 13.172 Rutas

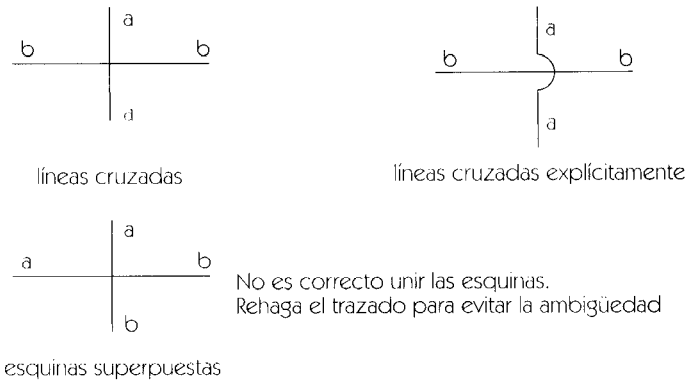


Figura 13.173 Cruces de rutas

En la mayoría de los diagramas, los cruces de líneas carecen de significación. Para evitar la ambigüedad respecto a la identidad de las líneas que se cruzan, se puede dibujar un pequeño semicírculo o discontinuidad en una de las líneas en el punto de cruce (Figura 13.173). Con mayor frecuencia, los creadores de modelos se limitan a tratar el cruce como dos líneas independientes y están de acuerdo en evitar la confusión que surgiría al hacer que dos ángulos rectos se tocaran en el vértice.

En algunas relaciones (tales como la agregación y la generalización), varias rutas del mismo tipo se pueden conectar a un mismo símbolo. Si las propiedades de los distintos elementos del modelo coinciden, entonces los segmentos de línea conectados con el símbolo se pueden combinar en un único segmento de línea, de tal modo que la ruta que sale del símbolo se ramifique formando una especie de árbol (Figura 13.174). Esto no pasa de ser una opción de

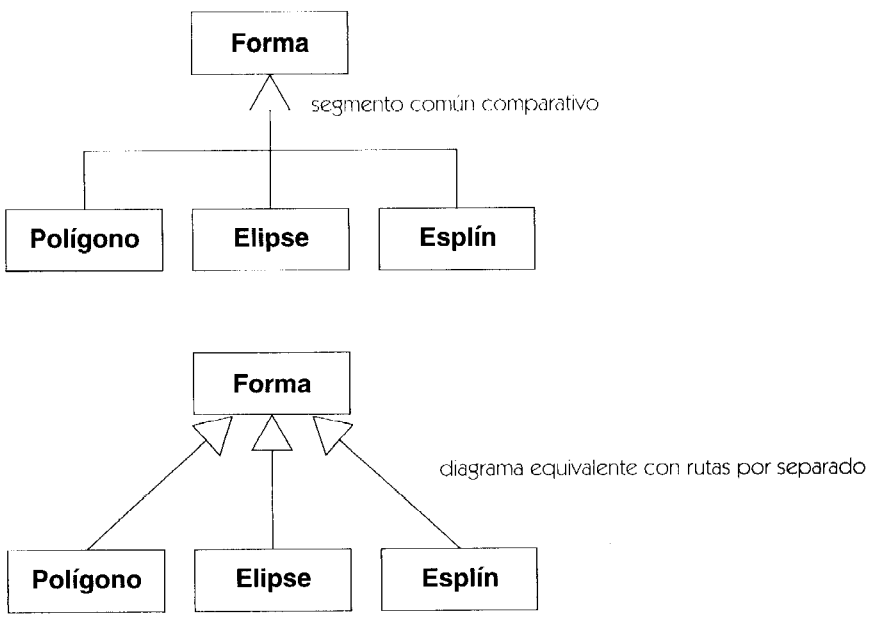


Figura 13.174 Rutas con segmento compartido

representación gráfica. Conceptualmente, las rutas individuales son diferentes. Esta opción de representación no puede utilizarse cuando la información de modelado de los diferentes segmentos no es idéntica.

secuencia de acciones

Sucesión de acciones que se ejecutan una detrás de otra. Es un tipo de acción.

Semántica

Una secuencia de acciones es una lista de acciones. Las acciones se ejecutan secuencialmente. La secuencia completa se considera una unidad atómica (es decir, no puede ser interrumpida). Una secuencia de acciones es una acción y puede por consiguiente ser asociada a transiciones o a pasos de interacción.

Notación

Una secuencia de acciones se representa mediante una cadena formada por una secuencia de cadenas separadas por signos de punto y coma.

Si las acciones se expresan en un determinado lenguaje de programación, se puede utilizar la sintaxis sobre secuencia de acciones del lenguaje en lugar de la representación descrita.

Ejemplo

```
contador:=0;reservas.borrar();send Quiosco.primeraPantalla()
```

semántica

Especificación formal del significado y comportamiento de algo.

señal

Especificación de una comunicación asíncrona entre objetos. Las señales pueden tener parámetros que se expresan como atributos.

Véase también evento, mensaje, enviar.

Semántica

Una señal es un clasificador explícito con nombre, destinado a una comunicación explícita entre objetos. Posee una lista explícita de parámetros, representados como sus atributos. Es enviada

explícitamente por un objeto a otro objeto o conjunto de objetos. La emisión general de una señal se puede considerar como enviar una señal al conjunto de todos los objetos —aunque en la práctica se implementaría de otra manera para mayor eficiencia—. El emisor especifica los argumentos de la señal en el momento en que se envía ésta. Enviar una señal es equivalente a instanciar un objeto señal y transmitir ese objeto al conjunto de objetos destino. La recepción de una señal es un evento que está destinado a disparar transiciones en la máquina de estados del receptor. Una señal que se envía a un conjunto de objetos puede disparar cero o una transiciones en cada objeto independientemente. Las señales son medios explícitos mediante los cuales los objetos se pueden comunicar entre sí asíncronamente. Para efectuar una comunicación síncrona, es preciso utilizar dos señales, una en cada dirección.

Las señales son elementos generalizables. Una señal hija se deriva de una señal padre. Hereda los atributos del padre y puede agregar sus propios atributos adicionales. Una señal hija dispara una transición que haya sido declarada para utilizar una de sus señales antecesoras.

La declaración de una señal tiene como alcance el paquete en que haya sido declarada. No está restringida a una única clase.

Una clase o una interfaz pueden declarar las señales que están dispuestas a manejar. Esta declaración es una recepción, que puede incluir la especificación de los resultados que se esperan cuando se reciba la señal. La declaración posee una propiedad que indica si es polimórfica. Si lo es, entonces una clase descendiente podrá manejar la señal, evitando quizá que la señal llegue a la clase actual. Si no es polimórfica, entonces ningún descendiente podrá interceptar el manejo de la señal.

Una señal es un clasificador y puede tener operaciones que accedan a sus atributos y los modifiquen. Además todas las señales comparten la operación implícita send.

send (conjuntoDestino)

La señal se envía a todos los objetos del conjunto destino.

Notación

La palabra reservada de estereotipo «**signal**» se pone delante de la declaración de aquellos parámetros que tengan el nombre de una señal, para indicar que hay una clase o interfaz que admite la señal. Los parámetros de la señal se incluyen en la declaración. La declaración no puede tener un tipo de retorno.

La declaración de una señal se puede expresar como un estereotipo de un símbolo de clase. La palabra reservada «**signal**» aparece en un rectángulo por encima del nombre de la señal. Los parámetros de la señal aparecen como atributos dentro del compartimento de atributos. El compartimento de operaciones puede contener operaciones de acceso.

La Figura 13.175 muestra la utilización de una notación de generalización para relacionar una señal hija con su padre. La hija hereda los parámetros de sus antecesores y puede agregar sus propios parámetros adicionales. Por ejemplo, la señal **BotonRatonPulsado** tiene los atributos **tiempo**, **dispositivo** y **localización**.

Para utilizar una señal como disparador de una transición, utilice la sintaxis:

nombre-evento ($\text{parámetros}_{\text{lista}}$)

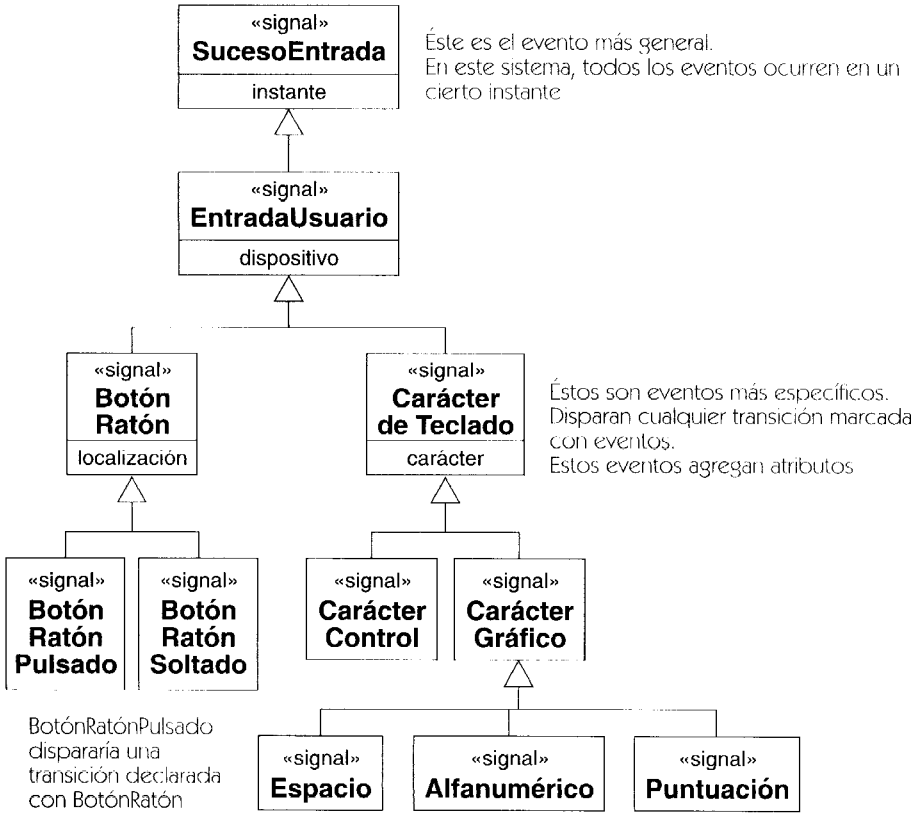


Figura 13.175 Declaraciones de señal

Los parámetros tienen la sintaxis:

nombre-parámetro :expresión-tipo

Los parámetros de una señal se declaran como atributos con valor inicial, que se puede anular durante la iniciación o al enviar. El valor inicial se usa si se crea una instancia de señal, se inicia y después se envía a un objeto. Si se envía una señal empleando una sintaxis de llamada a operación, entonces los valores iniciales son los valores por defecto de los parámetros de la señal.

Discusión

Una señal es la más fundamental comunicación entre objetos; posee una semántica más sencilla y más clara que las llamadas a procedimientos. Una señal es inherentemente a una comunicación asíncrona unidireccional que va de un objeto a otro y en la cual se pasa por valor toda la información. Es un modelo adecuado para la comunicación en sistemas concurrentes y distribuidos.

Para construir una comunicación asíncrona, utilice parejas de señales (una en cada dirección). Una llamada se puede ver como una señal con un parámetro implícito que es un puntero de retorno.

signatura

Nombre y propiedades de los parámetros de una característica de comportamiento, tal como una señal u operación. La signatura puede incluir tipos de retorno opcionales (para las operaciones, no para las señales).

Semántica

La signatura de una operación forma parte de su declaración. Parte de la signatura (pero no toda) se emplea para verificar la equivalencia de operaciones y métodos, con objeto de evitar conflictos o anulaciones. Para este propósito, se incluye el nombre de la operación y la lista ordenada de los tipos de los parámetros, pero no sus nombres ni direcciones y se excluyen los parámetros de retorno. Si coinciden dos signaturas pero las propiedades restantes son inconsistentes (por ejemplo, si un parámetro de entrada corresponde a uno de salida) entonces las declaraciones están en conflicto y el modelo está mal formado.

sistema

Colección de unidades conectadas que se organiza para lograr un propósito. Un sistema puede estar descrito por uno o más modelos, posiblemente desde distintos puntos de vista. El sistema es el "modelo completo".

Semántica

El sistema se modela mediante un subsistema de nivel máximo que contiene indirectamente todo el conjunto de elementos del modelo que funcionan como un todo para lograr algún propósito del mundo real.

Notación

Se puede mostrar el sistema como un paquete con el estereotipo «**system**». Sin embargo, no suele ser necesario mostrar el sistema como una unidad.

solicitud

Especificación de un estímulo que se envía a las instancias. Puede ser la llamada a una operación o bien el envío de una señal.

subclase

Hija de otra clase en una relación de generalización —esto es, la descripción más específica—. La clase hija se denomina subclase. La clase padre se denomina superclase.

Véase generalización, herencia.

Semántica

Una subclase hereda la estructura, relaciones y comportamiento de su superclase; además puede hacer sus propias adiciones.

subestado

Estado que forma parte de un estado compuesto.

Véase estado compuesto, subestado concurrente, subestado disjunto.

subestado concurrente

Un subestado que se puede llevar a cabo simultáneamente con otros subestados contenidos en el mismo estado compuesto.

Véase estado compuesto, subestado disjunto.

subestado disjunto

Un subestado que no se puede llevar a cabo simultáneamente con otros subestados contenidos en el mismo estado compuesto. Contraste: subestado concurrente.

Véase transición compleja, estado compuesto.

subestado ortogonal

Se trata de uno elemento de un conjunto de estados que descomponen un estado compuesto en subestados, todos los cuales están activados concurrentemente.

Véase estado compuesto, subestado concurrente.

submáquina

Máquina de estados que se puede invocar como parte de otra máquina de estados. Posee una semántica tal como si se hubiera duplicado su contenido para luego insertarlo en el estado al que hace referencia.

Véase estado, máquina de estados, estado de referencia a submáquina.

subsistema

Paquete de elementos que se tratan como una unidad, incluyendo una especificación de todo el contenido del paquete, que se trata como una unidad coherente. Un subsistema se modela

simultáneamente como paquete y como clase. Los subsistemas tienen un conjunto de interfaces que describen su relación con el resto del sistema y las circunstancias en que se puede utilizar.

Véase también interfaz, paquete, realización.

Semántica

Un subsistema es una pieza coherente de un sistema que se puede tratar como unidad abstracta. Representa el comportamiento emergente de una parte del sistema. Como unidad, posee su propia especificación de comportamiento y su parte de implementación. La especificación de comportamiento define su comportamiento emergente como unidad que puede interaccionar con otros subsistemas. La especificación de comportamiento se da en términos de caso de uso y otros elementos de comportamiento. La parte de implementación describe la implementación del comportamiento en términos de los elementos subordinados de que consta su contenido y se da como un conjunto de colaboraciones entre los elementos contenidos.

El sistema en sí constituye el subsistema de más alto nivel. La implementación de un subsistema se puede escribir como una colaboración de subsistemas de nivel inferior. De esta manera, se puede expandir todo el sistema como un jerarquía de subsistemas hasta definir los subsistemas del más bajo nivel en términos de clases ordinarias.

Los subsistemas pueden incluir elementos estructurales y elementos de especificación, tales como casos de uso y operaciones exportadas por el subsistema. Las especificaciones del subsistema se implementan mediante elementos estructurales. El comportamiento del subsistema es el comportamiento de los elementos que contiene.

Estructura

La especificación de un subsistema abarca los elementos indicados como elementos de especificación, junto con las operaciones e interfaces que se hayan definido para el subsistema como un todo. Entre los elementos de especificación se cuentan los casos de uso, las restricciones, las relaciones entre casos de uso y demás. Estos elementos y operaciones definen el comportamiento del subsistema como entidad emergente, que es el resultado neto de la actuación conjunta de sus partes. Los casos de uso especifican secuencias completas de interacciones del subsistema con actores externos. Las interfaces especifican operaciones que deban proporcionar el subsistema o sus casos de uso. La especificación no revela cuáles son las partes ni cómo interaccionan esas partes para realizar el comportamiento necesario.

El resto de los elementos del subsistema se encarga de dar vida a su comportamiento. Esto puede incluir distintas clases de clasificadores y las relaciones existentes entre ellos. Hay un conjunto de colaboraciones entre elementos de realización del subsistema que se encarga de dar vida a las especificaciones. En general, cada caso de uso requiere una o más colaboraciones. Cada colaboración describe la forma en que cooperan las instancias de los elementos de implementación para realizar conjuntamente el comportamiento especificado por un caso de uso u operación. Todos los mensajes entrantes y salientes en el subsistema en el nivel de especificación tienen que corresponderse con mensajes entre sus elementos de implementación y los de otros subsistemas.

Un subsistema es un paquete y tiene las propiedades de un paquete. En particular, la importación de subsistemas funciona tal como se ha descrito para los paquetes y la generalización entre subsistemas tiene las mismas consecuencias para la visibilidad de su contenido.

Notación

Los subsistemas se denotan con un símbolo de paquete (un rectángulo con una pequeña ficha) que contiene la palabra reservada «**subsystem**» por encima del nombre del subsistema (Figura 13.176).

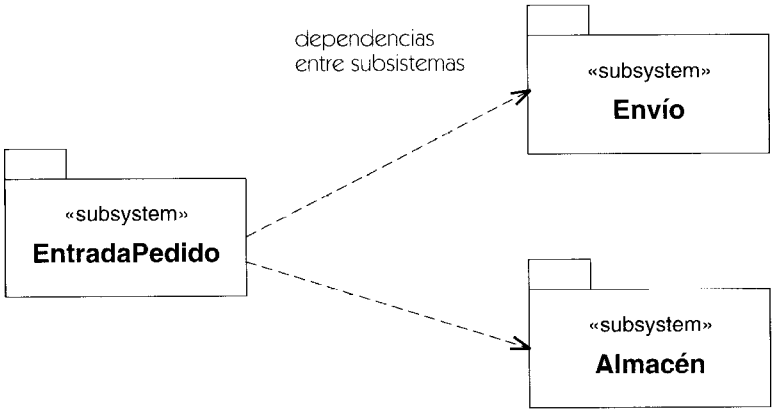


Figura 13.176 Subsistemas

Discusión

Un subsistema es un grupo emergente de elementos de diseño, tales como clases lógicas. Un componente es un grupo emergente de elementos de implementación, tales como clases del nivel de implementación. En muchos casos, los subsistemas se implementan como componentes. Esto simplifica la correspondencia entre diseño e implementación; por tanto, se trata de un enfoque corriente para la arquitectura. Además, muchos componentes se implementan como clases dominantes que implementan directamente las interfaces de los componentes. En este caso, un subsistema, un componente y una clase podrían tener todos ellos la misma interfaz.

subtipo

Tipo que es hijo de otro tipo. Se puede utilizar el término hijo, más neutro, para cualquier elemento generalizable

Véase generalización.

superclase

Padre de otra clase en una relación de generalización, esto es, la especificación de elemento más general. La clase hija se denomina subclase. La clase padre se denomina superclase.

Véase generalización.

Semántica

Las subclases heredan la estructura, relaciones y comportamiento de sus superclases y pueden hace adiciones.

supertipo

Sinónimo de superclase. Se puede emplear el término más neutro, padre, para cualquier elemento generalizable.

Véase generalización.

supratipo

Denota una metaclase cuyas instancias son subclases de una clase dada.

Véase también metaclase.

Semántica

Las subclases de una clase dada pueden considerarse a su vez instancias de una metaclase. Esta metaclase es lo que se denomina un supratipo. Por ejemplo, la clase **Árbol** puede tener las subclases **Roble**, **Olmo** y **Sauce**. Consideradas como objetos, estas subclases son instancias de la metaclase **EspecieDeÁrbol**. Entonces **EspecieDeÁrbol** es un supratipo de nivel superior a **Árbol**.

Notación

Un supratipo se representa mediante una clase con el estereotipo «**powertype**». Está conectado a un conjunto de rutas de generalización mediante una flecha discontinua que tiene el estereotipo «**powertype**» asociado a la flecha. (Figura 13.177).

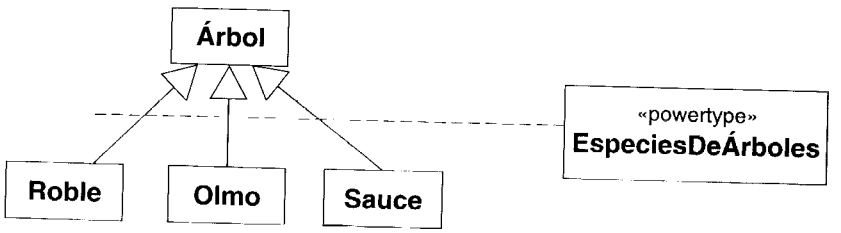


Figura 13.177 Supratipo

tiempo

Valor que representa un instante de tiempo absoluto o relativo.

Véase expresión temporal.

tiempo de análisis

Tiempo durante el cual se realiza la actividad de análisis en un proceso de desarrollo de software. No suponga que todo el análisis de un sistema se realiza al mismo tiempo o que precede a otras actividades como el diseño o la implementación, ya que las diversas actividades pueden ser secuenciales para un determinado elemento, pero cuando se trata del sistema completo las actividades se entremezclan.

Véase tiempo de diseño, tiempo de modelado.

tiempo de compilación

Se refiere a algo que ocurre durante la compilación de un módulo de software.

Véase tiempo de ejecución, tiempo de modelado.

tiempo de ejecución

Período de tiempo durante el cual se ejecuta un programa de computador. Contrastar con: tiempo de modelado.

tiempo de modelado

Denota todo aquello que se produce durante una actividad de modelado correspondiente al proceso de desarrollo del software. Se incluye el análisis y el diseño. Nota de utilización: cuando se discuten sistemas de objetos, suele ser importante distinguir entre los problemas del tiempo de modelado y del tiempo de ejecución.

Véase también proceso de desarrollo, fases de modelado.

tiempo de transición

Véase marca de tiempo.

tiempo de diseño

Se refiere a lo que ocurre durante la actividad de diseño del proceso de desarrollo de software. Contrastar con: tiempo de análisis.

Véase tiempo de modelado, fases de modelado.

tipo

Como adjetivo: El clasificador declarado que tienen que contener el valor de un atributo, parámetro o variable. El valor real tiene que ser una instancia del tipo o de uno de sus descendientes.

Como sustantivo: Estereotipo de una clase que se emplea para especificar un conjunto de instancias (tales como objetos) junto con las operaciones aplicables a esos objetos. Un tipo no puede contener métodos. Contrastar con: interfaz, clase de implementación.

Tipo y clase de implementación

Las clases se pueden estereotipar como tipos o clases de implementación (aunque también pueden quedar sin ser diferenciadas). Los tipos se emplean para especificar un dominio de objetos, junto con las parámetros aplicables a los objetos, sin definir la implementación física de los objetos o de sus operaciones. Los tipos no pueden contener métodos pero pueden proporcionar especificaciones de comportamiento para sus operaciones. También pueden incluir atributos y asociaciones para especificar el comportamiento de sus operaciones. Los atributos y asociaciones de un tipo no determinan la implementación de sus objetos.

Una clase de implementación define la estructura física de los datos y métodos de sus objetos, tal como se implementan en lenguajes tradicionales como C++ y SmallTalk. Se dice que la clase de implementación realiza un tipo si la clase de implementación incluye todas las operaciones del tipo con el mismo comportamiento. Una clase de implementación puede realizar múltiples tipos y múltiples clases de implementación pueden realizar un mismo tipo. Los atributos y asociaciones de una clase de implementación no tienen por qué ser los mismos que los de un tipo que realice. La clase de implementación puede proporcionar métodos para sus operaciones en términos de sus atributos y asociaciones físicas y puede declarar operaciones adicionales que no se encuentren en ningún tipo.

Un objeto sólo puede ser instancia de una clase de implementación, que especifica la implementación física del objeto. Sin embargo, un objeto puede ser instancia de múltiples tipos. Si el objeto es una instancia de una clase de implementación, entonces la clase de implementación tiene que realizar los tipos de los cuales sea instancia el objeto. Si se dispone de clasificación dinámica, entonces un objeto puede ganar y perder tipos a lo largo de su ciclo de vida. Los tipos que se emplean de este modo caracterizan el rol cambiante que puede adoptar un objeto para luego abandonarlo.

Aun cuando el uso de los tipos y de las clases de implementación es diferente, su estructura interna es la misma, así que se modelan como estereotipos de clases. Admiten la generalización en toda su extensión, así como el principio de capacidad de sustitución y la herencia de atributos, asociaciones y operaciones. Sin embargo, los tipos sólo pueden especializar a otros tipos y las clases de implementación sólo pueden especializar a otras clases de implementación. Los tipos sólo pueden estar relacionados con clases de implementación por realización.

Notación

Las clases sin diferenciar se muestran sin palabra reservada. Los tipos se muestran con la palabra clave «**type**» y las clases de implementación se muestran con la palabra clave «**clase de implementación**». La Figura 13.178 muestra un ejemplo.

La implementación de un tipo por parte de una clase de implementación se modela empleando la relación de realización, que se denota mediante una línea discontinua con una punta de flecha triangular rellena (una "flecha discontinua de generalización" o una "dependencia con

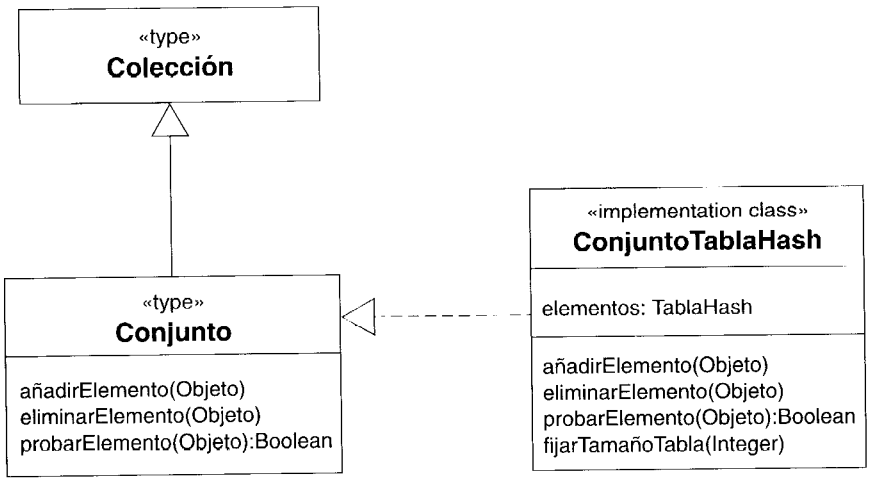


Figura 13.178 Notación para los tipos y clases de implementación

punta de flecha rellena"). Este símbolo implica la herencia de operaciones pero no de estructura (atributos o asociaciones). La realización puede emplearse entre dos clasificadores cualesquiera para los cuales uno admita las operaciones del otro, pero la forma habitual en que se usa es la implementación de un tipo por parte de una clase de implementación.

Discusión

Los tipos y las clases de implementación son conceptos relativamente limitados, destinados a los lenguajes de programación convencionales como C++ y SmallTalk. El concepto de clase de UML se puede emplear directamente de una forma más general. UML admite tanto la clasificación múltiple como la clasificación dinámica, lo cual elimina la necesidad de distinguir entre tipos y clases de implementación. Las diferencias, sin embargo, son útiles para diseñar código destinado a lenguajes convencionales.

tipo de dato

Un descriptor de un conjunto de valores que carecen de identidad (existencia independiente y la posibilidad de efectos secundarios). Los tipos de datos incluyen tipos primitivos predefinidos y tipos definibles por el usuario. Los tipos primitivos son números, cadenas, y tiempo.

Los definidos por el usuario son enumeraciones. Los tipos anónimos de datos previstos para la implementación en un lenguaje de programación se pueden definir usando tipos del lenguaje.

Véase también clasificador, identidad.

Semántica

Los tipos de datos son los primitivos predefinidos que se necesitan como base de los tipos definidos por el usuario. Su semántica se define matemáticamente fuera de los mecanismos de

construcción de tipos de un lenguaje. Los números están predefinidos. Incluyen números enteros y reales. También están predefinidas las cadenas. Estos tipos de datos no son definidos por el usuario.

Los tipos enumeración son conjuntos finitos, definidos por el usuario, de elementos nombrados que tienen un orden definido entre sí mismos y ninguna otra propiedad computacional. Un tipo enumeración tiene un nombre y una lista de las constantes de la enumeración. El tipo **booleano** se define con los literales de la enumeración **false** y **true**.

Las operaciones se pueden definir en los tipos de datos, y las operaciones pueden tener tipos de datos como parámetros. Debido a que un tipo de datos no tiene ninguna identidad y es sólo un valor, las operaciones en los tipos de datos no los modifican; en su lugar, simplemente vuelven valores. No tiene ningún sentido hablar de crear un nuevo valor del tipo de datos, porque carece de identidad. Todos los valores del tipo de datos se encuentran (conceptualmente) predefinidos. Una operación en un tipo de datos es una consulta que puede no cambiar el estado del sistema pero puede retornar un valor.

Un tipo de dato se puede también describir por un tipo del lenguaje —una expresión de un tipo de datos en un lenguaje de programación—. Dicha expresión señala un tipo de dato anónimo en el lenguaje en cuestión. Por ejemplo, la expresión **Persona*** (*) (**String**) denota una expresión de tipo en C++ que no se corresponde a ningún tipo simple de datos con un nombre.

tipo del lenguaje

Dícese de un tipo de datos anónimo que se define en la sintaxis de un lenguaje de programación.

Véase también tipo de dato.

Semántica

Un tipo del lenguaje es una expresión que debe ser interpretada como un tipo de dato de algún lenguaje de programación. Se puede emplear como tipo de un atributo, variable o parámetro. No posee nombre y no declara un nuevo tipo de dato.

Por ejemplo, el tipo de dato de C++ "**Persona*** (*)(**Contrato***,**int**)" se podría definir como un tipo del lenguaje C++.

La intención de un tipo del lenguaje es la implementación en un determinado lenguaje de programación. Para relaciones más lógicas deben emplearse asociaciones.

tipo primitivo

Este término denota un tipo de dato básico predefinido, tal como un entero o una cadena.

Véase también enumeración.

Semántica

Las instancias de los tipos primitivos carecen de identidad. Si dos instancias tienen la misma representación, entonces son indistinguibles y se pueden pasar por valor sin pérdida de información.

Entre los tipos primitivos se cuentan los números y las cadenas, así como posiblemente otros tipos de datos dependientes del sistema tales como fechas y valores monetarios, cuya semántica está predefinida fuera de UML.

Se supone que los tipos primitivos tendrán una fuerte correspondencia con los que se hallan en el lenguaje de programación que se tenga como destino.

Véase también enumeraciones, que son tipos de datos definibles por el usuario y que no son tipos primitivos predefinidos.

transición

Relación entre dos estados dentro de una máquina de estados que indica que un objeto del primer estado va a ejecutar las acciones especificadas para después pasar al segundo estado cuando se produzca un evento especificado y se cumplan las condiciones de guarda. En este cambio de estado se dice que se dispara la transición. Las transiciones simples poseen un único estado de origen y un único estado blanco. Una transición compleja posee más de un origen y/o más de un estado destino. Representa un cambio en el número de estados activos concurrentemente o una bifurcación o reunificación de control. Las transiciones internas poseen estado de origen pero no estado destino. Representan la respuesta a un evento sin cambio de estado. Los estados y las transiciones son los vértices y nodos de las máquinas de estados.

Véase también máquina de estados.

Semántica

Las transiciones representan rutas potenciales entre estados dentro de la historia de los objetos, así como las acciones que se llevan a cabo al cambiar de estado. Una transición indica la forma en que responde a un evento un objeto que está en un cierto estado. Los estados y las transiciones son los vértices y arcos de una máquina de estados que describe las posibles historias de las instancias de un clasificador.

Estructura

Toda transición puede poseer un estado de origen, un evento disparador, una condición de guarda, una acción y un estado destino. Las transiciones pueden carecer de alguno de estos elementos.

Estado de origen. El estado de origen es el estado que se ve afectado por la transición. Si un objeto se encuentra en el estado de origen, se puede disparar una transición saliente si el objeto recibe el evento disparador de la transición y se cumple la condición de guarda (si existe).

Estado destino. El estado destino es el estado que está activo una vez finalizada la transición. Es el estado al que pasa el objeto maestro. El estado destino no se usa en las transiciones internas, que no llevan a cabo cambios de estado.

Evento disparador. El evento disparador es el evento cuya recepción por parte del objeto cuando se encuentra en el estado de origen hace que la transición pueda dispararse, siempre y cuando se cumpla la condición de guarda. Si el evento posee parámetros, sus valores estarán disponibles para la transición y se pueden utilizar en expresiones para la condición de guarda y en acciones. El evento que dispara una transición pasa a ser el evento actual y es posible acceder a él desde las acciones subsiguientes que forman parte de la ejecución hasta finalizar iniciada por ese evento.

Una transición que carece de un evento disparador explícito es lo que se denomina una transición de finalización (o transición sin disparador) y se dispara implícitamente al finalizar cualquier actividad interna del estado. Si un estado no posee actividad interna o estados anidados, entonces cuando se entra en el estado después de haber ejecutado cualquier acción de entrada se dispara inmediatamente una transición de finalización. Observe que una transición de finalización tiene que satisfacer su condición de guarda para poder dispararse. Si la condición de guarda es falsa al producirse la terminación, entonces el evento implícito de finalización se consume y la transición no se disparará más tarde aun cuando la condición de guarda pase a ser verdadera. (Este tipo de comportamiento se puede modelar, de otro modo, empleando un evento de cambio.)

Observe que todas las apariciones de un evento dentro de una máquina de estados tienen que tener la misma signatura.

Condición de guarda. La condición de guarda es una expresión booleana que se evalúa cuando se dispara una transición por haber manejado un evento, incluyendo los eventos implícitos de finalización que se producen en una transición de finalización. Si la máquina de estados está realizando un paso que se ejecuta hasta finalizar cuando se produce un evento, entonces se guarda el evento hasta que el paso ha finalizado y la máquina de estados está en reposo. En caso, contrario, se maneja el suceso inmediatamente. Si la expresión produce el valor verdadero al ser evaluada, la transición puede dispararse. Si la expresión toma el valor falso, entonces la transición no se dispara. Si no hay ninguna expresión que pueda dispararse, se ignora el evento. Esto no es un error. Es posible que múltiples transiciones que posean distintas condiciones de guarda sean disparadas por el mismo evento. Si se produce el evento, entonces se comprueban todas las condiciones de guarda. Si hay más de una condición de guarda que sea verdadera, sólo se dispara una transición. Si no se da una regla de prioridad, la elección de la transición que se dispara no es determinista.

Observe que la condición de guarda sólo se evalúa una vez, en el momento en que se maneja el evento. Si la condición produce el valor falso al ser evaluada y después se vuelve verdadera, la transición no se disparará mientras no se produzca otro evento y la condición sea verdadera en ese momento. Observe que una condición de guarda no es la forma correcta de monitorizar continuamente un valor. Para una situación como ésta debería emplearse un evento de cambio.

Si una transición no posee una condición de guarda, entonces la condición de guarda se trata como si fuera verdadera y la transición se activa si se produce un evento disparador. Si están habilitadas varias transiciones, sólo se dispara una de ellas. La selección puede no ser determinista.

Por comodidad, se puede descomponer una condición de guarda en una serie de condiciones de guarda más sencillas. De hecho, se pueden ramificar varias condiciones de guarda a partir de un único evento disparador o condición de guarda. Toda ruta que pase por el árbol representa una única transición disparada por el (único) evento disparador con una condición de guarda efectiva diferente, que es la conjunción ("*and*") de las condiciones de guarda existentes a lo largo de su trayectoria. Todas las expresiones que haya a lo largo de la trayectoria se evaluarán antes de seleccionar la transición que se dispara. Una transición no puede dispararse parcialmente. De hecho, un conjunto de transiciones independientes puede compartir una parte de su descripción. La Figura 13.179 muestra un ejemplo.

Observe que los árboles de condiciones de guarda y la capacidad para ordenar transiciones que se pueden disparar son únicamente una cuestión de comodidad, porque se podría conseguir el mismo efecto mediante un conjunto de transiciones independientes, dotando a cada una de ellas de su propia condición de guarda disjunta.

Acción. Una transición puede contener una expresión que describe una acción. Se trata de una expresión para una computación procedimental que puede afectar al objeto que posee la máquina de estados (e indirectamente a otros objetos a los que tenga acceso). Esta expresión puede hacer uso de los parámetros del evento disparador, así como de los atributos y asociación del objeto que la posee. El evento disparador está disponible como evento actual durante todo el paso ejecutado hasta finalizar que ha sido iniciado por dicho evento, incluyendo los segmentos sin disparador y las acciones de entrada y de salida posteriores.

Una acción puede ser una secuencia de acción. Las acciones son atómicas, esto es, no se puede imponer su terminación externamente y tienen que ser ejecutadas por completo antes de que sea posible manejar ninguna otra acción o evento. Las acciones deberían tener una duración mínima, para evitar bloquear la máquina de estados. Los eventos que se reciben durante la ejecución de una acción se guardan hasta que finaliza la acción, momento en que son evaluados.

Ramificaciones. Por comodidad, varias transiciones que compartan el mismo evento disparador pero tengan diferentes condiciones de guarda se pueden agrupar en el modelo y en la notación para evitar la duplicación del disparador o de la parte común de la condición de guarda.

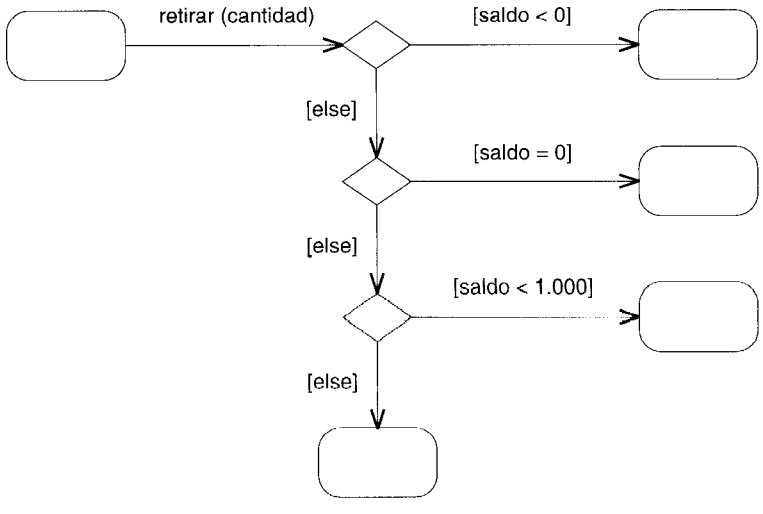


Figura 13.179 Árbol de condiciones de guarda

Esto no pasa de ser una forma cómoda de representación y no afecta a la semántica de las transiciones.

Véase bifurcación para más detalles de la representación y de la notación.

Notación

Las transiciones se representan mediante una flecha continua que va desde un estado (el de *origen*) a otro estado (el estado *destino*), etiquetadas con una cadena de transición. La Figura 13.180 muestra una transición entre dos estados y una que se ha fragmentado en segmentos.

Las transiciones internas se representan mediante una cadena de transición situada dentro de un símbolo de estado. Las cadenas de transición tienen el formato siguiente:

$\overline{\text{nombre :}}_{\text{opt}}\ \text{nombre-evento}_{\text{opt}}\ \overline{\text{(lista-parámetros)}}_{\text{opt}}\ \overline{\text{[condición-guarda]}}_{\text{opt}}$

$\overline{\text{/ lista-acciones}}_{\text{opt}}$

El *nombre* se puede utilizar para hacer referencia a la transición en las expresiones, especialmente para formar marcas de tiempo. Va seguido por dos puntos.

El *nombre-evento* da nombre a un evento y va seguido por sus parámetros. La lista de parámetros se puede omitir si no hay parámetros. El nombre del evento y la lista de parámetros se omiten en las transiciones de finalización. La lista-parámetros posee el formato siguiente:

$\overline{\text{nombre :tipo}}_{\text{lista,}}$

La *condición-guarda* es una expresión booleana escrita en términos de los parámetros del evento disparador y de los atributos y enlaces del objeto que describe la máquina de estados. La condición de guarda también puede implicar comprobaciones del estado de estados concurrentes de la máquina actual o estados indicados especialmente de algún objeto que se pueda alcanzar; entre los ejemplos se cuentan [**in** estado1] y [**not in** estado2]. Los nombres de estados pueden estar completamente calificados con los nombres de los estados anidados que los contienen, lo cual da lugar a nombres de ruta de la forma Estado1::Estado2::Estado3. Esto se puede utilizar si aparece el mismo nombre de estado en distintas regiones de estados compuestos dentro de la máquina global.

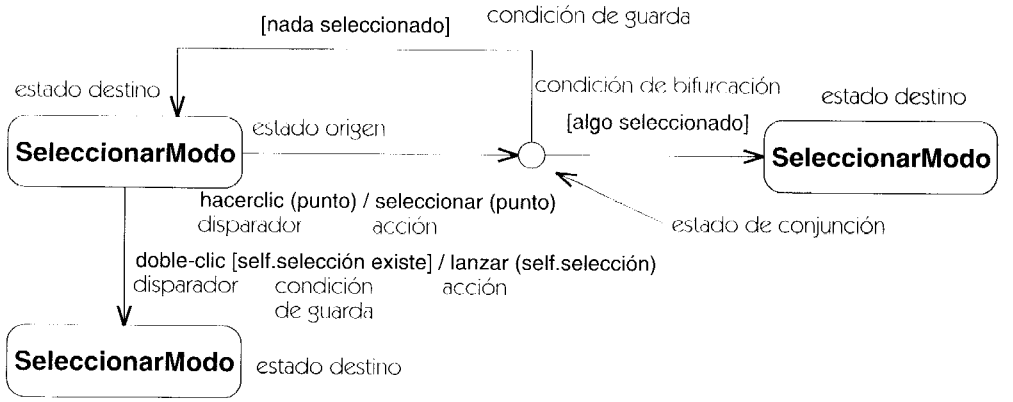


Figura 13.180 Transiciones

La *lista-acciones* es una expresión procedimental que se ejecuta siempre y cuando llegue a dispararse la transición. Puede estar escrita en términos de operaciones, atributos y enlaces del objeto poseedor y también empleando los parámetros del evento disparador. Entre las acciones se pueden incluir llamadas, envíos y otras clases de acciones. La lista-acciones puede contener más de una cláusula de acción separada por delimitadores formados por el signo de punto y coma:

acción$_{lista;}$

Ramificaciones. Una transición puede incluir un segmento con un evento disparador seguido por un árbol de estados de unión, que se dibujan como circulitos. Esto es equivalente a un conjunto de transiciones individuales, una para cada posible ruta a través del árbol, cuya condición de guarda es el "and" de todas las condiciones que se vayan encontrando a lo largo de la ruta. Sólo puede tener una acción el segmento final de la ruta.

Alternativamente, se puede dibujar un estado de unión como un rombo, si representa una ramificación o una reunificación. No hay diferencia de significados.

Discusión

Las transiciones representan un cambio atómico de un estado a otro, acompañado posiblemente por una acción atómica. Las transiciones no se pueden interrumpir. Las acciones de la transición deberían ser computaciones cortas, normalmente triviales, tales como sentencias de asignación y sencillos cálculos aritméticos.

transición abreviada

Notación que indica que una transición pasa directamente a un estado compuesto, pero se suprimen los detalles.

Véase también estado abreviado externo.

Notación

Una *abreviatura de estado* se representa mediante una pequeña línea vertical que se dibuja dentro de los límites del estado que la contiene (Figura 13.181). La abreviatura puede estar etiquetada con el nombre del estado, pero es frecuente que se omita cuando se suprimen los detalles. Indica que una transición está conectada a un estado interno suprimido. Las abreviaturas no se utilizan para transiciones a estados iniciales ni para salir de estados finales. Una abreviatura muestra que hay subestados adicionales en el modelo, pero faltan en el diagrama.

Un estado abreviado externo dentro de un estado de referencia a submáquina (Figura 13.91) hace referencia a un estado perteneciente a la definición de submáquina correspondiente (Figura 13.92). No se trata de un caso en que se hayan suprimido los detalles y es imprescindible incluir el nombre de la abreviatura.

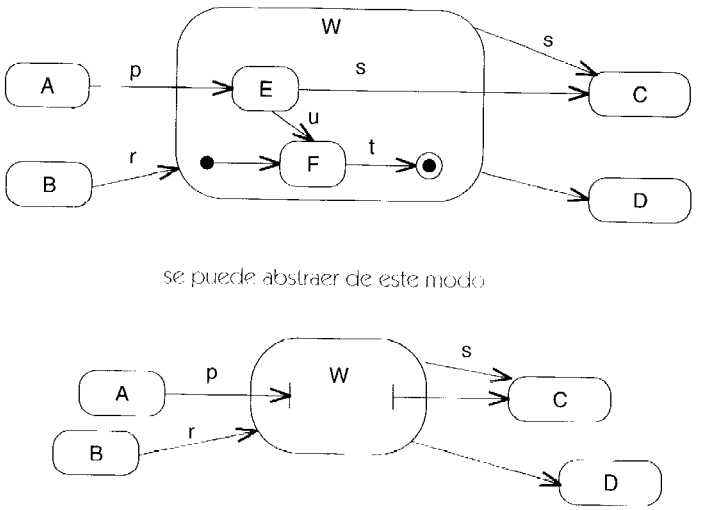


Figura 13.181 Transición abreviada

transición compleja

Transición con más de un estado origen y/o más de un estado destino. Representa una respuesta a un evento que causa un cambio en la cantidad de concurrencia. Es una sincronización de control, una división del control, o ambas, dependiendo del número de orígenes y destinos.

Véase también bifurcación, división, estado compuesto, reunificación, unión.

Semántica

A un alto nivel, un sistema pasa por una serie de estados, pero la visión monolítica de un sistema que tiene un único estado es demasiado restrictiva para grandes sistemas en los que interviene la distribución y la concurrencia. Un sistema puede estar en muchos estados simultáneamente. El conjunto de estados activos se denomina configuración del estado activo. Si está activo un estado anidado, el estado que lo contiene también está activo. Si el objeto permite concurrencia, entonces puede haber más de un subestado concurrente activo.

En muchos casos la actividad de un sistema puede modelarse como un conjunto de hilos de control que evolucionan independientemente unos de otros o que interactúan de forma limitada. Cada transición afecta, al menos, a unos pocos estados de la configuración del estado activo. Cuando se dispara una transición, los estados no afectados por la misma permanecen activos. El progreso de los hilos en un momento determinado se puede capturar como un subconjunto de estados dentro de la configuración del estado activo, de forma que exista un subconjunto por cada hilo. Cada conjunto de estados evoluciona de forma independiente en respuesta a eventos. Si el número de estados activos es constante, el modelo de estados no es sino una colección fija de máquinas de estados que interactúan. Sin embargo, en general el número de estados (y por tanto el número de hilos de control) puede variar a lo largo del tiempo. Se puede pasar de un estado a dos o más estados concurrentes (división del control), y se puede pasar de dos o más estados concurrentes a un único estado (unión de control). La máquina de estados del sistema controla el número de estados concurrentes y su evolución.

Una transición compleja es una transición hacia o desde un conjunto de subestados. Una transición compleja tiene más de un estado origen y/o más de un estado destino. Si tiene múltiples estados origen representa una unión de control, mientras que si tiene múltiples estados destino representa una división del control; si cuenta con múltiples estados origen y destino representa una sincronización de hilos paralelos.

Si una transición compleja tiene múltiples estados origen, todos ellos deben estar activos antes de que la transición sea candidata a ser disparada, siendo irrelevante el orden en que llegan a estar activos. Si todos los estados origen están activos y se produce el evento, se dispara la transición y puede ejecutarse si su condición de guarda es verdadera. Cada transición se dispara a partir de un único evento, incluso aunque haya múltiples estados origen, pues en UML no hay ocurrencia simultánea de eventos: cada evento debe disparar una transición distinta, aunque a los estados resultantes puede suceder una unión de estados.

Si una transición compleja con múltiples estados carece de un evento que la dispare (es decir, si es una transición de finalización) entonces se dispara cuando todos sus estados origen explícitos estén activos. Se lleva a cabo si en ese momento su condición de guarda es verdadera.

Cuando se lleva a cabo una transición compleja, todos los estados origen y todos sus iguales dentro del mismo estado compuesto dejan de estar activos y pasan a estarlo todos los estados destino y sus iguales.

En situaciones más complicadas, puede expandirse la condición de guarda para permitir la ejecución de la transición cuando esté activo algún subconjunto de los estados.

Ejemplo

La Figura 13.182 muestra un típico estado compuesto concurrente con transiciones complejas de entrada y de salida. La Figura 13.183 muestra una típica historia de ejecución de esta máquina (los estados activos se representan mediante letra en azul). La historia muestra la variación en el número de estados activos a lo largo del tiempo.

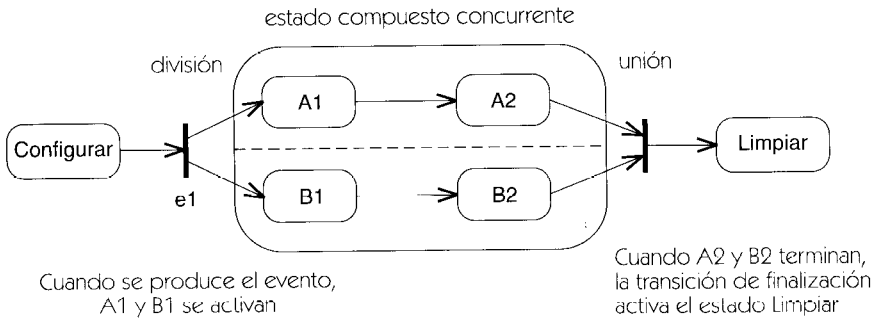


Figura 13.182 División y unión

Estados concurrentes

A menos que una máquina de estados haya sido cuidadosamente estructurada, un conjunto de transiciones complejas puede derivar en inconsistencias, incluyendo interbloqueos, ocupa-

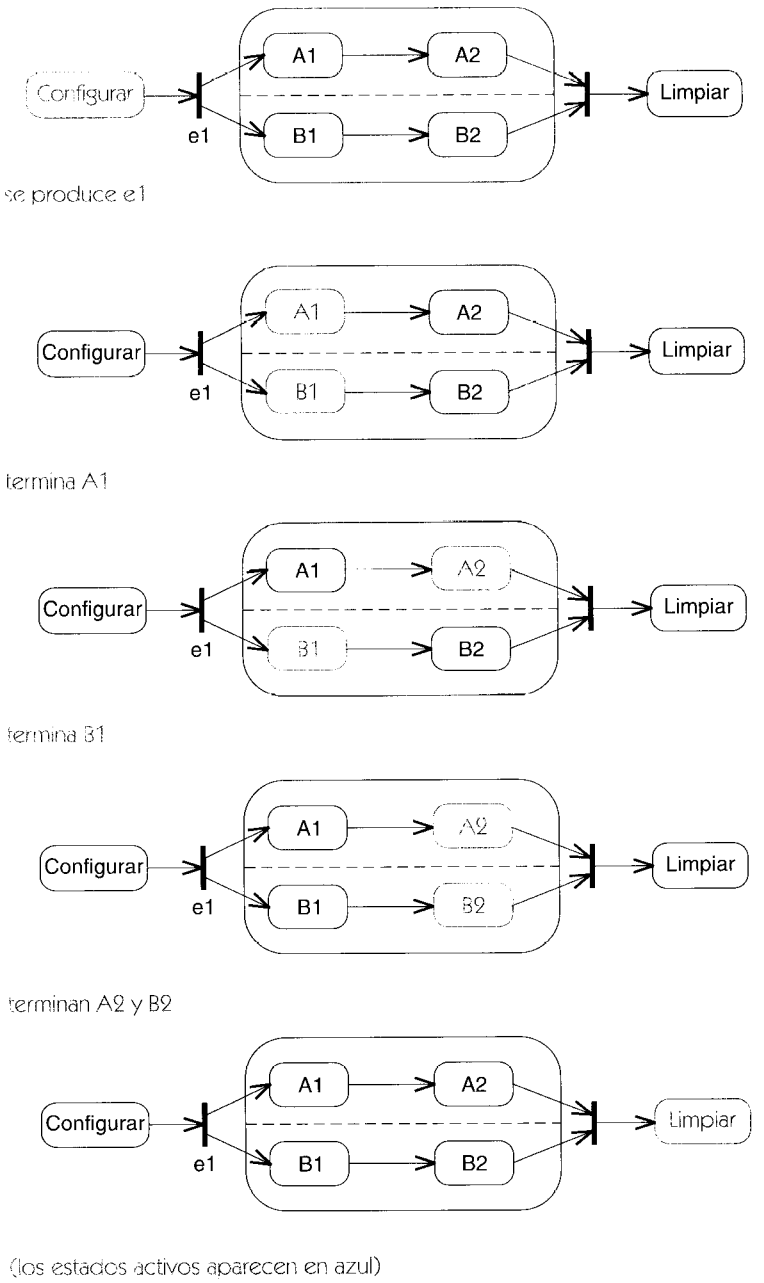


Figura 13.183 Historia de estados activos en una máquina de estados concurrente

ción múltiple de un estado, y otros problemas. El problema ha sido estudiado en profundidad a la luz de la teoría de redes de Petri, y la solución más habitual es imponer reglas de formación correcta a la máquina de estados para evitar el riesgo de inconsistencias, que son algo así como las reglas de "programación estructurada" para las máquinas de estados. Hay numerosos modelos, cada uno con sus ventajas e inconvenientes. Las reglas adoptadas por UML requieren que la máquina de estados se descomponga en estados más pequeños utilizando una especie de

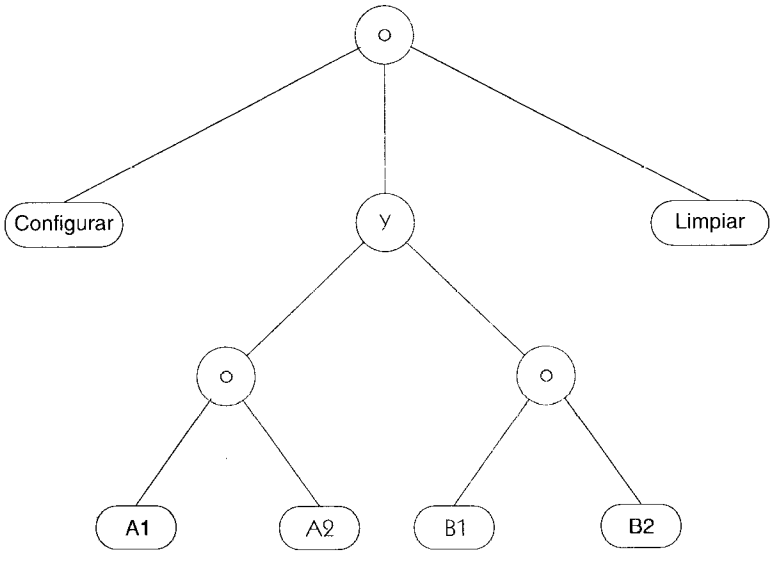
árbol Y/O. La ventaja es que una estructura bien anidada resulta sencilla de establecer, de mantener y de comprender. La desventaja es que hay ciertas configuraciones que están prohibidas aunque tienen significado. Haciendo balance, podemos decir que es algo similar a las soluciones de compromiso tomadas en referencia a las cláusulas "goto" en la programación estructurada.

Un estado complejo puede descomponerse en un conjunto de subestados mútuamente exclusivos (descomposición "O") o en un conjunto de subestados concurrentes (descomposición "Y"). La estructura es recursiva. Generalmente, las capas "Y" se alternan con las capas "O". Una capa "Y" representa una descomposición concurrente —todos los subestados están activos concurrentemente—, mientras que una capa "O" representa una descomposición secuencial —sólo hay un estado activo al mismo tiempo—. Se puede obtener un conjunto legítimo de subestados concurrentes expandiendo recursivamente los nodos del árbol empezando por la raíz, remplazando todos los estados "O" por todos sus hijos y remplazando todos los estados "Y" por todos sus hijos. El resultado obtenido se corresponde con la estructura anidada de los diagramas de estados.

Ejemplo

La Figura 13.184 muestra un árbol Y/O de estados correspondiente a la máquina de estados de la Figura 13.182. Se muestra un típico conjunto de estados activos en azul que se corresponde con el tercer paso de la Figura 13.183.

Si una transición entra en una región concurrente, entra en todos sus subestados, mientras que si entra en una región secuencial, entra exactamente en un subestado. El estado activo de una región secuencial puede cambiar, pero en una región concurrente todos los subestados concurrentes permanecen activos durante el tiempo en que la región está activa, aunque cada subestado concurrente se descompone a su vez en una región secuencial.



(El conjunto típico de estados activos aparece en azul)

Figura 13.184 Árbol Y/O de estados anidados

Por consiguiente, una transición simple (una que tiene sólo una entrada y una salida) debe conectar dos estados de la misma región secuencial o separados sólo por niveles "O". Una transición compleja debe conectar todos los subestados de una región concurrente con un estado externo a la región concurrente (se omiten deliberadamente casos más complicados, pero seguirían los mismos principios expuestos más arriba). En otras palabras, una transición de entrada en una región concurrente debe entrar en cada subestado, y una transición de salida de una región concurrente debe salir de todos los subestados.

Existe una representación abreviada: si una transición compleja entra en una región concurrente pero omite una o más de sus subregiones, existe una transición implícita al estado inicial de cada subregión omitida. Si alguna subregión no tiene estado inicial, el modelo está mal formado. Si una transición compleja sale de una región concurrente, existe una transición implícita para cada subregión omitida. Si se dispara la transición, finaliza toda actividad dentro de la subregión —es decir, representa una salida forzada—. Una transición puede conectarse con la propia región concurrente, lo que implica una transición al estado inicial de cada subregión —una situación muy común de modelado—. De forma parecida, una transición desde una región concurrente que engloba diversas subregiones provoca una salida forzada de cada una de las subregiones (si tiene un evento disparador) o una espera hasta que finaliza cada subregión (si no existen disparadores).

Las reglas sobre transiciones complejas aseguran que no pueda haber combinaciones de estados carentes de significado activas simultáneamente. Un conjunto de subestados concurrentes es una partición del estado compuesto que las engloba, donde o bien están todos activos o bien no lo está ninguno.

Hilo condicional

En un grafo de actividades, un segmento que sale de una división puede tener una condición de guarda. Esto es un hilo condicional. Cuando se dispara la transición, el hilo que comienza por el segmento que contiene la condición de guarda se inicia sólo si se satisface la condición de guarda. Los segmentos sin guardas siempre se inician cuando se dispara la transición. La concurrencia en un grafo de actividades debe estar bien anidada: cada división debe corresponderse con una unión posterior. Cuando un hilo condicional no puede iniciarse debido a que su condición de guarda se evalúa a falso, la actividad del segmento de entrada correspondiente en la unión posterior se considera completa, esto es, la transición no espera un flujo de control de dicho hilo condicional. Si ningún hilo de una división puede iniciarse, el control pasa inmediatamente a la unión.

Un hilo condicional es equivalente a un grafo con una bifurcación y una reunificación rodeando la parte condicional del grafo de actividades.

Ejemplo

La Figura 13.185 muestra un grafo de actividades con dos hilos condicionales, que representa el procedimiento de admisión que sigue una línea aérea. Antes de que nada pueda ocurrir, el cliente debe presentar el billete. A partir de ese momento hay tres hilos concurrentes, dos de los cuales son condicionales, ya que la asignación de asiento se realiza siempre. La comprobación del equipaje sólo se realiza si el pasajero tiene equipaje, y el pasaporte sólo se comprueba si el vuelo es internacional. Cuando se completan todos los hilos el control se une en un único hilo

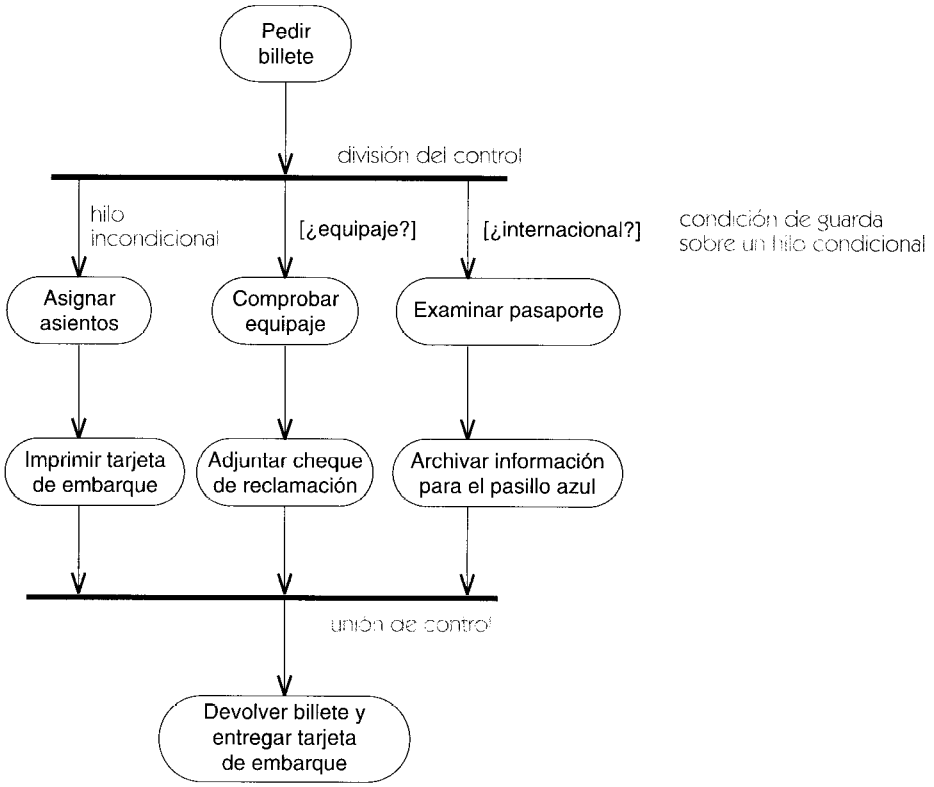


Figura 13.185 Hilos condicionales

en el cual se devuelve el billete al pasajero y se le entrega la tarjeta de embarque. Si no se inicia algún hilo condicional, se considera completado en la unión posterior.

Notación

Una transición compleja se representa mediante una pequeña barra gruesa (una barra de sincronización, que puede representar sincronización, división o ambas cosas). La barra puede tener una o más flechas de transición (dibujadas con línea continua) que van de los estados origen a la barra, y una o más flechas de transición desde la barra a los estados destino. Se puede mostrar una etiqueta de transición cerca de la barra para describir el evento disparador, la condición de guarda o la descripción de las acciones que se producen bajo la transición. Las flechas individuales no tienen sus propias cadenas de transición, ya que son meras partes de la transición global.

Ejemplo

La Figura 13.186 muestra la máquina de estados de la Figura 13.182 a la que se ha añadido una transición de salida adicional. También muestra una división implícita desde el estado **Configurar** hacia los estados iniciales de cada subregión y una unión implícita desde los estados finales de cada subregión hacia el estado **Limpiar**.

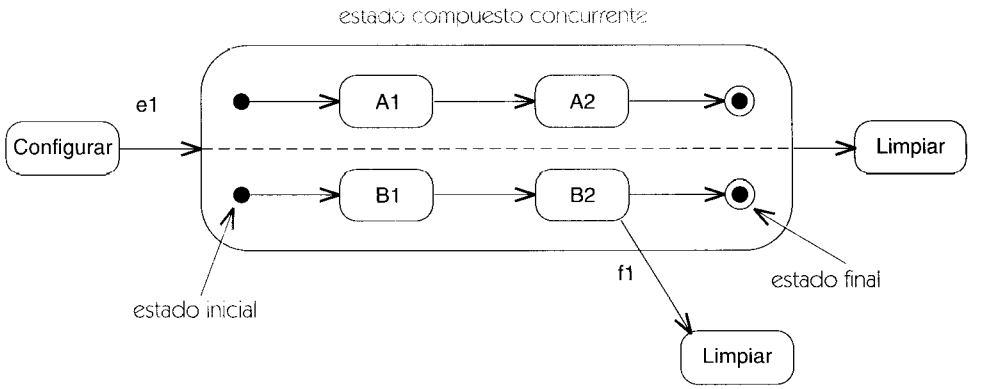


Figura 13.186 Transiciones complejas (división, unión)

Si se produce el evento **f1** cuando está activo el estado **B2**, se produce una transición al estado **Limpiar**. Esta transición es una unión implícita, ya que produce la finalización del estado **A2** a la vez que finaliza el estado **B2**.

transición de finalización

Transición que carece de un evento explícito de disparo y que se dispara por la finalización de la actividad del estado origen.

Véase también actividad, disparador, transición.

Semántica

Una transición de finalización se representa mediante una transición que no tiene un evento explícito de disparo. La transición se dispara implícitamente cuando su estado origen ha completado toda su actividad (incluyendo sus estados anidados). En un estado compuesto, la finalización de la actividad supone alcanzar el estado final. Si un estado no tiene actividad interna ni estados anidados, la transición de finalización se dispara inmediatamente después de la ejecución de la acción de entrada y de la acción de salida sin la intervención de otros eventos.

Si un estado tiene estados anidados o actividad interna pero carece de transiciones de salida disparadas por eventos, no está garantizada la ejecución de la transición de salida. Un evento podría disparar una transición sobre uno de los estados anidados mientras el estado que lo engloba está a la espera de completar su actividad, en cuyo caso la transición de finalización no se llevaría a cabo.

Una transición de finalización puede tener una condición de guarda y una acción. Normalmente no es deseable la existencia de una única transición de finalización con condición de guarda ya que si dicha condición es falsa nunca se dispara la transición (debido a que el disparador implícito sucede sólo una vez). Algunas veces esto puede ser útil para representar algún tipo de fallo, siempre que alguna transición saque al objeto de esa situación. Lo más común es que el conjunto de transiciones de finalización con condiciones de guarda tengan condiciones que cubran todas las posibilidades de forma que siempre se ejecute alguna de ellas cuando finalice el estado.

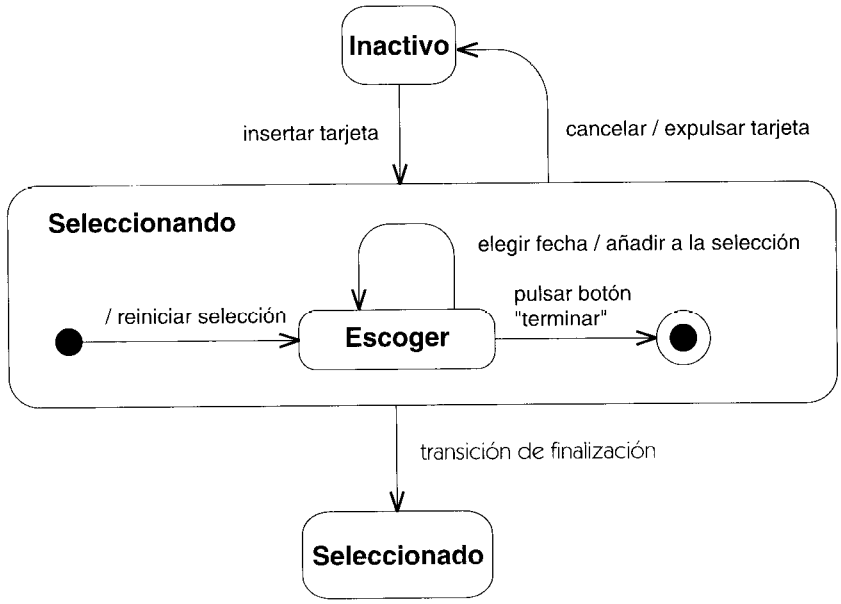


Figura 13.187 Transición de finalización

Las transiciones de finalización también se emplean para conectar estados iniciales y estados históricos con sus estados sucesores, ya que estos pseudoestados no pueden seguir activos después de la finalización de la actividad.

Ejemplo

La Figura 13.187 muestra un fragmento de una máquina de estados para una aplicación de un dispensador de entradas. El estado **Seleccionando** permanece activo durante todo el tiempo en que el cliente se encuentra eligiendo fechas. Cuando el cliente pulsa el botón "aceptar", el estado **Seleccionando** alcanza su estado final, lo que dispara su transición de finalización y lleva a la máquina de estados al estado **Seleccionado.**

transición (fase)

Cuarta fase del proceso de desarrollo del software, durante la cual el sistema implementado se configura para su ejecución en un contexto del mundo real. Durante esta fase, se completa la vista de despliegue, junto con cualquiera de las vistas restantes que pudieran no haber finalizado en las fases anteriores.

Véase proceso de desarrollo.

transición interna

Se trata de una transición asociada a un estado que posee una acción pero no implica un cambio de estado.

Véase también máquina de estados.

Semántica

La transición interna permite que un evento dé lugar a una acción sin que haya un cambio de estado. Una transición interna posee un estado de origen pero no un estado destino. Si se dispara, se ejecuta la acción pero no cambia el estado aun cuando la transición interna esté conectada a un estado que englobe al estado actual y herede (la transición) de él. Por tanto, no se ejecuta ninguna acción de entrada ni de salida. En este aspecto, difiere de una autotransición, que da lugar a la salida de los estados anidados y a la ejecución de las acciones de salida y de entrada.

Notación

La transición interna se representa mediante una entrada de texto en el compartimento del estado destinado a la transición interna. La entrada posee la misma sintaxis que el rótulo de texto correspondiente a una transición externa. Como no hay un estado destino, no hay necesidad de asociarla a una flecha.

nombre de evento / expresión de acción

Los nombres de evento **entry**, **exit**, **do** e **include** son palabras reservadas y no se pueden emplear como nombres de evento. Estas palabras reservadas se emplean para declarar una acción de entrada, una acción de salida, la ejecución de una actividad o la ejecución de una submáquina, respectivamente. Por uniformidad, estas acciones especiales hacen uso de la sintaxis de transición interna para especificar la acción. Sin embargo, no son transiciones internas y las palabras reservadas no son nombres de eventos.

La Figura 13.188 muestra la notación.

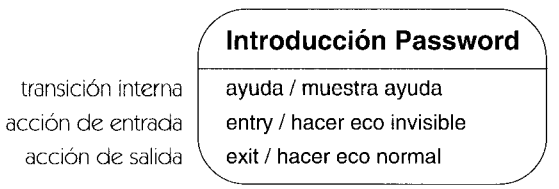


Figura 13.188 Sintaxis de transición interna

Discusión

Se puede pensar que una transición interna es una "interrupción" que da lugar a una acción sin afectar al estado actual; por tanto no invoca las acciones de salida ni las de entrada. La asociación de una transición interna a un estado compuesto es una buena forma de modelar una acción que debe suceder para un cierto número de estados pero no debe cambiar el estado activo —por ejemplo, para visualizar un mensaje de ayuda o para contar el número de apariciones de un cierto evento—. No es la forma correcta de modelar una salida por aborto o una excepción. Éstas deberían modelarse mediante transiciones a un nuevo estado, por cuanto su aparición invalida el estado actual.

transición simple

Transición con un único estado origen y un único estado destino. Representa la respuesta a un evento con un cambio de estado dentro de una región de estados mútuamente excluyentes. La cantidad de concurrencia no cambia cuando se ejecuta.

transición sin disparador

Transición que carece de un evento disparador explícito. Cuando sale de un estado normal, representa una transición de finalización, esto es, una transición que es disparada por la finalización de una actividad más que por un evento explícito. Cuando sale de un pseudoestado, representa un segmento de transición que se recorre automáticamente cuando el segmento precedente ha concluido su acción. Las transiciones sin disparador se utilizan para conectar estados iniciales y estados de historia con sus estados blanco.

traza

Dependencia que indica un proceso de desarrollo histórico o alguna otra relación más allá del modelo entre dos elementos que representan el mismo concepto sin reglas específicas para derivar uno a partir del otro. Se trata de la clase de dependencia menos específica y posee una semántica mínima. Su mayor aplicación es como recordatorio para el pensamiento humano durante el desarrollo.

Véase dependencia, modelo.

Semántica

Una traza es una variedad de dependencia que indica una conexión entre dos elementos que representan el mismo concepto en dos niveles diferentes de significado. No representa una semántica dentro de un modelo. Más bien, representa conexiones entre elementos de semántica diferente —esto es, de elementos pertenecientes a diferentes modelos situados en distintos planos de significado—. No hay una correspondencia explícita entre los elementos. Con cierta frecuencia, representa una conexión entre dos formas de capturar un concepto en distintas fases de desarrollo. Por ejemplo, dos elementos que sean variaciones de un mismo tema podrían estar relacionados por una traza. Una traza no representa una relación entre instancias en tiempo de ejecución. Más bien, es una dependencia entre elementos del modelo en sí.

Una aplicación importante de la traza es el seguimiento de requisitos que hayan ido cambiando a lo largo del desarrollo de un sistema. Las dependencias de traza pueden relacionar elementos de dos clases diferentes de modelos (tales como un modelo de caso de uso y un modelo de diseño) o pertenecientes a dos versiones de la misma clase de modelo.

Notación

Las trazas se denotan mediante una flecha de dependencia (una flecha discontinua el origen en el elemento más moderno y la punta de flecha en el elemento más antiguo) que posee la pala-

bra clave «**trace**». Normalmente, sin embargo, los elementos están en diferentes modelos que no se visualizan simultáneamente, así que en la práctica la relación se implementaría en forma de un hipervínculo dentro de alguna herramienta.

tupla

Lista ordenada de valores. Generalmente, el término implica que existe un conjunto de listas de forma similar. (Es un término matemático estándar.)

unidad de distribución

Un conjunto de objetos o componentes que se asignan a un proceso del sistema operativo o a un procesador como grupo. Una unidad de la distribución se puede representar por un compuesto de ejecución o por un agregado. Esto es un concepto de diseño en la vista de despliegue.

unión

Denota un cierto lugar de una máquina de estados, diagrama de actividades, o diagrama de secuencia en el cual se combinan dos o más hilos o estados concurrentes para formar un único hilo o estado; es una reunión o "confluencia". Antónimo: bifurcación.

Véase transición compleja, estado compuesto.

Semántica

Diremos que una unión es una transición con dos o más estados de origen y un único estado blanco. Si todos los estados de origen están activados y se produce el evento disparador, se ejecuta la acción de transición y se activa el estado blanco. Los estados de origen tienen que estar en regiones diferentes de un estado compuesto concurrente.

Notación

Las uniones se representan mediante una barra gruesa con dos o más flechas entrantes de transición y una flecha saliente de transición. Puede tener un rótulo de transición (condición de guarda, evento disparador y acción). La Figura 13.189 muestra una unión explícita de estados procedentes de un estado compuesto concurrente.

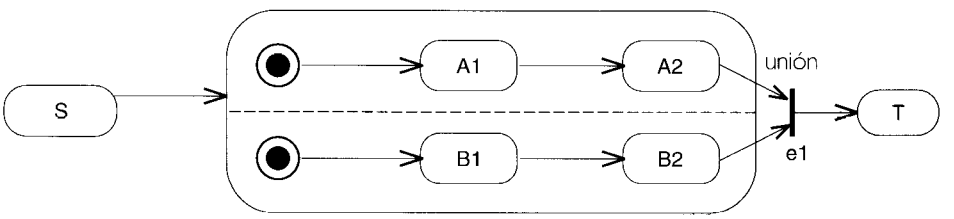


Figura 13.189 Una unión

Discusión

Véase reunificación.

use

Palabra clave para la dependencia de uso en la notación.

uso

Dependencia en la que un elemento (el cliente) requiere la presencia de otro elemento (el proveedor) para su correcto funcionamiento o implementación —generalmente es por razones de implementación.

Véase también colaboración, dependencia, enlace transitorio.

Semántica

Una dependencia de uso es una situación en la cual un elemento requiere la presencia de otro elemento para su correcta implementación o funcionamiento. Todos los elementos tienen que existir en el mismo nivel de significado —esto es, no implican un desplazamiento del nivel de abstracción o de realización (tal como una correspondencia entre una clase del nivel de análisis y una clase de nivel de implementación)—. Con cierta frecuencia las dependencias de uso implican elementos del nivel de implementación, tales como un archivo *include* de C++, lo que implica consecuencias para el compilador. Se puede estereotipar aún más el uso para indicar la naturaleza exacta de la dependencia, tal como llamar a una operación de otra clase o instanciar un objeto de otra clase.

Notación

El uso se denota mediante una flecha discontinua (dependencia) que tiene la palabra clave «**use**». La cabeza de flecha está en el elemento proveedor (independiente) y la cola se encuentra en el elemento cliente (dependiente).

Discusión

Un uso suele corresponder a un enlace transitorio, esto es, una conexión entre instancias de clases que no tiene sentido o no está presente en todo momento, sino sólo en algún contexto, tal como la ejecución de un procedimiento de subrutina. La estructura de dependencia no modela toda la información de esta situación, sino únicamente el hecho de su existencia. La estructura de colaboración proporciona la posibilidad estas relaciones con todo detalle.

Elementos estándar

call, create, instantiate, send.

uso de la tipografía

El texto puede ser distinguido con el uso de diversas tipografías y de otros marcadores gráficos.

Véase también marcador gráfico.

Discusión

Las letras cursivas se utilizan para indicar una clase abstracta, un atributo, o una operación. Otras distinciones de la tipografía están sobre todo para destacar o distinguir las partes de la notación. Se recomienda que los nombres de clasificadores y de asociaciones se muestren en negrillas y elementos subsidiarios, tales como atributos, operaciones, nombre de rol, etcétera, se muestre en tipo normal. Los nombres de compartimentos se deben mostrar en una tipografía distintiva, tal como una negrita pequeña, pero la elección se deja a la herramienta de edición. Una herramienta es también libre de utilizar las distinciones de la tipografía para destacar elementos seleccionados, para distinguir palabras y palabras claves reservadas, y para codificar características seleccionadas de un elemento, o puede permitir el uso de tales distinciones bajo control del usuario. Consideraciones similares se aplican al color, aunque su uso debe ser opcional porque muchas personas no distinguen el color.

Todos los usos mencionados son extensiones convenientes a la notación canónica descrita en este libro, la cual es suficiente para representar cualquier modelo.

utilidad

Un estereotipo de clase que agrupa variables y procedimientos globales en forma de una declaración de clase. Los atributos y operaciones de la utilidad pasan a ser variables y procedimientos globales, respectivamente. Una utilidad no es una estructura fundamental para el modelado, sino algo cómodo para la programación. No posee instancias.

Semántica

Los atributos de la utilidad son variables globales y las operaciones de la utilidad son operaciones globales. Las utilidades son innecesarias para la programación orientada a objetos, por cuanto los atributos y operaciones globales se pueden modelar mejor como miembros con alcance de clase.

Esta estructura se proporciona por compatibilidad con los lenguajes no orientados a objetos, tales como C.

Notación

Las utilidades se representan mediante un símbolo de clase con la palabra clave «**utility**» por encima de la cadena de nombre de clase. Los atributos y operaciones representan miembros globales. En este símbolo no se pueden declarar miembros con alcance de clase.

valor

Véase valor dato.

valor dato

Una instancia de un tipo de dato, un valor sin identidad.

Véase también tipo de dato, objeto.

Semántica

Un valor dato es un miembro de un dominio matemático —un valor puro—. Dos valores dato con la misma representación son indistinguibles; los valores dato no tienen ninguna identidad. Los valores dato son pasados por valor en un lenguaje de programación. No tiene ningún sentido pasarlos por referencia. No tiene sentido hablar de cambiar un valor dato; su valor está fijado permanentemente. De hecho, él es su valor. Cuando uno habla de cambiar un valor de un dato, esto significa generalmente cambiar una variable que contiene un valor de modo que contenga un nuevo valor. Pero los valores dato son invariables.

valor etiquetado

Pareja formada por un marcador y un valor, que se asocia a un elemento para almacenar alguna información. *Véase también* restricción, estereotipo. *Véase* el Capítulo 14, Elementos Estándar, para consultar una lista de marcadores predefinidos.

Semántica

Un valor etiquetado es una pareja valor-selector que se asocia a cualquier elemento (incluyendo elementos del modelo y elementos de presentación) para aportar distintos tipos de información, que suelen ser secundarios para la semántica del elemento pero posiblemente importantes para el esfuerzo global de modelado. El selector se denomina marcador; se trata de un valor de cadena. Cada marcador representa una propiedad que puede ser aplicable a una clase de elemento o a muchas clases. El nombre del marcador sólo puede aparecer una vez en cualquier elemento del modelo. El valor puede ser de distintos tipos, pero se codifica como una cadena. La interpretación del valor es una convención entre el creador del modelo y la herramienta de modelado. Se supone que los valores etiquetados se van a implementar como tablas de búsqueda indexadas por marcadores para tener un acceso eficiente.

Los valores etiquetados representan una información arbitraria expresada en forma de texto y que se suele utilizar para almacenar información relativa a la administración del proyecto, tal como el autor de un elemento, el estado de comprobación o la importancia del elemento para el sistema final (los marcadores podrían ser **autor**, **estado** e **importancia**).

Los valores etiquetados representan una modesta extensión de los metaatributos de las metaclases de UML. No es un mecanismo de extensión completamente general pero se puede utilizar para añadir información a las clases existentes del metamodelo para bien de las herramientas finales, tales como generadores de código, creadores de informes y simuladores. Para evitar la confusión, los marcadores deben ser distintos de los metaatributos existentes de los elementos del modelo al que se aplican. Esta comprobación se puede facilitar mediante una herramienta de modelado.

Ciertos marcadores están predefinidos en el UML; otros pueden ser definidos por el usuario. Los valores etiquetados son un mecanismo de extensión que permite asociar información arbitraria a los modelos.

Notación

Todos los valores etiquetados se muestran en la forma:

marcador = valor

en donde marcador es el nombre del marcador y valor es un valor literal. Los valores etiquetados se pueden incluir junto con otras palabras clave de propiedades en una lista de propiedades separadas por comas y encerrada entre llaves.

Se puede declarar una palabra clave de tal modo que haga las veces de un marcador con un cierto valor. En tal caso, la palabra clave se puede emplear en solitario. La ausencia del marcador se trata como equivalente a uno de los demás valores válidos del marcador.

marcador

Ejemplo

{autor = Héctor, estado = comprobado, requisito = 3.563.2a, suprimir}

Discusión

Casi todos los programas de edición de modelos ofrecen capacidades básicas para definir, visualizar y buscar valores etiquetados como cadenas, pero no los utilizan para extender la semántica de UML. Sin embargo, las herramientas finales, tales como los generadores de código, creadores de informes y cosas similares, pueden leer los valores etiquetados para alterar su semántica de manera flexible. Observe que las listas de valores etiquetados son una idea muy antigua; por ejemplo, las listas de propiedades del lenguaje Lisp.

Los valores etiquetados son una forma de asociar a los modelos información no semántica de administración y seguimiento del proyecto. Por ejemplo, el marcador autor podría contener el autor del elemento y el marcador **estado** podría contener el estado de desarrollo, tal como **incompleto**, **probado**, **defectuoso** y **completo**.

Además, los valores etiquetados son una forma de asociar condiciones dependientes del lenguaje de implementación a los modelos de UML sin incorporar a UML los detalles del lenguaje. Los indicadores propios de un generador de código, las pistas y los pragmas se pueden codificar como valores etiquetados sin afectar al modelo subyacente. Se pueden tener múl-

tiples conjuntos de marcadores para diferentes lenguajes en un mismo modelo. Ni el editor de modelos ni el analizador semántico tienen necesidad de comprender los marcadores —se pueden tratar como si fuesen cadenas—. Una herramienta final, tal como un generador de código, puede comprender y procesar los valores etiquetados. Por ejemplo, un valor etiquetado podría dar nombre a la clase de contenedor que se emplea para anular la implementación por defecto de una asociación con multiplicidad muchos.

Los valores etiquetados satisfacen la necesidad de muchas clases de información que es preciso asociar a los modelos, pero no están destinados a ser un mecanismo completo de extensión de metamodelos. Los marcadores forman un espacio de nombres plano que es preciso administrar empleando convenciones para evitar conflictos de nombres. No se ha previsto la posibilidad de especificar los tipos que poseen su valores. Tampoco están destinados a crear extensiones semánticas serias del lenguaje de modelado en sí. Los marcadores *vienen a ser* como atributos del metamodelo, pero *no* son atributos del metamodelo ni se han formalizado como tales.

El uso de marcadores, tal como el uso de procedimientos, en las bibliotecas de los lenguajes de programación, puede requerir un cierto período de evolución durante el cual puede haber conflictos entre los programadores. Con el tiempo, se desarrollarán estándares de utilización. UML no incluye un "registro" de marcadores ni ofrece la esperanza de que los primeros usuarios de los marcadores puedan "reservarlos" para evitar que se les den otros usos en el futuro.

valor inicial

Se trata de una expresión que especifica el valor que contiene un atributo de un objeto inmediatamente después de ser inicializado.

Véase también valor por defecto, iniciación.

Semántica

Un valor inicial es una expresión asociada a un atributo. La expresión es una cadena de texto. que también denota un lenguaje para interpretar la expresión. Cuando se instancia un objeto que contiene ese atributo, se evalúa la expresión según el lenguaje indicado y el valor actual del sistema. El resultado de la evaluación se emplea para iniciar el valor del atributo en el nuevo objeto.

El valor inicial es opcional. Si está ausente, entonces la declaración del atributo no especifica el valor que contendrá en un objeto nuevo (pero quizá otra parte del modelo total proporcione esa información).

Observe que un procedimiento explícito de inicialización para un objeto (un constructor, por ejemplo) puede anular la expresión del valor inicial y reemplazar el valor del atributo.

El valor inicial de un atributo cuyo alcance sea una clase se empleará para iniciarlo al comenzar la ejecución. UML no especifica el orden relativo de inicialización de atributos que posean distintos alcances de clase.

valor no especificado

Valor que todavía no ha sido especificado por el creador del modelo.

Discusión

Un valor no especificado no es un valor en estado correcto y no puede aparecer en un modelo complejo salvo para aquellas propiedades cuyo valor es innecesario o irrelevante. Por ejemplo, la multiplicidad no puede ser desconocida; tiene que tener algún valor. La falta de todo conocimiento es equivalente a una multiplicidad de muchos. La semántica de UML, por tanto, no permite ni trata la ausencia de valores o los valores no especificados.

Hay otro sentido en que el término "no especificado" es una parte útil de un modelo inacabado. Tiene el significado siguiente: "Todavía no he pensado en este valor y he tomado nota de él, así que lo recordaré posteriormente para darle un valor". Se trata de una indicación explícita de que el modelo está incompleto. Esto es una capacidad útil que muy bien podrían admitir las herramientas. Por su propia naturaleza, tal valor no puede aparecer en un modelo terminado y no tiene sentido definir su semántica. Una herramienta puede proporcionar automáticamente uno por defecto para los valores no especificados cuando se necesite un valor —por ejemplo, en la generación de código— pero esto no pasa de ser una cuestión de comodidad y no forma parte de la semántica. Los valores no especificados van más allá de la semántica de UML.

Análogamente, el valor por defecto de una propiedad carece de significado semántico. Las propiedades tienen un valor, simplemente. Una herramienta puede proporcionar automáticamente valores para las propiedades de los elementos recién creados. Ahora bien, esto no es más que una cierta comodidad operativa dentro de la herramienta; no forma parte de la semántica de UML. Los modelos semánticamente completos de UML no tienen valores por defecto ni valores no especificados; tienen valores, simplemente.

valor por defecto

Un valor proporcionado automáticamente como parte de un cierto lenguaje de programación o de una determinada herramienta. Los valores por defecto para las propiedades de los elementos no son parte de la semántica de UML y no aparecen en modelos.

Véase también valor inicial, parámetro, valor no especificado.

variabilidad

Propiedad que indica si puede cambiar el valor de un atributo o de un enlace.

Semántica

Esta propiedad se puede poner en un extremo de asociación o en un atributo. Si se aplica a una clase, significa que todos los atributos y enlaces de la misma la verifican (por ejemplo, un va-

lor *frozen* significa que el valor de un objeto de la clase no se puede modificar después de la inicialización). La variabilidad se representa mediante una palabra clave en una lista de propiedades con alguno de los siguientes valores:

changeable Los valores de los atributos pueden cambiar libremente, incluyendo la adición y eliminación de valores si lo permite la multiplicidad. Los enlaces pueden cambiar libremente, y ser añadidos y eliminados respetando las restricciones y la multiplicidad. Éste es el valor por defecto si no se especifica otro.

frozen Los valores de los atributos no pueden cambiar una vez inicializados, ni pueden añadirse o eliminarse valores. No se pueden añadir enlaces ni modificar los existentes tras la inicialización del objeto del extremo opuesto de la asociación (esto es, el extremo contrario a aquel que tiene el valor congelado). Sin embargo, cuando se crea un objeto en el extremo opuesto se pueden añadir enlaces al extremo congelado como parte de la inicialización.

addOnly Se pueden añadir valores de atributos adicionales si la multiplicidad no es un número fijo o ya tiene el valor máximo. Tras la creación, no se pueden borrar ni modificar los valores mientras vive el objeto. Se pueden añadir nuevos enlaces pero no se pueden eliminar ni modificar después de su creación. Si se destruye un objeto participante, los objetos que lo contienen son borrados, a pesar de su status **addOnly**.

vértice

Origen o destino de una transición en una máquina de estados. Un vértice puede ser bien un estado o un pseudoestado.

visibilidad

Una enumeración cuyo valor (**pública**, **protegida** o **privada**) denota si el elemento del modelo al que alude se puede o no ver fuera del espacio de nombres que lo contiene.

Véase también acceso para una discusión de las reglas de visibilidad en su aplicación a referencias entre paquetes.

Semántica

La visibilidad declara la capacidad de un elemento de modelado para hacer referencia a un elemento que se encuentra en un espacio de nombres distinto de aquel al que pertenece el elemento que hace la referencia. La visibilidad es parte de la relación existente entre un elemento y el contenedor que lo contiene. El contenedor puede ser un paquete, clase o algún otro espacio de nombres. Existen tres visibilidades predefinidas.

public Todo elemento que pueda ver el contenedor puede ver también el elemento indicado.

protected Sólo pueden ver el elemento indicado los elementos pertenecientes a su contenedor o a un descendiente de su contenedor.

private Sólo pueden ver el elemento indicado los elementos pertenecientes a ese mismo contenedor. Los demás elementos, incluyendo los pertenecientes a descendientes del contenedor, no pueden hacer referencia a él ni usarlo en modo alguno.

Se podrían definir otras clases de visibilidad para algunos lenguajes de programación, tales como la visibilidad *de implementación* propia de C++ (en realidad, todas las formas de visibilidad que no sea pública dependen del lenguaje). El uso de opciones adicionales tiene que hacerse por convenio entre el usuario y las posibles herramientas de modelado y generadores de código.

Notación

La visibilidad se puede mostrar mediante una palabra reservada de propiedad o bien mediante un signo de puntuación situado delante del nombre de algún elemento del modelo.

public +

protected #

private –

Se puede suprimir el marcador de visibilidad. La ausencia de un marcador de visibilidad indica que no se muestra la visibilidad, no que sea pública o no esté definida. Las herramientas deberían asignar visibilidades a los nuevos elementos, aun cuando no se muestre la visibilidad. El marcador de visibilidad es una notación taquigráfica de la cadena completa de especificación de la propiedad de visibilidad.

También se puede especificar la visibilidad mediante palabras clave (public, protected, private). Esta forma suele emplearse como elemento en medio de una lista, aplicándose a todo un bloque de atributos u otros elementos de la lista.

Toda opción de visibilidad que sea propia de un lenguaje o definida por el usuario deberá ser especificada mediante una cadena de propiedad o bien mediante alguna convención propia de una herramienta.

Clases. En las clases el marcador de visibilidad se sitúa en una lista de elementos, tales como atributos y operaciones. Muestra si otras clases pueden o no acceder a esos elementos.

Asociaciones. En una asociación, el marcador de visibilidad se sitúa en el nombre de rol de la clase destino (el extremo al que se accedería empleando esa especificación de visibilidad). Muestra si la clase del extremo opuesto puede o no recorrer la asociación en la dirección del extremo que tiene asociado el marcador de visibilidad.

Paquetes. En un paquete, el marcador de visibilidad se sitúa en elementos contenidos directamente dentro del paquete, tales como clases, asociaciones y paquetes anidados. Muestra si otro paquete que acceda al primero o lo importe puede o no ver los elementos.

vista

Proyección de un modelo, que se ve desde una cierta perspectiva o punto de observación y emite aquellas entidades que no son relevantes para esa perspectiva. Aquí, la palabra no se usa para denotar un elemento de presentación. Se refiere más bien a las proyecciones tanto en el modelo semántico como en la notación visual.

vista de actividades

Aspecto del sistema que tiene que ver con la especificación del comportamiento como actividades conectadas por flujos de control. Esta vista contiene grafos de actividades y se representa mediante diagramas de actividades. De forma un tanto libre se agrupa con otras vistas que tienen que ver con el comportamiento para formar la vista dinámica.

Véase grafo de actividades.

vista de administración del modelo

Dícese de aquel aspecto de un modelo que trata de la organización del modelo en sí en partes estructuradas, a saber, paquetes, subsistemas y modelos. La vista de administración del modelo se considera a veces parte de la vista estática y suele combinarse con la vista estática en los diagramas de clases.

vista de casos de uso

Aspecto del sistema dedicado a la especificación de comportamiento en términos de casos de uso. Un modelo de casos de uso es un modelo que se centra en esta visualización. La vista de casos de uso es parte del conjunto de conceptos de modelado que están relativamente agrupados como vista dinámica.

vista de comportamiento

Vista del modelo que pone de relieve el comportamiento de las instancias existentes en un sistema, incluyendo sus métodos, colaboraciones e historias de estados.

vista de despliegue

Una vista que muestra los nodos en un sistema distribuido, los componentes que se almacenan en cada nodo, y los objetos que se almacenan en componentes y nodos.

Véase despliegue, diagrama de despliegue.

vista de implementación

Es una vista de un modelo que contiene una declaración estática de los componentes de un sistema, de sus dependencias y posiblemente de las clases que sean implementadas por ese componente.

Véase diagrama de componentes.

vista de interacción

Se trata de la visualización de un modelo que muestra el intercambio de mensajes entre objetos para lograr algún propósito. Consta de colaboraciones e interacciones y se muestra empleando diagramas de colaboración y diagramas de secuencia.

vista de máquina de estados

Aspecto del sistema relacionado con la especificación del comportamiento de elementos individuales a lo largo de su vida. Esta vista contiene máquinas de estados. Está relativamente agrupada con otras vistas de comportamiento en la vista dinámica.

vista dinámica

Ese aspecto de un modelo que se ocupa de la especificación y de la implementación del comportamiento a lo largo del tiempo, contrapuesto a la estructura estática que se encuentra en la vista estática. La vista dinámica es un término de agrupación que incluye la vista de casos de uso, la vista de máquina de estados, la vista de actividades, y la vista de interacción.

vista estática

Vista del modelo global que caracteriza los elementos de un sistema y sus relaciones entre sí. Incluye a los clasificadores y sus relaciones: sucesos, generalización, dependencia y realización. A veces recibe el nombre de *vista de clases.*

Semántica

La vista estática muestra la estructura estática de un sistema y en particular las clases de elementos que existen (tales como las clases y tipos), su estructura interna y sus relaciones con otros elementos. Las vistas estáticas no muestran información temporal, aunque puede contener apariciones materializadas de cosas que tengan o describan un comportamiento temporal, tal como las especificaciones de operaciones o de eventos.

Entre los componentes de más alto nivel de una vista estática se cuentan los clasificadores (clase, interfaz, tipo de datos), las relaciones (asociación, generalización, dependencia, reali-

zación), las restricciones y los comentarios. También contiene paquetes y subsistemas como unidades organizativas. Los demás componentes están subordinados y contenidos en los elementos de alto nivel.

Las vistas de implementación, de despliegue y de administración del modelo están relacionadas con la vista estática y suelen combinarse con ella en otros diagramas.

La vista estática se puede contrastar con la vista dinámica, que la complementa y se basa en ella.

vista estructural

Vista global de un modelo que hace hincapié en la estructura de los objetos del sistema, incluyendo sus tipos, clases, relaciones, atributos y operaciones.

vista funcional

Una vista que se ocupa de la descomposición de un sistema en las funciones o las operaciones que proporcionan su funcionalidad. Una vista funcional no se considera normalmente orientada a objetos y puede conducir a una arquitectura difícil de mantener.

En los métodos de desarrollo tradicionales, el diagrama de flujo de datos es el corazón de la vista funcional. UML no apoya directamente una visión funcional, aunque los grafos de actividades tienen algunas características funcionales.

14 Elementos estándar

Los elementos estándar son palabras clave predefinidas que corresponden a restricciones, estereotipos y etiquetas. Representan conceptos de utilidad general que no son suficientemente significativos o suficientemente diferentes de los conceptos centrales para incluirlos como conceptos centrales de UML. Tienen la misma relación con los conceptos centrales de UML que la que posee una biblioteca de subrutinas con un lenguaje de programación. No forman parte del lenguaje en sí, pero forman parte del entorno con que puede contar el usuario cuando hace uso del lenguaje. La lista incluye también palabras clave de notación —son palabras reservadas que aparecen en el símbolo de otro elemento del modelo pero que denotan elementos incorporados del modelo, no estereotipos—. Para las palabras clave se enumera el símbolo de la notación.

Las referencias cruzadas aluden a artículos del Capítulo 13, Enciclopedia de Términos.

access

(estereotipo de la dependencia Permiso)

Dependencia estereotipada entre dos paquetes que denota que el contenido público del paquete destino es visible para el espacio de nombres del paquete origen.

Véase acceso.

association

(estereotipo de ExtremoAsociación)

Restricción aplicada al extremo de una asociación (incluyendo los extremos de un enlace o un extremo de un rol de asociación), que especifica que la instancia correspondiente es visible a través de una asociación virtual y no a través de un enlace transitorio, tal como un parámetro o variable local.

Véase asociación, extremo de asociación, rol de asociación.

become

(estereotipo de la relación Flujo)

Dependencia estereotipada cuyo origen y destino representan la misma instancia en distintos instantes de tiempo, pero pudiendo tener cada una valores, instancia de estado y roles diferentes.

Una dependencia de conversión que va desde **A** hasta **B** significa que la instancia a se convierte en **B** con (posiblemente) nuevos valores, instancia de estado y roles. La notación de conversión es una flecha discontinua que va del origen al destino con la palabra clave «**become**».

Véase become en la enciclopedia de términos.

bind

(palabra reservada en un símbolo de dependencia)

Palabra reservada de dependencia que denota una relación de ligadora. Va seguida por una lista de argumentos separados por comas y encerrada entre paréntesis.

Véase enlace, elemento ligado, plantilla.

call

(estereotipo de la dependencia Uso)

Dependencia estereotipada cuyo origen es una operación y cuyo destino es una operación. Una dependencia de llamada específica que el origen invoca a la operación destino. Una dependencia de llamada puede conectar una operación origen con cualquier operación destino situada a su alcance, incluyendo (pero sin tener como límite) las operaciones del clasificador que la contiene y las operaciones de otros clasificadores visibles.

Véase llamada, uso.

complete

(restricción de Generalización)

Restricción que se aplica a un conjunto de generalizaciones, para especificar que se han especificado todos los hijos (aunque se haya omitido alguno) y que no se espera declarar hijos adicionales posteriormente.

Véase generalización.

copy

(estereotipo de la relación Flujo)

Relación de flujo estereotipada cuyo origen y destino son instancias distintas, pero teniendo ambas los mismos valores, instancia de estado y roles (aunque con distinta identidad). Una dependencia de copia de A hacia B significa que B es una copia exacta de A. Los futuros

cambios de A no se reflejan necesariamente en B. La notación de copia es una flecha discontinua que va del origen al destino con la palabra clave «**copy**».

Véase acceder, copia.

create

(estereotipo de característicadecomportamiento)

Característica de comportamiento estereotipada que denota que la característica indicada crea una instancia del clasificador al que está asociada la característica.

(estereotipo de evento)

Evento estereotipado que denota que la instancia que encierra la máquina de estados a que se aplica el evento se crea. La creación se puede aplicar únicamente a una transición inicial situada en el nivel más elevado de la máquina de estados. De hecho, éste es el único tipo de disparador que se puede aplicar a una transición inicial.

(estereotipo de la dependencia Utilización)

Dependencia estereotipada que denota que el clasificador cliente crea instancias del clasificador proveedor.

Véase creación, uso.

derive

(estereotipo de la dependencia Abstracción)

Dependencia estereotipada cuyo origen y destino suelen ser (aunque no necesariamente) elementos del mismo tipo. Una dependencia de derivación especifica que el origen se puede computar a partir del destino. El origen puede haberse implementado por razones de diseño, tales como la eficiencia, aun cuando sea redundante desde un punto de vista lógico.

Véase derivación, elemento derivado.

destroy

(estereotipo de característicadecomportamiento)

Característica de comportamiento estereotipada que denota que la característica indicada destruye una instancia del clasificador al que está asociada la característica.

(estereotipo de evento)

Evento estereotipado que denota la destrucción de la instancia que contiene a la máquina de estados a la que se aplica ese tipo de evento.

Véase destrucción.

destroyed

(restricción de Rol de clasificador y de Rol de asociación)

Denota que existe una instancia de ese rol al principio de la ejecución de la interacción que lo contiene pero esa instancia es destruida antes de finalizar la ejecución.

Véase rol de asociación, rol de clasificador, colaboración, destrucción.

disjoint

(restricción de la Generalización)

Restricción que se aplica a un conjunto de generalizaciones, especificando que un objeto no puede ser instancia de más de un hijo en ese conjunto de generalizaciones. Esta situación únicamente surgiría con herencia múltiple.

Véase generalización.

document

(estereotipo de componente)

Componente estereotipado que representa un documento.

Véase componente.

documentation

(etiqueta de un elemento)

Comentario, descripción o explicación del elemento al que está asociada.

Véase comentario, cadena.

enumeration

(palabra clave de los símbolos de clasificador)

Palabra reservada de un tipo de datos de enumeración, cuyos detalles especifican un dominio que consta de un conjunto de identificadores que son los posibles valores de una instancia de ese tipo de datos.

Véase enumeración.

executable

(estereotipo de componente)

Componente estereotipado que denota un programa que se puede ejecutar en un nodo.

Véase componente.

extend

(palabra reservada del símbolo de Dependencia)

Palabra clave de un símbolo de dependencia que denota una relación de extensión entre casos de uso.

Véase extender.

facade

(estereotipo de paquete)

Paquete estereotipado que contiene únicamente referencias a elementos del modelo que son propiedad de otro paquete. Se emplea para proporcionar una vista pública de cierta parte del contenido de un paquete. Una fachada no contiene ningún elemento del modelo que sea propio de ella.

Véase paquete.

file

(estereotipo de componente)

Archivo es un componente estereotipado que representa un documento que contiene código fuente o bien datos.

Véase componente.

framework

(estereotipo de Paquete)

Paquete estereotipado que consta principalmente de patrones.

Véase paquete.

friend

(estereotipo de dependencia de permiso)

Dependencia estereotipada cuyo origen es un elemento del modelo tal como una operación, clase o paquete y cuyo destino es un elemento diferente del modelo, tal como una clase o

paquete. La relación de amistad concede al destino acceder al origen, independientemente de la visibilidad declarada. Extiende la visibilidad del origen de tal modo que el destino puede ver el interior del origen.

Véase acceso, amigo, visibilidad.

global

(estereotipo de ExtremoAsociacion)

Restricción que se aplica a un extremo de asociación (incluyendo los extremos de enlace y los extremos de roles de asociación), especificando que el objeto asociado es visible porque se encuentra en un alcance global respecto al objeto que está en el otro extremo del enlace.

Véase asociación, extremo de asociación, colaboración.

implementation

(estereotipo de generalización)

Generalización estereotipada que denota que el cliente hereda la lista implementación del proveedor (sus atributos, operaciones y métodos) pero no hace públicas las interfaces del proveedor ni garantiza que va a admitirlas, violando por tanto la capacidad de sustitución. Es una herencia privada.

Véase generalización, herencia privada.

implementationClass

(estereotipo de clase)

Clase estereotipada que no es un tipo y que representa una implementación de una clase en algún lenguaje de programación. Un objeto puede ser instancia de un máximo de una clase de implementación. Por contraste, un objeto puede ser una instancia de múltiples clases ordinarias en un instante y perder o ganar clases con el paso del tiempo. Una instancia de una clase de implementación también puede ser instancia de cero o más tipos.

Véase clase de implementación, tipo.

implicit

(estereotipo de asociación)

Estereotipo de una asociación, que especifica que la asociación no es manifiesta (no está implementada) sino sólo conceptual.

Véase asociación.

import

(estereotipo de dependencia de permiso)

Dependencia estereotipada entre dos paquetes, que denota que el contenido público del paquete destino se añade al espacio de nombres del paquete origen.

Véase acceso, importar.

include

(palabra clave de símbolo de dependencia)

Palabra clave de los símbolos de dependencia que denota una relación de inclusión entre casos de uso.

Véase incluir.

incomplete

(restricción de generalización)

Restricción que se aplica a un conjunto de generalizaciones, especificando que no se han especificado todos los hijos y que se espera la adición de otros hijos.

Véase generalización.

instanceOf

(palabra clave de símbolo de dependencia)

Una metarelación cuyo cliente es una instancia y cuyo proveedor es un clasificador. Una dependencia **instanceOf** que va desde **A** hacia **B** indica que **A** es una instancia de **B**. La notación de **instanceOf** es una flecha discontinua con la palabra clave «**instanceOf**».

Véase descriptor, instancia, instancia de.

instantiate

(estereotipo de dependencia de uso)

Dependencia estereotipada entre clasificadores, que indica que las operaciones que afectan al cliente crean instancias del proveedor.

Véase instanciación, uso.

invariant

(estereotipo de restricción)

Restricción estereotipada que es preciso asociar a un conjunto de clasificadores o relaciones. Denota las condiciones que debe imponer la restricción a efectos de los clasificadores o relaciones y sus instancias.

Véase invariante.

leaf

(palabra reservada de ElementoGeneralizable y CaracterísticaDeComportamiento)

Indica un elemento que no puede tener descendientes o no se pueden anular, esto es, que no es polimórfico.

Véase hoja, polimórfico/a.

library

(estereotipo de componente)

Componente estereotipado que representa una biblioteca estática o dinámica.

Véase componente.

local

(estereotipo de ExtremoAsociación)

Estereotipo de un extremo de asociación, extremo de enlace o extremo de rol de asociación, que especifica que el objeto asociado se encuentra en el alcance local del objeto que está en el otro extremo.

Véase asociación, extremo de asociación, colaboración, enlace transitorio.

location

(etiqueta de símbolo clasificador)

El componente que sirve de base al clasificador.

(palabra reservada de símbolo de instancia de componente)

Instancia de nodo en la que reside la instancia de componente.

Véase componente, localización, nodo.

metaclass

(estereotipo de clasificador)

Clasificador estereotipado que denota que la clase es una metaclase de alguna otra clase.

Véase metaclase.

new

(restricción de RolClasificador y Roldeasociación)

Denota que se crea una instancia del rol durante la ejecución de la interacción que lo contiene y que esa instancia sigue existiendo al finalizar la ejecución.

Véase rol de asociación, rol de clasificador, colaboración, creación.

overlapping

(restricción de generalización)

Restricción aplicada a un conjunto de generalizaciones, que especifica que un objeto puede ser instancia de más de un hijo dentro de ese conjunto de generalizaciones. Sólo puede surgir esta situación con herencia múltiple o clasificación múltiple.

Véase generalización.

parameter

(estereotipo de ExtremoAsociación)

Estereotipo de un extremo de asociación (incluyendo el extremo de un enlace y el extremo de un rol de asociación), que especifica que un objeto asociado es un argumento de una llamada a una operación del objeto situado en el otro extremo.

Véase rol de asociación, rol de clasificador, colaboración, parámetro, enlace transitorio.

persistence

(etiqueta de clasificador, asociación y atributo)

Denota si el valor de una instancia debe o no sobrevivir al proceso que lo ha creado. Los valores son persistentes y transitorios. Si se usa en un atributo, permite una mejor discriminación acerca de cuáles de los valores de los atributos deberían mantenerse dentro de un clasificador.

Véase objeto persistente.

postcondition

(estereotipo de restricción)

Restricción estereotipada que debe estar asociada a una operación. Denota las condiciones que deben cumplirse después de haber invocado a la operación.

Véase postcondición.

powertype

(estereotipo de clasificador)

Clasificador estereotipado que denota que el clasificador es una metaclase cuyas instancias son subclases de otra clase.

(palabra reservada de símbolo de dependencia)

Relación cuyo cliente es un conjunto de generalizaciones y cuyo proveedor es un supratipo. El proveedor es un supratipo del cliente.

Véase supratipo.

precondition

(estereotipo de restricción)

Restricción estereotipada que debe estar asociada a una operación. Denota las condiciones que deben cumplirse en el momento de invocar a la operación.

Véase precondición.

process

(estereotipo de clasificador)

Clasificador estereotipado que es una clase activa que representa a un proceso pesado.

Véase clase activa, proceso, hilo.

refine

(estereotipo de dependencia de abstracción)

Estereotipo de dependencia que denota una relación de refinamiento.

Véase refinamiento.

requirement

(estereotipo de comentario)

Comentario estereotipado que enuncia una responsabilidad u obligación.

Véase requisito, responsabilidad.

responsibility

(estereotipo de comentario)

Contrato u obligación del clasificador. Se expresa mediante una cadena de texto.

Véase responsabilidad.

self

(estereotipo de ExtremoAsociación)

Estereotipo de un extremo de asociación (incluyendo los extremos de enlace y los extremos de rol de asociación), que especifica un pseudoenlace de un objeto consigo mismo con el propósito de invocar a una operación del mismo objeto dentro de una interacción. No implica una estructura de datos real.

Véase rol de asociación, rol de clasificador, colaboración, parámetro, enlace transitorio.

semantics

(etiqueta de clasificador)

Especificación del significado del clasificador.

(etiqueta de operación)

Especificación del significado de la operación.

Véase semántica.

send

(estereotipo de dependencia de uso)

Dependencia estereotipada cuyo cliente es una operación o clasificador y cuyo proveedor es una señal, que especifica que el cliente envía la señal a algún destino no especificado.

Véase enviar, señal.

stereotype

(palabra clave de símbolo clasificador)

Palabra clave para la definición de un estereotipo. Se puede utilizar el nombre como nombre de estereotipo en otros elementos del modelo.

Véase estereotipo.

stub

(estereotipo de paquete)

Paquete estereotipado que representa un paquete que proporciona las partes públicas de otro paquete, pero nada más.

Observe que la palabra también se usa en UML para describir las transiciones abreviadas.

Véase paquete.

system

(estereotipo de paquete)

Paquete estereotipado que contiene un conjunto de modelos de un sistema, describiéndolo desde distintos puntos de vista no necesariamente disjuntos —es la estructura de más alto nivel en la especificación del sistema—. También contiene relaciones entre elementos del modelo pertenecientes a diferentes modelos. Estas relaciones y restricciones no añaden información semántica a los modelos. Lo que hacen es describir las relaciones de los modelos en sí; por ejemplo para el seguimiento de requisitos y de la historia del desarrollo.

Un sistema puede ser realizado por un conjunto de sistemas subordinados, cada uno de los cuales es descrito por su propio conjunto de modelos reunidos en un paquete de sistema por separado. Un paquete de sistema sólo puede estar contenido en un paquete de sistema.

Véase paquete, modelo, sistema.

table

(estereotipo de componente)

Componente estereotipado que representa una tabla de una base de datos.

Véase componente.

thread

(estereotipo de clasificador)

Clasificador estereotipado que es una clase activa y representa un flujo de control ligero.

Observe que en este libro se utiliza la palabra con un sentido más amplio, denotando cualquier centro de ejecución independiente y concurrente.

Véase clase activa, hilo.

trace

(palabra clave de dependencia de abstracción)

Palabra clave de un símbolo de dependencia que denota una relación de traza.

Véase traza.

transient

(restricción de rol de clasificador y rol de asociación)

Afirma que se crea una instancia del rol durante la ejecución dc la interacción que lo contiene, pero esa instancia se destruye antes de haber finalizado la ejecución.

Véase rol de asociación, rol de clasificador, colaboración, creación, destrucción, enlace transitorio.

type

(estereotipo de clase)

Clase estereotipada que se emplea para la especificación de un dominio de instancias (objetos) junto con las operaciones que son aplicables a esos objetos. Un tipo no puede contener métodos pero puede poseer atributos y asociaciones.

Véase clase de implementación, tipo.

use

(palabra clave de símbolo de dependencia)

Palabra clave de dependencia que denota una relación de uso.

Véase uso.

utility

(estereotipo de clasificador)

Clasificador estereotipado que carece de instancias. Describe una colección con nombre de atributos y operaciones que no son miembros, todos los cuales tienen alcance de clase.

Véase utilidad.

xor

(restricción de asociación)

Restricción aplicada a un conjunto de asociación que comparten una conexión con una clase; especifica que todo objeto de la clase compartida tendrá enlaces procedentes de una sola de las asociaciones. Se trata de una restricción o-exclusivo (no o-inclusivo).

Véase asociación.

Parte 4: Apéndices

Apéndice A
Metamodelo de UML

Documentos de definición de UML

UML está definido en un conjunto de documentos publicados por el Object Management Group [UML-98]. Estos documentos se han incluido en el CD que acompaña al libro. Este capítulo explica la estructura del modelo semántico de UML que se describe en los documentos.

El UML se define formalmente empleando un metamodelo —esto es, un modelo de las estructuras de UML. El metamodelo, a su vez, se expresa en UML. Esto es un ejemplo de intérprete metacircular— esto es, un lenguaje definido en términos de sí mismo. Pero las cosas no son completamente circulares. Sólo se emplea un pequeño subconjunto de UML para definir el metamodelo. En principio, el punto fijo de la definición se podría basar en alguna definición más básica. En la práctica, no es necesario llegar a estos extremos.

Cada sección del documento semántico contiene un diagrama de clases que muestra una porción del metamodelo; una descripción textual de las clases del metamodelo definidas en esa sección, con sus atributos y relaciones; una lista de restricciones aplicables a los elementos expresadas en lenguaje natural y en OCL y por último una descripción textual de la semántica dinámica de las estructuras de UML definidas en la sección. Por tanto, la semántica dinámica es informal, pero una descripción completamente formal sería a la vez poco práctica y prácticamente ilegible para casi todos.

La notación se describe en otro capítulo que hace referencia al capítulo de semántica y establece la correspondencia entre los símbolos y las clases del metamodelo.

Estructura del metamodelo

El metamodelo está dividido en tres paquetes principales (Figura A1).

- El paquete de fundamentos define la estructura estática de UML.
- El paquete de elementos de comportamiento define la estructura dinámica de UML.
- El paquete de administración del modelo define la estructura organizativa de los modelos de UML.

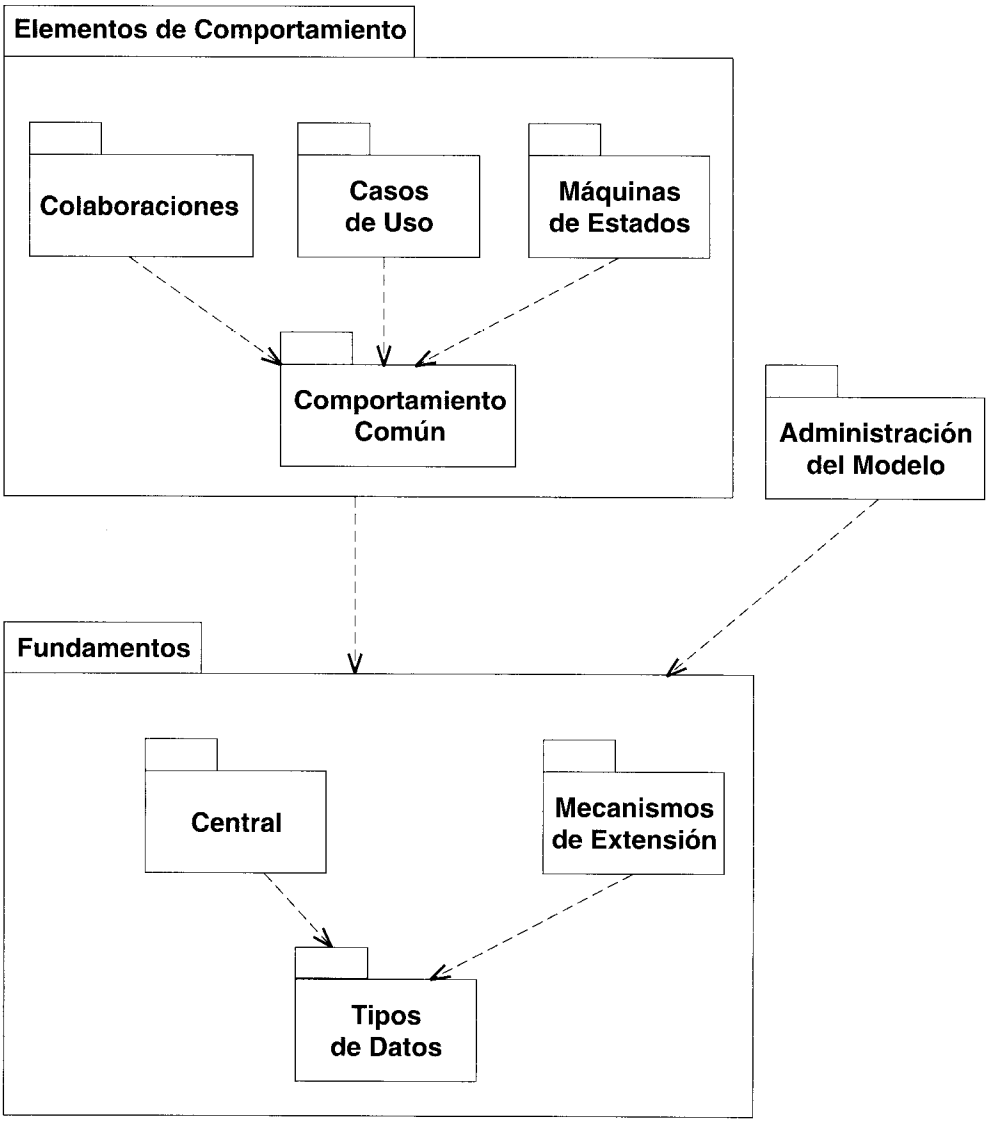


Figura A-1 Estructura de paquetes del metamodelo de UML

Paquete de fundamentos

El paquete de fundamentos contiene cuatro fundamentos.

Núcleo

El paquete núcleo describe las principales estructuras estáticas de UML. Entre ellas se cuentan los clasificadores, su contenido y sus relaciones. El contenido incluye atributo, operación, método y parámetro. Entre las relaciones se cuentan generalización, asociación y dependencia. También se definen varias metaclases abstractas tales como elemento generalizable, espacio de

nombres y elemento del modelo. El paquete también define plantilla y distintos tipos de subclases de dependencia así como componente, nodo y comentario.

Tipos de datos

El paquete de tipos de datos describe las clases de tipos de datos que se utilizan en el metamodelo.

Mecanismos de extensión

El paquete de mecanismos de extensión describe los mecanismos restricción, estereotipo y valor etiquetado.

Paquete de elementos de comportamiento

El paquete de comportamiento tiene un subpaquete para cada vista principal y además un paquete para estructuras de comportamiento que comparten las tres vistas principales.

Comportamiento común

El paquete de comportamiento común describe señal, operación y acción. También describe instancias de clases correspondientes a diferentes descriptores.

Colaboraciones

El paquete colaboraciones describe colaboración, interacción, mensaje, rol de clasificador y asociación.

Casos de uso

El paquete casos de uso describe actor y caso de uso.

Máquinas de estados

El paquete máquinas de estados describe la estructura de una máquina de estados, incluyendo estado y distintas clases de pseudoestado, evento, señal, transición y condición de guarda. También describe estructuras adicionales para modelos de actividad tales como estado de acción, estado de actividad y estado de flujo de objeto.

Paquete de administración de modelos

El paquete de administración de modelos describe los paquetes, modelos y subsistemas. También describe las propiedades de propiedad y visibilidad de los espacios de nombres y de los paquetes. No posee subpaquetes.

Apéndice B

Resumen de la notación

Este capítulo contiene un breve resumen visual de la notación. Se han incluido los elementos principales de la notación, pero no constan todas las versiones u opciones. Para estudiar todos los detalles, consulte la entrada de cada elemento en la enciclopedia.

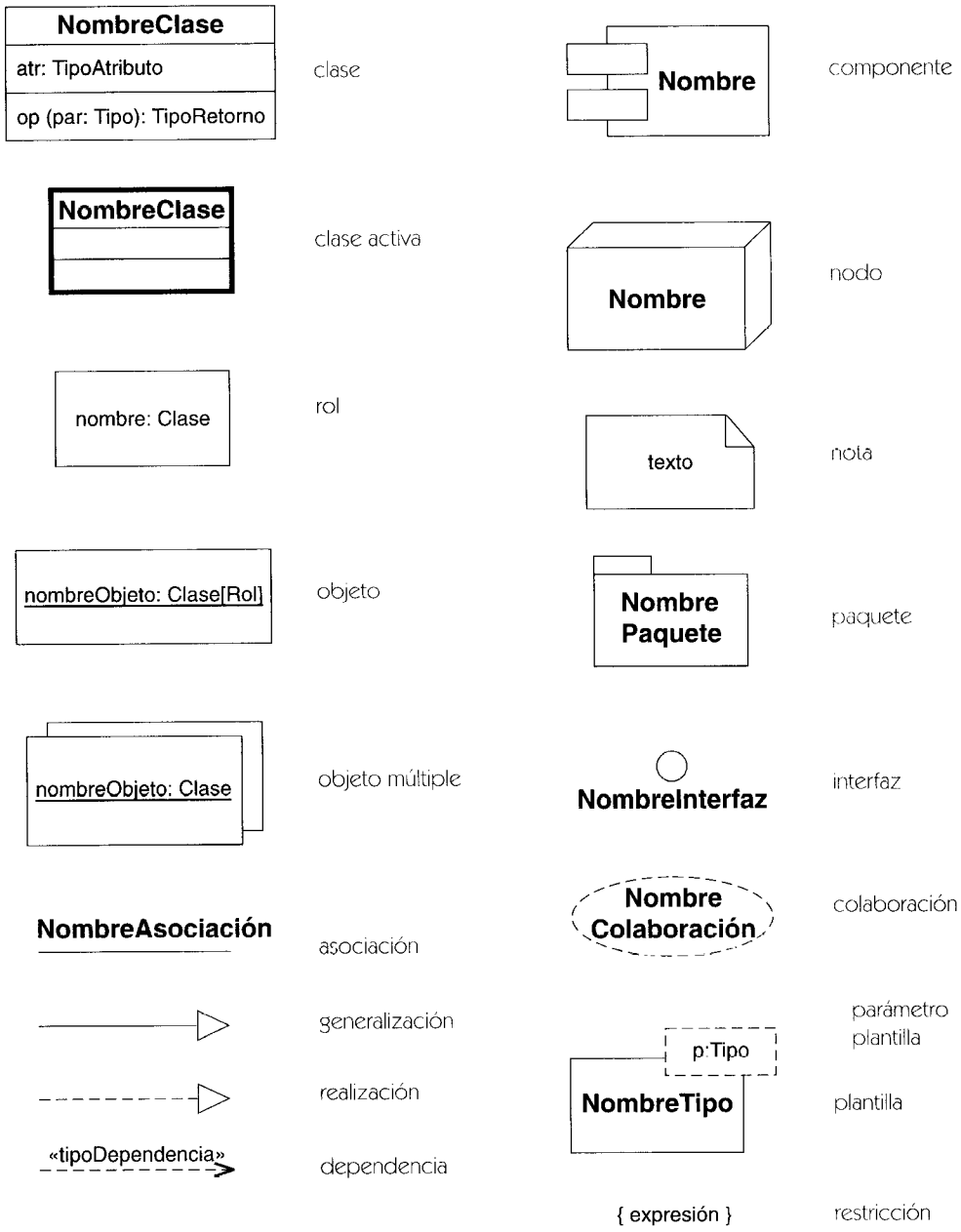


Figura B-1 Iconos de los diagramas de clase, componente, despliegue y colaboración

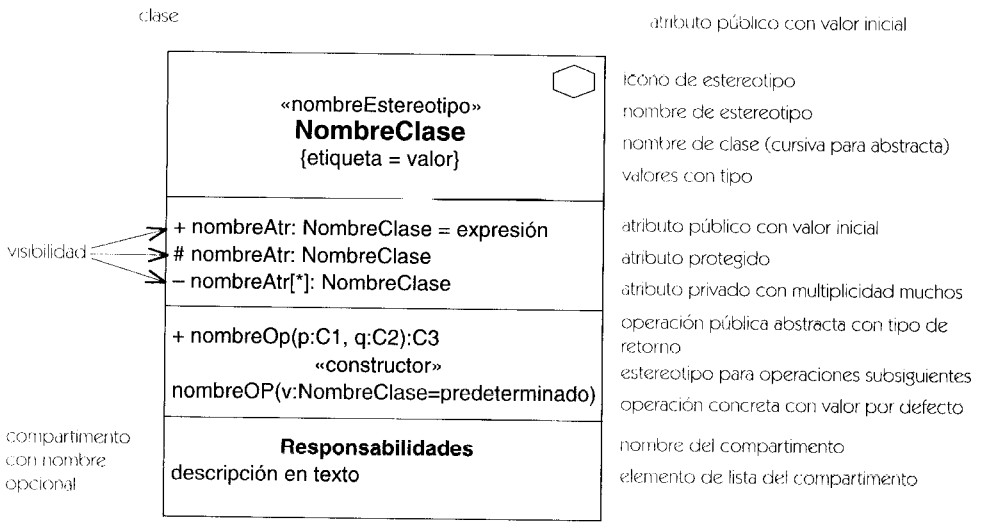


Figura B-2 Contenido de una clase

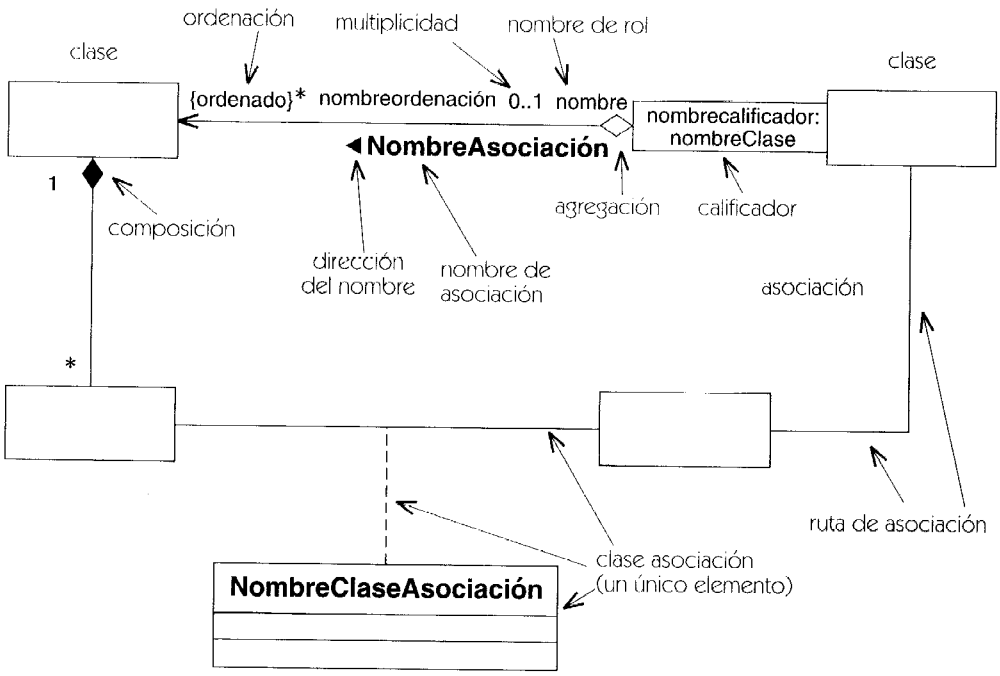


Figura B-3 Adornos de asociación dentro de un diagrama de clases

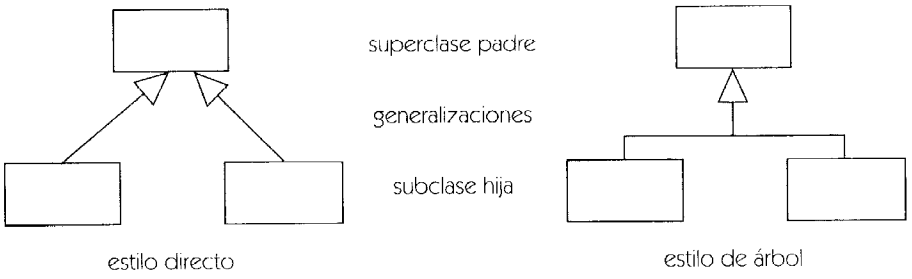


Figura B-4 Generalización

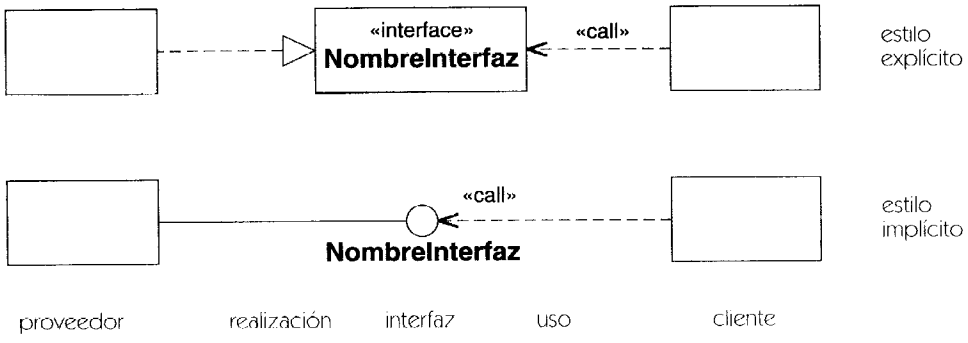


Figura B-5 Realización de una interfaz

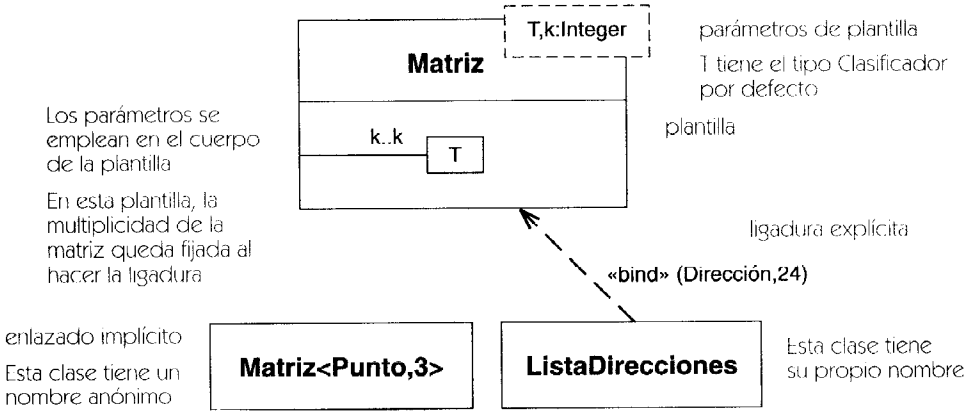


Figura B-6 Plantilla

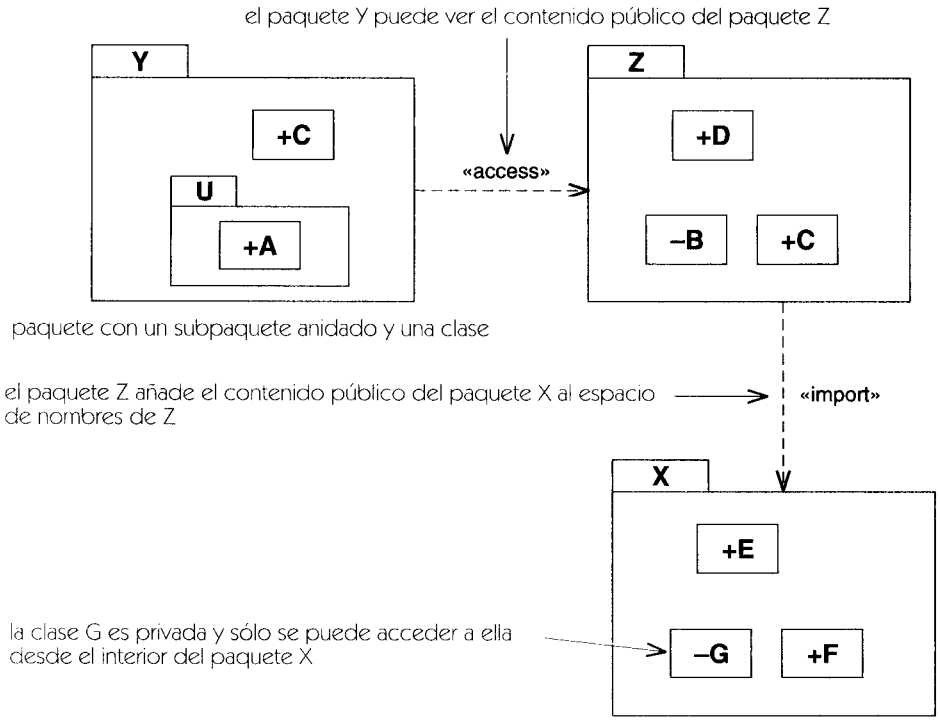


Figura B-7 Notación de paquetes

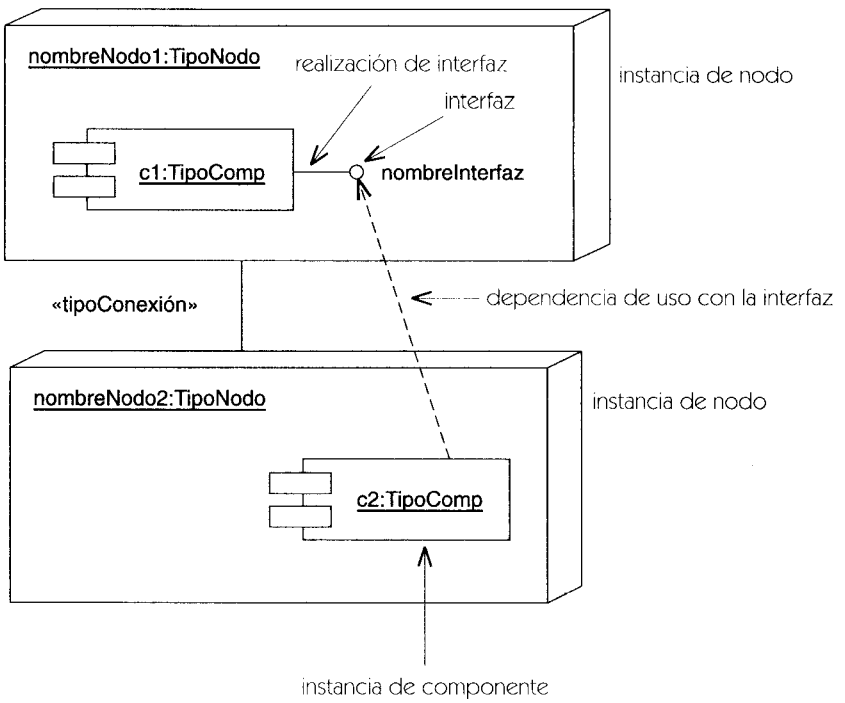


Figura B-8 Notación de componentes y nodos

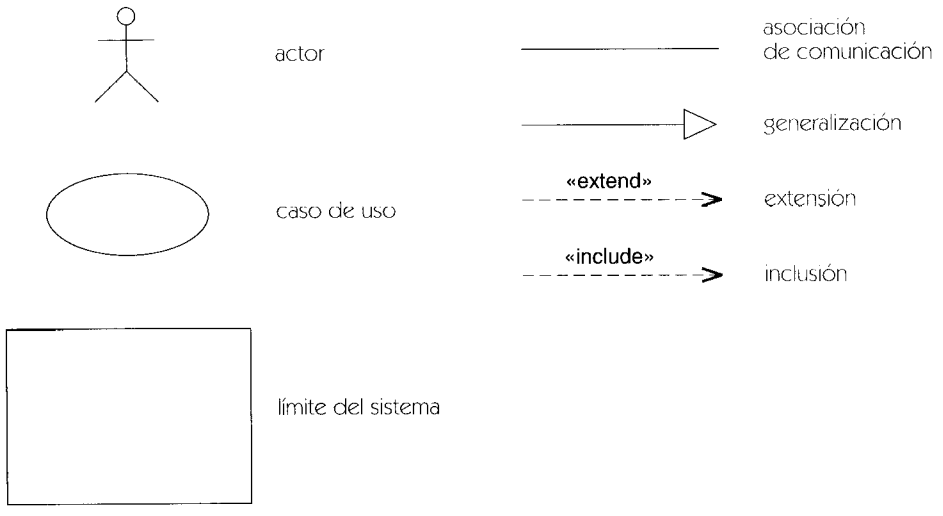


Figura B-9 Iconos de los diagramas de caso de uso

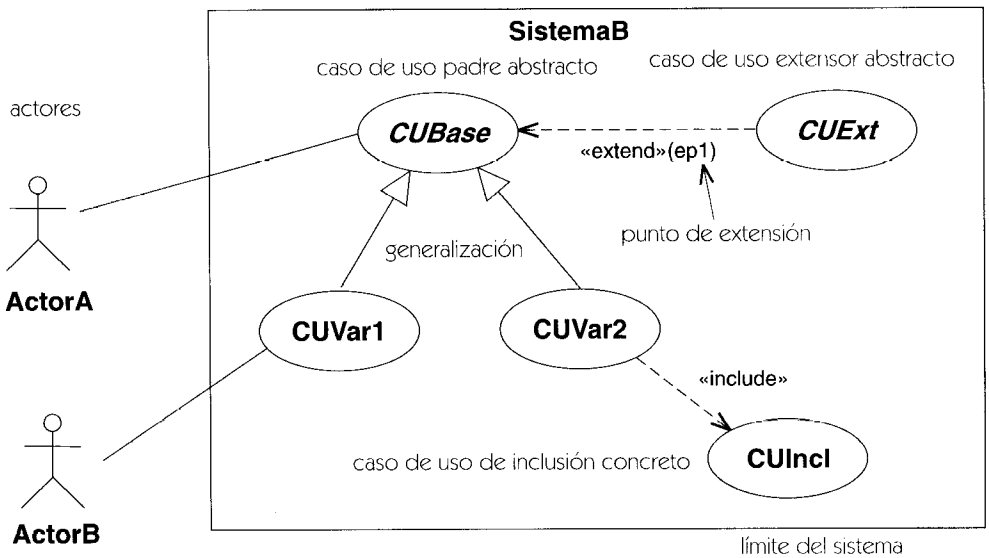


Figura B-10 Notación de los diagramas de caso de uso

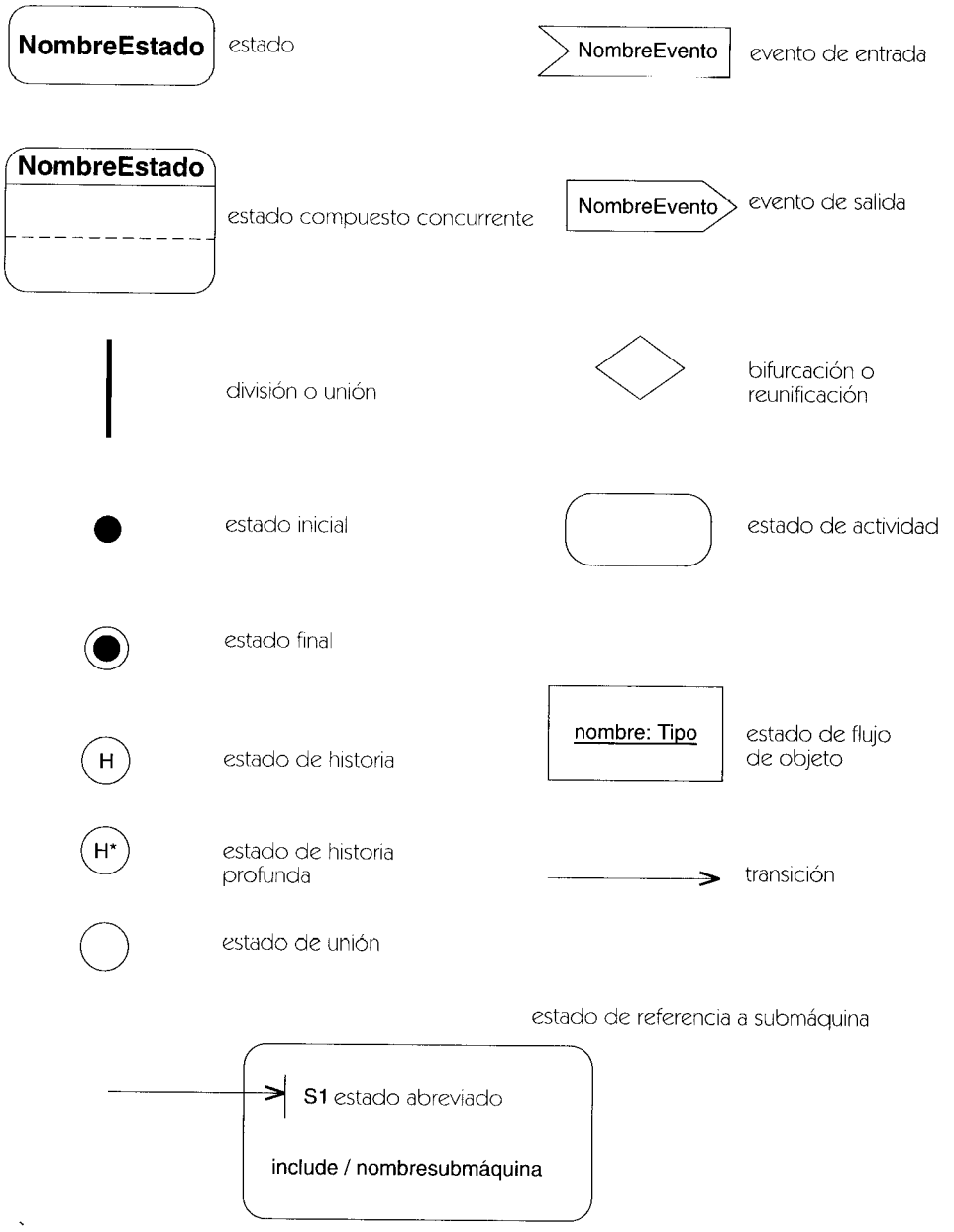


Figura B-11 Iconos de un diagrama de estados y diagramas de actividad

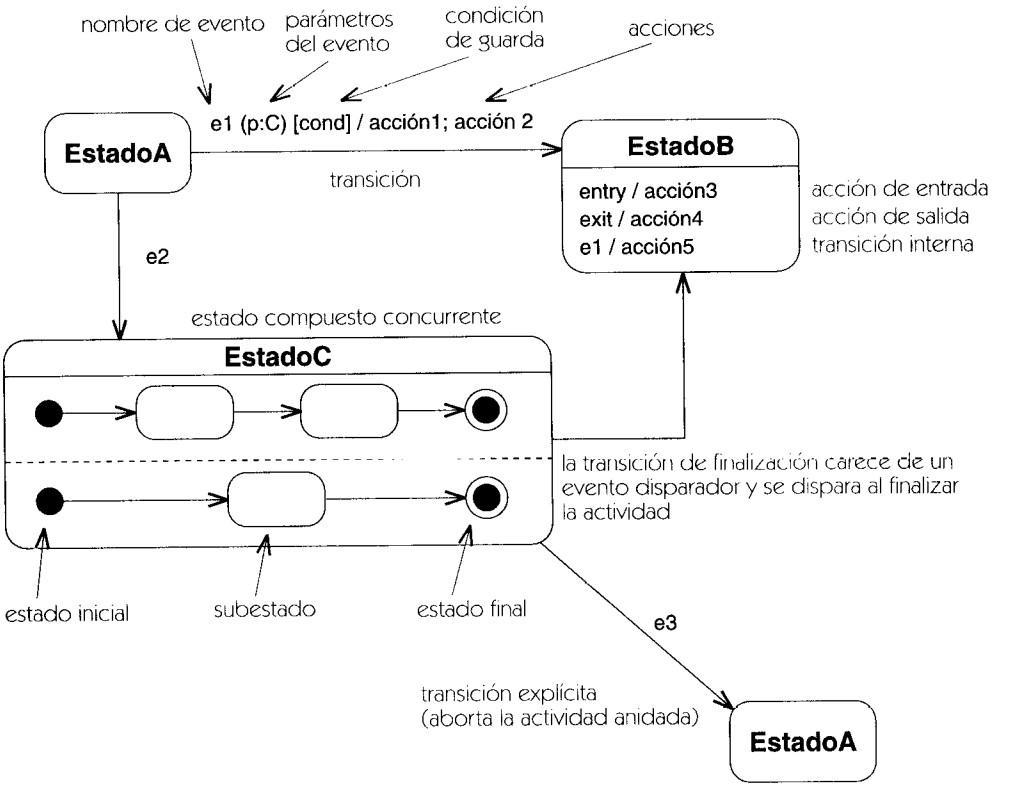


Figura B-12 Notación de diagramas de estados

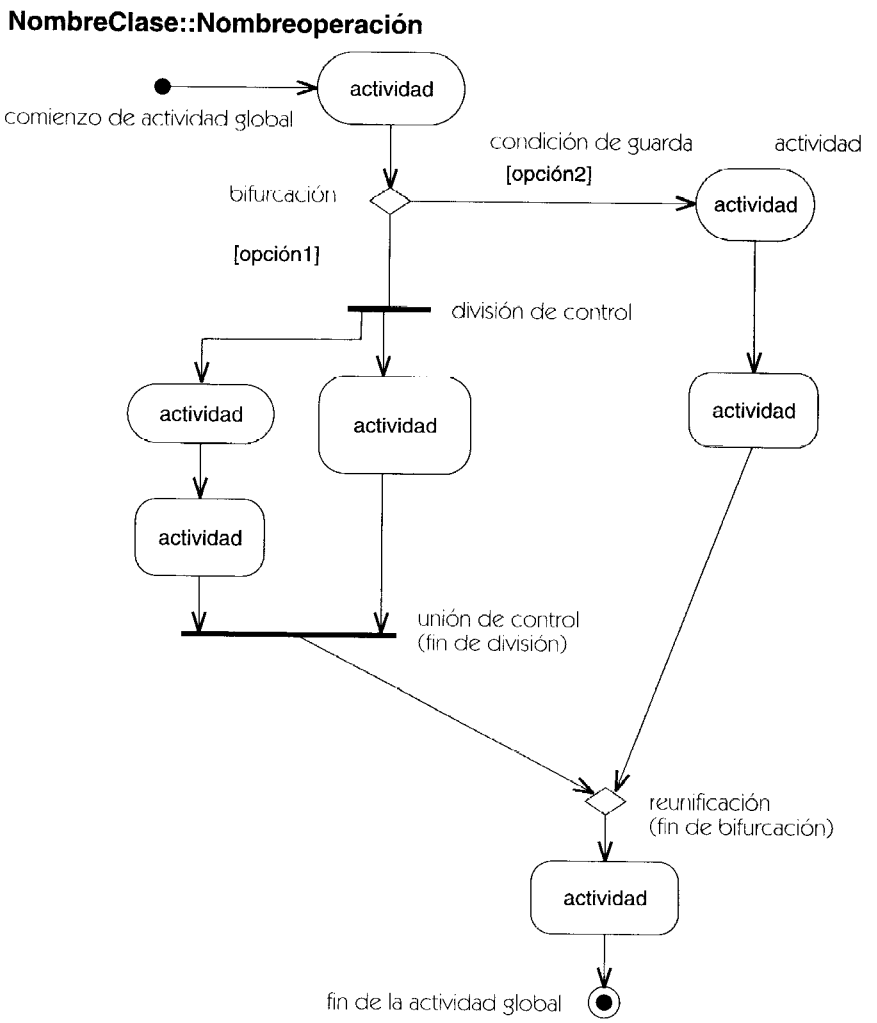


Figura B-13 Notación de los diagramas de actividad

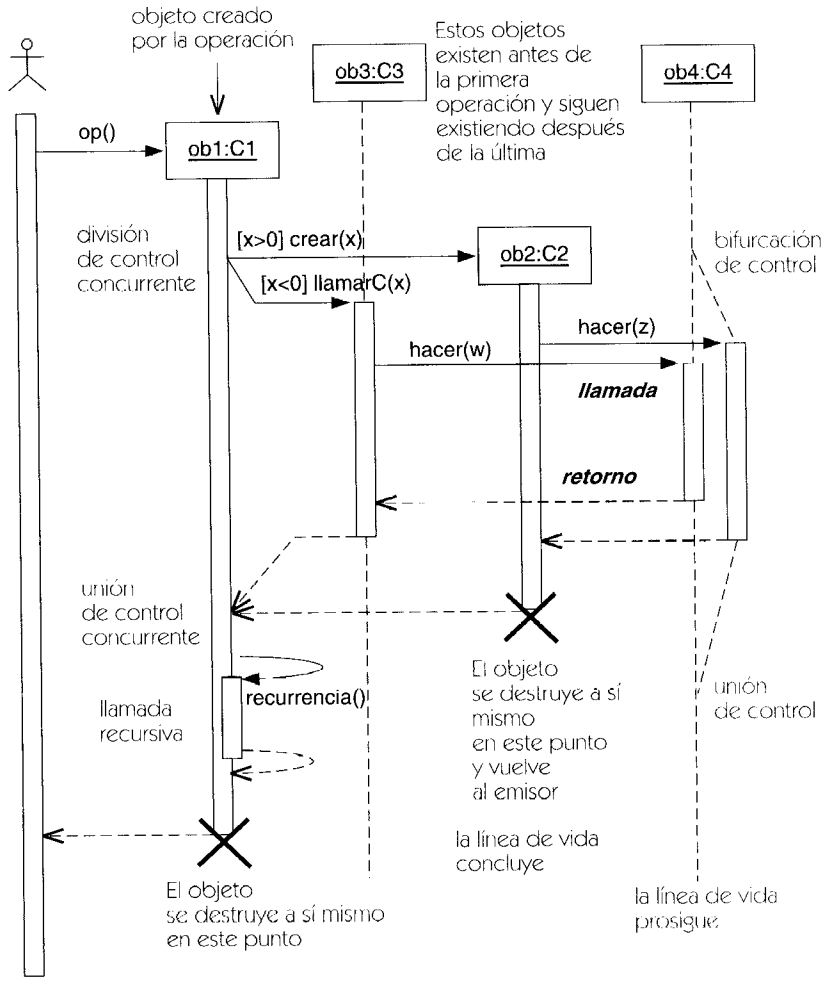


Figura B-14 Notación de diagramas de secuencia

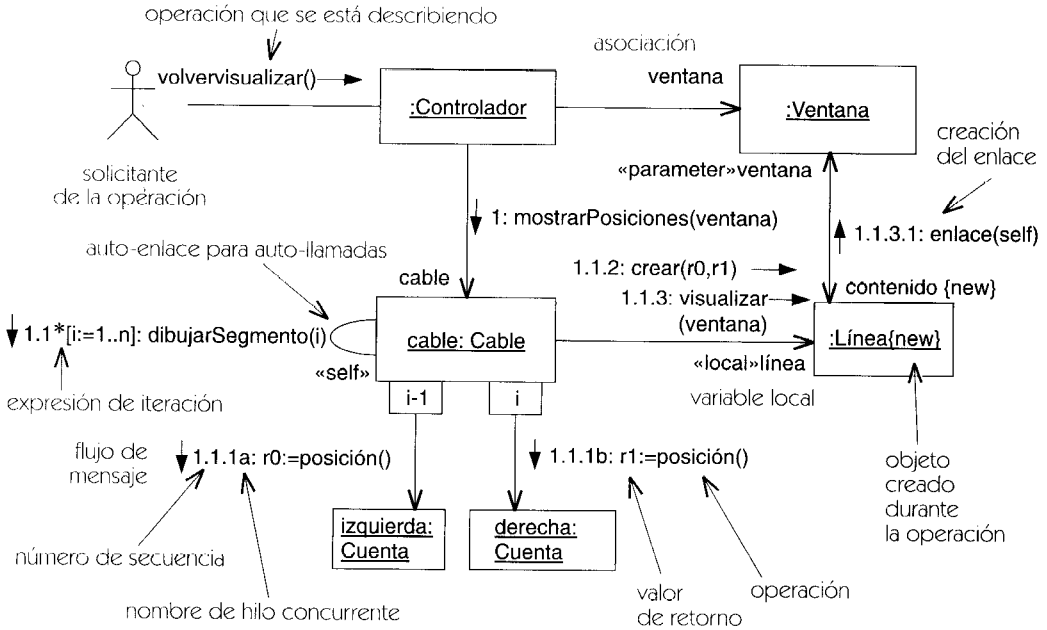


Figura B-15 Notación de diagrama de colaboración

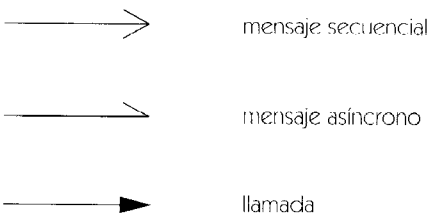


Figura B-16 Notación de mensajes

Apéndice C

Extensiones del proceso

Cómo personalizar UML

UML está diseñado para ser utilizable con múltiples propósitos. Aunque UML es un lenguaje universal que puede expresar un elevado número de propiedades fundamentales de modelado, existen ocasiones en que resulta deseable una jerga hecha a la medida de un determinado dominio de conocimiento. En los lenguajes naturales, la jerga, las germanías y los vocabularios especializados suelen facilitar la comunicación dentro de un arte o disciplina especializado, pero impiden la comunicación con los extraños. UML se puede personalizar de forma similar, empleando distintos mecanismos tales como convenciones de denominación, reglas de estilo, clases predefinidas y correspondencias implícitas con el software. En particular, se puede definir una extensión de UML empleando estereotipos, restricciones y valores etiquetados predefinidos.

UML posee muchas estructuras expresivas así que debería evitarse una extensión especializada a no ser que sea absolutamente necesaria. Por definición, una extensión sirve tan sólo a una comunidad limitada y puede dar lugar a malentendidos y confusiones por parte de los extraños. Sin embargo, si una comunidad destino está muy centrada, la potencia expresiva de una extensión puede compensar a veces la pérdida de uniformidad. Con el tiempo, algunas extensiones pueden ser tan útiles que acabarán por ser añadidas a UML; otras pueden caer en desuso.

Este capítulo describe dos extensiones que se adjuntan con los documentos de UML. Su utilización no es obligatoria y ni siquiera se recomienda necesariamente, pero indican la forma en que se puede definir una extensión para un proceso de desarrollo. En la literatura se ha descrito un cierto número de extensiones similares.

Extensiones del proceso de desarrollo de software

Estas extensiones están basadas en el proceso de desarrollo Objectory, que es un precursor del Proceso Unificado. Están destinadas a ser utilizadas en un proceso de desarrollo de software que consta de cuatro fases: captura de casos de uso, análisis, diseño e implementación.

Estas extensiones incluyen estereotipo de unidades de empaquetado, clases y asociación así como algunas restricciones relativas a la conexión de elementos. No hay etiquetas nuevas.

Estereotipos organizativos

La Tabla C-1 muestra los estereotipos organizativos, esto es, los estereotipos de modelo, clase y subsistema. Hay varios estereotipos aplicables a cada fase del desarrollo. En el proceso Objectory, cada fase posee un modelo distinto, con relaciones de traza entre los elementos de los diferentes modelos. Avanzando desde arriba hacia abajo, los términos sistema, subsistema y paquete de servicio describen capas de empaquetado.

Los estereotipos organizativos no tienen iconos especiales. Se representan mediante iconos de carpeta con el nombre del estereotipo entre comillas, tal y como aparece en la Figura C-1.

Figura C-1 Notación organizativa de estereotipos para el proceso de desarrollo de software

Tabla C-1 Estereotipos organizativos para el proceso de desarrollo de software

Clase Base	*Estereotipos por Fase del Proceso*			
	Caso de Uso	*Análisis*	*Diseño*	*Implementación*
modelo	modelo de caso de uso	modelo de análisis	modelo de diseño	modelo de implementación
paquete	sistema de caso de uso paquete de caso de uso			sistema de implementación subsistema de implementación
subsistema		sistema de análisis subsistema de análisis paquete de servicio	sistema de diseño subsistema de diseño paquete de servicio	

Estereotipos de clase

En las clases están definidos tres estereotipos: control, frontera y entidad. Se muestran en la Figura C-2.

Una *clase de control* describe objetos que administran interacciones, tales como un administrador de transiciones, un controlador de dispositivos o un monitor de un sistema operativo.

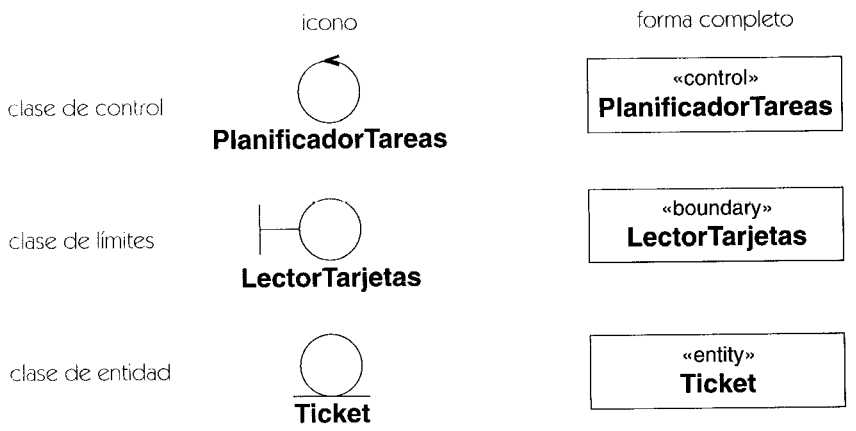


Figura C-2 Estereotipo de clase para el proceso de desarrollo de software

Tiene un comportamiento específico de un caso de uso. Normalmente, las clases de control no sobreviven a los casos de uso que admiten. Las clases de control se muestran como un círculo que tiene una punta de flecha.

Las *clases de frontera* describen objetos que median entre el sistema y los actores externos, tales como un formulario para hacer pedidos o un sensor. Se muestran como un círculo que tiene asociado un segmento en forma de T. Los objetos de frontera suelen existir durante toda la vida del sistema.

Las *clases de entidad* describen objetos pasivos. No inician las interacciones. Los objetos de entidad pueden participar en muchos casos de uso y normalmente sobreviven a las transiciones individuales. Se muestran como un círculo con una línea debajo.

Estereotipos de asociación

Existen dos estereotipos de asociación: comunicar y suscribir. La notación es una ruta de asociación con el nombre del estereotipo entre comillas.

Las *asociaciones de comunicación* conectan un actor con un caso de uso con el cual se comunica. Ésta es la única asociación posible entre actores y casos de uso, así que se puede omitir la palabra clave.

Las *asociaciones de suscripción* conectan la clase cliente (el suscriptor) con la clase proveedora (el editor). El suscriptor especifica un conjunto de eventos que pueden ser producidos por el editor. Cuando se produce uno de estos eventos, se le notifica al suscriptor.

Extensiones de modelado de negocios

Estas extensiones están destinadas a modelar organizaciones de negocios del mundo real y para comprender situaciones reales, más que para la implementación de software. Estos estereotipos no son necesarios para el modelado de negocios, pero abarcan algunas situaciones comunes.

Estas extensiones incluyen estereotipos de unidades de empaquetado, clases y asociaciones así como algunas restricciones relativas a la conexión de elementos. No hay etiquetas nuevas.

Estereotipos organizativos

La Tabla C-2 muestra los estereotipos organizativos correspondientes al modelado de procesos de negocios.

El modelo de caso de uso es un modelo de la vista de casos de uso. El sistema de casos de uso y el paquete de casos de uso son dos capas de organización de su contenido.

El modelo de objetos es un modelo de la estructura interna del sistema de negocios. El sistema objeto es su subsistema de más alto nivel, que contiene unidades organizativas y unidades de trabajo como capas inferiores. Una unidad organizativa corresponde a una unidad organizativa del negocio real, y una unidad de trabajo es un agrupamiento significativo, aun siendo de menor entidad.

No existen iconos especiales para la unidades organizativas del modelado de negocios; utilizan el símbolo de carpeta con un estereotipo entre comillas.

Tabla C-2 Estereotipos organizativos para el modelado de negocios

Clase base	*Captura de casos de uso*	*Modelo de Objetos*
modelo	modelo de casos de uso	modelo de objetos
paquete	sistema de caso de uso paquete de caso de uso	
subsistema		sistema de objetos unidad organizativa unidad de trabajo

Estereotipos de clase

Además de los actores (que están definidos en UML estándar), los objetos de negocios tiene varios sistemas de clase: trabajador, trabajador de caso, trabajador interno, entidad. Se muestran en la Figura C-3.

Un trabajador representa un ser humano que actúa en el interior del sistema. Un trabajador de caso es un trabajador que interacciona directamente con actores externos. Un trabajador interno es un trabajador que interacciona con trabajadores y entidades situadas dentro del sistema.

Las entidades de clase describen objetos pasivos. No inician las interacciones. Los objetos de entidad pueden participar en muchos casos de uso y normalmente sobreviven a interacciones individuales. En una situación de trabajo, las entidades suelen representar productos del trabajo.

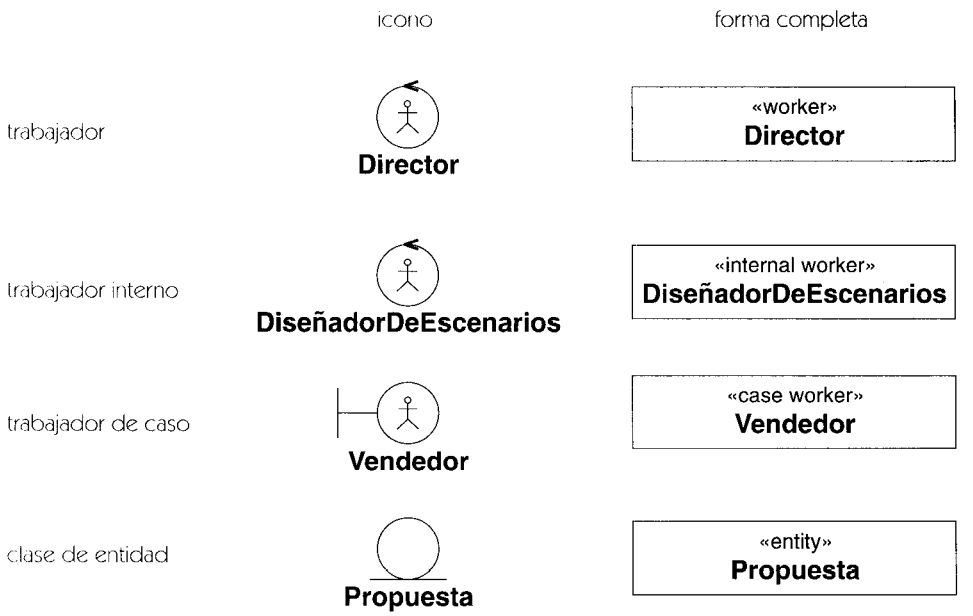


Figura C-3 Estereotipos de clase para el modelado de negocios

Estereotipos de asociación

Existen dos estereotipos de asociación: comunicar y suscribir. Son los mismos que para un proceso de desarrollo de software. La notación es una ruta de asociación con la palabra clave del estereotipo entre comillas.

Una asociación de comunicación conecta un actor con aquel caso de uso con el cual se comunica. Se trata de la única asociación existente entre actores y casos de uso, así que se puede omitir la palabra clave.

Una asociación de suscripción conecta una clase cliente (el suscriptor) con una clase proveedora (el editor). El suscriptor especifica un conjunto de eventos que puede producir el editor. Cuando se produce uno de estos eventos, se le comunica al suscriptor.

Bibliografía

[**Blaha-98**] Michael Blaha, William Premerlani. *Object-Oriented Modeling and Design for Database Applications*. Prentice Hall, Upper Saddle River, N.J., 1998.

[**Booch-91**] Grady Booch. *Object-Oriented Analysis and Design with Applications, 1.ª Edición Benjamin/Cummings*, Redwood City, Calif, 1991.

[**Booch-94**] Grady Booch. *Object-Oriented Analysis and Design with Applications, 2.ª Edición Benjamin/Cummings*, Redwood City, Calif., 1994.

[**Booch-96a**] Grady Booch. *Object Solutions: Managing the Object-Oriented Project*. Addison-Wesley, Menlo Park, Calif, 1996.

[**Booch-96b**] Grady Booch. *Best of Booch: Designing Strategies for Object Technology*. SIGS Books, Nueva York, N.Y., 1996.

[**Booch-99**] Grady Booch, James Rumbaugh, Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, Mass., 1999.

[**Buschmann-96**] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. *Pattern-Oriented Software Architecture: A System of Pattems*. Wiley, Chichester, U.K., 1996.

[**Coad-91**] Peter Coad, Edward Yourdon. *Object-Oriented Analysis, 2.ª Edición* Yourdon Press, Englewood Cliffs, N.J., 1991.

[**Coleman-94**] Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilchrist, Fiona Hayes, Paul Jeremaes. *Object-Oriented Development: The Fusion Method*. Prentice Hall, Englewood Cliffs, N.J., 1994.

[**Cox-86**] Brad J. Cox. *Object-Oriented Programming.-An Evolutionary Approach*. Addison-Wesley, Reading, Mass., 1986.

[**Embley-92**] Brian W. Embley, Barry D. Kurtz, Scott N. Woodfield. *Object-Oriented Systems Analysis: A Model-Driven Approach*. Yourdon Press, Englewood Cliffs, N.J., 1992.

[**Gamma-95**] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Mass., 1995.

[**Goldberg-83**] Adele Goldberg, David Robson. *Smalltalk-80: The Language and Its Implementación*. Addison-Wesley, Reading, Mass., 1983.

[**Harel-98**] David Harel, Michal Politi. *Modeling Reactive Systems With Statecharts: The STATEMATE Approach*. McGraw-Hill, Nueva York, N.Y., 1998.

[**Jacobson-92**] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Overgaard. *Object- Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Wokingham, Inglaterra, 1992.

[**Jacobson-95**] Ivar Jacobson, Maria Ericsson, Agneta Jacobson. *The Object Advantage: Business Process Reengineering with Object Technology*. Addison-Wesley, Wokingham, Inglaterra, 1995.

[**Jacobson-97**] Ivar Jacobson, Martin Griss, Patrik Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success*. Addison-Wesley, Harlow, Inglaterra, 1997.

[**Jacobson-99**] Ivar Jacobson, Grady Booch, James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, Reading, Mass., 1999.

[**Martin-92**] James Martin, James Odell. *Object-Oriented Analysis and Design*. Prentice Hall, Englewood Cliffs, N.J., 1992.

[**Meyer-88**] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, Nueva York, N.Y., 1988.

[**Rumbaugh-91**] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen. *Object- Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, N.J., 1991.

[**Rumbaugh-96**] James Rumbaugh. *OMT Insights: Perspectives on Modeling from the Journal of Object-Oriented Technology*. SIGS Books, Nueva York, N.Y., 1996.

[**Rumbaugh-99**] James Rumbaugh, Ivar Jacobson, Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, Mass., 1999.

[**Selic-94**] Bran Selic, Garth Gullekson, Paul T. Ward. *Real-Time Object-Oriented Modeling*. Wiley, Nueva York, N.Y., 1994.

[**Shlaer-88**] Sally Shlaer, Stephen J. Mellor. *Object-Oriented Systems Analysis: Modeling the World in Data*. Yourdon Press, Englewood Cliffs, N.J., 1988.

[**Shlaer-92**] Sally Shlaer, Stephen J. Mellor. *Object Lifecycles: Modeling the World in States*. Yourdon Press, Englewood Cliffs, N.J., 1992.

[**UML-98**] *Unified Modeling Language Specification*. Object Management Group, Framingham, Mass., 1998. Internet: www.omg.org.

[**Ward-85**] Paul Ward, Stephen J. Mellor. *Structured Development for Real-Time Systems: Introduction and Tools*. Yourdon Press, Englewood Cliffs, N.J., 1985.

[**Warmer-99**] Jos B. Warmer, Anneke G. Kleppe. *The Object Restriction Language: Precise Modeling with UML*. Addison-Wesley, Reading, Mass., 1999.

[**Wirfs-Brock-90**] Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener. *Designing Object-Oriented Software*. Prentice Hall, Englewood Cliffs, N.J., 1990.

[**Yourdon-79**] Edward Yourdon, Larry L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Yourdon Press, Englewood Cliffs, N.J., 1979.

Índice alfabético

Las referencias principales aparecen en negrita.

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

JAMES RUMBAUGH • IVAR JACOBSON • GRADY BOOCH

EL LENGUAJE UNIFICADO DE MODELADO. MANUAL DE REFERENCIA

Los autores han hecho un trabajo excepcional con este libro de UML. Las definiciones de los términos son de lo mejor que he visto. La organización y material de la enciclopedia son fantásticos.

— **Perry Cole**, MCIWorldCom

El Lenguaje Unificado de Modelado (UML) se ha convertido rápidamente en una notación estándar para el modelado de sistemas con gran cantidad de software. Este libro proporciona la descripción definitiva de **UML**, realizada por sus creadores originales, James Rumbaugh, Ivar Jacobson y Grady Booch. Este es un libro útil para todos aquellos que trabajan en captura de requisitos, para los que desarrollan una arquitectura software, o para los que desean diseñar la implementación o comprender un sistema ya existente.

La mayor parte de este libro es sólo, una lista ordenada alfabéticamente de entradas que cubren cada uno de los aspectos de **UML**, permitiendo tanto una rápida referencia como un estudio detallado. Este formato permite cubrir completamente los detalles de **UML** y los temas de más alto nivel, sin confundir al lector con constantes cambios de nivel. La primera parte del libro es un resumen de los conceptos de **UML** organizados por área temática, que proporciona una introducción a **UML** para los principiantes, a la vez que permite ser punto de entrada hacia temas más detallados.

Algunos de los puntos más importantes del libro son:

- ✦ Gran cantidad de diagramas, con muchas anotaciones aclaratorias.
- ✦ Tratamiento a fondo, tanto de la semántica como de la notación, separados en entradas diferentes para permitir una más fácil y rápida referencia.
- ✦ Amplias explicaciones de aquellos conceptos cuyo significado o propósito no está muy claro en las especificaciones formales.
- ✦ Secciones de discusión que ofrecen consejos de utilización e ideas adicionales sobre conceptos complicados.
- ✦ Un CD-ROM con la versión del libro en inglés en formato Acrobat Reader, un excelente recurso para navegar o buscar en el texto información específica.
- ✦ Texto completo en el CD- ROM de los documentos de la especificación de **UML** versión 1.3 de junio de 1999, cortesía del Object Management Group.
- ✦ Resumen en CD-ROM de la notación, con hiperenlaces a cada entrada individual.

James Rumbaugh, **Ivar Jacobson** y **Grady Booch** son los diseñadores originales del **Lenguaje Unificado de Modelado (UML)**. Son mundialmente reconocidos por sus significativas contribuciones al desarrollo de la tecnología de objetos, entre las que se encuentra el Proceso Objectory (OOSE) el Método Booch, y la Técnica de Modelado de Objetos (OMT), actualmente los tres trabajan en Rational Software Corporation.

Obtenga más información sobre la serie Object Technology Series en http://www.awl.com/cseng/otseries

ISBN 84-7829-037-0

Pearson Educación