EL LENGUAJE UNIFICADO DE MODELADO MANUAL DE REFERENCIA

BOOCH
JACOBSON
RUMBAUGH

ADDISON-WESLEY
OBJECT TECHNOLOGY
SERIES
SERIES EDITORS

UML 2.0

2ª EDICIÓN

James Rumbaugh Ivar Jacobson Grady Booch

Aprenda UML directamente de sus creadores

PEARSON
Addison Wesley

EL LENGUAJE UNIFICADO DE MODELADO MANUAL DE REFERENCIA

Segunda edición

EL LENGUAJE UNIFICADO DE MODELADO MANUAL DE REFERENCIA

Segunda edición

James RUMBAUGH
Ivar JACOBSON
Grady BOOCH

Traducción

Héctor Castán Rodríguez
Óscar Sanjuán Martínez
Mariano de la Fuente Alarcón
Facultad de Informática
Universidad Pontificia de Salamanca (Campus de Madrid)

Coordinación general y Revisión técnica

Luis Joyanes Aguilar
Facultad de Informática
Universidad Pontificia de Salamanca (Campus de Madrid)

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. Segunda edición
PEARSON EDUCACIÓN, S.A, Madrid, 2007

ISBN: 978-84-7829-087-1
Materia: Informática 004

Formato 195 x 250 Páginas 688

J. Rumbaugh, I. Jacobson, G. Booch
EL LENGUAJE UNIFICADO DE MODELADO.
MANUAL DE REFERENCIA. SEGUNDA EDICIÓN

ISBN: 978-84-7829-087-1
Depósito Legal:

Authorized translation from the English language edition, entitled UNIFIED MODELING LANGUAGE REFERENCE MANUAL, THE, 2nd Edition, 0321245628 by RUMBAUGH, JAMES; JACOBSON, IVAR; BOOCH, GRADY, published by Pearson Education, Inc, publishing as Addison Wesley Professional, Copyright©2005

SPANISH language edition published by PEARSON EDUCATION, S.A.,; Copyright©2007

Equipo editorial:
Editor: Miguel Martín-Romo
Técnico Editorial: Marta Caicoya
Equipo de producción:
Director: José Antonio Clares
Técnico: José Antonio Hernán
Composición: Ángel Gallardo Servicios Gráficos, S.L.
Diseño de cubierta: Equipo de diseño de Pearson Educación, S. A.
Impreso por:

IMPRESO EN ESPAÑA - PRINTED IN SPAIN

Este libro ha sido impreso con papel y tinta ecológicos

Para Madeline, Nick y Alex

—Jim

Contenido

Prefacio

Objetivos

Este libro tiene la intención de ser una referencia útil y completa sobre el Lenguaje Unificado de Modelado (UML) para el desarrollador, arquitecto, gestor de proyecto, ingeniero de sistemas, programador, analista, contratista, cliente, y para cualquiera que necesite especificar, diseñar construir o comprender sistemas software complejos. Proporciona una referencia completa sobre los conceptos y construcciones de UML, incluyendo su semántica, notación y propósito. Está organizado para ser una referencia práctica, pero minuciosa, para el desarrollador profesional. También intenta ofrecer detalles adicionales sobre temas que pueden no estar claros en la documentación estándar y ofrece una base lógica para muchas decisiones que se tomaron en UML.

Este libro no pretende ser una guía sobre la documentación estándar o sobre la estructura interna del metamodelo contenida en ella. Los detalles del metamodelo son de interés para los metodólogos y para los constructores de herramientas UML, pero la mayoría de los desarrolladores no necesitan de los misteriosos detalles de la documentación del Object Management Group (OMG). Este libro proporciona todos los detalles de UML que la mayoría de los desarrolladores necesita; en muchos casos, proporciona información explícita que, de lo contrario, habría que buscar entre líneas en la documentación original. Para aquéllos que deseen consultar la información original, ésta se encuentra disponible en el sitio web del OMG (www.omg.org).

Este libro está pensado como una referencia para aquéllos que ya tienen algún conocimiento sobre la tecnología de orientación a objetos. Para los principiantes, en la bibliografía hay una lista de libros escrita por nosotros y por otros autores; aunque la notación ha cambiado, libros como [Rumbaugh-91], [Jacobson-92], [Booch-94] y [Meyer-88]* proporcionan una introducción a los conceptos de la orientación a objetos que continúa siendo válida y que, por tanto, no es necesario duplicar aquí. [Blaha-05] actualiza [Rumbaugh-91] utilizando la notación de UML. Para una introducción a UML que muestre cómo modelar varios problemas comunes, véase *Guía de Usuario del Lenguaje Unificado de Modelado* [Booch-99] o *UML Distilled* [Fowler-04]*.

UML no exige un proceso de desarrollo concreto. Aunque UML puede utilizarse con distintos procesos de desarrollo, fue diseñado para servir de apoyo a un proceso interactivo, incremental, guiado por casos de uso y centrado en la arquitectura, el tipo de desarrollo que nosotros consideramos que es más apropiado para el desarrollo de sistemas complejos modernos. Para

---

* *N. del T.:* Todos estos libros están traducidos al castellano por Pearson Educación, S. A.

situar UML en su contexto como herramienta para el desarrollo de software, este libro define las etapas de tal proceso, aunque no son parte del estándar UML. *El Proceso Unificado de Desarrollo de Software* [Jacobson-99] describe en detalle el tipo de proceso que nosotros creemos que complementa UML y que ayuda a mejorar el desarrollo de software.

Segunda edición y versión de UML

Esta segunda edición ha sido extensamente modificada a partir de la primera edición que fue publicada en 1999. Esta edición está basada en la especificación de UML versión 2.0 "adoptada" por el OMG, que prevé cambios sobre la especificación "diponible" que está siendo preparada por una Finalization Task Force del OMG.

La documentación original de la especificación y la información actualizada sobre el trabajo relativo a UML y a temas relacionados puede encontrarse en el sitio web del OMG en www.omg.org.

Manual de referencia y especificación del OMG

UML es un lenguaje de modelado extenso con diversas características. Un manual de referencia que se limite a repetir la documentación original de la especificación no ayuda demasiado a los lectores. De la misma forma que en un diccionario o en una enciclopedia, hemos resumido la información de la forma más clara posible al tiempo que reducíamos el volumen de material incluido. Frecuentemente hemos elegido hacer hincapié en los usos comunes, omitiendo los oscuros casos especiales o los significados redundantes de representación de algunos conceptos. Esto no significa que estas técnicas carezcan de utilidad, sino que la mayoría de los lectores pueden ser capaces de obtener resultados sin utilizarlos. No obstante, el *Manual de referencia* no debe ser considerado cono la autoridad última sobre el lenguaje UML. Como en cualquier estándar, la autoridad final reside en las especificaciones oficiales, que deben ser consultadas para solucionar cualquier conflicto.

Hemos intentado seguir los siguientes principios:

- Explicar el propósito principal de un concepto sin perdernos en los detalles de la representación del metamodelo.
- Evitar discusiones sobre metaclases abstractas. Los modeladores deben, en última instancia, utilizar metaclases concretas, las cuales pueden ser descritas de una forma más sencilla si las capas abstractas internas son colapsadas.
- Evitar discusiones sobre el empaquetamiento del metamodelo. Los paquetes deben ser importantes para los constructores de herramientas, pero los modeladores no necesitan tener conocimiento sobre ellos la mayor parte del tiempo. En cualquier caso, si necesita tener conocimiento, necesita echar un vistazo en detalle a la especificación.
- Describir conceptos a partir de la especificación completa. La especificación del OMG tiene varias capas intermedias y puntos de conformidad que complican enormemente la compresión de UML. Describimos UML con todas sus características. Si su herramienta no

implementa todas las posibilidades, entonces carecerá de algunas de las características, aunque no le hace daño tener conocimiento de ellas.

- Describir conceptos desde el punto de vista de su uso habitual. A menudo la especificación del OMG se toma muchas molestias para expresar conceptos de forma general. Esto es propio de una especificación, pero consideramos que los lectores a menudo comprenden mejor los conceptos si se presentan en un contexto específico y son generalizados después. Si está preocupado por la aplicación de un concepto en una situación compleja y ambigua y considera que la explicación del *Manual de referencia*, puede ser inadecuada, revise la especificación original. Sin embardo, desafortunadamente incluso la especificación del OMG es ambigua en situaciones complejas.

Esquema general del libro

El *Manual de referencia de UML* se encuentra organizado en cuatro partes: (1) una descripción de la historia de UML y del modelado, (2) una visión de conjunto de los conceptos de UML, (3) un diccionario alfabético de los términos y conceptos de UML, y (4) unos breves apéndices.

La primera parte es una vista general de UML —su historia, propósitos y usos— para ayudarle a entender el origen de UML y las necesidades que intenta cubrir.

La segunda parte es una revisión de los conceptos de UML, de forma que pueda tener una perspectiva de todas sus características. Esta 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 contiene un capítulo para cada tipo de vista de UML. Esta visión no pretende ser un curso completo ni una extensa descripción de conceptos. Sirve principalmente para resumir y relacionar varios conceptos de UML, proporcionando puntos de partida para lecturas detalladas del diccionario.

La tercera parte contiene el material de referencia organizado para acceder fácilmente a cada tema. La mayor parte del libro es un diccionario alfabético de todos los conceptos y construcciones de UML. Cada término de cierta importancia en UML tiene su propia entrada en el diccionario. El diccionario pretende ser completo, por lo que todos los conceptos vistos de forma general en la Parte 2, son repetidos con mayor detalle en el diccionario. En algunos casos se ha repetido la misma o similar información en varias partes del diccionario, de forma que el lector pueda encontrarla cómodamente. Se han incluido algunos términos comunes de la orientación a objetos, que no son conceptos oficiales de UML, para proporcionar un contexto en los ejemplos y discusiones.

Los apéndices muestran el metamodelo de UML y un resumen de la notación de UML. Hay una bibliografía concisa de los principales libros sobre la orientación a objetos, pero no intenta ser una guía completa sobre las fuentes de ideas de UML y otras aproximaciones. Muchos de los libros de la bibliografía contienen listas excelentes de referencias a libros y artículos para aquéllos que estén interesados en seguir el desarrollo de las ideas.

Convenciones en el formato de las entradas del diccionario

El diccionario está organizado como una lista alfabética de entradas, cada una de las cuales describe un concepto con mayor o menor detalle. Los artículos representan una lista plana de conceptos UML a distintos niveles conceptuales. Un concepto de alto nivel contiene típicamente un resumen de sus conceptos subordinados, cada uno de los cuales se describe completamente en un artículo separado. Los artículos tienen muchas referencias cruzadas. La organización plana del diccionario permite presentar la descripción de cada concepto con un nivel bastante uniforme de detalle, sin los continuos cambios de nivel que serían necesarios en una representación secuencial de las descripciones anidadas. El formato de hipertexto del documento también debería hacerlo útil como referencia. No debería ser necesario utilizar el índice en exceso; en su lugar, para cualquier término de interés, vamos directamente al artículo principal en el diccionario y seguimos las referencias cruzadas. Este formato no es necesariamente el ideal para el aprendizaje del lenguaje; se aconseja a los principiantes leer la descripción general de UML que se encuentra en la Parte 2 o leer libros introductorios sobre UML, como la *Guía de usuario de UML [Booch-99]*.

Los artículos del diccionario tienen las siguientes divisiones, aunque no todas las divisiones aparecen en todos los artículos.

Palabra clave y breve descripción

El nombre del concepto aparece en negrita, colocado a la izquierda del texto del artículo. A continuación le sigue una breve descripción en letra normal. Esta descripción tiene como propósito capturar la idea principal del concepto, pero puede simplificar el concepto para su presentación concisa. Es necesario remitirse al artículo principal para una semántica precisa.

Los estereotipos predefinidos son incluidos como entradas. Un comentario entre paréntesis después del estereotipo identifica el elemento de modelado al cual puede ser aplicado.

Semántica

Esta sección contiene una descripción detallada del significado del concepto, incluyendo restricciones en su uso y las consecuencias de su aplicación. Esta sección no incluye la notación, aunque los ejemplos utilizan la notación adecuada. En primer lugar se proporciona la semántica general. Para aquellos conceptos que tienen propiedades estructurales subordinadas, a la semántica general le sigue una lista con dichas propiedades, a menudo bajo el título *Estructura*. En la mayoría de los casos, las propiedades se muestran como una tabla ordenada alfabéticamente por el nombre de la propiedad, con la descripción de cada una a su derecha. Si una propiedad tiene una lista determinada de opciones, éstas se proporcionan mediante una sublista sangrada. En casos más complejos, la propiedad tiene su propio artículo para evitar anidamientos excesivos. Cuando las propiedades necesitan una explicación mayor que la permitida en una tabla, éstas son descritas mediante cabeceras en negrita y cursiva. En algunos casos, el concepto principal se describe mejor mediante varias subdivisiones lógicas en lugar de con una lista. En estos casos, las secciones adicionales se encuentran a continuación o reemplazan a la subsección *Estructura*. Aunque se han usado varios mecanismos de organización, su estructura debería ser obvia para el lector. Los nombres de las propiedades se indican mediante lenguaje corriente en lugar de utilizar los identificadores internos del metamodelo de UML, con la intención de que la correspondencia sea obvia.

Formato de las entradas del diccionario

nombre de la entrada

Una breve descripción del concepto en una o dos frases.

Véase también concepto relacionado.

Semántica

Una descripción de la semántica en varios párrafos.

Estructura

Una lista de conceptos subordinados dentro del concepto principal.

elemento	Descripción de un elemento. Normalmente los nombres del metamodelo de UML son convertidos en lenguaje sencillo.
elemento enumerado	Una enumeración con varios valores. Lista de valores:
valor	El significado de este valor para este elemento.

Otro elemento. Los temas más complicados se describen en párrafos separados.

Ejemplo

Se puede incluir un ejemplo dentro de la semántica, de la notación o de forma independiente.

Notación

Descripción de la notación que normalmente incluye un diagrama o sintaxis.

Opciones de presentación

Describe variantes en la notación, normalmente opcionales.

Pautas de estilo

Plantea prácticas recomendadas aunque no son obligatorias.

Discusión

Las opiniones del autor o explicaciones de fondo más allá de UML.

Historia

Cambios con respecto a UML versión 1.x.

entrada de estereotipo (estereotipo de Clase)

Descripción del significado del estereotipo.

Notación

Esta sección contiene una descripción detallada de la notación del concepto. Normalmente, la sección de notación tiene un formato semejante a la sección precedente de semántica, a la cual referencia, y a menudo tiene las mismas divisiones. La sección de notación a menudo incluye uno o más diagramas para ilustrar el concepto. La notación real está impresa en negro. Para ayudar al lector a entender la notación, muchos diagramas contienen anotaciones en negrita cursiva. Cualquier material en negrita cursiva es un comentario y no es parte de la notación real.

Pautas de estilo

Esta sección opcional describe convenciones de estilo comunes. Estas no son obligatorias, pero las sigue la propia especificación de UML. También se pueden proporcionar opciones o pautas de presentación en una sección aparte.

Ejemplo

Esta subsección contiene ejemplos de la notación o del uso del concepto. A menudo, los ejemplos también tratan situaciones complicadas o potencialmente confusas. Si los ejemplos fueran breves, pueden ser incluidos dentro de otra sección.

Discusión

Esta sección describe asuntos sutiles, clarifica los puntos difíciles y habitualmente confusos y contiene otros detalles que, de otra forma serían tratados en una sección semántica más descriptiva. Hay pocos artículos que dispongan de una sección de discusión.

Esta sección también explica ciertas decisiones de diseño que fueron tomadas durante el desarrollo de UML, en particular aquellas que pueden parecer menos intuitivas o que han provocado más controversia. Las sencillas diferencias de opinión generalmente no se contemplan.

Algunas veces expresamos una opinión sobre el valor (o la falta de él) de ciertos conceptos. Reconocemos que otros pueden estar en desacuerdo con estas valoraciones. Hemos intentado restringir las opiniones a la sección de discusión.

Historia

Esta sección describe los cambios entre UML1 y UML2, incluyendo en algunos casos las razones de dichos cambios. Los cambios menores no suelen ser enumerados. La ausencia de esta sección no implica que no se hayan dado cambios.

Convenciones de sintaxis

Expresiones sintácticas. Las expresiones sintácticas se dan en formato BNF escrito con formato de fuente sans serif (Myriad). Los valores literales que aparecen en la frase destino aparecen impresos en negro, mientras que los nombres de las variables sintácticas y de los operadores sintácticos especiales aparecen impresos en ***`negrita cursiva`***.

El texto impreso en negro aparece literalmente en la cadena destino.

Las marcas de puntuación (impresas siempre en negro) aparecen literalmente en la cadena destino.

Cualquier palabra impresa en **`courier negrita`** representa una variable que debe ser reemplazada por otra cadena o por otra producción sintáctica en la cadena destino. Las palabras pueden contener letras y guiones. Si una palabra en **`courier negrita`** aparece en cursiva o subrayada, la palabra real también debe aparecer en cursiva o subrayada.

En los ejemplos de código, los comentarios se muestran en **`courier negrita`** a la derecha del texto del código.

Los subíndices y los paréntesis en L se utilizan como operadores sintácticos de la siguiente forma:

`expresión`$_{\texttt{opc}}$	La expresión es opcional.
`expresión`$_{\texttt{lista,}}$	Puede aparecer una lista de expresiones separada por coma. Si hay cero o una repetición no se utiliza el separador. Si aparece un signo de puntuación distinto de la coma en el subíndice se utiliza dicho signo como separador.
$\lfloor$**`=expresión`**$\rfloor_{\texttt{opc}}$	Una pareja de paréntesis en L enlaza dos o más términos que son considerados una unidad para optatividad o la repetición de ocurrencias. En este ejemplo, el signo igual y la expresión forman una unidad que puede ser omitida o incluida.

Se evita que existan dos niveles de anidamiento. La sintaxis especialmente enrevesada puede simplificarse algo para su presentación, pero en cualquier caso el uso de este tipo de sintaxis tiende a ser confusa para las personas y debe ser evitada.

Cadenas de literales. El texto ejecutable, las palabras clave del lenguaje, los nombres de elementos del modelo y las cadenas de ejemplo de los modelos se muestran con formato de fuente sans serif (Myriad).

Diagramas. En los diagramas, el texto y las flechas en negrita cursiva son anotaciones, es decir, explicaciones sobre la notación de los diagramas que no deben aparecer en diagramas reales. Cualquier texto y símbolo en negro 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® Reader® (PDF). Mediante el uso del Adobe Reader el lector puede buscar en el libro de una forma sencilla una palabra o frase. La versión en CD también contiene una tabla de contenidos cuyos términos son enlaces al texto donde aparecen definidos, un índice, marcadores Adobe Reader y gran cantidad de hiperenlaces (en rojo) en el cuerpo de los artículos. Basta con pulsar en uno de estos para saltar al artículo del diccionario para encontrar esa palabra o frase. Esperamos que este CD será una referencia útil para los lectores avanzados.

Los creadores de UML

Queremos dar las gracias a los muchos colaboradores que han construido la especificación de UML a lo largo de años de reuniones, acaloradas discusiones, redacción e implementación

de ideas. La lista de colaboradores ha crecido significativamente desde UML1, y la especificación del OMG ya no enumera a los colaboradores principales, cuyo número oscila entre veinte y cincuenta dependiendo del umbral para su inclusión, e incluso más si se incluye el trabajo que ha influenciado UML. Ya no parece posible confeccionar una lista completa sin pasar por alto a muchas personas.

Sobre todo, queremos elogiar a los cientos de personas que han contribuido a la comunidad de ideas de la que UML ha partido —ideas sobre tecnología de orientación a objetos, metodología de software, lenguajes de programación, interfaces de usuario, programación visual y otras numerosas áreas de la ciencia de la computación. Es imposible enumerarlos a todos, o incluso seguirle la pista a las principales cadenas de influencia, sin un enorme esfuerzo de estudio, teniendo en cuenta que este es un libro de ingeniería y no un estudio histórico. Muchos de ellos son sobradamente conocidos, pero muchas buenas ideas también provienen de aquéllos que no tienen la buena fortuna de volverse conocidos. La bibliografía incluye algunos de los libros menos conocidos que influyeron en los autores.

Agradecimientos

Nos gustaría agradecer a los revisores que han hecho posible este libro. Entre ellos, en esta segunda edición, se encuentran Conrad Bock, Martin Gogolla, Øystein Haugen, Birger Møller-Pedersen y Ed Seidewitz. De la primera edición, hemos recibido observaciones de Mike Blaha, Conrad Bock, Perry Cole, Bruce Douglass, Martin Fowler, Eran Gery, Pete McBreen, Gunnar Övergaard, Karin Palmkvist, Guus Ramackers, Tom Schultz, Ed Seidewitz y Bran Selic.

En un tono más personal, me gustaría dar las gracias al Profesor Jack Dennis, que inspiró mi trabajo sobre modelado y el de otros muchos estudiantes hace más de treinta años. Las ideas de su Computations Structures Group (grupo de estructuras de computación) en el MIT han dado muchos frutos y no son las menores de las fuentes de UML. También quiero dar las gracias a Mary Loomis y Ashwin Shah, con los que desarrollé las ideas originales de OMT y a mis anteriores colegas en el GE R&D Center, Mike Blaha, Bill Premerlani, Fred Eddy y Bill Lorensen, con los que escribí el libro de OMT.

Por último, sin la paciencia de mi mujer, Madeline, y de mis hijos, Nick y Alex, no hubiera existido UML ni ningún libro sobre él.

James Rumbaugh
Cupertino, California
Junio 2004

Parte 1: Antecedentes

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

1 Perspectiva general de UML

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

Breve resumen de UML

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual de propósito general que se utiliza para especificar, visualizar, construir y documentar los artefactos de un sistema software. Captura decisiones y conocimiento sobre sistemas que deben ser construidos. Se usa para comprender, diseñar, ojear, configurar, mantener y controlar la información sobre tales sistemas. Está pensado pasa ser utilizado 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 las técnicas de modelado e incorporar las mejores prácticas de software actuales en una aproximación estándar. UML incluye conceptos semánticos, notación y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Está pensado para ser apoyado por herramientas de modelado visuales e interactivas que dispongan de generadores, tanto de código, como 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 existentes.

UML capta la información sobre la estructura estática y el comportamiento dinámico del sistema. Un sistema es modelado como una colección de objetos discretos que interactúan para realizar un trabajo que en última instancia beneficia a un usuario externo. La estructura estática define 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 a lo largo del tiempo y la comunicación entre objetos para cumplir los objetivos. El modelado de 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 con las que se pueda trabajar, comprender y controlar las dependencias entre paquetes y gestionar las versiones de las unidades del modelo, en un entorno de desarrollo complejo. Contiene construcciones para representar las decisiones de implementación y para organizar elementos de tiempo de ejecución en componentes.

Ante todo, UML no es un lenguaje de programación. Puede ser utilizado para escribir programas, pero carece de las facilidades sintácticas y semánticas que proporcionan la mayoría de

los lenguajes de programación para facilitar la tarea de programar. Las herramientas pueden proporcionar generadores de código para UML sobre diversos lenguajes de programación, así como construir modelos de ingeniería inversa a partir de programas existentes. UML no es un lenguaje altamente formal pensado para demostrar teoremas. Existen algunos lenguajes de este tipo, pero no son ni fáciles de comprender ni de utilizar para la mayoría de los propósitos. UML es un lenguaje de modelado de propósito general. Para dominios especializados, como el diseño de IGU, el diseño de circuitos VLSI o la inteligencia artificial basada en reglas, será necesario utilizar una herramienta especializada con un lenguaje especial. UML es un lenguaje de modelado discreto. No se creó para modelar sistemas continuos, como los que podemos encontrar en ingeniería o en física. UML pretende ser un lenguaje de modelado universal, de propósito general, para sistemas discretos, 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 orientados a objetos que habían surgido.

Los métodos de desarrollo orientados a objetos

Los métodos de desarrollo para los lenguajes de programación tradicionales, como Cobol y Fortran, surgieron en los años 70 y se generalizaron en los años 80. El más destacado entre ellos era el Análisis estructurado y diseño estructurado [Yourdon-79] y sus variantes, entre las que se encontraba el Diseño estructurado de tiempo real [Ward-85]. Esos métodos, originalmente desarrollados por Constantine, DeMarco, Mellor, Ward, Yourdon y otros, alcanzaron cierta prenegación en el área de los grandes sistemas, especialmente en sistemas contratados por el gobierno en los campos aeroespacial y de defensa, en los que los contratistas insistían en un proceso de desarrollo organizado y una amplia documentación del diseño y la implementación. Los resultados no siempre fueron tan buenos como se esperaba —muchos sistemas software asistidos por computadora (CASE) fueron poco más que generadores de informes que extraían el diseño una vez finalizada la implementación— pero los métodos incluían buenas ideas que fueron ocasionalmente utilizadas en la construcción de grandes sistemas con eficiencia. Las aplicaciones comerciales fueron más reticentes a la hora de adoptar grandes sistemas CASE y métodos de desarrollo. La mayoría de las empresas desarrollaban el software internamente según sus propias necesidades, sin la relación de enfrentamiento entre el cliente y el contratista que caracterizaba los grandes proyectos gubernamentales. Los sistemas comerciales se percibían como más simples, tanto si realmente lo eran, como si no, y había una menor necesidad de revisión por parte de una organización externa.

El primer lenguaje que es en general reconocido como orientado a objetos es Simula-67 [Birtwistle-75], desarrollado en noruega en 1967. Este lenguaje nunca ha tenido un seguimiento significativo, aunque influyó mucho en los desarrolladores de varios lenguajes orientados a objetos posteriores. El trabajo de Dahl y Nygaard tuvo una profunda influencia en el desarrollo de la orientación a objetos. El movimiento de la orientación a objetos se volvió activo con la amplia difusión de Smalltalk a principios de los años 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 cinco años después de que Smalltalk se convirtiera en ampliamente 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 Booch [Booch-94], Rumbaugh/Blaha/Premerlani/Eddy/Lorensen [Rumbaugh-91] (actualizado en [Blaha-05]) y Wirfs-Brock/Wilkerson/Wiener [Wirfs-Brock-90] (nótese que los años de los derechos editoriales a menudo comienzan en julio del año anterior). Estos 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 Objectory [Jacobson-92] fue publicado poco después, basado en los trabajos que habían aparecido en publicaciones anteriores. Este libro trajo una aproximación un tanto diferente, puesto que se centra en los casos de uso y el proceso de desarrollo.

Durante los siguientes cinco años aparecieron gran cantidad de libros sobre metodología orientada a objetos, cada uno con su propio conjunto de conceptos, definiciones, notación, terminología y proceso. Algunos añadieron nuevos conceptos, pero en general había una gran similitud entre los conceptos propuestos por los diferentes autores. Muchos de los libros nuevos comenzaban con uno o más de los métodos existentes sobre los que se realizaban extensiones o cambios menores. Los autores originales tampoco estuvieron inactivos; la mayoría actualizaron su trabajo original, a menudo utilizando buenas ideas de otros autores. En general, surgió un núcleo de conceptos comunes, junto con una gran variedad de conceptos adoptados por uno o dos autores pero no ampliamente utilizados. Incluso en el núcleo de conceptos comunes, había discrepancias menores entre los métodos que hacían que las comparaciones detalladas fueran traicioneras, sobre todo para el lector ocasional.

Esfuerzo de unificación

Hubo algunos intentos tempranos de unificar conceptos entre métodos. Un ejemplo notable fue Fusión, de Coleman y sus colegas [Coleman-94], que incluyó conceptos de OMT [Rumbaugh-91], Booch [Booch-94] y CRC [Wirfs-Brock-90]. Como no involucró a los autores originales, fue considerado como otro nuevo método en lugar de como un reemplazo para varios métodos existentes. El primer intento con éxito de combinar y reemplazar las aproximaciones existentes llegó cuando Rumbaugh se unió a Booch en Racional Software Corporation en 1994. Ellos empezaron combinando los conceptos de OMT y de los métodos de Booch, obteniendo una primera propuesta en 1995. En ese momento, Jacobson también se unió a Racional y comenzó a trabajar con Booch y Rumbaugh. Su trabajo conjunto recibió el nombre de Lenguaje Unificado de Modelado (UML). El impulso ganado al tener a los autores de los tres métodos más relevantes trabajando juntos para unificar sus aproximaciones 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 en abandonar algunos de sus propios conceptos para alcanzar la armonía.

En 1996, el Object Management Group (OMG) publicó una petición de propuestas para una aproximación estándar al modelado orientado a objetos. Los autores de UML, Booch, Jacobson y Rumbaugh, comenzaron a trabajar con metodólogos y desarrolladores de otras compañías para generar, tanto una propuesta atractiva para los miembros del OMG, como un lenguaje de modelado que pudiera ser ampliamente aceptado por los fabricantes de herramientas, los metodólogos y los desarrolladores, que serían usuarios finales del lenguaje. Finalmente, todas las propuestas se fusionaron en la propuesta final de UML que se envió al OMG en septiembre de1997. El producto final es una colaboración entre mucha gente. Nosotros empezamos el esfuerzo de UML y aportamos unas pocas buenas ideas, pero las ideas contenidas en él son el producto de muchas mentes.

Estandarización

El Lenguaje Unificado de Modelado fue adoptado unánimemente como estándar por los miembros del OMG en noviembre de 1997 [UML-98]. El OMG asumió la responsabilidad de los futuros desarrollos del estándar UML. Incluso antes de que se adoptara definitivamente, se publicaron varios libros en los que se esbozaban los puntos clave de UML. Muchos proveedores de herramientas anunciaron apoyo o planes de apoyo para UML, y muchos metodólogos anunciaron que utilizarían UML para sus futuros trabajos. En la actualidad, UML ha reemplazado a la mayoría, si no a todas, de las notaciones de modelado de los procesos de desarrollo, de las herramientas de modelado y de los artículos en la literatura técnica. La aparición de UML parece haber sido atractiva para el público informático en general, ya que consolida la experiencia de muchos autores con un estatus oficial que ha reducido las divergencias gratuitas entre herramientas.

Dignas de mención son las series de conferencias internacionales de investigación con el título *UML aaaa*, donde aaaa es el año, que comenzó en 1998 y que continúa anualmente [UML-Conf]. También hay que fijarse en las conferencias anuales [ECOOP] y [OOPSLA] que se encargan de la tecnología de la orientación a objetos en general.

UML2

Después de varios años de experiencia utilizando UML, el OMG solicitó propuestas para mejorar UML solucionando problemas descubiertos por la experiencia en el uso y extendiendo el lenguaje con características adicionales que eran deseadas en varios dominios de aplicación. Las propuestas fueron desarrolladas entre noviembre de 2000 y julio de 2003, y poco después los miembros del OMG adoptaron la especificación de UML versión 2.0. La especificación adoptada experimentó el proceso de finalización habitual del OMG para solucionar los errores y problemas encontrados en la implementación inicial, por lo que se espera disponer de una especificación definitiva a finales de 2004 o comienzos de 2005.

En este libro se utiliza el término UML1 para hacer referencia a las versiones 1.1 y 1.5 de la especificación de UML y UML2 para hacer referencia a la versiones 2.0 y superiores de la especificación de UML.

Nuevas características. UML2 es en lo fundamental lo mismo que UML1, en especial, en lo que se refiere a las características principales utilizadas habitualmente. Algunas áreas problemáticas han sido modificadas, se han añadido algunas mejoras importantes y se han solucionado muchos pequeños fallos, pero los usuarios de UML1 podrían tener algunos problemas al utilizar UML2. La nueva versión puede ser considerada como la nueva versión de un lenguaje de programación o de una aplicación. Algunos de los cambios más importantes visibles para los usuarios son:

- La construcción y notación de los diagramas de secuencia se basa en gran parte en el estándar ITU Message Sequence Chart, adaptándolo para hacerlo más orientado a objetos.
- Se elimina la relación existente entre los conceptos del modelado de actividad y las máquinas de estados y del uso de notación popular en la comunidad de modelado de negocio.
- Se unifica el modelado de actividad con el modelado de acción añadido en la versión 1.5 de UML, para proporcionar un modelo procedimental más completo.

- Construcciones del modelado contextual para la composición interna de clases y colaboraciones. Estas construcciones permiten, tanto la encapsulación libre, como la encapsulación estricta, y el enlace de estructuras internas a partir de partes más pequeñas.
- El reposicionamiento de componentes como construcciones de diseño y artefactos como entidades físicas que son desplegadas.

Mecanismos internos. Otros cambios afectan a la representación interna de las construcciones de UML (el metamodelo) y su relación con otras especificaciones. Estos cambios no preocuparán directamente a la mayoría de los usuarios, pero son importantes para los fabricantes de herramientas porque afectan a la interoperatividad entre múltiples especificaciones, lo que afectará a los usuarios indirectamente:

- Unificación del núcleo de UML con partes del modelado conceptual de MOF (Meta-Object Facility). Esto permite que los modelos de UML sean manejados por herramientas MOF genéricas y repositorios.
- Reestructuración del metamodelo de UML para eliminar construcciones redundantes y permitir la reutilización de subconjuntos bien definidos por otras especificaciones.
- Disponibilidad de perfiles para definir extensiones de UML específicas del dominio y de la tecnología.

Otras fuentes

Además de los diversos métodos de desarrollo citados anteriormente y de algunos más que vendrán un poco después, ciertas vistas de UML muestran fuertes influencias de fuentes concretas no orientadas a objetos.

La vista estática, con clases interconectadas mediante diversas relaciones, está fuertemente influenciada por el modelo Entidad-Relación (Entity-Relationship) (ER) de Peter Chen, desarrollado en 1976. Esta influencia llegó a UML a través de la mayoría de los primeros métodos orientados a objetos. El modelo ER también influyó en los sistemas de bases de datos. Desafortunadamente, el mundo de los lenguajes de programación y el mundo de las bases de datos han seguido, en general, caminos separados.

Los modelos de máquinas de estados han sido utilizados en informática y en ingeniería eléctrica durante muchos años. Los diagramas de estados de David Harel son una extensión importante a las máquinas de estados clásicas, que añaden conceptos de estados enlazados y ortogonales. Las ideas de Harel fueron adaptadas por OMT, y desde ahí a otros métodos y finalmente a UML, donde forman la base de la vista de máquina de estados.

La notación del diagrama de secuencia de UML2 está tomado del estándar ITU Message Sequence Chart (MSC) [ITU-T Z.120], adaptado para hacerlo encajar mejor con otros conceptos de UML. Este estándar, que ha sido ampliamente utilizado en la industria de las telecomunicaciones, reemplaza la notación de los diagramas de secuencia de UML1 añadiendo varias construcciones estructuradas para superar los problemas de la notación previa de UML1. El ITU está considerando si adoptar algunos o todos los cambios en el estándar ITU.

Los conceptos de clasificadores estructurados de UML2 estaban fuertemente influenciados por las construcciones de la ingeniería de tiempo real de SDL [ITU-T Z.100], MSC y el método ROOM [Selic-94].

La notación de los diagramas de actividad de UML1, e incluso la de UML2, está fuertemente influenciada por varias notaciones de modelado de procesos de negocio. Dado que no existía una única notación dominante de modelado de procesos de negocio, La notación empleada por UML fue tomada de varias fuentes.

Existen otras muchas influencias sobre UML, y a menudo la fuente original de una idea precede a la persona que se ha hecho famosa por popularizarla. En torno a 20 personas fueron los principales colaboradores de la especificación de UML1, con muchos otros que participaron en menor medida. Quizá 30, más o menos, desarrollaron los papeles principales en el desarrollo de UML2, con otros muchos que mandaron sugerencias, revisaron propuestas y escribieron libros. Es imposible enumerar a todos los que contribuyeron en UML, y las breves referencias que hemos incluido indudablemente pasan por alto a algunos colaboradores importantes, para los que pedimos su comprensión.

¿Qué significa *unificado*?

La palabra *unificado* tiene los siguientes significados relevantes para UML.

A través de los métodos históricos y notaciones. UML combina los 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 a 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 hasta la implantación. El mismo conjunto de conceptos y notación puede ser utilizado en distintas etapas del desarrollo e incluso mezcladas en un único modelo. No es necesario traducir de una etapa a otra. Esta continuidad es crítica para el desarrollo iterativo e incremental.

A través de los dominios de aplicación. UML está pensado para modelar la mayoría de los dominios de aplicación, incluyendo aquéllos que involucran a sistemas que son grandes, complejos, de tiempo real, distribuidos, con tratamiento intensivo de datos o cálculo intensivo, entre otras propiedades. Puede haber áreas especializadas en las que un lenguaje de propósito especial sea más útil, pero UML pretende ser tan bueno o mejor que otros lenguajes 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 usable en aquellos sistemas implementados con varios lenguajes de implementación y varias 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 en todos los casos, mientras que el trabajo de la capa inferior diferirá en algo para cada medio.

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

A través de los conceptos internos. En la construcción del metamodelo de UML, hemos realizado un esfuerzo deliberado por descubrir y representar las relaciones subyacentes entre varios conceptos, tratando de capturar conceptos de modelado en un sentido amplio, aplicables

a muchas situaciones conocidas y desconocidas. Este proceso llevó a una mejor comprensión de los conceptos y a una forma más general de aplicarlos. Éste no fue el propósito general del trabajo de unificación, pero es 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 utilizar todos los modeladores. No tiene propietario y está basado en el común acuerdo de gran parte de la comunidad informática. Esto significa que incluye conceptos de los métodos líderes de forma que puede ser utilizado como su lenguaje de modelado. Como mínimo, está pensado para sustituir los modelos de OMT, Booch, Objectory, así como al resto de los participantes de la propuesta. Está pensado para ser tan familiar como sea posible; siempre que ha sido posible, hemos utilizado la notación de OMT, Booch, Objectory y de otros métodos líderes. Esto significa incorporar buenas prácticas de diseño, tales como la encapsulación, la separación de temas y la captura de la intención del modelo construido. Pretende abordar los problemas actuales del desarrollo de software, tales como el gran tamaño, la distribución, la concurrencia, los patrones y los equipos de desarrollo.

UML no pretende ser un método completo de desarrollo. No incluye un proceso de desarrollo paso por 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 que UML y un proceso para utilizar UML son dos cosas distintas. UML está pensado para dar soporte a todos o, al menos, a la mayoría de los procesos de desarrollo. UML incluye conceptos que creemos son necesarios para respaldar un proceso de desarrollo iterativo moderno basado, en la construcción de una arquitectura sólida para resolver los requisitos dirigidos por casos de uso.

El último objetivo 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 surgen en un sistema moderno, como la concurrencia y la distribución, así como los mecanismos de la ingeniería de software, como el encapsulamiento y los componentes. Desafortunadamente, esto significa que no puede ser pequeño si queremos que maneje algo más que sistemas de juguete. Los lenguajes modernos y los sistemas operativos modernos son más complicados que los de hace 50 años, porque 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 alguno de sus antecesores porque intenta ser más amplio. Pero no hay que aprendérselo entero de una vez, no más de lo que haría con un lenguaje de programación, un sistema operativo o una aplicación compleja de usuario, sin mencionar un lenguaje natural o una técnica.

Complejidad de UML

UML es un lenguaje de modelado extenso y variado, pensado para ser utilizado a muy diferentes niveles y en múltiples etapas del ciclo de vida de desarrollo. Ha sido criticado por ser extenso y complejo, pero la complejidad es inherente en cualquier aplicación universal que está pensado para un uso realista en problemas del mundo real, como sistemas operativos, lenguajes de programación, aplicaciones de edición multimedia, editores de hoja de cálculo y sistemas de publicación asistidos por computadora. Tales aplicaciones pueden ser mantenidas como pequeñas sólo pagando el precio de hacerlas de juguete, y los desarrolladores de UML no quisieron que fuera un juguete.

La complejidad de UML debe ser entendida a la luz de su historia:

- UML es producto del consenso entre personas con distintos objetivos e intereses. Comparte las cualidades del producto de un proceso democrático. No es tan limpio o coherente como lo sería un producto individual. Contiene características superfluas (aunque distintas personas pueden no estar exactamente de acuerdo sobre lo que es superfluo). Contiene características solapadas que no están siempre bien integradas. Sobre todo, carece de un punto de vista consistente. A diferencia de un lenguaje de programación, que tiene un uso bastante ajustado, se pretende que UML sirva para cualquier cosa, desde el modelado de negocio, a la programación gráfica. A menudo, una gran amplitud de aplicación trae un coste de especificación.
- En un principio era la fusión de las cuatro o cinco principales aproximaciones de modelado, siendo después el destino en el que albergar diversas notaciones existentes, como es el caso de SDL (Specification and Description Lenguaje, [ITU-T Z.100]), diversos lenguajes de modelado de negocio (los cuales no disponen de un estándar propio), lenguajes de acción, notaciones de máquina de estados, etcétera. El deseo de conservar notaciones anteriores a menudo crea inconsistencias entre características e incluye notación redundante con la intención de satisfacer el conocimiento de ciertos grupos de uso.
- Los documentos de la especificación oficial han sido escritos por equipos de aptitudes irregulares. Hay grandes variaciones en estilo, completitud, precisión y consistencia entre diversas secciones del documento.
- UML no es una especificación precisa de la forma en la que lo es un lenguaje formal. Aunque la comunidad informática mantiene la formalidad como una virtud, pocos de los lenguajes de programación dominantes están definidos con precisión, y los lenguajes formales son a menudo inaccesibles incluso para los expertos. También es necesario resaltar que modelar no es lo mismo que codificar. En la industria de la construcción, los anteproyectos se realizan mediante un estilo informal utilizando muchas convenciones que dependen del sentido común de los artesanos, pero los edificios son construidos con éxito a partir de ellos.
- Algunas veces, las secciones semánticas contienen sentencias vagas sin la explicación adecuada ni ejemplos. Los términos son introducidos en los metamodelos sin contar con una diferenciación clara con otros términos. Hay demasiadas buenas diferencias que alguien puede considerar importantes pero que no están explicadas con claridad.
- Se utiliza en exceso la generalización a costa de diferencias esenciales. El mito de que la herencia siempre es buena ha sido una maldición de la orientación a objetos desde sus primeros días.
- Hay tensiones entre los conceptos del modelado conceptual y la representación en los lenguajes de programación, sin pautas consistentes.

Valoración de UML

- UML es desordenado, impreciso, complejo y enmarañado. Esto es, tanto un fallo, como una virtud. Cualquier cosa pensada para un uso general va a ser desordenada.
- No es necesario saber o utilizar más características de UML de las que se necesitan, de la misma forma que no es necesario saber o utilizar todas las características de una gran apli-

cación software o de un lenguaje de programación. Hay un pequeño conjunto de conceptos principales que son ampliamente utilizados. Otras características pueden aprenderse gradualmente y utilizarse cuando se necesite.

- UML puede ser y ha sido utilizado de muchas formas diferentes en proyectos de desarrollo del mundo real.
- UML es más que una notación visual. Los modelos de UML pueden ser utilizados para generar código y casos de prueba. Esto exige un perfil UML adecuado, el uso de herramientas que se correspondan con la plataforma destino y elegir entre varios compromisos de implementación.
- Es innecesario escuchar demasiado a los abogados del lenguaje UML. No hay una única forma correcta de utilizarlo. Es una de las muchas herramientas que los buenos desarrolladores utilizan. No tiene por qué ser utilizada para cualquier cosa. Se puede modificar para adaptarlo a las necesidades propias, siempre que se tenga la cooperación de los colegas y de las herramientas software.

Áreas conceptuales de UML

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

Estructura estática. Cualquier modelo preciso debe definir primero 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 objetos discretos que almacenan la 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 su estructura común utilizando la generalización. Una clase hija añade de forma incremental estructuras y comportamientos a las estructuras y comportamientos que obtiene mediante la herencia de la clase padre común. Los objetos también tienen conexiones en tiempo de ejecución con otros objetos individuales. Estas relaciones objeto a objeto se modelan mediante asociaciones entre clases. Algunas relaciones entre elementos son agrupadas como relaciones de dependencia, incluyendo las relaciones entre niveles de abstracción, el enlace de los parámetros de una plantilla, otorgar permisos y el uso de un elemento por otro. Las clases pueden tener interfaces, los cuales describen su comportamiento visible desde el exterior. Se incluyen otras relaciones y extiende dependencias de casos de uso. La vista estática se realiza mediante diagramas de clases y sus variantes. La vista estática puede ser utilizada para generar la mayoría de las declaraciones de estructuras de datos de un programa. Hay muchos otros 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 similar a las clases con ciertos añadidos y restricciones en cada tipo de clasificador.

Construcciones de diseño. Los modelos de UML están pensados, tanto para el análisis lógico, como para el diseño destinado a la implementación. Determinadas construcciones representan elementos de diseño. Un clasificador estructurado expande una clase en su implementación como una colección de partes mantenidas juntas mediante conectores. Una clase puede encapsular su estructura interna detrás de puertos externos visibles. Una colaboración modela una colección de objetos que desempeñan roles en un contexto transitorio. Un componente es una

parte reemplazable de un sistema que se ajusta y proporciona una realización de un conjunto de interfaces. Está pensado para que sea fácilmente sustituible por otros componentes que cumplan con la misma especificación.

Construcciones de despliegue. Un nodo es un recurso de cálculo de tiempo de ejecución que define una ubicación. Un artefacto es una unidad física de información o de descripción de comportamiento en un sistema informático. Los artefactos se despliegan en los nodos. Un artefacto puede ser una manifestación, esto es, una implementación, de un componente. La vista de despliegue describe la configuración de los nodos en un sistema en ejecución y la situación de los artefactos en ellos, incluyendo las relaciones de manifestación con los componentes.

Comportamiento dinámico. Hay tres formas de modelar el comportamiento. Una es la historia de la vida de un objeto, que muestra cómo interactúa con el resto del mundo; otra son los patrones de comunicación de un conjunto de objetos conectados, que muestra cómo interactúan para implementar un comportamiento; la tercera es la evolución del proceso de ejecución, que muestra cómo pasa a través de varias actividades.

La vista de un objeto aislado es una máquina de estados —una vista de un objeto que muestra cómo responde a los eventos basándose en su estado actual, cómo realiza acciones como parte de su respuesta, y las transiciones a un nuevo estado. Las máquinas de estados se muestran en diagramas de máquinas de estados.

Una interacción envuelve a un clasificador estructurado o a una colaboración con el flujo de mensajes entre las partes. Las interacciones se muestran en los diagramas de secuencia y en los diagramas de comunicación. Los diagramas de secuencia hacen énfasis en las secuencias de tiempo, mientras que los diagramas de comunicación hacen mayor hincapié en las relaciones entre objetos.

Una actividad representa la ejecución de un cálculo. Se modela como un conjunto de nodos de actividad conectados mediante flujos de control y flujos de datos. Las actividades pueden modelar comportamientos, tanto secuenciales, como concurrentes. Incluyen las construcciones tradicionales de flujo de control, así como decisiones y bucles. Los diagramas de actividad pueden ser utilizados para mostrar cálculos además de flujos de trabajo en organizaciones humanas.

Para guiar todas las vistas de comportamiento está un conjunto de casos de uso, cada uno con una descripción de una parte de la funcionalidad del sistema tal y como lo ve un actor, que es un usuario externo del sistema. La vista de casos de uso incluye, tanto la estructura estática de los casos de uso y sus actores, como las secuencias dinámicas de mensajes entre actores y el sistema, normalmente expresadas como diagramas de secuencia o simplemente texto.

Organización del modelo. Las computadoras pueden manejar grandes modelos “planos”, pero los humanos no. En un sistema grande, la información de modelado puede ser dividida en piezas, de forma que los equipos puedan trabajar sobre las diferentes partes de forma concurrente. Incluso en un sistema más pequeño, el entendimiento humano necesita de la organización del contenido del modelo en paquetes de tamaño modesto. Los paquetes son unidades de propósito general, organizativas y jerárquicas, de los modelos de UML. Pueden utilizarse para el almacenamiento, el control de acceso, la gestión de la configuración y la construcción de bibliotecas que contengan fragmentos del modelo reutilizables. Una dependencia entre paquetes resume las dependencias entre los contenidos del paquete. En una dependencia entre paquetes puede ser impuesta 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.

Perfiles. No importa cuan completo sea el conjunto de "facilidades" de un lenguaje, la gente querrá hacer sus extensiones. UML contiene una capacidad limitada de extensión que debería permitir la mayoría de las necesidades de extensión del día a día, sin recurrir a modificar el lenguaje básico. Un estereotipo es un nuevo tipo de elemento del modelo con la misma estructura que un elemento existente, pero con restricciones adicionales, una interpretación y un icono diferentes, y un tratamiento distinto por parte de los generadores de código y otras herramientas de bajo nivel. Un estereotipo define un conjunto de valores etiquetados. Un valor etiquetado es un atributo definido por el usuario que se aplica a los propios elementos del modelo, en lugar de a los objetos en tiempo de ejecución. Por ejemplo, los valores etiquetados deben indicar información para la gestión del proyecto, pautas para el generador de código e información específica del dominio. Una restricción es una condición bien formada expresada como una cadena de texto en algún lenguaje restrictivo, como un lenguaje de programación, un lenguaje especial de restricciones o lenguaje natural. UML incluye un lenguaje de restricciones denominado **OCL**[1]. Un perfil puede ser desarrollado con un propósito específico y ser almacenado en bibliotecas para su uso en modelos de usuario. Como en cualquier mecanismo de extensibilidad, estos mecanismos deben ser utilizados con cuidado, ya que es un riesgo producir un dialecto privado ininteligible para el resto. Sin embargo, pueden evitar la necesidad de más cambios radicales.

---

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

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 varios niveles de los modelos: ideal, parcial y basado en herramientas.

¿Qué es un modelo?

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

Un modelo se expresa en un medio adecuado para el trabajo. Los modelos de edificios o construcciones pueden pintarse en papel, las figuras tridimensionales son construidas con cartón y pasta de papel, o las ecuaciones de elementos finitos en una computadora. Un modelo de construcción de un edificio muestra la apariencia del edificio, pero también puede utilizarse 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, tanto semántica, como notación, y puede adoptar diversos formatos que incluyen el texto y los gráficos. Se pretende que el modelo sea más fácil de utilizar, para ciertos propósitos, que el sistema final.

¿Para qué sirven los modelos?

Los modelos se utilizan para varios propósitos.

Para capturar y enumerar exhaustivamente los requisitos y el dominio del conocimiento, de forma que todos los implicados puedan entenderlos y estar de acuerdo con ellos. Diferentes modelos de un edificio capturan requisitos sobre la apariencia, patrones de tráfico, varios tipos de servicio, fortaleza frente al viento y a los terremotos, costes y muchas otras cosas. Entre los implicados se encuentra el arquitecto, el ingeniero de estructuras, el contratista principal, varios subcontratistas, el propietario, los inquilinos y la ciudad. Los distintos modelos de un sistema software pueden capturar requisitos sobre el dominio de aplicación, las formas en que los usuarios lo utilizarán, su división en módulos, patrones comunes utilizados en su construcción, y otras cosas. Entre los implicados se encuentra el arquitecto, el analista, los programadores, el encargado del proyecto, los clientes, los inversores, los usuarios finales y los operadores. Se utilizan diferentes tipos de modelos UML.

Para pensar en el diseño de un sistema. Un arquitecto utiliza modelos en papel, en una computadora, o con construcciones tridimensionales para visualizar y experimentar con posibles diseños. La simplicidad de crear y modificar pequeños modelos permite un pensamiento creativo con poco coste.

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

Para capturar las decisiones de diseño en un formato alterable independiente de los requisitos. Un modelo de un edificio muestra el aspecto externo convenido con el cliente. Otro modelo muestra la disposición interna de los cables, tuberías y 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 verifica esta información, pero, a menudo, los clientes no se preocupan por los detalles mientras funcionen.

Un modelo de un sistema software puede capturar el comportamiento externo de un sistema y la información del dominio del mundo real representada por el sistema. Otro modelo muestra las clases y operaciones internas que implementan el comportamiento externo. Hay muchas formas de implementar el comportamiento; el modelo de diseño final muestra una aproximación que el diseñador considera que es buena.

Para generar productos usables para el trabajo. Un modelo de un edificio puede ser utilizado para generar diversos tipos de productos. Estos incluyen una factura con los materiales, una simulación animada de un paseo, una tabla de desviaciones a varias velocidades del viento, y una visualización de las desviaciones en diversos puntos del armazón.

Un modelo de un sistema software puede ser utilizado para generar las declaraciones de las clases, los cuerpos de los procedimientos, las interfaces de usuario, las bases de datos, los escenarios de uso válidos y una lista de 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, eléctrica, fontanería, ventilación, decoración, etcétera. Sin embargo, a menos que el modelo esté en una computadora, no es fácil encontrar cosas y modificarlas. Si se encuentran en una computadora los cambios se pueden realizar y recordar fácilmente, y pueden explorarse, de forma sencilla, múltiples diseños mientras comparten algunos elementos comunes.

Un modelo de un sistema software organiza la información en varias vistas: estructura estática, máquinas de estados, interacciones, requisitos, etcétera. Cada vista es una proyección de la información seleccionada, para un propósito, del modelo completo.

Mantener un modelo de cualquier tamaño correcto es imposible sin disponer de una herramienta de edición que maneje el modelo. Un editor gráfico e interactivo del modelo puede presentar la información en diferentes formatos, ocultando la información que no es necesaria para un determinado propósito y mostrándola de nuevo más tarde, agrupando operaciones relacionadas, realizando cambios, tanto sobre elementos individuales, como sobre grupos de elementos, con un comando, etcétera.

Para explorar económicamente múltiples soluciones. Las ventajas y riesgos de los diferentes métodos de diseño de un edificio pueden no estar claros al principio. Por ejemplo, distin-

tas subestructuras pueden interactuar de formas complicadas, que no pueden evaluarse en la cabeza de un ingeniero. Los modelos pueden explorar los distintos diseños y permitir cálculos de costes y riesgos antes de que el edificio real sea construido.

Los modelos de un gran sistema software permiten que se propongan y comparen varios diseños. Por supuesto, los modelos no se construyen con todos los detalles, pero incluso un modelo rudimentario puede poner de manifiesto muchas cuestiones que deben ser tenidas en cuenta en el diseño final, El modelado permite considerar varios diseños, con un pequeño coste al implementar cualquiera de ellos.

Para dominar 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 real no puede producirse a voluntad y, de todas formas, podría destruir los instrumentos de medición. Muchos procesos físicos rápidos, pequeños o violentos pueden ser comprendidos utilizando modelos físicos.

Un modelo de un gran sistema software permite ocuparse de la complejidad que es demasiado difícil de tratar directamente. Un modelo puede abstraer a un nivel que sea comprensible para las personas, sin perderse en los detalles. Una computadora puede realizar análisis complicados sobre un modelo en un esfuerzo por encontrar posibles puntos problemáticos, como errores de sincronización y desbordamiento de los recursos. Un modelo puede determinar el impacto potencial de la realización de un cambio antes de que se realice, mediante la exploración de las dependencias en el sistema. Un modelo también puede mostrar cómo reestructurar un sistema para reducir tales efectos.

Niveles de los modelos

Los modelos adquieren distintas formas para diversos propósitos, y aparecen en diversos niveles de abstracción. La cantidad de detalles en el modelo debe ser adaptada a uno de los siguientes propósitos.

Guías para el proceso de pensamiento. La construcción de modelos de alto nivel al principio de un proyecto sirve para centrarse en el proceso de pensamiento de los participantes y destacar determinadas opciones. La captura de requisitos representa un punto de partida hacia el diseño del sistema. Los primeros modelos ayudan a los creadores a explorar las posibles opciones antes de converger en un concepto de sistema. A medida que el diseño progresa, los primeros modelos son reemplazados por otros más precisos. No es necesario conservar cada giro y vuelta del proceso de exploración inicial. Su propósito es producir ideas. Sin embargo, los modelos de pensamiento finales deben ser conservados incluso después de que el foco de atención se desplace hacia cuestiones de diseño. Los primeros modelos no necesitan los detalles o la precisión de un modelo de implementación, y no necesitan de una gama completa de conceptos de implementación. Tales modelos utilizan un subconjunto de las construcciones de UML, un subconjunto más limitado que los modelos posteriores de diseño.

Cuando un primer modelo es una vista completa de un sistema con una precisión determinada —por ejemplo, un modelo de análisis de lo que debe ser hecho— entonces debe ser conservado cuando el desarrollo pasa a la siguiente fase. Hay una diferencia importante entre añadir detalles (en cuyo caso, se debe preservar la cadena de razonamiento) y el proceso habitual de caminar al azar, para explorar muchos callejones sin salida, antes de alcanzar la solución correc-

ta. En este último caso, es abrumador e innecesario guardar la historia completa, excepto en situaciones excepcionales en las que la trazabilidad es obligatoria.

Especificaciones abstractas de la estructura esencial de un sistema. Los modelos en el análisis o en las etapas preliminares del diseño se centran en los conceptos clave y en los mecanismos del posible sistema. Estos se corresponden en cierta manera con el sistema final. Pero faltan los detalles en el modelo, que deben ser añadidos explícitamente durante el proceso de diseño. El propósito de los modelos abstractos es obtener de forma correcta los aspectos principales de alto nivel, antes de abordar los detalles más localizados. Estos modelos están pensados para hacerlos evolucionar hacia los modelos finales mediante un proceso cuidadoso que garantiza que el sistema final implementa correctamente la intención de esos primeros modelos. Debe existir trazabilidad entre esos modelos esenciales y los modelos completos; de otra forma, no hay garantías de que el sistema final incorpore correctamente las propiedades clave que el modelo esencial se esfuerza en mostrar. Los modelos esenciales se centran en la semántica. No necesitan la gama completa de opciones de implementación. Es más, las distinciones de rendimiento de bajo nivel a menudo oscurecen la semántica lógica. Sin embargo, el camino desde un modelo esencial hacia un modelo de implementación completo debe ser claro y sencillo, con independencia de si es generado automáticamente por un generador de código o desarrollado manualmente por un diseñador.

Especificaciones completas de un sistema final. Un modelo de implementación incluye suficiente información para construir el sistema. No sólo debe incluir la semántica lógica del sistema y los algoritmos, estructuras de datos y mecanismos que aseguren un rendimiento adecuado, además es necesario incluir las decisiones organizativas sobre los artefactos del sistema que se necesitan para el trabajo cooperativo de las personas y el procesamiento de las herramientas. Este tipo de modelo debe incluir construcciones para el empaquetamiento del modelo, tanto para su comprensión por parte de las personas, como para la conveniencia de la computadora. Estas no son características de la aplicación propiamente dicha. En realidad, son propiedades del proceso de construcción.

Ejemplos de sistemas típicos o posibles. Los ejemplos bien elegidos pueden proporcionar entendimiento a las personas y pueden validar las especificaciones y la implementación del sistema. Sin embargo, incluso una gran colección de ejemplos, necesariamente carece de una descripción definitiva.

En última instancia, necesitamos modelos que especifiquen el caso general; es decir, lo que es un programa después de todo. Sin embargo, ejemplos de estructuras de datos típicas, secuencias de interacción o historias de los objetos pueden ayudar a las personas a intentar entender una situación complicada. Los ejemplos deben ser utilizados con cierto cuidado. Es lógicamente imposible inducir el caso general a partir de un conjunto de ejemplos, pero los prototipos bien elegidos son la forma de pensar de la mayoría de la gente. Un modelo de ejemplo incluye instancias más que descriptores generales. Por lo tanto, tiende a dar una visión diferente de la que da un modelo descriptivo genérico. Los modelos de ejemplo normalmente utilizan sólo un subconjunto de las construcciones de UML, las que tienen que ver con instancias. Tanto los modelos descriptivos, como los de ejemplo son útiles en el modelado de un sistema.

Descripciones completas o parciales de un sistema. 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 y discretas, cada una de las cuales debe ser almacenada y manipulada por separado, como parte de la descripción completa. Tales modelos tienen “cosas pendientes”

que deben ser enlazadas con otros modelos para proporcionar un sistema completo. Debido a que las piezas tienen coherencia y significado, deben ser combinadas con otras piezas de diversas formas para proporcionar muchos sistemas diferentes. Alcanzar la reutilización es una meta importante del buen modelado.

Los modelos se desarrollan durante un cierto tiempo. 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 una vista de alto nivel del sistema completo, con unos pocos servicios clave con pocos detalles y sin adornos. En un cierto plazo, se añaden muchos más detalles y se introducen variantes. También, durante un cierto plazo, el enfoque cambia de la parte externa, la vista lógica centrada en el usuario, a la parte interna, la vista física centrada en la implementación. A medida que los desarrolladores trabajan con el sistema y lo comprenden mejor, el modelo debe iterarse a todos los niveles para capturar esa comprensión; es imposible comprender un sistema en una única pasada lineal. No hay una única forma correcta para un modelo.

¿Qué hay en un modelo?

Semántica y presentación. Los modelos tienen dos aspectos principales: la información semántica (semántica) y la presentación visual (notación).

El aspecto semántico captura el significado de una aplicación en forma de una red de construcciones lógicas, como clases, asociaciones, estados, casos de uso y mensajes. Los elementos del modelo semántico llevan el significado del modelo —es decir, transportan la semántica. Los elementos del modelado semántico se utilizan para la generación de código, la comprobación de la validez, las métricas de complejidad, etcétera. La apariencia visual es irrelevante para la mayoría de las herramientas que procesan los modelos. La información semántica suele ser denominada a menudo como *el modelo*. Un modelo semántico tiene estructura sintáctica, reglas para asegurar su corrección y dinámicas de ejecución. Estos aspectos se describen a menudo por separado (como en la documentación de definición de UML), pero hay partes de un único modelo coherente que se encuentran fuertemente interrelacionadas.

La presentación visual muestra la información semántica de forma que pueda ser vista, hojeada y corregida por los seres humanos. Los elementos de la presentación se encargan de la presentación del modelo —es decir, la muestran de forma que sea comprensible para los seres humanos. No añaden significado, pero organizan la presentación para enfatizar la organización del modelo de una manera útil. Por lo tanto, dirigen la comprensión humana del modelo. Los elementos de la presentación obtienen su semántica de los elementos del modelo semántico. Pero a la vista de que son las personas las que proporcionan disposición de los diagramas, los elementos de la presentación no son completamente derivables de los elementos lógicos. La disposición de los elementos de la presentación puede llevar connotaciones sobre relaciones semánticas que son demasiado débiles o ambiguas para ser formalizadas en el modelo semántico pero que, sin embargo, son sugerentes para las personas.

Contexto. Los modelos son, en sí mismos, artefactos en un sistema informático, y se utilizan dentro de un contexto mayor que les da su significado completo. Esta descomposición no es necesaria por motivos semánticos —un gran modelo monolítico debería ser tan preciso como un conjunto de modelos organizados en paquetes coherentes, quizá incluso más precisos porque los límites organizativos complican el trabajo de definir una semántica precisa. Pero los equipos de

trabajo no podrían trabajar de una forma efectiva en un gran modelo monolítico sin meterse en el camino de los otros. Además, un modelo monolítico no tiene piezas que puedan ser reutilizadas en otras situaciones. Por último, los cambios en un gran modelo tienen consecuencias que son difíciles de determinar. Los cambios en una pequeña pieza aislada de un modelo grande pueden ser tratables si el modelo está estructurado correctamente en subsistemas con interfaces bien definidas. En cualquier caso, la división de grandes sistemas en una jerarquía de piezas correctamente seleccionadas es la forma más fiable de diseñar los grandes sistemas que los seres humanos han diseñado durante miles de años.

Los modelos capturan la información semántica de un sistema, pero también necesitan almacenar muchos tipos de información sobre el propio proceso de desarrollo, como el autor de una clase, el estado en la depuración de un procedimiento y a quién se le permite corregir un diagrama. Tal información es, en el mejor de los casos, anexa a la semántica del sistema, pero es importante para el proceso de desarrollo. Por lo tanto, un modelo de un sistema tiene que incluir ambos puntos de vista. Esto es más fácil de alcanzar considerando la información de gestión del proyecto como anotaciones en el modelo semántico —es decir, descripciones arbitrarias adjuntas a los elementos del modelo, pero cuyo significado se encuentra fuera del lenguaje de modelado. En UML, estas anotaciones se implementan como cadenas de texto cuyo uso está definido en perfiles optativos.

Las órdenes utilizadas para crear y modificar un modelo no son parte de la semántica del lenguaje de modelado, de la misma forma que no son parte de un lenguaje de programación las órdenes de un editor de texto o de un navegador. Las propiedades de los elementos del modelo no tienen valores *por defecto*; en un modelo específico, simplemente tienen *valores*. Sin embargo, para el desarrollo práctico, las personas necesitan construir y modificar modelos sin necesidad de especificarlo todo con todo detalle. Los valores por defecto existen en el límite entre el lenguaje de modelado y la herramienta de edición que le da soporte. Realmente son valores por defecto en los comandos de la herramienta que crean el modelo, aunque pueden transcender a una herramienta en particular y convertirse en expectativas del usuario sobre la implementación del lenguaje por las herramientas en general.

Los modelos no se construyen ni utilizan de forma asilada. Son parte de un entorno mayor que incluye herramientas de modelado, lenguajes y compiladores, sistemas operativos, redes de computadoras, restricciones de implementación, etcétera. La información sobre un sistema incluye información sobre todas las partes del entorno. Algunas de ellas serán almacenadas en el modelo incluso si no son información semántica. Como ejemplo se pueden citar las anotaciones para la gestión del proyecto (discutidas anteriormente), las ayudas y directivas para la generación de código, el empaquetado del modelo y los comandos de configuración por defecto para la herramienta de edición. El resto de la información se puede almacenar por separado. Como ejemplo se puede señalar el código fuente del programa y los comandos de configuración del sistema operativo. Incluso si parte de la información pertenece al modelo, la responsabilidad de su interpretación puede recaer en varios lugares, como el lenguaje de modelado, la herramienta de modelado, el generador de código, el compilador, un leguaje de comandos, etcétera. Este libro describe la interpretación de los modelos que está definida en la especificación de UML, y que, por lo tanto, se aplica a todos los usos de UML. Pero cuando se opera en un entorno de desarrollo físico, otras fuentes pueden añadir interpretaciones adicionales, más allá de las dadas en la especificación de UML.

¿Cuál es el significado de un modelo?

Un modelo es un *generador* de potenciales configuraciones de sistemas; los posibles sistemas son *extensiones* o valores. Idealmente, todas las configuraciones consistentes con el modelo deberían ser posibles. Sin embargo, algunas veces no es posible representar todas las restricciones en el modelo. Un modelo también es una descripción de la estructura genérica y del significado de un sistema. Las descripciones son su *propósito* o significado. Un modelo siempre es una abstracción a un cierto nivel. Captura los aspectos *esenciales* de un sistema e ignora algunos de los detalles. En los modelos, se deben tener en cuenta los siguientes aspectos.

Abstracción frente a detalle. Un modelo captura los aspectos esenciales de un sistema e ignora 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 permitir modelos que sean especificados a diferentes niveles de detalle. Un primer modelo, o modelo de alto nivel, puede no necesitar todos los detalles, porque los detalles adicionales pueden ser irrelevantes para el propósito que tenemos en mente. Modelos con distintos niveles de precisión pueden ser utilizados a lo largo de la vida del proyecto. Un modelo previsto para la generación de código necesita que, al menos, se tengan en cuenta algunos aspectos del lenguaje de programación. Típicamente, los modelos tienen menos precisión durante el comienzo del análisis. Ganan en detalles a medida que el ciclo de desarrollo avanza, de forma que los modelos finales tienen un detalle y una precisión considerables.

Especificación frente a implementación. Un modelo puede decir *qué* es lo que hace algo (*especificación*), también *cómo* se consigue la función (*implementación*). Estos aspectos deben ser separados en el modelado. Es importante obtener el *qué* correctamente antes de invertir mucho tiempo en el *cómo*. Abstraerse de la implementación es una faceta importante del modelado. Puede haber una cadena de relaciones especificación-implementación, en la que cada implementación define la especificación para la capa siguiente.

Descripción frente a instancia. Los modelos son descripciones, Las cosas que describen son instancias, las cuales normalmente aparecen en los modelos sólo como ejemplo. La mayoría de las instancias existen sólo como parte de la ejecución. No obstante, algunas veces las instancias de tiempo de ejecución son descripciones de otras cosas. Nosotros denominamos a estos objetos híbridos *metadatos*. Si lo miramos con mayor profundidad, no es realista insistir en que todo es, o una instancia, o una descripción. Algo es una instancia o una descripción, no de forma aislada, sino en relación con algo más, y la mayoría de las cosas pueden ser consideradas desde múltiples puntos de vista.

Variaciones en la interpretación. Hay muchas interpretaciones posibles de los modelos en un lenguaje de modelado. Uno puede definir ciertos *puntos de variación semántica* —lugares donde son posibles distintas interpretaciones— y asignar a cada interpretación un nombre como *variación semántica*, de forma que uno pueda saber qué variación semántica se está utilizando. Por ejemplo, el lenguaje Self tiene distintos mecanismos, que el lenguaje Smalltalk, para localizar los métodos; un punto de variación semántica en el mecanismo de resolución de métodos permite que sean admitidos ambos lenguajes de programación. Los puntos de variación semántica permiten soportar distintos modelos de ejecución.

Parte 2: Conceptos de UML

Esta parte contiene una descripción de los conceptos de UML para mostrar cómo encajan entre sí en el modelado de un sistema. Esta parte no pretende describir los conceptos con todo detalle. Para ver todos los detalles de los conceptos de UML, véase la sección del *diccionario de términos* de este libro.

3

Un paseo por UML

Este capítulo presenta un breve recorrido por los conceptos y diagramas de UML utilizando un ejemplo sencillo. El propósito del capítulo es organizar los conceptos de alto nivel de UML en un pequeño conjunto de vistas y diagramas que presentan los conceptos de forma visual. Muestra cómo diversos conceptos se utilizan para describir un sistema y cómo las vistas encajan entre sí. Este resumen no está pensado para ser exhaustivo; se omiten muchos conceptos. Para más detalles, véanse los siguientes capítulos que esbozan las vistas semánticas de UML, además del material de referencia detallado en el capítulo del diccionario.

El ejemplo que se utiliza es una taquilla de un teatro que ha automatizado sus operaciones. Es un ejemplo inventado, cuyo propósito es destacar diversas construcciones de UML en un espacio reducido. Se ha simplificado a propósito, y no se muestra con todos los detalles. La presentación de un modelo completo de un sistema implementado, ni cabría en un espacio tan pequeño, ni destacaría un rango suficiente de construcciones sin caer en una repetición excesiva.

Vistas de UML

No hay una línea clara entre los diversos conceptos y construcciones en UML, pero, por comodidad, nosotros los dividimos en varias vistas. Una vista simplemente es un subconjunto de las construcciones de modelado de UML que representan un aspecto del sistema. La división en diferentes vistas es un tanto arbitraria, pero esperamos que sea intuitiva. Uno o dos tipos de diagramas proporcionan una notación visual para los conceptos de cada vista. Las vistas utilizadas en este libro no son parte de la especificación de UML, pero las utilizamos como ayuda para organizar y presentar los conceptos de UML.

En el nivel superior, las vistas pueden ser divididas en las siguientes áreas: clasificación estructural, comportamiento dinámico, diseño físico y gestión del modelo.

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

La clasificación estructural describe los elementos del sistema y sus relaciones con otros elementos. El concepto de clasificador modela los elementos en un sistema. Los clasificadores incluyen clases, casos de uso, actores nodos, colaboraciones y componentes.

Tabla 3.1 Vistas y diagramas de UML

Área principal	*Vista*	*Diagrama*	*Conceptos principales*
estructural	vista estática	diagrama de clases	asociación, clase, dependencia, generalización, interfaz, ralización
	vista de diseño	estructura interna	conector, interfaz, interfaz obligatoria, interfaz proporcionada, parte, puerto
		diagrama de colaboración	colaboración, conector, rol, uso de la colaboración
		diagrama de componentes	componente, dependencia, interfaz proporcionada, interfaz obligatoria, puerto, realización, subsistema
	vista de casos de uso	diagrama de casos de uso	actor, asociación caso de uso, extensión, generalización de casos de uso, inclusión
dinámica	vista de máquina de estado	diagrama de máquina de estados	actividad hacer, disparador, efecto, estado, evento, región, transición, transición de finalización
	vista actividad	diagrama de actividad	acción, actividad, control de flujo, división, excepción, flujo de datos, nodo de control, nodo objeto, pin, región de expansión, unión
	vista de interacción	diagrama de secuencia	especificación de la ejecución, especificación del suceso, fragmento de la interacción, interacción, línea de vida, mensaje, operando de la interacción, señal
		diagrama de comunicación	colaboración, condición de guarda, mensaje, rol, número de secuencia
física	vista de despliegue	diagrama de despliegue	artefacto, dependencia, manifestación, nodo
gestión del modelo	vista de gestión del proyecto	diagrama de paquetes	importar, modelo, paquete
	perfil	diagrama de paquetes	estereotipo, perfil, restricción, valor etiquetado

Los clasificadores proporcionan las bases sobre las que se construye el comportamiento dinámico. Las vistas estructurales incluyen la vista estática, vista de diseño y vista de casos de uso.

El comportamiento dinámico describe el comportamiento de un sistema o de un clasificador a lo largo del tiempo. El comportamiento puede ser descrito como una serie de cambios sobre

instantáneas del sistema extraídas de la vista estática. Las vistas del comportamiento dinámico incluyen la vista de máquina de estados, la vista de actividades y la vista de interacción.

El diseño físico describe los recursos computacionales del sistema y el despliegue de artefactos en ellos. Incluye la vista de despliegue.

La gestión del modelo describe la organización de los propios modelos en unidades jerárquicas. Un modelo es una jerarquía de paquetes que proporciona una abstracción semántica completa de un sistema desde un punto de vista específico. La vista de gestión del modelo cruza las otras vistas y las organiza para el trabajo de desarrollo y el control de la configuración.

Las extensiones a UML se organizan en perfiles. Los perfiles de UML declaran diversas construcciones con la intención de proporcionar una capacidad de extensión limitada pero útil. Estas construcciones incluyen restricciones, estereotipos y definiciones de etiquetas. Los perfiles se declaran en los diagramas de clases y se aplican en los diagramas de paquetes. Los estereotipos normalmente son aplicados en los diagramas de clases, aunque pueden aparecer también en otros lugares. Los perfiles pueden incluir también bibliotecas de clases específicas del dominio.

Vista estática

La vista estática modela conceptos del dominio de la aplicación, así como los conceptos internos inventados como parte de la implementación de una aplicación. Esta vista es estática porque no describe el comportamiento dependiente del tiempo del sistema, que se describe en otras vistas. Los principales componentes de la vista estática son las clases y sus relaciones: asociación, generalización y varios tipos de dependencia, como la realización y el uso. Una clase es la descripción de un concepto del dominio de la aplicación o del dominio de la solución. Las clases son el centro, alrededor del cual se organiza la vista de clases; otros elementos pertenecen o se unen a las clases. La vista estática se muestra en los diagramas de clases, llamadas así porque centran fundamentalmente su atención en las clases.

Las clases se dibujan como rectángulos. La lista de atributos y operaciones se muestran en compartimentos separados. Se pueden suprimir los compartimentos cuando no se necesitan todos los detalles. Una clase puede aparecer en varios diagramas. Los atributos y operaciones se muestran a menudo en un diagrama (el diagrama “raíz”) y se suprimen del resto de los diagramas.

Las relaciones entre clases se muestran como las líneas que conectan los rectángulos de las clases. Los diferentes tipos de relaciones se distinguen 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 del teatro. Este diagrama contiene parte del dominio del modelo “venta de entradas”. Muestra varias clases importantes, como **Cliente, Reserva, Entrada** y **Representación.** Los clientes pueden tener muchas reservas, pero cada reserva es hecha por un único cliente. Las reservas son de dos tipos: suscripción a un ciclo y reservas individuales. Ambos reservan entradas: en el primer caso varias entradas, mientras que en el segundo sólo una entrada. Cada entrada puede ser parte de una suscripción a un ciclo o de una reserva individual, pero no de ambos. Cada representación tiene muchas entradas disponibles, cada una con un único número de asiento. Una representación se puede identificar mediante la obra, la fecha y la hora.

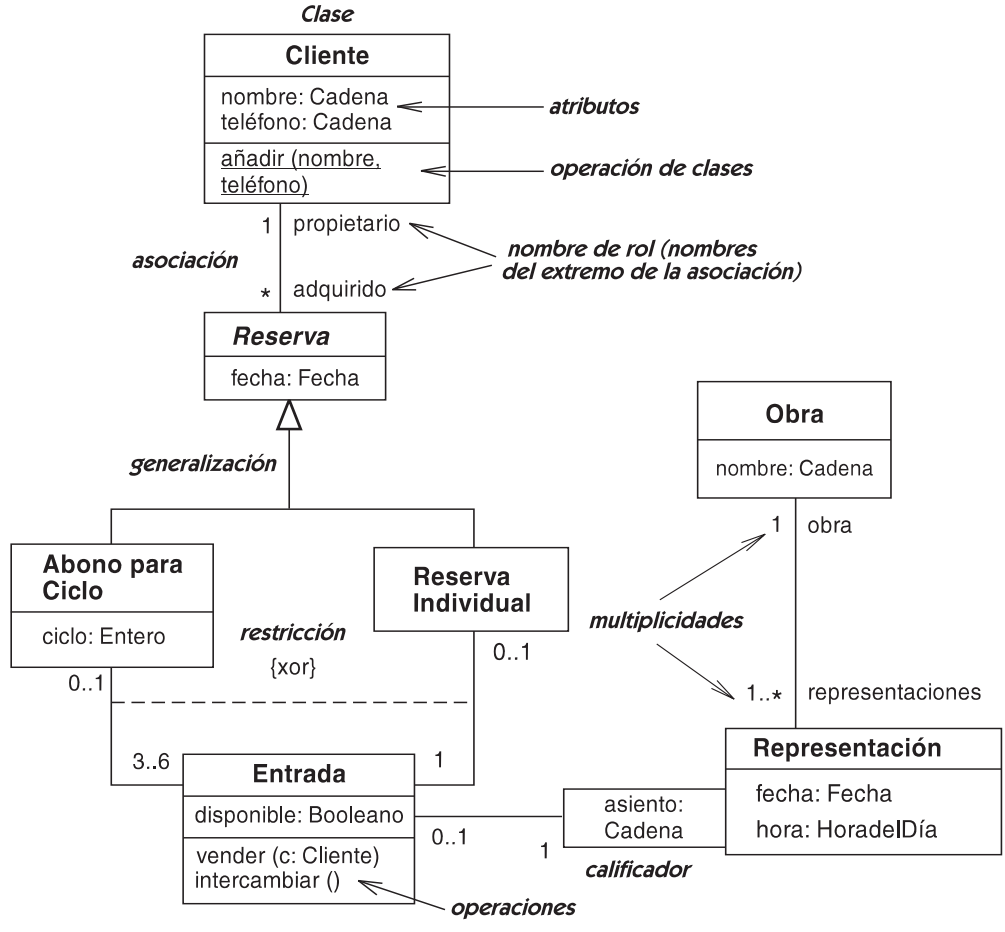


Figura 3.1 Diagrama de clases

Las clases se pueden describir con distintos niveles de detalle y de concreción. En las primeras etapas del diseño, el modelo captura los aspectos más lógicos del problema. En las últimas etapas, el modelo también captura las decisiones de diseño y los detalles de implementación. La mayoría de las vistas tienen una evolución en la calidad similar.

Vistas de diseño

Las vistas previas modelan los conceptos de la aplicación desde un punto de vista lógico. Las vistas de diseño modelan la estructura de diseño de la propia aplicación, como su expansión en clasificadores estructurados, las colaboraciones que proporcionan funcionalidad y su ensamblado a partir de componentes con interfaces bien definidas. Estas vistas proporcionan una oportunidad para establecer una correspondencia entre las clases y los componentes de implementación, y expandir las clases de alto nivel en una estructura de soporte. Los diagramas de implementación incluyen el diagrama de estructura interna, el diagrama de colaboración y el diagrama de componentes.

Diagrama de estructura interna

Una vez que comienza el proceso de diseño, las clases se deben descomponer en colecciones de partes conectadas que, posteriormente, se deben descomponer por turnos. Un clasificador estructurado modela las partes de una clase y sus conectores contextuales. Una clase estructurada puede ser encapsulada forzando a que las comunicaciones desde el exterior pasen a través de los puertos cumpliendo con las interfaces declaradas.

Un diagrama de estructura interna muestra la descomposición de una clase. La Figura 3.2 muestra el diagrama de estructura interna de la aplicación de la taquilla de teatro para el sistema de venta de entradas. Esta clase se descompone en tres partes: la interfaz del expendedor de entradas, una guía de representación, que recupera representaciones según la fecha u otro criterio, y un conjunto de bases de datos que contienen los datos de las representaciones y de las entradas. Cada parte interactúa con el exterior a través de un puerto. Lo mensajes sobre este puerto son enviados a la clase expendedor de entradas, pero la estructura interna de la clase taquilla está oculta de los clientes externos.

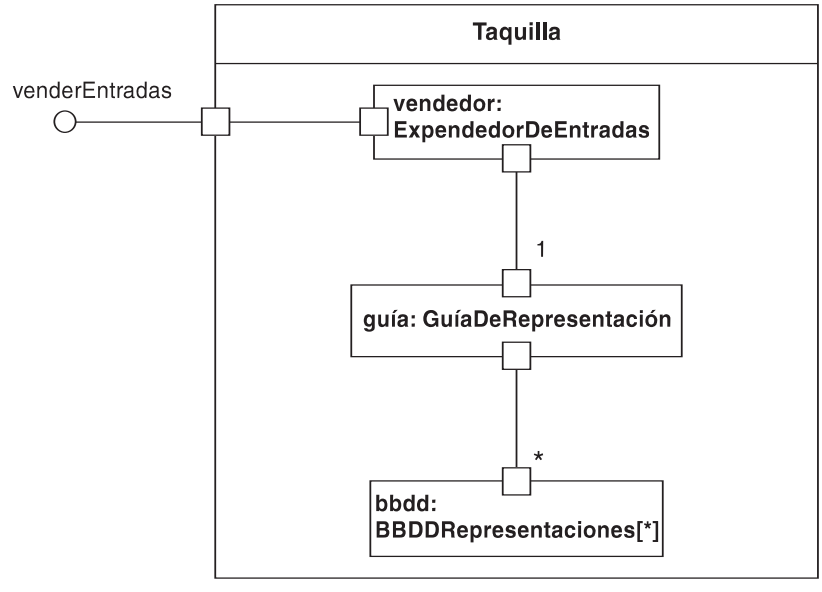


Figura 3.2 Diagrama de la estructura interna

Diagrama de colaboración

Una colaboración es una relación contextual entre un conjunto de objetos que trabajan juntos para lograr un propósito. Contiene una colección de roles —ranuras contextuales dentro de un patrón genérico, que pueden ser representadas por objetos individuales, o vinculadas a ellos.

La Figura 3.3 muestra un diagrama de colaboración para el sistema de venta del teatro. En él interactúan tres tipos distintos de componentes para proporcionar la funcionalidad al sistema: quiosco, terminales de ventas y la aplicación de la taquilla. Los diferentes componentes no pertenecen a una única clase global, sino que cooperan de maneras bien definidas para ofrecer servicios a los usuarios.

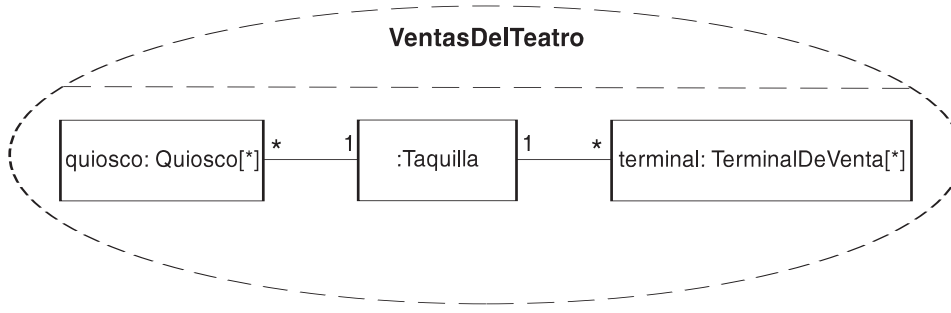


Figura 3.3 Diagrama de colaboración

Diagrama de componentes

Un componente es un tipo de clasificador estructurado, por lo que debe definirse su estructura interna en un diagrama de estructura interna. La Figura 3.4 muestra la estructura interna del sistema de venta del teatro. El sistema de venta del teatro se define como un componente cuya estructura interna contiene usos de otros componentes. Hay tres interfaces de usuario: la de los clientes que utilizan el quiosco, la de los empleados que utilizan el sistema de reserva automatizado y la de los supervisores que realizan consultas sobre la venta de entradas. Hay un componente expendedor de entradas que ordena las peticiones de los quioscos y de los empleados, un componente que procesa los cargos a las tarjetas de crédito, y un repositorio que almacena la información de las entradas. El diagrama de definición de componentes proporciona la estructura de un tipo de componente; una configuración específica de la aplicación puede utilizar más de una copia del componente.

Un pequeño círculo vinculado a un componente o a una clase es una interfaz proporcionada —un conjunto coherente de servicios proporcionados por el componente o la clase. Un pequeño semicírculo vinculado a un componente o a una clase es una interfaz obligatoria —una declaración de que el componente o la clase necesita obtener servicios de un elemento que los proporcione. Por ejemplo, el expendedor de entradas proporciona las ventas de suscripción y las ventas de grupo; las ventas de suscripción son accesibles tanto para los quioscos como para los empleados, pero las ventas de grupo sólo son accesibles para los empleados. Anidando una interfaz proporcionada con una interfaz obligatoria se indica que un componente llamará a otro para obtener los servicios que necesita. Hay que tener en cuenta que las interfaces pueden ser utilizadas con todos los clasificadores, y no sólo con los componentes.

Un diagrama de componentes muestra los componentes de un sistema —es decir, las unidades software con las que se construye la aplicación— así como las dependencias entre componentes, de forma que se pueda valorar el impacto de un cambio propuesto. La Figura 3.5 muestra un diagrama de componentes para los componentes utilizados en el componente agencia de tarjetas de crédito. Las líneas discontinuas de dependencia muestran interfaces proporcionadas y obligatorias compatibles entre sí, pero cuando las interfaces tienen los mismos nombres las líneas de dependencia son redundantes. En este ejemplo, el diagrama de componentes añade poco al diagrama de estructura interna. En un ejemplo más extenso, el diagrama de componentes combinaría componentes utilizados en diferentes lugares.

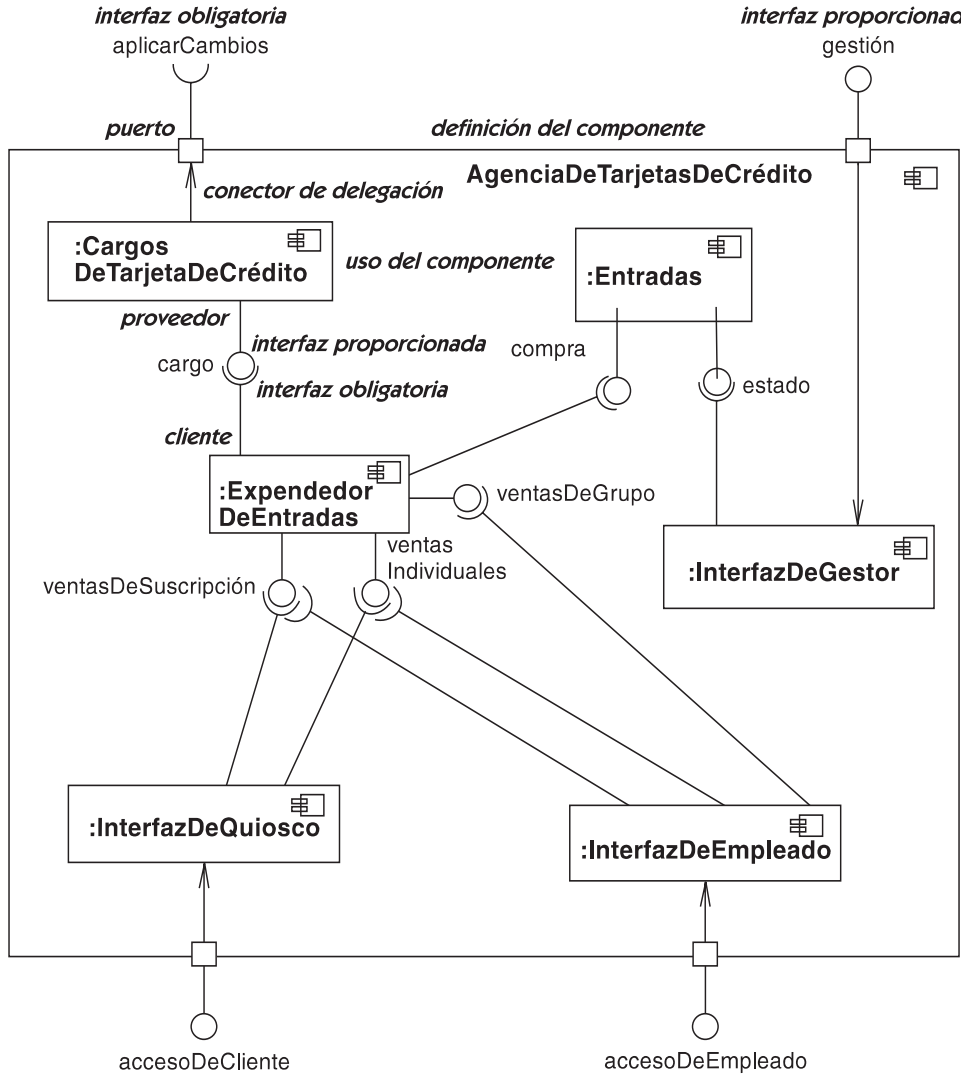


Figura 3.4 Definición del componente

Vista de casos de uso

La vista de casos de uso modela la funcionalidad de un sistema tal como lo perciben los agentes externos, denominados actores, que interactúan con el sistema desde un punto de vista particular. Un caso de uso es una unidad de funcionalidad expresada como una transacción entre los actores y el sistema. El propósito de la vista de casos de uso es enumerar los actores y casos de uso, y mostrar qué actores participan en cada caso de uso. El comportamiento de los casos de uso se expresa mediante las vistas dinámicas, especialmente la vista de interacción.

La Figura 3.6 muestra un diagrama de casos de uso para el ejemplo de la taquilla. Los actores son el empleado, el supervisor y el quiosco. El quiosco es un sistema distinto que acepta peticiones de un cliente. El cliente no es un actor de la aplicación taquilla porque no se encuentra

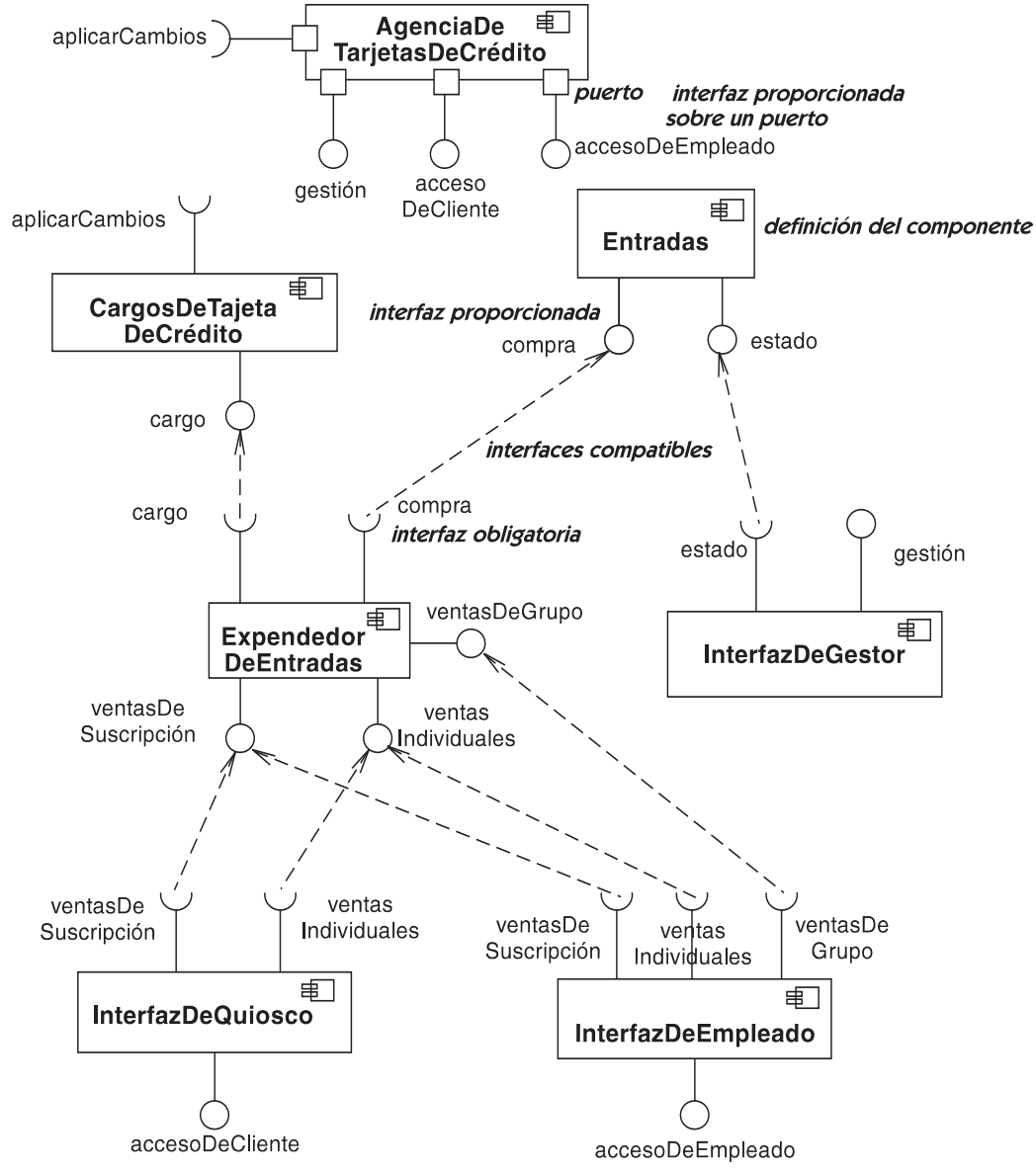


Figura 3.5 Diagrama de componentes

directamente conectado a la aplicación. Los casos de uso incluyen comprar entradas a través del quiosco o del empleado, comprar suscripciones (sólo a través del empleado y examinar las ventas (a petición del supervisor). La compra de entradas y de suscripciones incluyen un fragmento común —que es, realizar cargos en la tarjeta de crédito. (Una descripción completa del sistema implicaría muchos otros casos de uso, como el cambio de entradas y la comprobación de la disponibilidad.)

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

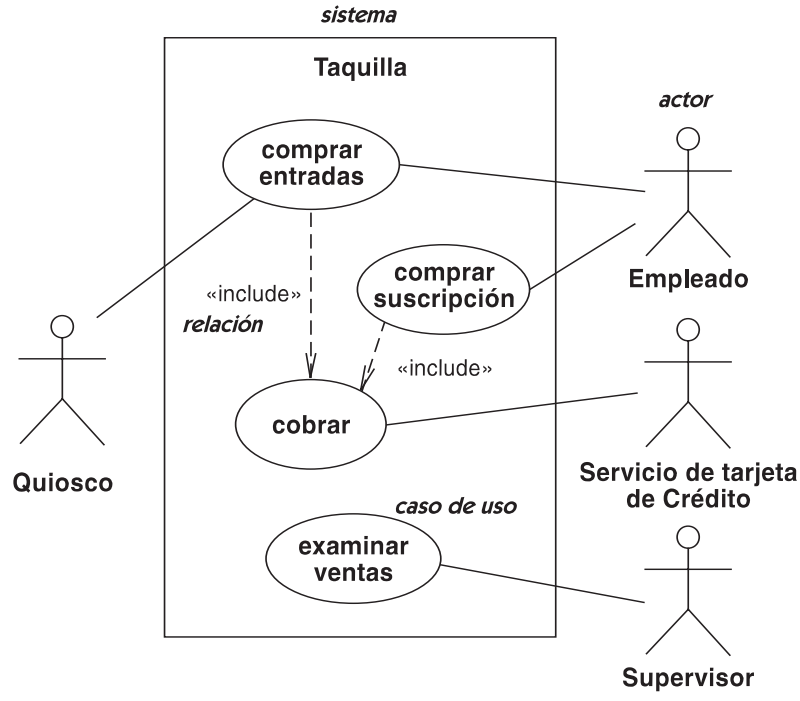


Figura 3.6 Diagrama de casos de uso

Vista de máquina de estados

Una máquina de estados modela las posibles historias de vida de un objeto de una clase. Una máquina de estados contiene estados conectados por transiciones. Cada estado modela un periodo 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 al objeto a un nuevo estado. Cuando se dispara una transición, se ejecuta un efecto (acción o actividad) asociada a la transición. Las máquinas de estados se muestran como diagramas de máquina de estados.

La Figura 3.7 muestra un diagrama de máquina 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 de que comience la temporada, se asignan los asientos para los abonados de la temporada. Las entradas individuales adquiridas interactivamente se bloquean primero mientras los clientes realizan una selección. Después de esto, se venden o se liberan si son rechazadas. Si el cliente tarda demasiado tiempo en realizar la selección, finaliza el tiempo para la transacción y el asiento se libera. Los asientos vendidos a los abonados de la temporada se pueden cambiar para otras representaciones, en cuyo caso, vuelven a estar disponibles de nuevo.

Las máquinas de estados pueden ser utilizadas para describir interfaces de usuario, controladores de dispositivos y otros subsistemas reactivos. También pueden ser utilizadas para describir objetos pasivos que pasan por varias fases cualitativamente distintas durante su tiempo de vida, cada una de las cuales tiene su propio comportamiento especial.

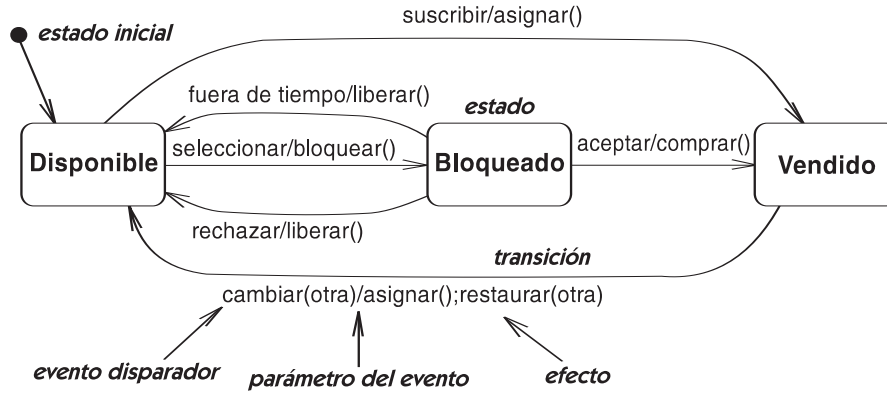


Figura 3.7 Diagrama de máquina de estados de uso

Vista de actividad

Una actividad muestra el flujo de control entre las actividades computacionales involucradas en la realización de un cálculo o un flujo de trabajo. Una acción es un paso computacional primitivo. Un nodo de actividad es un grupo de acciones o subactividades. Una actividad describe, tanto el cómputo secuencial, como el concurrente. Las actividades se muestran en los diagramas de actividad.

La Figura 3.8 muestra un diagrama de actividad para la taquilla. Este diagrama muestra las actividades involucradas en el montaje de una obra. (¡No se tome este ejemplo demasiado en serio si tiene experiencia teatral!) Las flechas muestran dependencias secuenciales. Por ejemplo, las obras deben ser seleccionadas antes de poder planificarlas. Las barras gruesas muestran las divisiones o uniones del control. Por ejemplo, después de que una obra se planifica, el teatro puede comenzar a anunciarla, a comprar los guiones, a contratar a los artistas, a construir los decorados, a diseñar la iluminación y a hacer el vestuario, todo ello de forma concurrente. No obstante, antes de que puedan comenzar los ensayos, se deben haber encargado los guiones y contratado a los artistas.

Este ejemplo muestra un diagrama de actividad, cuyo propósito es modelar los flujos de trabajo del mundo real de una organización humana. Este modelado de negocio es el propósito principal de los diagramas de actividad, aunque los diagramas de actividad también se pueden utilizar para modelar actividades software. Un diagrama de actividad es útil para comprender el comportamiento de alto nivel de la ejecución de un sistema, sin profundizar en los detalles internos del paso de mensajes, necesarios en un diagrama de colaboración.

Los parámetros de entrada y salida de una actividad pueden mostrarse utilizando relaciones de flujo que conecten la acción y nodos objeto.

Vista de interacción

La vista de interacción describe el intercambio de secuencias de mensajes entre las partes de un sistema. Una interacción está basada en un clasificador estructurado o en una colaboración. Un

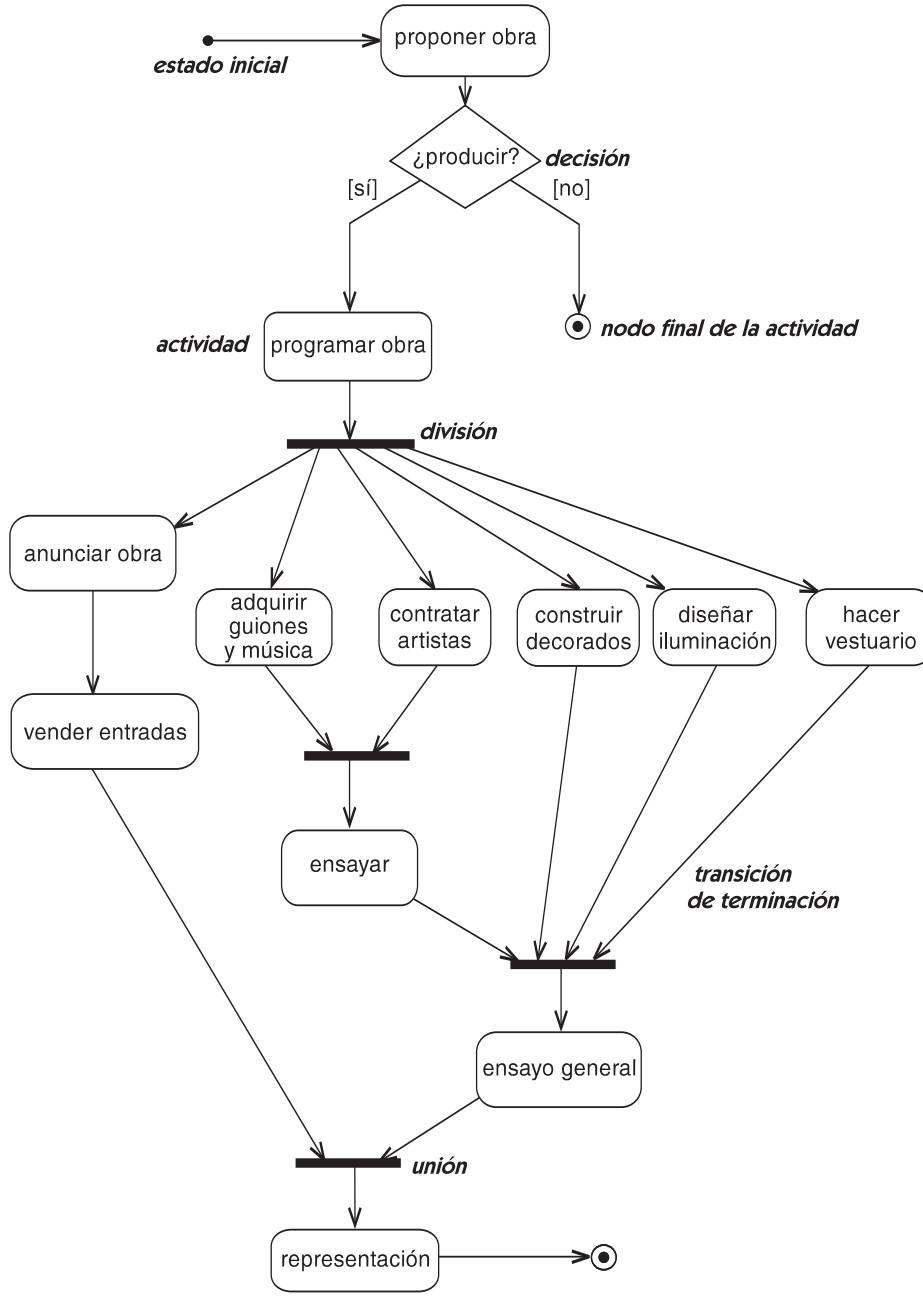


Figura 3.8 Diagrama de actividad

rol es una ranura que debe ser rellenada con objetos en un uso concreto de una interacción. Esta vista proporciona una visión integral del comportamiento de un sistema —es decir, muestra el flujo de control a través de varios objetos. La vista de interacción se muestra mediante dos diagramas que se centran en aspectos distintos: diagramas de secuencia (Figura 3.9) y diagramas de comunicación (Figura 3.10).

Diagrama de secuencia

Un diagrama de secuencia muestra un conjunto de mensajes ordenados en una secuencia temporal. Cada rol se muestra como una línea de vida —es decir, una línea vertical que representa al rol a lo largo del tiempo a través de la interacción completa. Los mensajes se muestran con flechas entre líneas de vida. Un diagrama de secuencia puede mostrar un escenario —una historia individual de una transacción. Las construcciones de control estructurado, como los bucles, las condiciones y las ejecuciones en paralelo, se muestran como rectángulos anidados con palabras clave y una o más regiones.

Un uso de un diagrama de secuencia es mostrar la secuencia de comportamiento de un caso de uso. Cuando el comportamiento se implementa, cada mensaje del diagrama de secuencia responde a una operación de una clase o a un evento disparado en una transición de una máquina de estados.

La Figura 3.9 muestra un diagrama de secuencia para el caso de uso **comprar entradas**. El contexto de la ejecución del caso de uso es una colaboración que involucra tres roles: cada uno de los

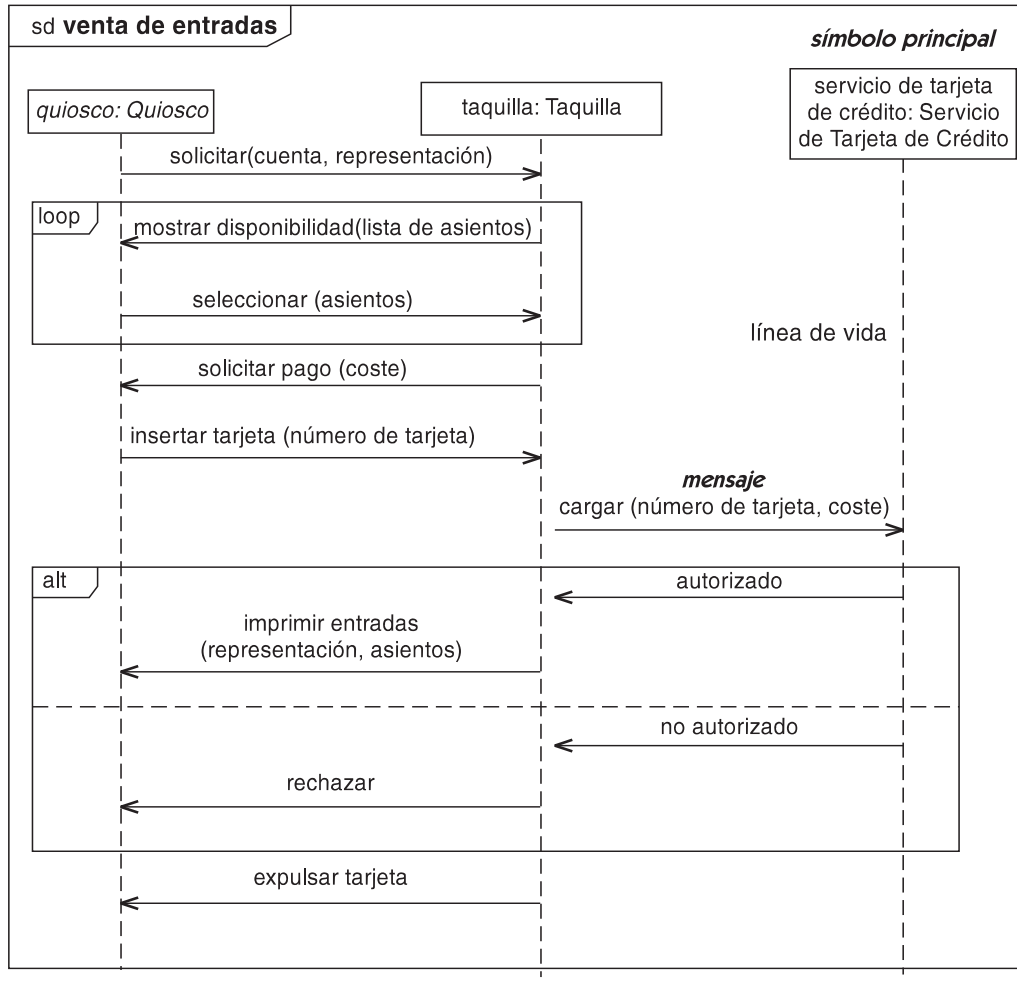


Figura 3.9 Diagrama de secuencia

tipos **Quiosco**, **Taquilla** y **Servicio de Tarjeta de Crédito**. Este caso de uso lo inicia el cliente en el quiosco que se comunica con la taquilla. Los pasos para el caso de uso **cobrar** se incluyen dentro de la secuencia, que involucra la comunicación, tanto con el quiosco, como con el servicio de tarjeta de crédito. Tanto el escenario en el que la operación se realiza con éxito, como el escenario en el que la operación falla, se muestran como alternativas. Este diagrama de secuencia es de una etapa temprana del desarrollo y no muestra todos los detalles de la interfaz de usuario. Por ejemplo, el formato exacto de la lista de asientos y el mecanismo para especificar asientos todavía debe determinarse, pero se ha especificado la comunicación esencial de la interacción del caso de uso.

Diagrama de comunicación

Un diagrama de comunicación muestra roles en una interacción con una disposición geométrica (Figura 3.10). Cada rectángulo muestra un rol —una línea de vida que representa la vida de un objeto a lo largo del tiempo. Los mensajes entre los objetos que desempeñan los roles se muestran como flechas vinculadas a los conectores. La secuencia de mensajes se indica mediante números de secuencia que preceden a la descripción de los mensajes.

Un uso de un diagrama de comunicación es mostrar la implementación de cualquier operación. Una colaboración muestra los parámetros y las variables locales de las operaciones como roles, así como asociaciones más permanentes. Cuando se implementa el comportamiento, la secuencia de los mensajes en un diagrama de comunicación se corresponde con la estructura de llamadas anidada y el paso de señales del programa.

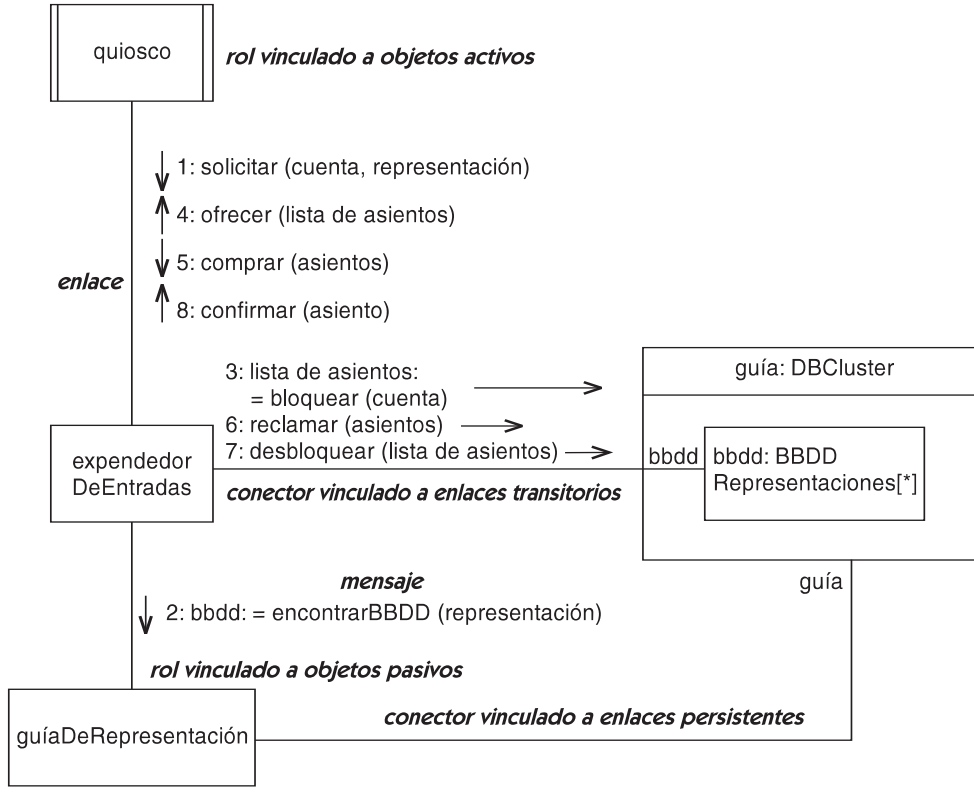


Figura 3.10 Diagrama de comunicación

La Figura 3.10 muestra un diagrama de comunicación para parte de la interacción de la reserva de entradas en una fase ulterior del desarrollo. Muestra la interacción entre objetos internos de 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 **bbdd** que se le devuelve al objeto **expendedorDeEntradas** representa un enlace local transitorio a una base de datos de representaciones que se mantiene durante la interacción y que se descarta posteriormente. El expendedor de entradas solicita un número de asientos para la representación; se encuentra una selección de asientos en varias gamas de precios, se bloquean temporalmente y se devuelven al quiosco para su selección por parte del cliente. Cuando el cliente realiza la selección a partir de la lista de asientos, los asientos seleccionados son reclamados y el resto son desbloqueados.

Tanto los diagramas de secuencia, como los diagramas de comunicación, muestran interacciones, pero cada uno acentúa aspectos distintos. Un diagrama de secuencia muestra la secuencia temporal como una dimensión geométrica, pero las relaciones entre roles son implícitas. Un diagrama de comunicación muestra geométricamente las relaciones entre roles y relaciona los mensajes con los conectores, pero las secuencias temporales son menos claras porque vienen dadas por los números de secuencia. Cada diagrama debe ser utilizado cuando su aspecto principal es el foco de atención.

Vista de despliegue

Un diagrama de despliegue representa el despliegue de artefactos de tiempo de ejecución sobre nodos. Un artefacto es una unidad de implementación física, como un archivo. Un nodo es un recurso de tiempo de ejecución, como una computadora, un dispositivo o la memoria. Un artefacto puede ser una manifestación (implementación) de uno o más componentes. Esta vista permite valorar las consecuencias de la distribución y de la asignación de recursos.

La Figura 3.11 muestra un diagrama de despliegue de nivel de descriptor para el sistema de la taquilla. Este diagrama muestra los tipos de nodos que hay en el sistema y los artefactos que contienen. Un nodo se muestra como un cubo. Un artefacto se muestra como un rectángulo con una palabra clave. La relación de manifestación muestra qué artefacto implementa cada componente.

Un tipo artefacto puede ser situado en diferentes tipos de nodos, y diferentes artefactos pueden manifestar el mismo tipo de componente.

La Figura 3.12 muestra un diagrama de despliegue de nivel de instancia para el sistema de la taquilla. El diagrama muestra los nodos individuales y sus enlaces en una configuración específica del sistema. La información de este modelo es consistente con la información de nivel de descriptor de la Figura 3.11.

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 las clases, máquinas de estados y casos de uso. Los paquetes pueden contener otros paquetes: por lo tanto, un modelo comienza con un paquete raíz que indirectamente alberga todos los contenidos del

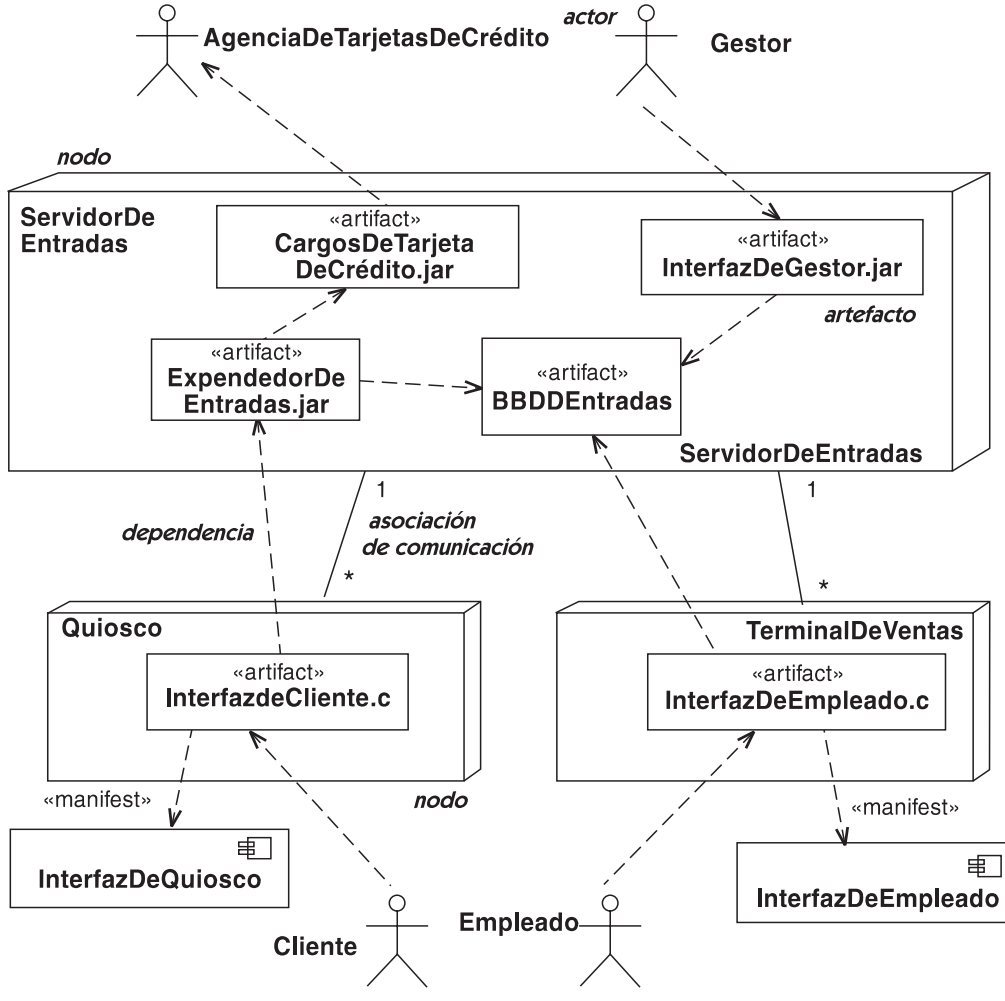


Figura 3.11 Diagrama de despliegue (nivel de descriptor)

modelo. Los paquetes son unidades para manipular los contenidos de un modelo, así como unidades para el control de acceso y el control de la 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 varios puntos de vista —por ejemplo, un modelo de análisis y un modelo de diseño. Un modelo puede ser mostrado como un tipo especial de paquete, pero habitualmente es suficiente con mostrar sólo los paquetes.

La información de la gestión del modelo se suele mostrar en diagramas de paquetes, que son una variedad de los diagramas de clases.

La Figura 3.13 muestra la descomposición del sistema completo del teatro en paquetes y sus relaciones de dependencia. El paquete de taquilla contiene paquetes para los ejemplos anteriores de este capítulo; la aplicación completa también incluye operaciones para la gestión del teatro y subsistemas de planificación. Cada subsistema consta de varios paquetes.

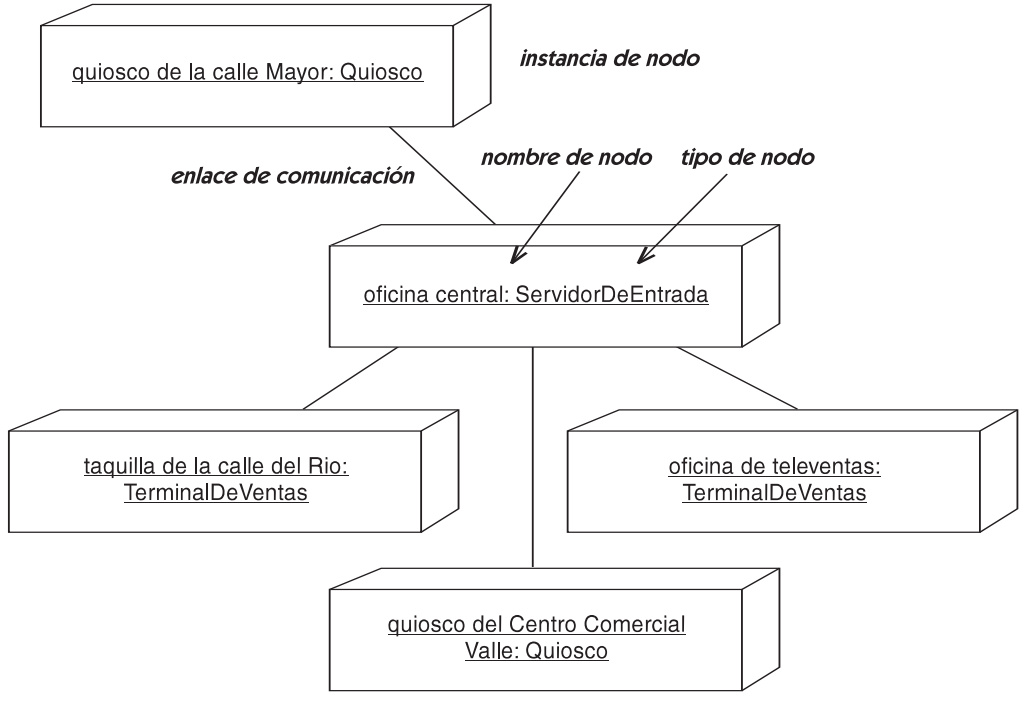


Figura 3.12 Diagrama de despliegue (nivel de instancia)

Perfiles

UML está definido utilizando un metamodelo, es decir, un modelo del propio lenguaje de modelado. La modificación del metamodelo es complicada y peligrosa. Además, muchas herramientas son construidas a partir del metamodelo estándar, y no funcionarían correctamente con un metamodelo diferente. El mecanismo de los perfiles permite cambios limitados sobre UML sin modificar el metamodelo subyacente. Los perfiles y las restricciones permiten que UML sea adaptado a dominios o plataformas específicas mientras mantiene la interoperatividad entre herramientas.

UML incluye tres constructores principales de extensibilidad: restricciones, estereotipos y valores etiquetados. Una restricción es una declaración de texto de una relación semántica expresada en algún tipo de lenguaje formal o en lenguaje natural. Un estereotipo es un nuevo tipo de elemento del modelo concebido por el modelador y basado en un tipo de elemento del modelo existente. Un valor etiquetado es una pieza de información con nombre vinculada a cualquier elemento del modelo.

Estos constructores permiten muchos tipos de extensiones sobre UML sin necesidad de cambios sobre el propio metamodelo básico de UML. Deben ser utilizados para crear versiones adaptadas de UML para un área de aplicación. Un conjunto coherente de estereotipo con su definición de etiquetas y restricciones se modela con un perfil.

La Figura 3.14 muestra ejemplos de restricciones, estereotipos y valores etiquetados. La restricción sobre la clase **Obra** asegura que los nombres de las obras sean únicos. La Figura 3.1

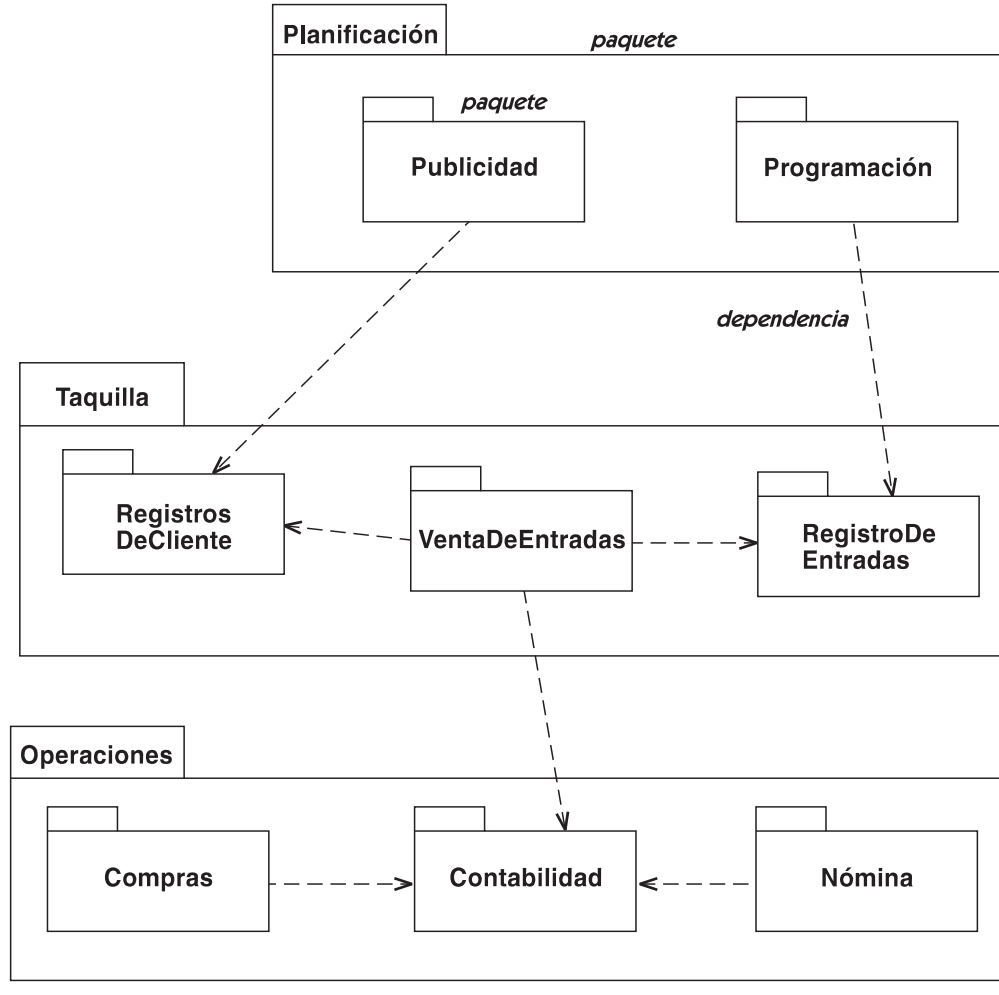


Figura 3.13 Diagrama de paquetes

muestra una restricción **xor** entre dos asociaciones; un objeto puede tener un enlace de cualquiera de ellas en un instante. Las restricciones son útiles para realizar declaraciones que puedan ser expresadas en lenguaje de texto, pero que no son soportadas directamente por los constructores de UML.

El estereotipo en el componente **BBDDEntradas** indica que el componente es una base de datos, lo que permite omitir las interfaces soportadas por el componente, ya que son soportadas por todas las bases de datos. Los modeladores pueden añadir nuevos estereotipos para representar elementos especiales. Se puede vincular un conjunto de restricciones, valores etiquetados o propiedades para la generación de código a un estereotipo. Un modelador puede definir un icono para un determinado nombre de estereotipo como ayuda visual, como se muestra en el diagrama. De todas formas, siempre se puede utilizar la forma textual.

Los valores etiquetados del paquete **Programación** muestran que Frank Martin es el responsable de acabarla antes del final de la década. Esta información para la gestión de proyecto se

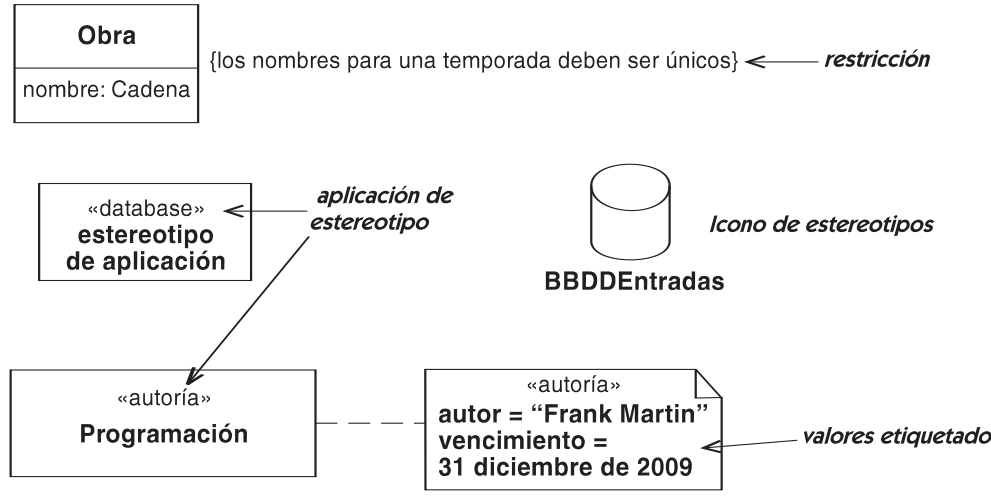


Figura 3.14 Constructores de extensión

define en el estereotipo **autoría** que se ha aplicado a la clase **Programación**. Los valores etiquetados son especialmente útiles para la información de la gestión del proyecto y para los parámetros de la generación de código. La mayoría de los valores etiquetados serían almacenados como información emergente en una herramienta de edición y no aparecerían habitualmente en las figuras impresas.

Los estereotipos y sus etiquetas se definen en los perfiles. Un perfil es un paquete que agrupa extensiones pensadas para ajustar el modelo a un dominio, tecnología o implementación específicos. Habitualmente, los perfiles serán predefinidos y almacenados en bibliotecas. Los perfiles se pueden aplicar a paquetes para extender los elementos que contienen.

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 —todo tipo de conceptos encontrados en los sistemas. Por ejemplo, un sistema de entradas para un teatro tiene conceptos como entradas, reservas, planes de suscripción, algoritmos para la asignación de asientos, páginas Web interactivas para hacer pedidos y datos de archivo para la redundancia.

La vista estática captura la estructura de los objetos. Un sistema orientado a objetos unifica la estructura de datos y las características de comportamiento en una única estructura de objetos. La vista estática incluye todo lo concerniente a la estructura de datos tradicional, así como la organización de las operaciones sobre los datos. Tanto los datos, como las operaciones, son cuantificadas en clases. Desde la perspectiva de la orientación a objetos, los datos y el comportamiento están íntimamente relacionados. Por ejemplo, un objeto **Entrada** lleva datos, como su precio, fecha de la representación y número de asiento, así como las operaciones sobre él, como reservar o calcular su precio con un descuento especial.

La vista estática describe las declaraciones de comportamiento, como las operaciones, como elementos discretos de modelado, pero no contiene los detalles de su comportamiento dinámico. Los trata como elementos que deben ser nombrados, pertenecer a las clases y ser invocados. Su ejecución dinámica se describe mediante otras vistas que describen los detalles internos de su faceta dinámica. Estas otras vistas incluyen a la vista de interacción y a la vista de máquina de estados. Las vistas dinámicas necesitan de la vista estática para describir los elementos que interactúan dinámicamente —no se puede decir *cómo* algo interactúa sin decir primero *qué* está interactuando. La vista estática es la base sobre la que se construyen el resto de las vistas.

Los elementos clave de la vista estática son los clasificadores y sus relaciones. Un clasificador es un elemento de modelado que describe cosas que contienen valores. Hay varios tipos de clasificadores, entre los que se incluyen las clases, las interfaces y los tipos de datos. Los aspectos de comportamiento, como los casos de uso y las señales, también se materializan como clasificadores. Los propósitos de implementación se encuentran detrás de algunos clasificadores, como los componentes, las colaboraciones y los nodos.

Los modelos grandes deben ser organizados en unidades más pequeñas para la comprensión humana y la reutilización. Un paquete es una unidad organizativa de propósito general que alberga y gestiona los contenidos de un modelo. Todo elemento está contenido en algún paquete. Un

modelo es un conjunto de paquetes que describe una vista completa del sistema, que puede ser utilizado de forma más o menos independiente de otros modelos; es un paquete raíz que indirectamente contiene los paquetes que describen el sistema.

Un objeto es una unidad discreta fuera de la cual el modelador comprende 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 varios tipos de dependencia, entre las que se incluye la realización y el uso.

Clasificadores

Un clasificador modela un concepto discreto que describe cosas (objetos) que tiene identidad, estado, comportamiento, relaciones y una estructura interna opcional. Los tipos de clasificadores incluyen clases, interfaces y tipos de datos. Otros tipos de clasificadores son materializaciones de conceptos de comportamiento, elementos en el entorno o estructuras de implementación. Estos clasificadores incluyen caso de uso, actor colaboración, componente y nodo, así como varios tipos de comportamiento. La Tabla 4.1 enumera los distintos tipos de clasificadores y sus funciones. El término clasificador del metamodelo incluye todos estos conceptos, pero dado que el término más familiar es clase, lo trataremos primero y definiremos el resto de los conceptos mediante diferencias con ella.

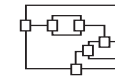
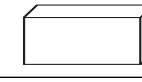
Clase. Una clase representa un concepto discreto dentro de la aplicación que se está modelando, que representa un elemento de un tipo particular —un elemento cosa física (como un avión), un elemento de negocio (como una solicitud), un elemento lógico (como la programación de la retransmisión de un evento), un elemento de una aplicación (como el botón de cancelar), un elemento de computación (como una tabla indexada) o un elemento de comportamiento (como una tarea). Una clase es el descriptor para un conjunto de objetos con similar estructura, comportamiento y relaciones. Todos los atributos y operaciones se vinculan a clases u otros clasificadores. Las clases son el punto alrededor del cual se organizan los sistemas orientados a objetos.

Un objeto es una entidad discreta con identidad, estado y comportamiento que se puede invocar. Los objetos son las piezas individuales de un sistema en tiempo de ejecución; las clases son los conceptos individuales con los que comprender y describir multitud de objetos individuales.

Una clase define un conjunto de objetos que tiene estado y comportamiento. El estado se describe mediante atributos y asociaciones. Los atributos se utilizan habitualmente para valores puros de datos sin identidad, como números y cadenas, y las asociaciones se utilizan para conexiones entre objetos con identidad. Las piezas individuales de comportamiento que se pueden invocar se describen como operaciones; un método es una implementación de una operación. La historia de la vida de un objeto se describe mediante una máquina de estados vinculada a una clase. La notación para una clase es un rectángulo con compartimentos para el nombre de la clase, atributos y operaciones, como se muestra en la Figura 4.1.

Un conjunto de clases puede utilizar la relación de generalización y el mecanismo de herencia construido en ella para compartir partes comunes del estado y la descripción del comportamiento. La generalización relaciona clases más específicas (subclases) con clases más generales (superclases) que contienen propiedades comunes a varias subclases. Una clase puede tener cero

Tabla 4.1 Tipos de clasificadores

Clasificador	*Función*	*Notación*
actor	Un usuario externo al sistema	
artefacto	Una pieza física del sistema de información	«artifact» **Nombre**
caso de uso	Una especificación del comportamiento de una entidad en su interacción con los agentes externos	
clase	Un concepto del sistema modelado	**Nombre**
clasificador estructurado	Un clasificador con estructura interna	
colaboración	Una relación contextual entre objetos que desempeñan roles	**Nombre**
componente	Una parte modular de un sistema con interfaces bien definidas	**Nombre**
enumeración	Un tipo de datos con valores literales predefinidos	«enumeration» **Nombre**
interfaz	Un conjunto con nombre de operaciones que caracterizan un comportamiento	«interface» **Nombre**
nodo	Un recurso computacional	
rol	Una parte iinterna en el contexto de una colaboración o clasificador estructurado	**rol:Nombre**
señal	Una comunicación asíncrona entre objetos	«signal» **Nombre**
tipo primitivo	Un descriptor de un conjunto de valores primitivos que carecen de identidad	**Nombre**

o más padres (superclases) y cero o más hijos (subclases). Una clase hereda el estado y la descripción del comportamiento de sus padres y otros antecesores, y define el estado y la descripción del comportamiento que sus hijos y otros descendientes heredan.

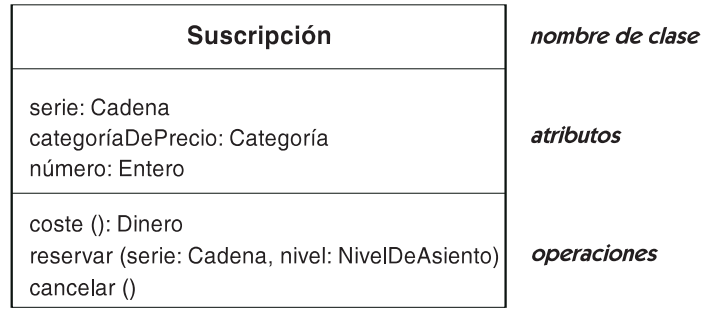


Figura 4.1 Notación de clase

Una clase tiene un nombre único dentro de su contenedor, que habitualmente es un paquete, pero que puede ser otra clase. La clase tiene una visibilidad con respecto a su contenedor; la visibilidad especifica cómo puede ser utilizada por otras clases fuera del contenedor.

Interfaz. Una interfaz es la descripción del comportamiento de los objetos sin proporcionar su implementación o estado. Una o más clases o componentes pueden realizar una interfaz, y cada clase implementa las operaciones que se encuentran en la interfaz.

Tipo de datos. Un tipo primitivo es la descripción de valores primitivos que carecen de identidad (existencia independiente y posibilidad de efectos laterales). Los enteros y las cadenas se encuentran entre los tipos primitivos. Los tipos primitivos se pasan por valor y son entidades inmutables. Un tipo primitivo no tiene atributos pero puede tener operaciones. Las operaciones no modifican los valores de los datos, pero pueden devolver valores de datos como resultado.

Los modelos de usuario también pueden declarar tipos enumerados. Un tipo enumerado declara un conjunto de literales enumerados que pueden ser utilizados como valores.

Niveles de significado. Las clases pueden existir con varios niveles de significación en un modelo, incluyendo los niveles de análisis, diseño e implementación. Cuando se representan conceptos del mundo real, es importante capturar estado, comportamiento y relaciones del mundo real. Pero los conceptos de implementación, como la ocultación de la información, eficiencia, visibilidad y métodos, no son conceptos relevantes del mundo real. *Son* conceptos relevantes del diseño. Muchas propiedades potenciales de una clase son irrelevantes a nivel de análisis. Una clase de nivel de análisis representa un concepto lógico en el dominio de la aplicación o en la propia aplicación. El modelo de análisis debe ser una representación mínima del sistema que modela, suficiente para capturar la lógica esencial del sistema sin entrar en temas de rendimiento o construcción.

Cuando se representa un diseño de alto nivel, conceptos como la localización del estado de una clase en particular, la eficiencia de la navegación entre objetos, la separación entre el comportamiento externo y la implementación interna, y la especificación de operaciones precisas son relevantes para una clase. Una clase de nivel de diseño representa la decisión de empaquetar la información del estado y de las operaciones sobre ella en una unidad discreta. Captura las decisiones clave de diseño, la ubicación de la información y la funcionalidad en los objetos. Las clases de nivel de diseño contienen tanto aspectos del mundo real como aspectos de un sistema informático.

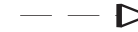
Finalmente, cuando se representa código de un lenguaje de programación, la forma de una clase se asemeja mucho al lenguaje de programación elegido, y muchas de las capacidades de una clase general pueden descartarse si no tienen implementación directa en el lenguaje. Una clase de nivel de implementación se corresponde directamente con el código del lenguaje de programación.

El mismo sistema puede ser modelado desde múltiples puntos de vista, como un modelo lógico captura la información del mundo real y el modelo de diseño captura las decisiones de representación interna. Una clase orientada a la implementación puede ser la realización de una clase en otro modelo. Una clase lógica captura los atributos y relaciones de la información del mundo real. Una clase de implementación representa la declaración de una clase como la encontramos en un determinado lenguaje de programación. Captura la forma exacta de una clase, según lo requerido por el lenguaje. Sin embargo, en muchos casos, la información del análisis, diseño e implementación se pueden anidar en una sola clase.

Relaciones

Las relaciones entre clasificadores son asociación, generalización y varios tipos de dependencia, entre los que se incluyen la realización y el uso (*véase* Tabla 4.2).

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	
generalización	Una relación entre una descripción más específica y una más general, utilizada para la herencia y para declaraciones de tipo polimórfico	
realización	Relación entre una especificación y su implementación	
uso	Una situación en la cual un elemento necesita de otro para su correcto funcionamiento	

La relación de asociación describe conexiones semánticas entre objetos individuales de clases dadas. Las asociaciones proporcionan las conexiones con las que pueden interactuar objetos de diferentes clases. Las relaciones restantes relacionan las descripciones de los propios clasificadores, no de sus instancias.

La relación de generalización relaciona descripciones generales de los clasificadores padre (superclases) con clasificadores hijo más especializados (subclases). La generalización facilita la descripción de clasificadores a partir de piezas de declaración incremental, cada una de las 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 la relación de generalización. La generalización y la herencia permiten que diferentes clasificadores compartan los atributos, operaciones y relaciones que tienen en común, sin repeticiones.

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

La relación de dependencia relaciona clases cuyo comportamiento o implementación afecta a otras clases. Existen varios tipos de dependencia, además de la de realización, incluyendo la dependencia de traza (conexión leve entre elementos en distintos modelos), refinamiento (correspondencia entre dos niveles de significación), uso (un requisito para la presencia de otro elemento en un modelo concreto) y ligadura (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 a nivel de código. La dependencia es particularmente útil cuando está resumida en unidades de organización del modelo, como los paquetes, en las que muestra la estructura arquitectónica de un sistema. Por ejemplo, las restricciones de compilación se pueden representar mediante dependencias.

Asociación

Una asociación describe conexiones discretas entre objetos u otras instancias en un sistema. Una asociación relaciona una lista ordenada (tupla) de dos o más clasificadores, en la que se permiten repeticiones. El tipo más habitual de asociación es la asociación binaria entre un par de clasificadores. Una instancia de una asociación es un enlace. Un enlace consta de una tupla (una lista ordenada) de objetos, cada uno obtenido a partir de su clase correspondiente. Un enlace binario comprende un par de objetos.

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

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

Cada conexión de una asociación con una clase se denomina extremo de la asociación. La mayor parte de la información sobre una asociación se vincula a uno de sus extremos. Los extremos de la asociación pueden tener nombres (nombres de rol) y visibilidad. Su propiedad más importante es la multiplicidad —cuántas instancias de una clase pueden estar relacionadas con una instancia de otra clase. La multiplicidad es más útil en las asociaciones binarias porque su definición en las asociaciones *n*-arias es más complicada. Si el límite superior de la multiplicidad es mayor que uno, se debe especificar en el extremo de la asociación la ordenación y unicidad de los valores asociados.

La notación para las asociaciones binarias es una línea o camino que conecta las clases participantes. El nombre de la asociación se sitúa a lo largo de la línea con el nombre del rol y la multiplicidad en cada extremo, como se muestra en la Figura 4.2.

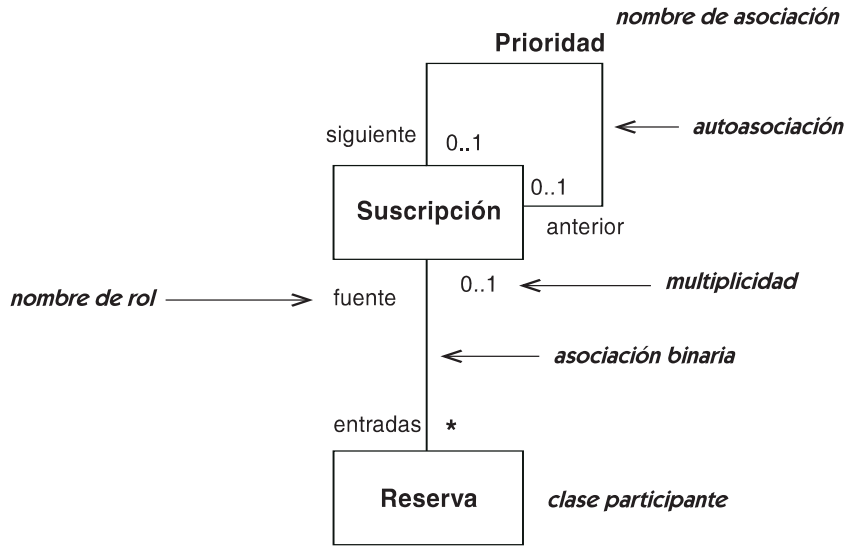


Figura 4.2 Notación de asociación

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

Durante el análisis, las asociaciones representan relaciones lógicas entre objetos. No hay una gran necesidad de imponer la dirección o de preocuparse de su implementación. Las asociaciones redundantes deben evitarse porque no aportan información lógica. Durante el diseño, las asociaciones capturan las decisiones de diseño sobre la estructura de los datos, así como la separación de responsabilidades entre clases. La dirección de las asociaciones es importante, se pueden

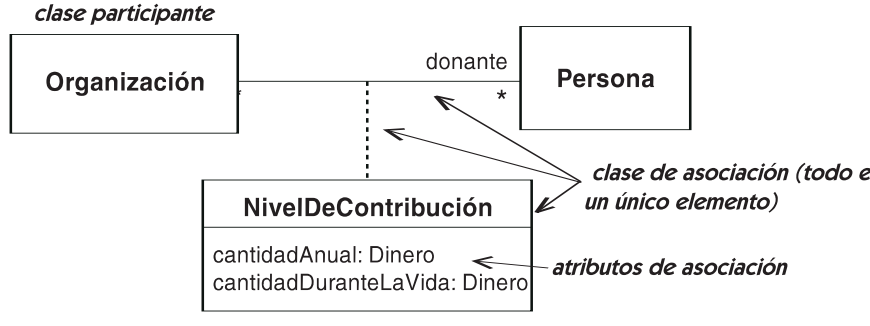


Figura 4.3 Clase de asociación

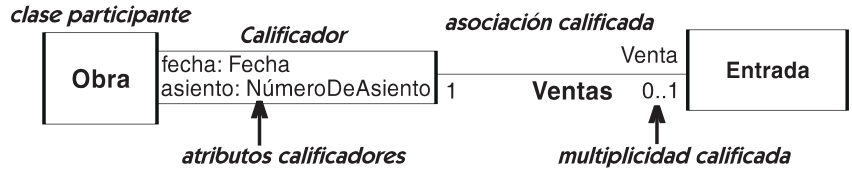


Figura 4.4 Asociación calificada

incluir asociaciones redundantes para mejorar la eficiencia del acceso a los datos, así como para localizar la información de una clase en particular. Sin embargo, en esta etapa del modelado, las asociaciones no se pueden equiparar con punteros de C++. Una asociación navegable en la etapa de diseño representa información de estado disponible para una clase, pero se puede implementar el código de un lenguaje de programación de distintas formas. La implementación puede ser un puntero, una clase contenedora embebida en una clase o incluso una tabla de objetos completamente separada. Otros tipos de propiedades de diseño incluyen la visibilidad y la posibilidad de cambio de enlaces. La Figura 4.5 muestra algunas propiedades de diseño de las asociaciones.

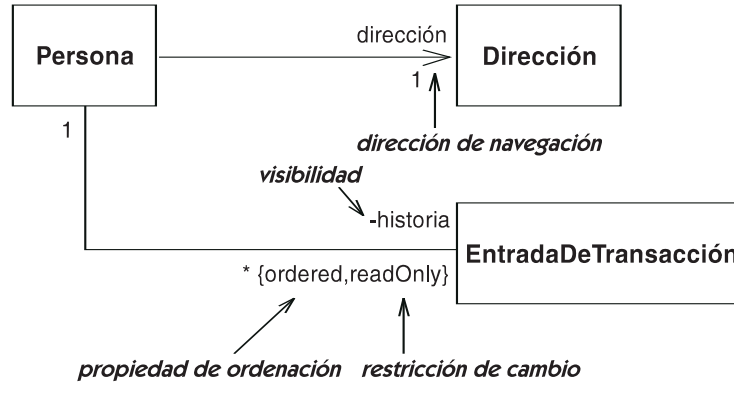


Figura 4.5 Propiedades de diseño de las asociaciones

Agregación y composición. Una agregación es una asociación que representa una relación todo-parte. Se representa 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, como en el caso de su asignación y desasignación. Se representa adornando con un diamante relleno el extremo compuesto. Hay una asociación independiente entre cada clase que representa una parte y la clase que representa el todo, pero por comodidad las trayectorias unidas al todo pueden ensamblarse juntas para dibujar el conjunto de asociaciones como si fuera 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 (tupla) de referencias a objetos, cada una de las 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 sis-

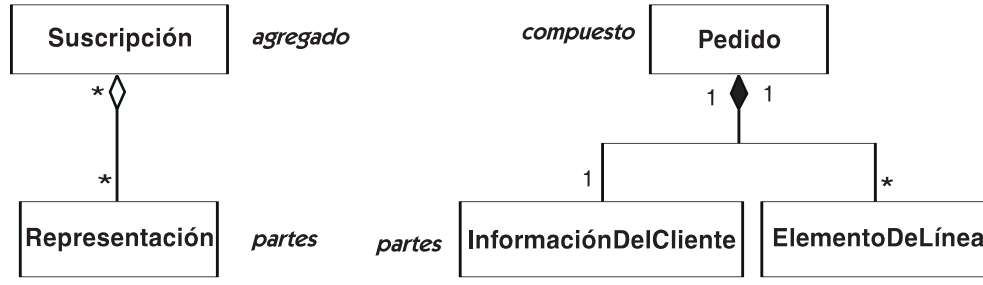


Figura 4.6 Agregación y composición

tema constituyen parte del estado del sistema. Los enlaces no existen independientemente de los objetos; toman su identidad de los objetos que relacionan (en términos de bases de datos, la lista de objetos es la *clave* para el enlace). Sin embargo, en el caso de las bolsas, puede haber múltiples enlaces que se correspondan con una tupla de objetos. Conceptualmente, una asociación es distinta de las clases que relaciona. En la práctica, las asociaciones son implementadas a menudo utilizando punteros en las clases participantes, pero pueden ser implementadas como un contenedor de objetos independiente de las clases a las que conecta.

Bidireccionalidad. Los distintos extremos de una asociación son distinguibles, incluso si dos de ellos involucran a la misma clase. Esto simplemente significa que se pueden relacionar objetos de la misma clase. Dado 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. En algunos casos, se dice que una asociación es bidireccional. Esto significa que la relación lógica funciona en ambos sentidos. Esta frase se suele interpretar mal, incluso por algunos metodólogos. No significa que cada clase "conozca" a la otra clase, o que, en una implementación, sea posible acceder a una clase desde la otra. Simplemente significa que cualquier relación lógica tiene una inversa, con independencia de que la inversa sea o no fácil de calcular. Las asociaciones se pueden marcar con direcciones de navegación, para establecer la capacidad de recorrer una asociación en una dirección pero no en la otra, como una decisión del diseño.

¿Por qué es relacional el modelo básico, en lugar de con punteros, que son un modelo más frecuente en los lenguajes de programación? La razón es que un modelo intenta capturar la intención subyacente a una implementación. Sin embargo, si la relación entre dos clases se modela como una pareja de punteros, los punteros están relacionados. El concepto de asociación reconoce que las relaciones son significativas en ambas direcciones, sin importar cómo estén implementadas. Es sencillo convertir una asociación en un par de punteros para la implementación, pero es muy difícil reconocer que cada uno de los dos punteros es el inverso del otro, 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 va extendiendo. 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 un tipo más específico de préstamo. Una hipoteca mantiene las características básicas de un crédito, pero añade características adicionales, tales como una casa como garantía para el préstamo. El descriptor más general se denomina padre; un elemento en la clausura transitiva es un antecesor. La descripción más específica se denomina hijo; un elemento en la clausura transitiva es un descendiente. En el ejemplo, **Préstamo** es la clase padre, e **Hipoteca** es la clase hija. La generalización se utiliza para los clasificadores (clases, interfaces, tipos de datos, casos de uso, actores, señales, etcétera). Para las clases, los términos superclase y subclase se utilizan para el padre y el hijo.

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

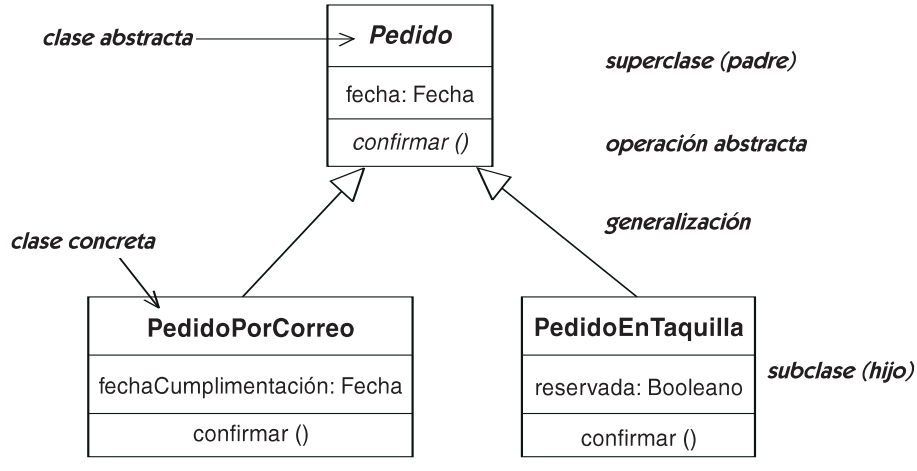


Figura 4.7 Notación de generalización

Propósito de la generalización. La generalización tiene dos propósitos. El primero es definir las condiciones bajo las cuales una instancia de una clase (u otro elemento) puede ser utilizado cuando se declara una variable (como un parámetro o variable de un procedimiento) conteniendo valores de una clase dada. Esto se denomina principio de sustitución (de Barbara Liskov). La regla es que una instancia de un descendiente puede ser utilizada dondequiera siempre que esté declarado el antecesor. Por ejemplo, si una variable se declara para contener préstamos, entonces un objeto hipoteca es un valor legal.

La generalización permite operaciones polimórficas —es decir, operaciones cuya implementación (método) se determina por la clase de objeto a la que se aplican, en lugar de ser explícitamente indicadas por el llamador. Esto funciona porque la clase padre puede tener muchos hijos posibles, cada uno de los cuales implementa su propia variación de una operación, la cual se define a través de todo el conjunto de clases. Por ejemplo, el cálculo de intereses podría funcionar de forma distinta para una hipoteca que para el préstamo para un automóvil, pero cada uno

de ellos es una variación del cálculo de intereses de la clase padre **Préstamo**. Cuando se declara una variable para albergar a la clase padre, se puede utilizar cualquier objeto de las clases hijas, cada una de las cuales tiene sus propias operaciones específicas. Esto es especialmente útil, puesto que se pueden añadir nuevas clases posteriormente sin necesidad de modificar las llamadas polimórficas existentes. Por ejemplo, se puede añadir posteriormente un nuevo tipo de préstamo, y el código existente que utiliza la operación **calcular intereses** seguiría funcionando. Una operación polimórfica se puede declarar sin ninguna implementación en la clase padre, con la intención de que sea proporcionada por cada clase descendiente. Tal operación incompleta es abstracta (y se muestra poniendo su nombre en cursiva).

El otro propósito para la generalización es permitir la descripción incremental de un elemento que comparte las descripciones con sus antecesores. Esto se denomina herencia. La herencia es el mecanismo mediante el cual una descripción de un objeto de una clase se construye a partir de fragmentos de declaraciones de la clase y de sus antecesores. La herencia permite que las partes compartidas de la descripción se declaren una vez y sean compartidas por muchas clases, en lugar de repetirlas en cada clase que las utiliza. Al compartir se reduce el tamaño del modelo. Más importante aún, reduce el número de cambios que se deben realizar en una actualización del modelo y reduce el riesgo de la inconsistencia accidental. La herencia funciona de forma similar para otros tipos de elementos, como estados, señales y casos de uso.

Herencia

Cada tipo de elemento generalizable tiene un conjunto de propiedades que puede heredar. Para cualquier elemento del modelo, éstas incluyen a las restricciones. Para los clasificadores, también incluyen características (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.

Para un clasificador, no se puede declarar más de una vez (directamente o heredado) un atributo con la misma signatura. De lo contrario, hay un conflicto y el modelo está mal formado. En otras palabras, un atributo declarado en un antecesor no puede ser redeclarado en un descendiente. Una operación puede ser declarada en más de un clasificador, siempre que la declaración sea consistente (los mismos parámetros, restricciones y significado). Las declaraciones adicionales son, simplemente, redundantes. Un método puede ser declarado por múltiples clases en una jerarquía. Un método vinculado a un descendiente suplanta y reemplaza (anula) 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 entran en conflicto y el modelos está mal formado. (Algunos lenguajes de programación permiten elegir explícitamente uno de los métodos. Encontramos más sencillo y seguro redefinir el método en la clase hija.) Las restricciones sobre un elemento son la unión de las restricciones del propio elemento y de todos sus antecesores; si cualquiera de ellas es inconsistente el modelo está mal formado.

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

Bajo algunas circunstancias, una redefinición de una definición heredada debe ser declarada en una subclase. La redefinición puede cambiar el nombre o algunas de las propiedades de una característica, pero ello puede crear confusión y debe ser utilizado en contadas ocasiones.

Herencia múltiple

Si un clasificador tiene más de un padre, hereda de cada uno de ellos (Figura 4.8). Sus características (atributos, operaciones y señales) son la unión de las de sus padres. Sin embargo, si la misma clase aparece como un antecesor por más de un camino, sólo contribuye con una copia de cada uno de sus miembros. Si dos o más clases declaran una característica con la misma signatura, que no heredan del mismo antecesor (declaraciones independientes), entonces las declaraciones entran en conflicto y el modelo está mal formado. UML no proporciona una regla de resolución de conflictos para esta situación porque la experiencia ha mostrado que el diseñador debe resolverla explícitamente. Algunos lenguajes, como Eiffel, permiten que los conflictos sean explícitamente resueltos por el programador, lo que es mucho más seguro que las reglas implícitas de resolución, que conducen con frecuencia a sorpresas para el desarrollador.

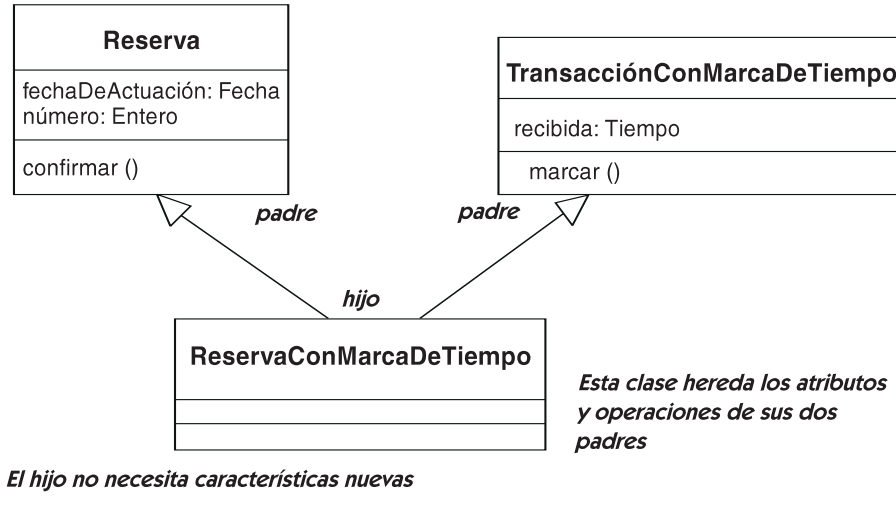


Figura 4.8 Herencia múltiple

Clasificación simple y múltiple

Muchos lenguajes orientados a objetos asumen clasificación simple —un objeto tiene una única clase directa. No hay necesidad lógica de que un objeto tenga una sola clase. Nosotros observamos los objetos del mundo real desde muchos ángulos simultáneamente. En la formulación general de UML, un objeto puede tener una o más clases directas —clasificación múltiple. El objeto se comporta como si perteneciera a una clase implícita que es hija de cada una de las clases directas —de hecho, es herencia múltiple sin la necesidad real de declarar la nueva clase.

Clasificación estática y dinámica

En la formulación más simple, un objeto no puede cambiar su clase después de ser creado (clasificación estática). De nuevo, no hay necesidad lógica para esta restricción. Sobre todo, tiene la intención de hacer la implementación de los lenguajes de programación orientados a objetos más sencillos. En una formulación más general, un objeto puede cambiar su clase directa

dinámicamente (clasificación dinámica). Haciendo esto, puede ganar o perder atributos y operaciones. Si los pierde, la información que contienen se pierde y no se puede recuperar más tarde, incluso si se vuelve a cambiar a la clase original. Si gana atributos o asociaciones, entonces debe ser inicializado en el momento del cambio, de una forma similar a la inicialización de un objeto nuevo.

Cuando se combina la clasificación múltiple con la clasificación dinámica, un objeto puede añadir y quitar clases a su clasificación durante su vida. La clasificación dinámica se denomina a veces roles o tipos. Un patrón de modelado común es exigir que cada objeto tenga una única clase inherente estática (una que no pueda cambiar durante la vida del objeto) más uno o más roles que puedan ser añadidos o eliminados a lo largo de la vida del objeto. La clase inherente describe sus propiedades fundamentales, y las clases de rol describen propiedades que son transitorias. Aunque muchos lenguajes de programación no soportan la clasificación dinámica múltiple en la jerarquía de declaración de clases, no obstante es un concepto de modelado valioso que se puede representar con asociaciones.

Realización

La relación de realización (Figura 4.9) conecta a un elemento del modelo, como una clase, con otro elemento del modelo, como una interfaz, que le proporciona su especificación de comportamiento, pero no su estructura o implementación. El cliente debe tener (por herencia o por declaración directa) al menos todas las operaciones que tiene el proveedor. Aunque se pretende que la realización sea utilizada con elementos de especificación, como interfaces, también puede ser utilizada con elementos concretos de implementación para indicar que su especificación (pero no su implementación) debe estar presente. Por ejemplo, esto podría ser utilizado para mostrar la relación de una versión optimizada de una clase con una versión más sencilla pero ineficiente.

Tanto la generalización, como la realización relacionan una descripción más general con versiones más completas de la misma. La generalización relaciona dos elementos del mismo nivel

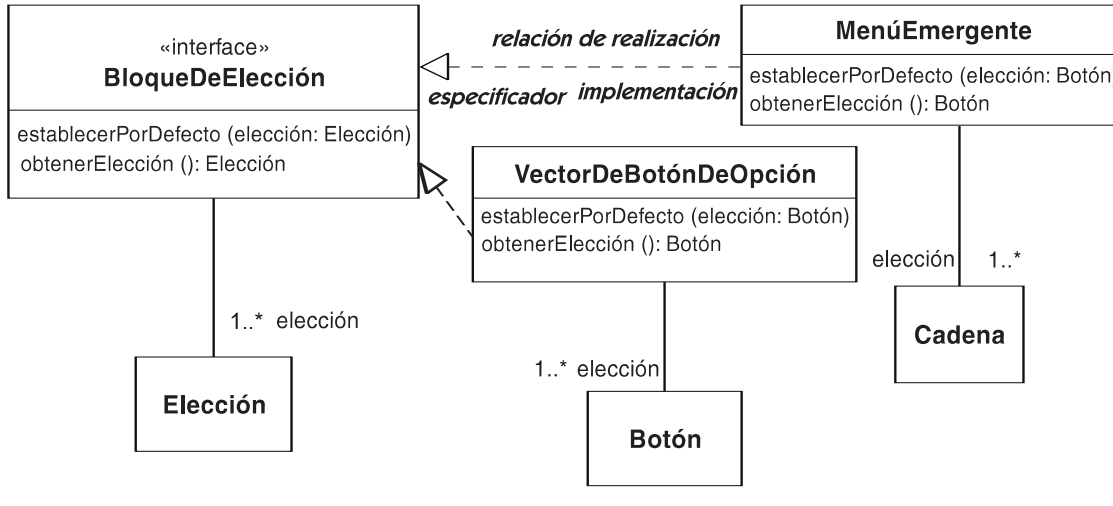


Figura 4.9 Relación de realización

semántico (por ejemplo, del mismo nivel de abstracción), normalmente, dentro del mismo modelo; la realización relaciona dos elementos de diferentes niveles semánticos (por ejemplo, una clase de análisis y una de diseño, o una interfaz y una clase), que a menudo se encuentran en distintos modelos. Puede haber dos o más jerarquías completas de clases en diferentes etapas del desarrollo cuyos elementos estén relacionados por la realización. Las dos jerarquías no necesitan tener la misma forma porque la realización de las clases puede tener dependencias de implementación que no son relevantes para la especificación de las clases.

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

Una interfaz realizada por una clase se denomina interfaz proporcionada, porque la clase proporciona los servicios de la interfaz a los que los invocan desde el exterior. Esta relación se debe mostrar vinculando, con una línea, un pequeño círculo al rectángulo de la clase; el círculo se etiqueta con el nombre de la interfaz. Una interfaz utilizada por una clase para implementar su comportamiento interno se denomina interfaz obligatoria. Esta relación se debe mostrar vinculando, con una línea, un pequeño semicírculo al rectángulo de la clase. Esta notación no se utiliza para declarar interfaces, sino para mostrar sus relaciones con las clases. *Véase* Figura 4.10.

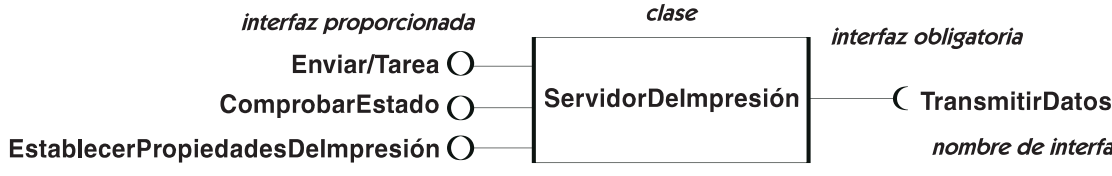


Figura 4.10 Interfaces proporcionadas y obligatorias

Dependencia

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

Según esta definición, las relaciones de asociación y de generalización son dependencias, pero éstas tienen semánticas específicas con consecuencias importantes. Por lo tanto, ellas tienen sus propios nombres y su semántica detallada. Nosotros utilizamos habitualmente la palabra dependencia para todas las demás relaciones que no encajan en categorías más definidas. La Tabla 4.3 enumera los tipos de dependencias aplicables a la vista estática.

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

Tabla 4.3 Tipos de dependencias

Relación	*Función*	*Palabra clave*[1]
acceso	Importación privada de los contenidos de otro paquete	**access/accede**
creación	Establece que una clase crea instancias de otra clase	**create/crea**
dependencia de traza	Establece que existe alguna conexión entre elementos en diferentes modelos, pero menos preciso que una correspondencia	**trace*/traza**
derivación	Establece que una instancia puede ser calculada a partir de otra instancia	**derive/deriva**
envío	Relación entre el emisor de una señal y el receptor de la señal	**send/envía**
instanciación	Establece que un método de una clase crea instancias de otra clase	**instantiate/usa instancias**
ligadura	Asignación de valores a los parámetros de una plantilla, para generar un nuevo elemento del modelo	**bind*/ligado**
llamada	Establece que un método de una clase llama a una operación de otra clase	**call/llama**
permiso	Permiso para que un elemento utilice los contenidos de otro elemento	**permit/permite**
realización	Correspondencia entre una especificación y su implementación	**realize/realiza**
refinamiento	Establece que existe una correspondencia entre elementos a dos niveles semánticos distintos	**refine/refina**
sustitución	Establece que la clase origen soporta las interfaces y contratos de la clase destino y puede ser sustituida por ella	**substitute/ sustituye**
uso	Establece que un elemento necesita de la presencia de otro elemento para su correcto funcionamiento (incluye llamada, creación, instanciación, envío, pero está abierto a otros tipos)	**use*/usa**

[1] Estas palabras clave son estereotipos (*véase* Capítulo 11) y por tanto pueden estar en cualquier idioma. La forma inglesa puede ser 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.*)

Un refinamiento es una relación entre dos versiones de un concepto en diferentes etapas del desarrollo o a diferentes niveles de abstracción, expresada como dos elementos separados del modelo. Los dos elementos del modelo no pretenden coexistir en el modelo detallado final. Normalmente, uno de ellos es una versión menos terminada del otro. En principio, hay una correspondencia entre el elemento menos terminado y el más terminado. Esto no implica que la traducción sea automática. Es habitual que el elemento más detallado contenga decisiones de diseño tomadas por el diseñador, decisiones que podrían haber sido tomadas de muchas formas. En principio, los cambios sobre un modelo podrían ser validados contra el otro, etiquetando las desviaciones. En la práctica, las herramientas no pueden hacer todas estas cosas en la actualidad, aunque se pueden implementar correspondencias más sencillas. Por lo tanto, un refinamiento es, sobre todo, un recordatorio para el modelador de que múltiples modelos se encuentran relacionados de una manera predecible.

Una dependencia de derivación indica que un elemento puede ser calculado a partir de otro elemento (pero el elemento derivado debe ser incluido explícitamente en el sistema para evitar un recálculo costoso). Derivación, realización, refinamiento y 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 afecta al comportamiento o implementación de otro. 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 derivan del código y no necesitan ser declaradas explícitamente, a menos que sean parte de un diseño de estilo descendente que limita la organización del sistema (por ejemplo, utilizando componentes predefinidos y bibliotecas). Se debe especificar el tipo concreto de dependencia de uso, pero a menudo se omite porque el propósito de la relación es resaltar la dependencia. Con frecuencia, los detalles exactos se pueden obtener del código de implementación. Los estereotipos de uso incluyen la llamada y la instanciación. La dependencia de llamada indica que un método de una clase llama a una operación de otra clase; la instanciación indica que un método de una clase crea una instancia de otra clase.

Algunas variedades de dependencia añaden elementos a un espacio de nombres. La dependencia de importación añade los nombres de los contenidos del espacio de nombres destino al espacio de nombres desde el que se importa. La dependencia de acceso también añade nombres a un espacio de nombres, pero los nombres añadidos no son visibles fuera del espacio de nombres al que se añadieron.

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.

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

Una dependencia se dibuja como una flecha de línea discontinua del cliente al proveedor, con la palabra clave del estereotipo para distinguir su tipo, como se muestra en la Figura 4.11.

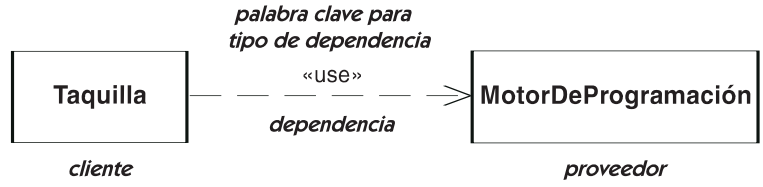


Figura 4.11 Dependencias

Restricción

UML proporciona un conjunto de conceptos y relaciones para modelar sistemas como grafos de elementos de modelado. Sin embargo, algunas cosas se expresan mejor lingüísticamente —es decir, utilizando la potencia del lenguaje textual. Una restricción es una expresión Booleana representada como una cadena que debe ser interpretada en un determinado lenguaje. Para expresar restricciones se pueden utilizar el lenguaje natural, notación de conjuntos, lenguajes de restricción y algunos lenguajes de programación. UML incluye la definición de un lenguaje de restricción, llamado OCL, adecuado para expresar restricciones de UML y que se espera que tenga un amplio soporte. Para más información sobre OCL, véase la referencia OCL y el libro [Warmer-99]. La Figura 4.12 muestra algunas restricciones.

Se pueden utilizar las restricciones para indicar varias restricciones no locales, como restricciones en las asociaciones. En particular, se pueden utilizar las restricciones 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 en UML, como las asociaciones en una relación de "o exclusivo" o varias restricciones entre las subclases en la generalización.

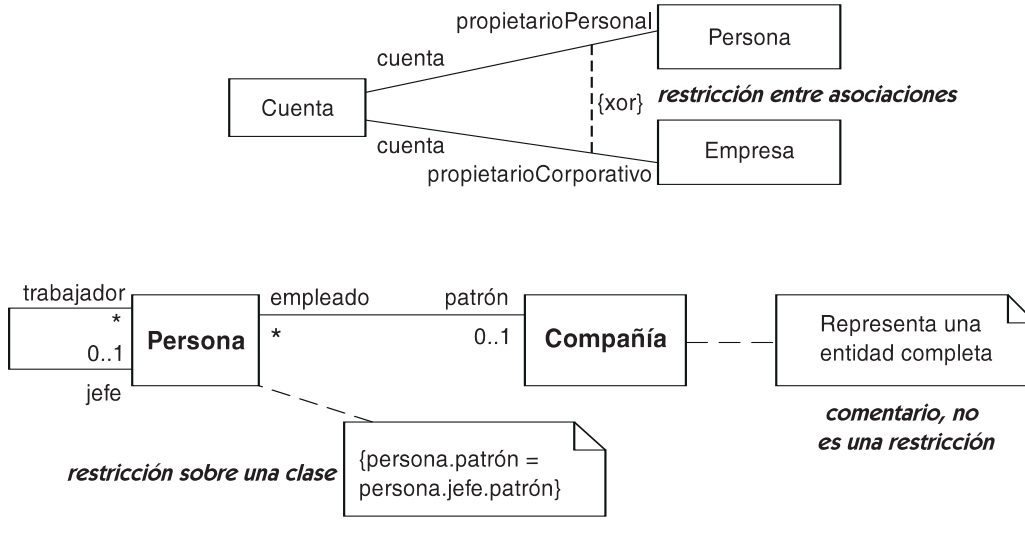


Figura 4.12 Restricciones

Las restricciones se muestran como expresiones de texto entre llaves. Se deben escribir en un lenguaje formal o en lenguaje natural. La cadena de texto se debe poner en una nota o vinculada a una flecha de dependencia.

Las restricciones predefinidas se pueden expresar en OCL. Por ejemplo, en la Figura 4.12, la restricción xor (o exclusivo) se puede escribir en OCL de la siguiente forma:

```
context Cuenta inv
   propietarioPersonal -> tamaño > 0   xor   propietarioCorporativo -> tamaño > 0
```

Instancia

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

Un propósito de un modelo es describir los posibles estados de un sistema y sus comportamientos. Un modelo es una declaración de potencial, de las posibles colecciones de objetos que podrían existir y de la posible historia de comportamiento que los objetos podrían 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 formas en las que 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 del sistema.

Una configuración estática específica de un sistema en un instante se llama instantánea. Una instantánea consta de 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 completamente y una instancia indirecta de los antecesores de esa clase. (Si se permite la clasificación múltiple, entonces un objeto puede ser instancia directa de más de una clase.) De la misma forma, 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 una multiplicidad opcional o múltiple, entonces podrá albergar cero o más valores. Un enlace consta de una tupla de valores, cada uno de los cuales es una referencia a un objeto de una clase dada (o una de sus descendientes). Los objetos y los enlaces deben obedecer cualquier restricción sobre las clases o asociaciones de las que son instancias (incluyendo, tanto restricciones explícitas, como restricciones intrínsecas, como la multiplicidad).

El estado de un sistema es una *instancia válida del sistema* si cada instancia en él es una instancia de algún elemento de un modelo de sistema bien formado y si todas la restricciones impuestas por el modelo son satisfechas por las instancias.

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

Las partes de comportamiento de UML describen las secuencias válidas de las instantáneas que pueden darse como resultado de los efectos del comportamiento, tanto interno, como externo. 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 del sistema en un instante en el tiempo. Dado que contiene imágenes de objetos, se denomina diagrama de objetos. Puede ser útil como ejemplo del sistema para, por ejemplo, ilustrar estructuras de datos complejas o para mostrar el comportamiento a través de una secuencia de instantáneas a lo largo del tiempo (Figura 4.13). Un diagrama de objetos no está restringido a la especificación de objetos. Puede incluir especificaciones de los valores de objetos en los que algunos de los valores pueden estar especificados de forma incompleta, indicando, por ejemplo, un rango de valores en lugar de un valor específico.

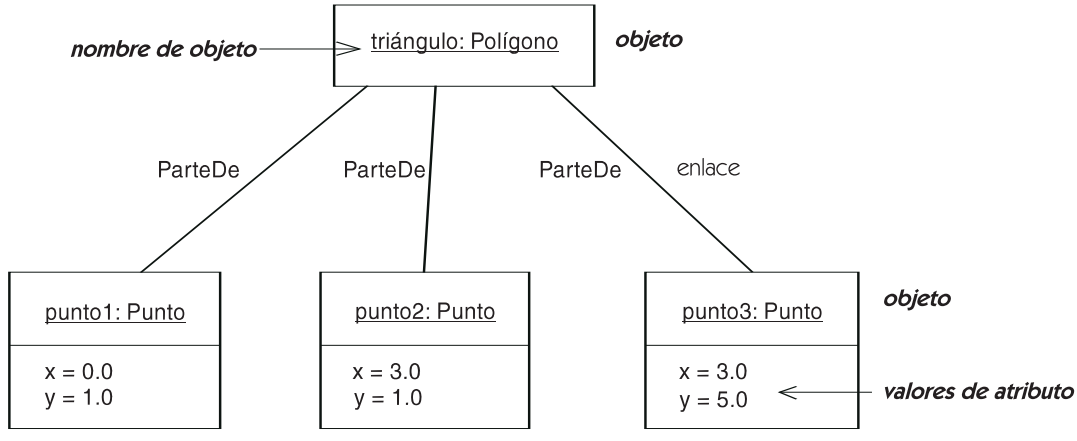


Figura 4.13 Diagrama de objetos

Las instantáneas son ejemplos de sistemas, no definiciones de sistemas. La definición de la estructura y el comportamiento de un sistema es el objetivo del modelado y el diseño. Los ejemplos pueden ayudar a aclarar significados a las personas, pero no son definiciones.

5
La vista de diseño

Descripción

Una buena parte del modelado de un sistema tiene la intención de mostrar los aspectos lógico y de diseño de un sistema, con independencia de su empaquetamiento final en el medio de implementación. No obstante, los aspectos de implementación son importantes, tanto para propósitos de reutilización, como de rendimiento. La vista de diseño muestra las decisiones sobre la descomposición de un sistema en unidades modulares con límites de encapsulamiento e interfaces externas. Aunque los elementos en la vista de diseño son más abstractos que el código final, necesitan de conocimientos sobre los compromisos de implementación que se reflejarán finalmente en el código.

Los sistemas complejos necesitan de múltiples niveles de estructura. Durante las primeras etapas del modelado, una clase se encuentra definida por sus propiedades externas. Durante el modelado de diseño, el diseño interno de las clases de alto nivel puede expandirse en partes constituyentes. Un clasificador estructurado es un clasificador con partes internas conectadas dentro del contexto del clasificador. Un clasificador estructurado puede tener una frontera débil o una fuerte, donde todas las comunicaciones tienen lugar sobre puntos de interacción bien definidos denominados puertos. Los propios tipos de partes internas deben ser clasificadores estructurados; por tanto la descomposición del sistema debe abarcar varios niveles.

En un diseño, objetos independientes suelen trabajar juntos para realizar operaciones y otros comportamientos. Durante un tiempo limitado, los objetos se encuentran relacionados por su participación en un contexto compartido. Una colaboración es una descripción de un grupo de objetos que tienen una relación provisional dentro del contexto de la realización de un comportamiento. Una colaboración es una relación conceptual, no un objeto concreto, aunque puede haber objetos en una implementación relacionados con ella. Las conexiones entre objetos pueden incluir varios tipos de relaciones transitorias, tales como parámetros, variables y relaciones de derivación, así como asociaciones normales.

La vista de diseño muestra la organización lógica de piezas reutilizables del sistema en unidades sustituibles, denominadas componentes. Un componente tiene un conjunto de interfaces externas y una implementación interna y oculta. Los componentes interactúan a través de interfaces para evitar las dependencias sobre otros componentes específicos. Durante la implementación, cualquier componente que soporte una interfaz puede ser sustituido por él, permitiendo que distintas partes de un sistema sean desarrolladas sin la dependencia de sus implementaciones internas.

Clasificador estructurado

Un clasificador estructurado es un clasificador con estructura interna. Contiene un conjunto de partes conectadas mediante conectores. Una instancia de una clase estructurada contiene un objeto con un conjunto de objetos que se corresponden con cada parte. Una parte tiene un tipo y una multiplicidad dentro de su contenedor. Un objeto que es una parte sólo puede pertenecer a un objeto estructurado. Todos los objetos en un único objeto estructurado se encuentran relacionados implícitamente por estar contenidos en el mismo objeto. Esta relación implícita puede ser aprovechada en la implementación del comportamiento en la clase estructurada.

Un conector es una relación contextual entre dos partes en un clasificador estructurado. Define una relación entre objetos que sirven de partes del mismo objeto estructurado. Un conector entre dos partes de una clase estructurada es diferente de una asociación entre dos clases que están asociadas por composición de la misma clase. En la asociación, no hay necesidad de que ésta conecte dos objetos que se encuentran contenidos en el mismo objeto compuesto. Cada asociación a una única clase es independiente, mientras que todos los conectores de una única clase estructurada comparten un único contexto durante la ejecución. Los conectores se pueden implementar mediante asociaciones normales o mediante relaciones transitorias, tales como parámetros, variables, valores globales y otros mecanismos.

La Figura 5.1 muestra un ejemplo de un pedido de entrada que contiene partes.

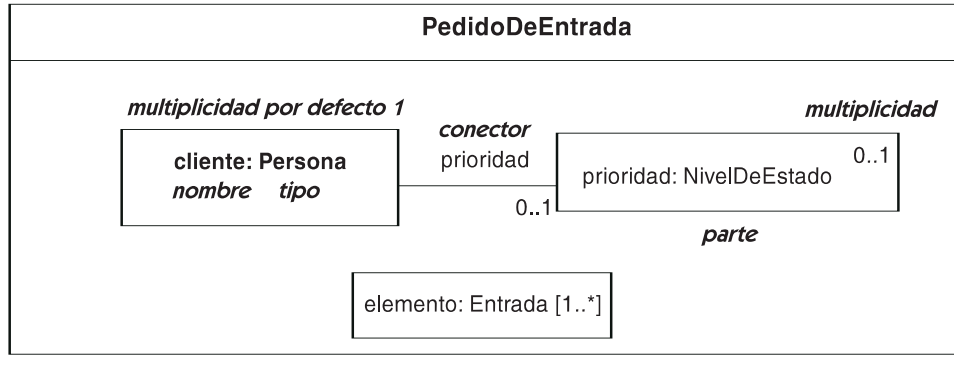


Figura 5.1 Clase estructurada

Los clasificadores estructurados pueden estar fuertemente encapsulados si se fuerza a que todas las interacciones entre el entorno externo y las partes internas pasen a través de puertos. Un puerto es un punto de interacción con una interfaz bien definida. El puerto se puede conectar con las partes internas o con los puertos de partes internas, o se puede conectar directamente con el comportamiento principal del clasificador estructurado. Los mensajes recibidos por un puerto se reenvían automáticamente a la parte o comportamiento (o viceversa en las salidas). Cada puerto tiene un conjunto de interfaces proporcionadas y de interfaces obligatorias que definen sus interacciones externas. Una interfaz proporcionada especifica los servicios que puede solicitar un mensaje al puerto. Una interfaz obligatoria especifica los servicios que un mensaje desde el puerto puede solicitar del entorno externo. Las conexiones externas a un clasificador encapsulado sólo pueden darse sobre los puertos. Una conexión externa es legal si las interfaces coinciden.

En el caso de una solicitud de entrada, la interfaz proporcionada debe soportar al menos el servicio solicitado por la conexión externa. En el caso de solicitudes salientes, la interfaz obligatoria no debe solicitar más servicios que los proporcionados por la conexión externa.

La Figura 5.2 muestra los puertos y las interfaces de una videocámara.

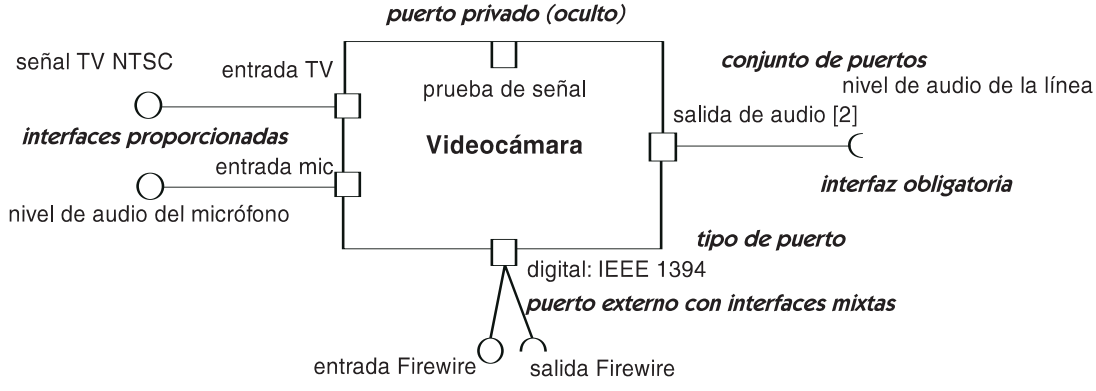


Figura 5.2 Clase estructurada con puertos

Colaboración

Una colaboración es una descripción de una colección de objetos que interactúan para implementar algún comportamiento dentro de un contexto. Describe una sociedad de objetos cooperativos reunidos para llevar a cabo algún propósito. Una colaboración contiene roles que son desempeñados por objetos durante la ejecución. Un rol representa una descripción de objetos que pueden participar en la ejecución de una colaboración. Un conector representa una descripción de asociaciones entre roles de la colaboración. Las relaciones entre roles y conectores dentro de una colaboración sólo tienen significado en ese contexto. Roles y conectores pueden tener tipos (clasificadores y asociaciones) que especifican qué objetos se pueden ligar a ellos. Los tipos de asociación son optativos, porque las relaciones en una colaboración pueden ser transitorias e implementadas mediante otros mecanismos, como los parámetros. Las relaciones descritas por las colaboraciones sólo aplican objetos ligados a roles dentro de una instancia específica de una colaboración; no se aplican a los clasificadores y asociaciones subyacentes aparte de la colaboración.

La vista estática describe las propiedades inherentes de una clase. Por ejemplo, una **Entrada** tiene una obra y un decorado asociado a ella. Estas relaciones se aplican a todas las instancias de la clase. Una colaboración describe las propiedades que tiene una instancia de una clase cuando desempeña un rol específico en una colaboración. Por ejemplo, una **entrada** en una colaboración **VentaDeEntradas** tiene un **vendedor**, algo que no es relevante para una **Entrada** en general, pero que es una parte esencial de la colaboración de venta de entradas.

Un objeto en un sistema puede participar en más de una colaboración. Las colaboraciones en las que aparezca no necesitan estar directamente relacionadas, aunque sus ejecuciones estén conectadas (quizás a propósito) a través del objeto compartido. Por ejemplo una persona puede

ser un **comprador** en una colaboración y un **vendedor** en otra. Algo menos frecuente es que un objeto pueda desempeñar más de un rol en la misma colaboración.

Una colaboración es un tipo de clasificador estructurado.

La Figura 5.3 muestra una colaboración para la venta de entradas.

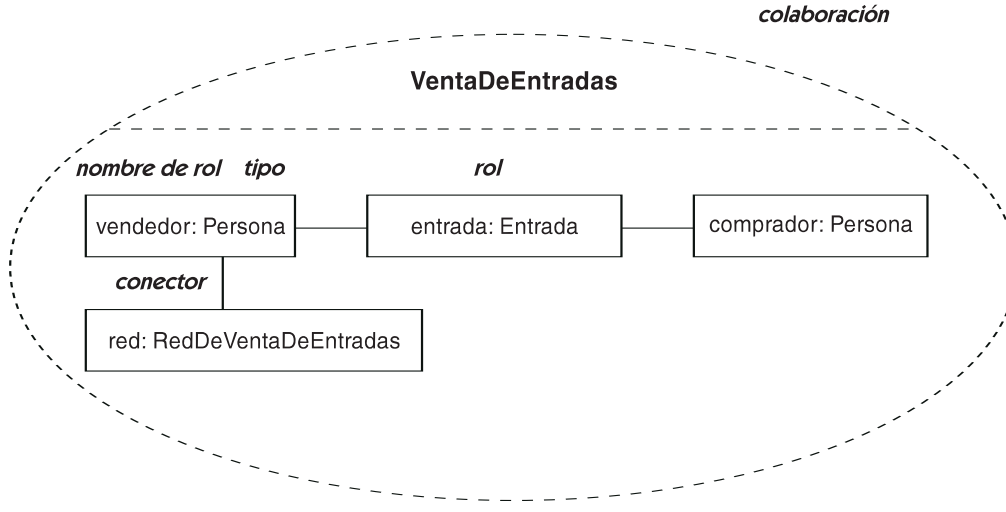


Figura 5.3 Definición de colaboración

Patrones

Un patrón es una colaboración parametrizada, junto con las pautas sobre cuándo utilizarlo. Un parámetro puede ser reemplazado por diferentes valores para producir diferentes colaboraciones. Los parámetros normalmente señalan huecos para las clases. Cuando se instancia un patrón, sus parámetros se ligan a las clases reales dentro de un diagrama de clases o a los roles dentro de una colaboración más grande.

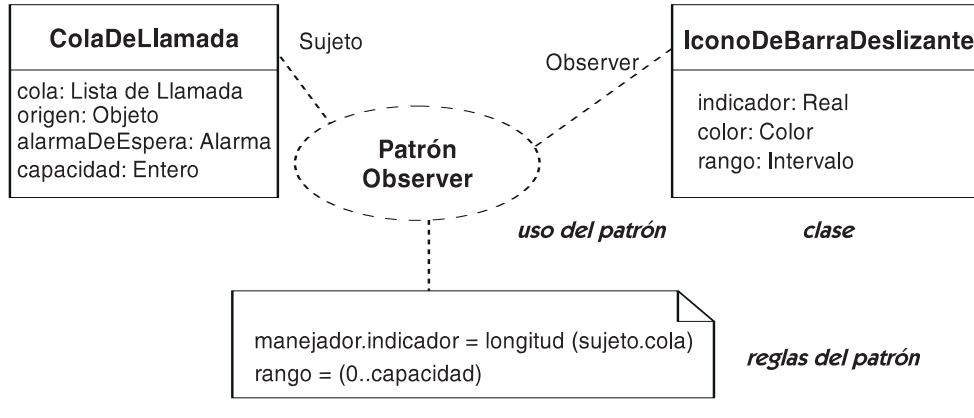


Figura 5.4 Uso de patrones

El uso de un patrón se muestra mediante una elipse discontinua conectada a cada una de sus clases mediante una línea discontinua que se etiqueta con el nombre del rol. Por ejemplo, la Figura 5.4 muestra el uso del patrón **Observer** de [Gamma-95]. En este uso del patrón, **ColaDeLlamada** reemplaza al rol **Sujeto** e **IconoDeBarraDeslizante** reemplaza al rol **Observer**.

Los patrones pueden aparecer a nivel de análisis, arquitectura, diseño detallado e implementación. Son una forma de capturar estructuras que se dan con frecuencia para su reutilización. La Figura 5.4 muestra el uso del patrón Observer.

Componente

Un componente representa una pieza modular de un sistema lógico o físico, cuyo comportamiento externo visible puede ser descrito mucho más concisamente que su implementación. Los componentes no dependen directamente de otros componentes, sino de las interfaces que los componentes soportan. Un componente en un modelo puede ser reemplazado por otro componente que soporta la interfaz adecuada, y la instancia de un componente dentro de una configuración de un sistema puede ser reemplazada por una instancia de cualquier componente que soporte la misma interfaz.

Los componentes (puesto que son clases) tienen interfaces que soportan (interfaces proporcionadas) e interfaces que necesitan de otros componentes (interfaces obligatorias). El uso de interfaces con nombre permite evitar dependencias directas entre componentes, facilitando la sustitución de un nuevo componente. Un diagrama de componentes muestra la red de dependencias entre componentes. La vista de componentes puede presentarse de dos formas. Puede mostrar un conjunto de componentes disponibles (biblioteca de componentes) con sus dependencias; este es el material con el que se ensamblan los sistemas. También puede mostrar un sistema configurado, con la selección de componentes (fuera de la biblioteca) utilizados para construirlo. En esta forma, cada componente se conecta a otros componentes cuyos servicios utiliza; estas conexiones deben ser consistentes con las interfaces de los componentes.

El icono de componente se dibuja mediante un rectángulo con dos pequeños rectángulos a un lado. Un componente se dibuja mediante un rectángulo con el icono de componente en la esquina superior derecha. El rectángulo contiene el nombre del componente. Las interfaces proporcionadas se muestran con un pequeño círculo conectado con el componente con una línea. Las interfaces obligatorias se muestran con un pequeño semicírculo conectado con el componente con una línea. La Figura 5.5 muestra un ejemplo.

Los componentes pueden tener puertos. El componente puede distinguir los mensajes recibidos por diferentes puertos y se pueden implementar de diferentes formas. Un puerto se muestra como un pequeño cuadrado en el borde del símbolo de un componente. El símbolo de una interfaz puede estar vinculado a un puerto.

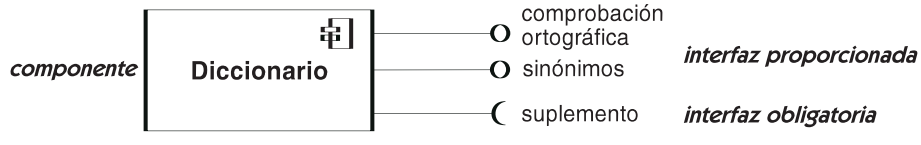


Figura 5.5 Componente con interfaces

Los componentes pueden contener otros componentes como su implementación. La "conexión" de dos componentes en una implementación se muestra mediante el anidamiento del círculo de una interfaz proporcionada en el semicírculo de una interfaz obligatoria en una notación de "articulación de rótula" (Figura 5.6).

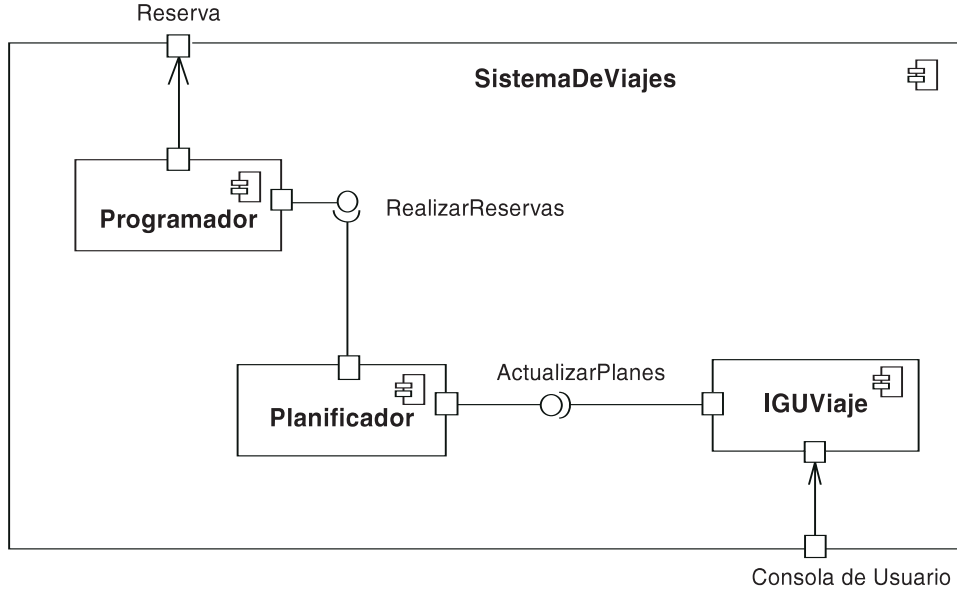


Figura 5.6 Estructura interna de un componente

Un componente es un clasificador estructurado. La distinción semántica entre un clasificador estructurado y un componente no es muy grande. Un componente es más una declaración de una intención de diseño que una diferencia semántica.

6

La vista de casos de uso

Descripción

La vista de casos de uso captura el comportamiento de un sistema, subsistema, clase o componente tal y como se muestra a un usuario externo. Divide la funcionalidad del sistema en transacciones que tienen significado para los actores —usuarios ideales de un sistema. Las piezas de funcionalidad interactiva se denominan casos de uso. Un caso de uso describe una interacción con actores como una secuencia de mensajes entre el sistema y uno o más actores. El término *actor* incluye tanto a personas, como a otros sistemas informáticos y procesos. La Figura 6.1 muestra un diagrama de casos de uso para una aplicación de telefónica de venta por catálogo. El modelo se ha simplificado como ejemplo.

Actor

Un actor es una idealización de un rol desempeñado por una persona externa, un proceso o cosa que interactúe con el sistema, subsistema o clase. Un actor caracteriza la interacción que una clase de usuarios externos puede tener con el sistema. Durante la ejecución, un usuario físico puede estar ligado con múltiples actores dentro del sistema. Diferentes usuarios pueden estar ligados con el mismo actor y, por lo tanto, representan múltiples instancias de la misma definición de actor. Por ejemplo, una persona puede ser un cliente y un cajero de una tienda en diferentes momentos.

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

Los actores pueden ser definidos en jerarquías de generalización, en las cuales la descripción de un actor abstracto es compartida y aumentada por la descripción de uno o más actores concretos.

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

Un actor se dibuja como una persona pequeña con trazos lineales y el nombre situado debajo de él.

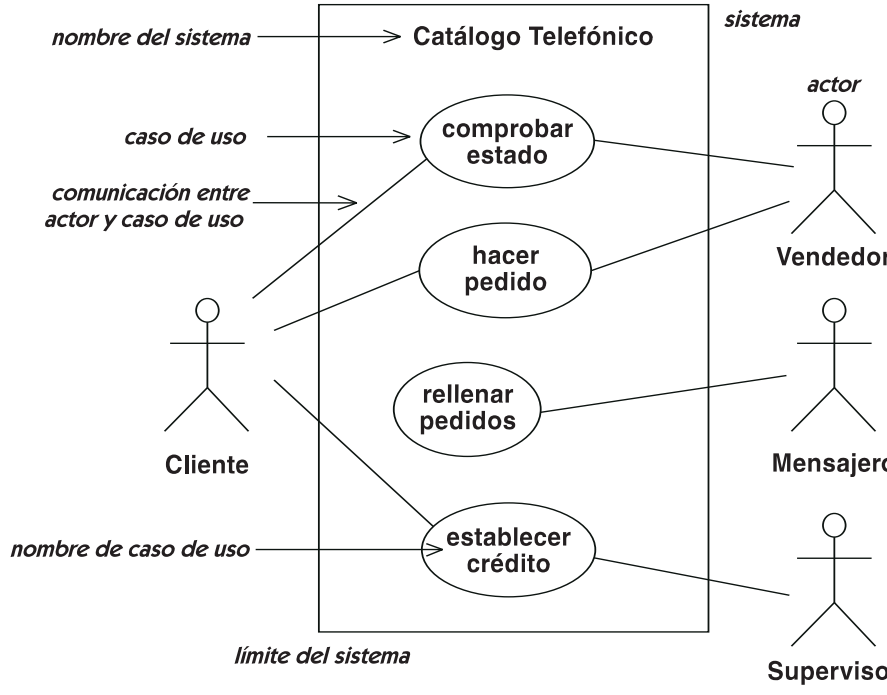


Figura 6.1 Diagrama de casos de uso

Caso de uso

Un caso de uso es una unidad coherente de funcionalidad externamente visible proporcionada por un clasificador (denominado sistema) y expresada mediante secuencias de mensajes intercambiados por el sistema y uno o más actores de la unidad del sistema. 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 se le supone —las secuencias principales, distintas variaciones del comportamiento normal y todas las condiciones de excepción que pueden darse con dicho comportamiento, junto con la respuesta deseada. Desde el punto de vista del usuario, éstas pueden ser situaciones anormales. Desde el punto de vista del sistema, son variaciones adicionales que deben ser descritas y manejadas.

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

La dinámica de un caso de uso se puede especificar mediante interacciones de UML, materializadas en diagramas de estados, diagramas de secuencia, diagramas de comunicación o descripciones informales de texto. Cuando se implementan los casos de uso, éstos 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 del sistema, los casos de uso representan el comportamiento externo del sistema tal y como lo ven los usuarios externos. Sin embargo, a diferencia de una operación, un caso de uso puede continuar recibiendo entradas de sus actores durante su ejecución. Los casos de uso se pueden aplicar al sistema completo, pudiéndose aplicar también internamente a unidades más pequeñas del sistema, como subsistemas y clases individuales. Un caso de uso interno representa el comportamiento que presenta un subsistema al resto del sistema. Por ejemplo, un caso de uso de una clase representa un trozo coherente de funcionalidad que la clase proporciona a otras clases que juegan 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. No es una construcción manifiesta de 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. En la medida en la que una clase puede desempeñar múltiples roles en la implementación de un sistema, puede por tanto realizar partes de varios casos de uso. Parte de la tarea de diseño es encontrar clases de diseño que combinen limpiamente los roles adecuados 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 la realización de un caso de uso.

Un caso de uso puede participar en varias relaciones, además de en la asociación con los actores (Tabla 6.1).

Tabla 6.1 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	
generalización de casos de uso	Una relación entre un caso de uso general y un caso de uso más específico que hereda le añade propiedades	
inclusión	La inserción de comportamiento adicional en un caso de uso base que describe explícitamente la inserción	

Un caso de uso se dibuja como una elipse con su nombre en el interior o debajo de ella. Se conecta mediante líneas de trazo continuo con los actores que se comunican con él.

La descripción de un gran caso de uso se puede descomponer en otros casos de uso más sencillos. Esto es similar a la manera en la que las clases se pueden definir incrementalmente a partir de la descripción de una superclase. Un caso de uso puede incorporar el comportamiento de otros casos de uso como fragmentos de su propio comportamiento. Esto se denomina relación de inclusión. El caso de uso incluido no es una especialización de caso de uso original y no puede ser sustituido por él.

Un caso de uso también puede ser definido como una extensión incremental de un caso de uso base. Esto se denomina relación de extensión. Puede haber varias extensiones del mismo caso de uso base, pudiéndose aplicar todas juntas. Las extensiones al caso de uso base se agregan a su semántica; se instancia el caso de uso base, no los casos de uso de la extensión.

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

Un caso de uso también puede ser especializado en uno más casos de uso hijos. Esta es la generalización de casos de uso. Cualquier caso de uso hijo puede ser utilizado en una situación en la cual se espera al caso de uso padre.

La generalización de casos de uso se muestra de la misma forma que cualquier generalización, mediante una línea desde el caso de uso hijo a caso de uso padre, con una punta de flecha triangular grande en el extremo del padre. La Figura 6.2 muestra las relaciones de casos de uso en la aplicación de venta por catálogo.

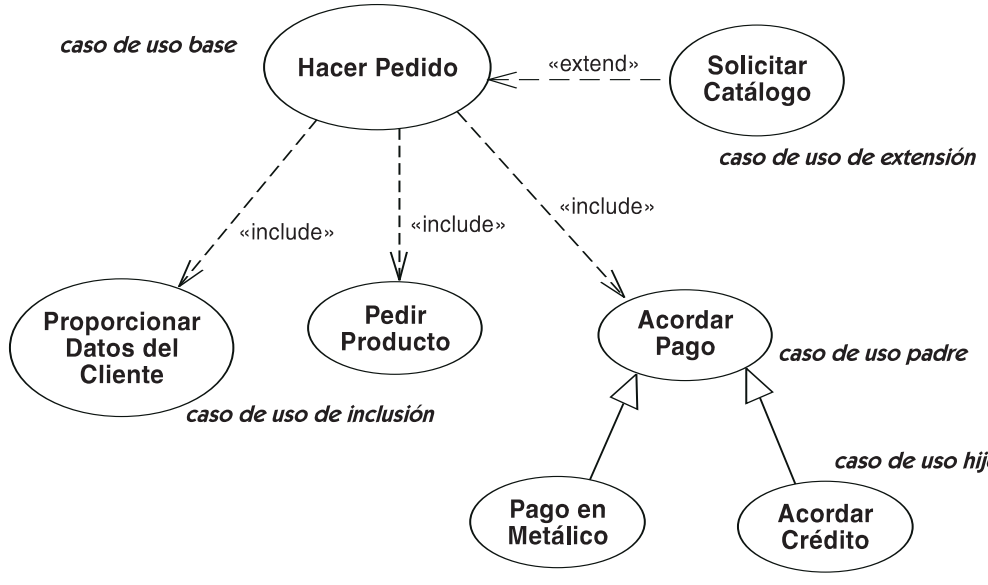


Figura 6.2 Relaciones de casos de uso

7

La vista de la máquina de estados

Descripción

La vista de la máquina de estados describe el comportamiento dinámico de los objetos, durante un periodo de tiempo, mediante el modelado de los ciclos de vida 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 el tipo de cambios que un objeto puede detectar —la recepción de llamadas o señales explícitas de un objeto a otro, un cambio en determinados valores o el paso del tiempo. Cualquier cosa que pueda influir en un objeto se puede caracterizar como un evento. Los sucesos del mundo real se modelan como señales del mundo exterior al sistema.

Un estado es un conjunto de valores de un objeto para una clase dada que tiene la misma respuesta cualitativa a los eventos que ocurren. En otras palabras, todos los objetos con el mismo estado reaccionan de la misma forma general ante un evento, dado que todos los objetos en un estado dado ejecutan el mismo efecto —una acción o una actividad— cuando reciben el mismo evento. Sin embargo, los objetos en estados diferentes pueden reaccionar de diferente forma al mismo evento, realizando distintos efectos. Una máquina de estados describe un número pequeño de estados que pueden ser albergados por un objeto. Para cada estado, la máquina de estados especifica las consecuencias de la recepción de cada tipo de evento como un efecto y un cambio a un estado nuevo. Por ejemplo, un cajero automático reacciona de una forma al botón de cancelación cuando está procesando una transacción y de otra forma cuando está inactivo.

Las máquinas de estados describen el comportamiento de las clases, así como el comportamiento dinámico de los casos de uso, colaboraciones y métodos. Para uno de estos objetos, un estado representa un paso en su ejecución. Nosotros normalmente hablamos en términos de clases y objetos para la descripción de las máquinas de estados, pero éstas se pueden aplicar a otros elementos de manera directa.

Máquina de estados

Una máquina de estados es un grafo de estados y transiciones. Normalmente, una máquina de estados está vinculada a una clase, y describe la respuesta de una instancia de la clase a los eventos que recibe. Las máquinas de estados también se pueden vincular a comportamientos, casos de uso y 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 del resto del mundo se resu-

me en un evento. Cuando el objeto detecta el evento, responde de una manera que depende de su estado actual. La respuesta puede incluir la ejecución de un efecto y el cambio a un nuevo estado. Las máquinas de estados se pueden estructurar para que compartan transiciones y puedan modelar la concurrencia.

Una máquina de estados es una vista localizada de un objeto, una vista que lo separa del resto del mundo y examina su comportamiento aislado. Es una vista reduccionista de un sistema. Esta es una buena forma de especificar el comportamiento con precisión, pero, a menudo, no es una buena forma de comprender el funcionamiento de un sistema en conjunto. Para hacerse una idea mejor de los efectos completos del comportamiento de un sistema, las vistas de interacción son mucho más útiles. Sin embargo, las máquinas de estados son útiles para comprender mecanismos de control, como las interfaces de usuario y los controladores de dispositivos.

Evento

Un evento es un tipo de ocurrencia significativa que tiene una localización en tiempo y espacio. Ocurre en un punto en el tiempo y no tiene duración. Se modela algo como evento si su ocurrencia tiene consecuencias. Cuando usamos la palabra *evento* por sí misma, normalmente queremos decir que se trata de un descriptor de evento —es decir, una descripción de todas las ocurrencias de los eventos individuales que tienen la misma forma general, de la misma forma que la palabra *clase* hace referencia a todos los objetos individuales que tienen la misma estructura. Una ocurrencia específica de un evento se denomina ocurrencia. Los eventos pueden tener parámetros que caractericen cada ocurrencia de un evento individual, de la misma forma que las clases tiene atributos que caracterizan cada objeto. 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 7.1 es una lista de los tipos de evento y sus descripciones.

Tabla 7.1 Tipos de eventos

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

Señal. Una señal es un tipo de clasificador que está pensado explícitamente como un vehículo de comunicación entre dos objetos; la recepción de una señal es un evento para el objeto receptor. El objeto que lo envía, crea e inicializa explícitamente una instancia de la señal y se la

envía a uno o a un conjunto de objetos explícitos. Las señales incorporan comunicaciones asíncronas unidireccionales, que son el tipo más importante. El emisor no espera a que el receptor se ocupe de la señal, sino que continúa con su trabajo de forma independiente. Para modelar comunicaciones bidireccionales se pueden utilizar múltiples señales, al menos una en cada dirección. El emisor y el receptor pueden ser el mismo objeto.

Las señales se pueden declarar en los diagramas de clases como clasificadores, utilizando la palabra clave **«signal»**; los parámetros de la señal se declaran como atributos. Como clasificadores, las señales pueden tener relaciones de generalización. Las señales pueden ser hijas de otras señales; heredan los atributos de sus padres y disparan las transiciones que contienen la señal padre (Figura 7.1).

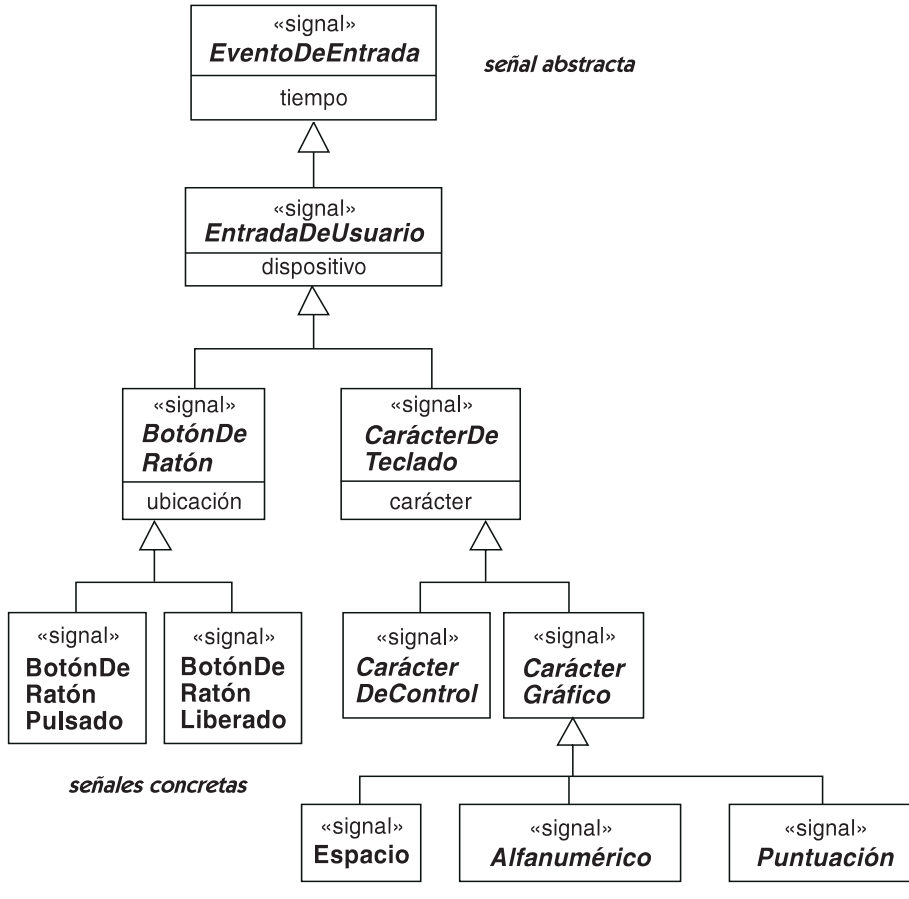


Figura 7.1 Jerarquía de señales

Evento de llamada. Un evento de llamada es la recepción de una llamada de una operación por un objeto. La clase receptora elije si una operación se implementará como método o como disparador de un evento de llamada en una máquina de estados (o posiblemente ambos). Los parámetros de la operación son parámetros del evento de llamada. Una vez que el objeto

receptor procesa el evento de llamada tomando una transición disparada por el evento o no pudiendo tomar ninguna transición, devuelve el control al objeto que realizó la llamada. Sin embargo, a diferencia de un método, una transición de una máquina de estados disparada por un evento de llamada puede continuar su propia ejecución en paralelo con el que realizó la llamada.

Evento de cambio. Un evento de cambio es la satisfacción de una expresión Booleana que depende ciertos valores señalados de los atributos. Es una manera declarativa de esperar hasta que se satisfaga una condición, pero se debe utilizar con cuidado, porque representa un cómputo continuo y potencialmente no local (acción a distancia, porque el valor o valores verificados pueden ser distantes). Esto es, al mismo tiempo, bueno y malo. Es bueno porque centra el modelo en la verdadera dependencia —un efecto que ocurre cuando se satisface una condición dada— en lugar de en el mecanismo de comprobación de la condición. Es malo porque oscurece la relación causa-efecto entre la acción que cambia el valor subyacente y el efecto consiguiente. El coste de comprobar un evento de cambio es potencialmente grande, ya que, en principio, es continuo. Sin embargo, en la práctica, hay optimizaciones que evitan la computación innecesaria. Los eventos de cambio se deben utilizar sólo cuando no es natural una forma más explícita de comunicación.

Observe las diferencias entre una condición de guarda y un evento de cambio. Una condición de guarda se evalúa una vez cuando ocurre el evento disparador de la transición y el receptor maneja el evento. Si es falsa, la transición no se dispara y la condición no se vuelve a evaluar. Un evento de cambio se evalúa continuamente hasta que se convierte en verdadero, momento en el cual se dispara la transición.

Evento de tiempo. Los eventos de tiempo representan el paso del tiempo. Un evento de tiempo puede ser especificado de un modo absoluto (hora) o de un modo relativo (tiempo transcurrido desde un evento dado). En un modelo de alto nivel, los eventos de tiempo pueden ser tomados como eventos provenientes del universo; en un modelo de implementación, son causados por señales provenientes de un objeto, tanto del sistema operativo, como de un objeto de la aplicación.

Estado

Una máquina de estados define un pequeño número de estados con nombre. Un estado puede estar caracterizado de tres formas complementarias: como un conjunto de valores que son cualitativamente similares en cierta forma; como un periodo de tiempo durante el cual un objeto espera que ocurra algún evento o eventos o como un periodo de tiempo durante el cual un objeto realiza alguna actividad hacer. Un estado puede tener un nombre, aunque a menudo son anónimos y se describen simplemente mediante sus efectos y relaciones. En realidad, los estados son las unidades de control a partir de las que se construyen las máquinas de estados.

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

Un estado se muestra como un rectángulo con las esquinas redondeadas (Figura 7.2).

Figura 7.2 Estado

Transición

Una transición que deja un estado define la respuesta de un objeto en ese estado a una ocurrencia de un evento. En general, una transición tiene un evento que la dispara, una condición de guarda, un efecto y un estado destino. La Tabla 7.2 muestra los tipos de transiciones y efectos implícitos invocados por las transiciones.

Tabla 7.2 Tipos de transiciones

Tipo de transición	*Descripción*	*Sintaxis*
transición de entrada	La especificación de una actividad de entrada que se ejecuta cuando se entra en el estado	**entry/`actividad`** o **entrada/`actividad`**
transición de salida	La especificación de una actividad de salida que se ejecuta cuando se sale del estado	**exit/`actividad`** o **salida/`actividad`**
transición externa	Una respuesta a un evento que causa un cambio de estado o una transición a sí mismo, junto con el efecto especificado. También puede causar la ejecución de las actividades de entrada y/o salida de los estados de los que se sale o entra	**`e`**(**`a:T`**)[**`exp`**]/**`actividad`**
transición interna	Una respuesta a un evento que causa la ejecución de un efecto pero no causa el cambio de estado o la ejecución de las actividades de entrada o salida	**`e`**(**`a:T`**)[**`exp`**]/**`actividad`**

Transición externa. Una *transición externa* es una transición que cambia el estado activo. Este es el tipo de transición más común. Se dibuja como una flecha desde el estado origen al destino, mostrando el resto de las propiedades como una cadena de texto vinculada a la flecha (Figura 7.3).

Evento disparador. El disparador especifica el evento cuya ocurrencia permite la transición. El evento puede tener parámetros, los cuales se encuentran disponibles para el efecto especificado como parte de la transición. Si la señal tiene descendientes, la recepción por parte de cualquier descendiente de la señal permite la transición. Por ejemplo, si una transición tiene como disparador la señal **BotónDeRatón** (*véase* Figura 7.1), la recepción de la señal **BotónDeRatónPulsado** disparará la transición.

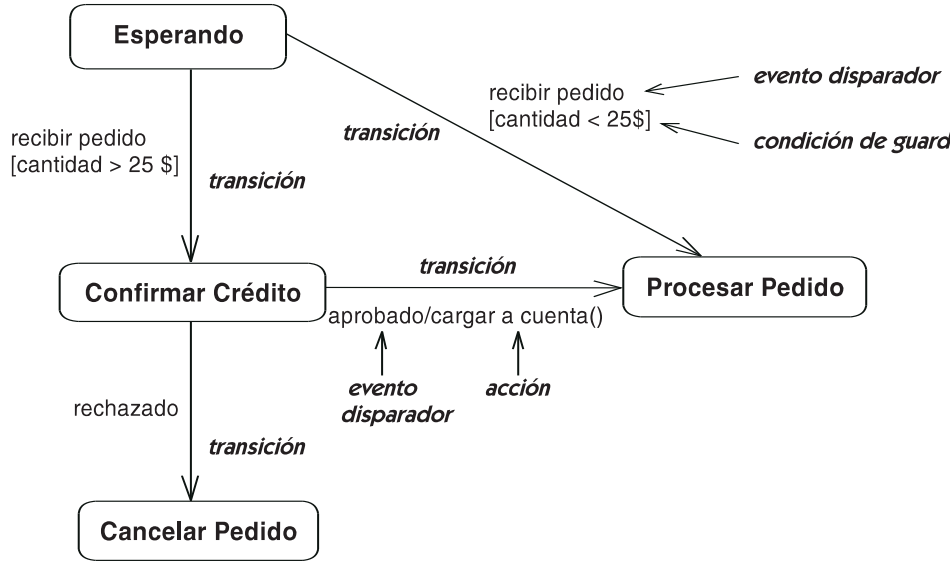


Figura 7.3 Transición externa

Un evento no es algo continuo, ocurre en un punto en el tiempo. Cuando un objeto recibe la ocurrencia de un evento, dicha ocurrencia se coloca en la lista de eventos del objeto. Un objeto maneja una ocurrencia de un evento cada vez. Cuando el objeto está libre, se saca una ocurrencia de la lista de eventos. Se debe disparar una transición en el momento en el que el objeto maneja el evento; la ocurrencia del evento no es "recordada" hasta después (excepto en el caso especial de los eventos diferidos, que permanecen en la lista de eventos hasta que disparan una transición o hasta que el objeto entra un estado en el que no se difieren). Si dos eventos suceden simultáneamente, se manejan de uno en uno. Una ocurrencia de un evento que no dispara ninguna transición simplemente se ignora y se pierde. Esto no es un error. Es mucho más sencillo ignorar eventos no deseados que intentar especificarlos todos.

Condición de guarda. Una transición puede tener una condición de guarda, que es una expresión Booleana. Puede referenciar atributos del objeto al que pertenece la máquina de estados, así como parámetros del evento disparador. La condición de guarda se evalúa cuando ocurre un evento disparador. Si la expresión se evalúa como verdadera, se dispara la transición —es decir, ocurren sus efectos. Si la expresión se evalúa como falsa, no se dispara la transición. La condición de guarda sólo se evalúa una vez, en el momento en el que ocurre un evento disparador. Si la condición es falsa y después se convierte en verdadera, es demasiado tarde para disparar la transición.

El mismo evento puede ser el disparador de más de una transición que parte de un único estado. Cada transición con el mismo evento debe tener distinta condición de guarda. Si ocurre el evento, una transición disparable por el evento puede dispararse si la condición es verdadera. A menudo, el conjunto de condiciones de guarda cubre todas las posibilidades, de forma que se garantice que la ocurrencia del evento dispare alguna transición. Si no están cubiertas todas las posibilidades y no se habilita ninguna transición, entonces el evento simplemente se ignora. Sólo se puede disparar una transición (dentro de un hilo de control) en respuesta a una ocurrencia de un evento. Si un evento permite más de una transición, sólo se dispara una de ellas. Una transición en un estado anidado tiene prioridad sobre una transición sobre uno de los estados que incluye. Si dos transiciones que están

en conflicto se activan al mismo tiempo, una de ellas se dispara, no quedando determinado cuál de ellas lo hace. La elección puede ser aleatoria o puede depender de los detalles de implementación, pero el modelador no podría contar con un resultado predecible.

Si está activo un estado ortogonal, cada región en él está activa, lo que significa que múltiples estados (al menos uno en cada región) pueden estar activos concurrentemente. Si están activos múltiples estados, una única ocurrencia de un evento puede disparar una transición en cada región ortogonal. Las transiciones concurrentes se ejecutan concurrentemente y no interactúan, salvo de forma indirecta debido a los efectos de los objetos compartidos.

Transición de finalización. Una transición que carece de un evento disparador específico se dispara cuando finaliza la actividad en el estado que deja (esto es una transición de finalización). Una transición de finalización puede tener una condición de guarda, la cual se evalúa en el momento en el que la actividad del estado finaliza (y no después). Las transiciones de finalización tienen prioridad sobre los eventos normales y no esperan el paso normal de "ejecución hasta la finalización".

Efecto. Cuando una transición se dispara, su efecto (si lo hay) se ejecuta. Un efecto puede ser una acción o una actividad. Una acción es una computación atómica, como una sentencia de asignación o un cálculo aritmético sencillo. También son acciones el envío de una señal a otro objeto, la llamada a una operación, la creación o destrucción de un objeto y la obtención y la modificación de los valores de los atributos. Un efecto también puede ser una actividad —es decir, una lista de acciones o actividades más sencillas. Una acción o actividad no puede ser finalizada o influida por efectos simultáneos. (Un perfil avanzado podría añadir una acción que finalizara o interrumpiera otras actividades.) Conceptualmente, una actividad se procesa en un tiempo; sin embargo, no se puede manejar un segundo evento durante la ejecución de un efecto.

El sistema en conjunto puede realizar múltiples actividades simultáneamente. Un sistema implementado puede procesar interrupciones hardware y compartir el tiempo entre varias acciones. Una actividad no se puede interrumpir dentro de su propio hilo de control. Una vez que ha comenzado debe finalizar, y no puede interactuar con otro efecto activo simultáneamente. Esto se denomina semántica de "ejecución hasta la finalización". Los efectos no se deberían utilizar como un mecanismo de transacciones largas. Su duración debería ser breve en comparación con el tiempo de respuesta que necesitan los eventos externos. De otra forma, el sistema podría ser incapaz de responder de forma oportuna.

Un efecto puede utilizar parámetros del evento disparador y atributos del objeto al que pertenece como parte de sus expresiones.

Cambio de estado. Cuando se completa la ejecución de un efecto, el estado destino de la transición pasa a ser el estado activo. Esto puede disparar una actividad de salida o una actividad de entrada (o varias si la máquina de estados atraviesa varios estados anidados desde el estado origen al estado destino).

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

Actividades de entrada y salida. Una transición a través de uno o más niveles de anidamiento puede salir y entrar en estados. Un estado puede especificar actividades para que se realicen cuando se entra o sale del estado. Si la transición deja el estado original, entonces se ejecuta la actividad de salida, antes del efecto de la transición, y la actividad de entrada del nuevo estado.

Las actividades de entrada se utilizan a menudo para realizar la configuración necesaria dentro del estado. Igualmente, una actividad de salida es una actividad que ocurre siempre que se sale de un estado, una oportunidad de realizar una limpieza. Esto es especialmente útil cuando hay transiciones de alto nivel que representan condiciones de error que abortan estados anidados. La actividad de salida puede limpiar tales casos, de modo que el estado del objeto permanezca consistente. Las actividades de entrada y salida se podrían vincular a transiciones entrantes y salientes, pero declararlas como efectos especiales del estado permite que el estado sea definido independientemente de sus transiciones y, por tanto, esté encapsulado.

Transición interna. Una transición interna tiene estado origen, pero no estado destino. Las reglas de disparo para una transición interna son las mismas que para una transición que cambia de estado. Una transición interna no tiene estado destino, por lo que el estado activo no cambia como resultado de su disparo. Si una transición interna tiene un efecto, el efecto se ejecuta, pero no ocurre ningún cambio de estado, y, por tanto, no se ejecutan ni actividades de entrada, ni actividades de salida. Las transiciones internas son útiles para modelar situaciones de interrupción que no cambian el estado (como el recuento de ocurrencias de un evento o el despliegue de una pantalla de ayuda).

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

Una transición a sí mismo invoca actividades de entrada y salida en el estado (conceptualmente, sale y vuelve a entrar en el mismo estado); por tanto, no es equivalente a una transición interna. La Figura 7.4 muestra actividades de entrada y salida, así como transiciones internas.

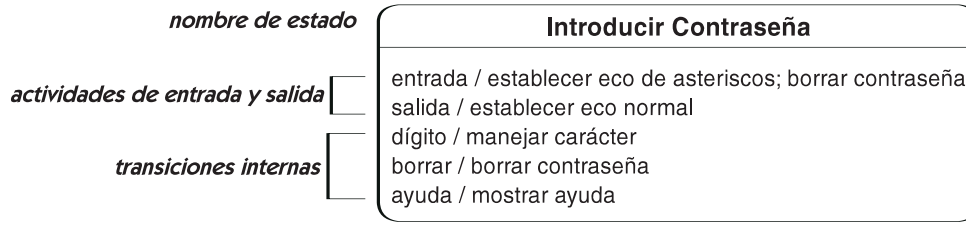
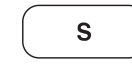
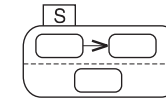
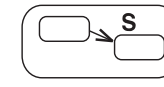
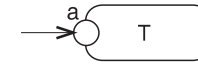


Figura 7.4 Transiciones internas y acciones de entrada y salida

Estado compuesto

Un estado simple no tiene estructura interna, sólo un conjunto de transiciones y posibles actividades de entrada y salida. Un estado compuesto es aquél que se ha descompuesto en regiones, cada una de las cuales contiene uno o más subestados directos. La Tabla 7.3 enumera los distintos tipos de estados.

Tabla 7.3 Tipos de estados

Tipo de estado	*Descripción*	*Notación*
estado simple	Un estado sin estructura	
estado ortogonal	Un estado que se divide en dos o más regiones. Un subestado directo de cada región se encuentra activo concurrentemente cuando el estado compuesto está activo	
estado no ortogonal	Un estado compuesto que contiene uno o más subestados directos. Exactamente uno de los subestados se encuentra 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 finalizado su actividad	
finalizador	Un estado especial cuya activación finaliza la ejecución del objeto al que pertenece la máquina de estados	
conjunción	Un pseudoestado que encadena segmentos de transición en una transición de ejecución hasta la finalización	
elección	Un pseudosestado que realiza una bifurcación dinámica dentro de una transición de ejecución hasta la finalización	
estado de historia	Un pseudoestado cuya activación restaura el estado que se encontraba previamente activo dentro de un estado compuesto	
estado de referencia a submáquina	Un estado que referencia a la definición de una máquina de estados, que conceptualmente reemplaza la submáquina de estados	
punto de entrada	Un pseudoestado externo y visible dentro de la máquina de estados que identifica un estado interno como destino	
punto de salida	Un pseudoestado externo y visible dentro de la máquina de estados que identifica un estado interno como origen	

Una descomposición de un estado no ortogonal en subestados directos es similar a la especialización de una clase. Un estado externo se descompone en varios estados internos, cada uno de los cuales hereda las transiciones del estado externo. Sólo un subestado directo por estado no ortogonal puede ser activado en un instante de tiempo. El estado externo representa la condición de estar en alguno de los estados internos.

Las transiciones hacia dentro o hacia fuera del estado compuesto invocan cualquier actividad de entrada o actividad de salida del estado. Si hay varios estados compuestos, una transición a través de varios niveles puede invocar múltiples actividades de entrada (primero las más externas) o varias actividades de salida (primero las más internas). Si hay un efecto en la propia transición, el efecto se ejecuta después de que se ejecuten las actividades de salida y antes de que lo hagan las de entrada.

Cada región de un estado compuesto puede tener un estado inicial. Una transición al límite del estado compuesto es implícitamente una transición al estado inicial. Un objeto nuevo comienza en el estado inicial del estado más exterior. De igual forma, un estado compuesto puede tener un estado final. Una transición al estado final dispara una transición de finalización en el estado compuesto. Si un objeto alcanza el estado final del estado más exterior se destruye. Los estados iniciales, los estados finales, las actividades de entrada y las actividades de salida per-

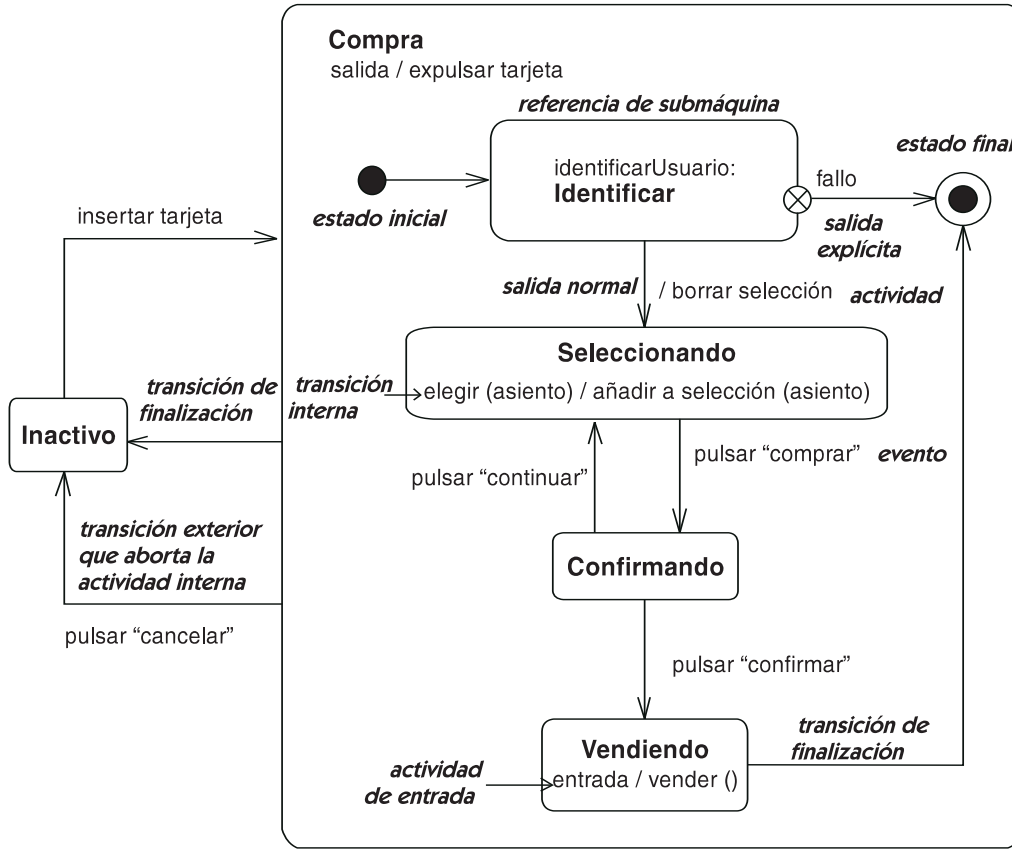


Figura 7.5 Máquina de estados

miten que la definición de un estado sea encapsulada independientemente de las transiciones a y desde él.

La Figura 7.5 muestra la descomposición secuencial de un estado, incluyendo un estado inicial. Este es el control para una máquina de venta de entradas.

Una descomposición de un estado ortogonal en regiones ortogonales representa cómputos independientes. Cuando se entra en un estado ortogonal, el número de hilos de control se incrementa a medida que un subestado directo de cada región ortogonal se vuelve activo. Cuando se sale del estado ortogonal, el número de hilos de control se decrementa. A menudo, la concurrencia se implementa mediante un objeto distinto para cada subestado, pero los estados ortogonales también pueden representar concurrencia lógica dentro de un único objeto. La Figura 7.6 muestra la descomposición concurrente de tomar una clase en la universidad.

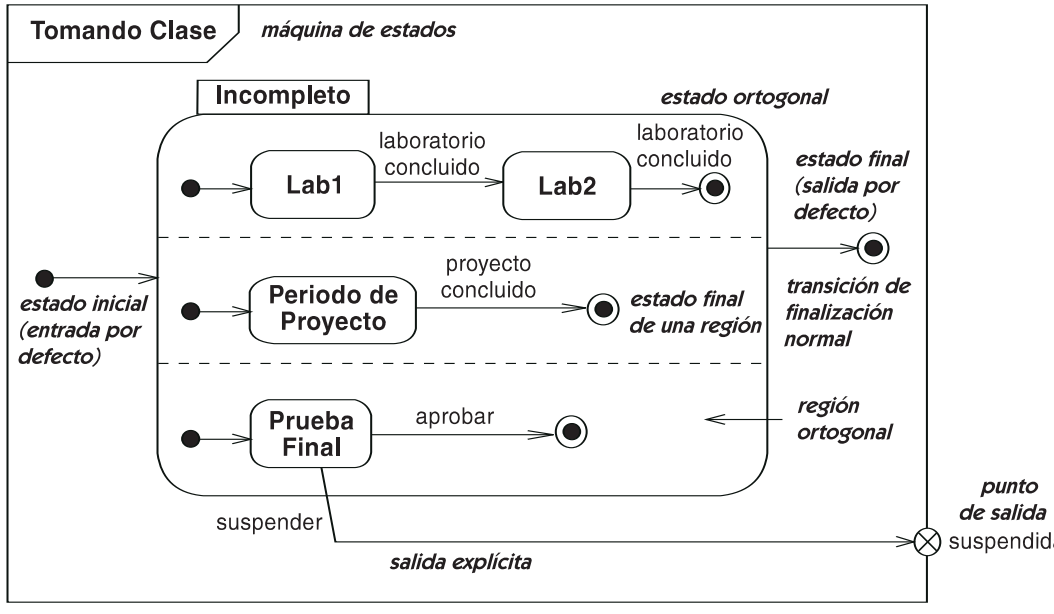


Figura 7.6 Máquina de estados con un estado compuesto ortogonal

A menudo es conveniente reutilizar un fragmento de una máquina de estados en otra máquina de estados. Se le puede dar un nombre a una máquina de estados y referenciarla desde un estado de una o más máquinas de estados. La máquina de estados destino es una submáquina, y el estado que la referencia se denomina estado de referencia a submáquina. Esto implica la sustitución (conceptual) de una copia de la máquina de estados referenciada en el lugar de la referencia. Una submáquina puede definir puntos de entrada y de salida con nombre que conecten los estados internos. Las transiciones a un estado de referencia a submáquina pueden utilizar puntos de conexión que hagan referencia a dichos puntos de entrada y salida, ocultando la estructura interna de la submáquina de los clientes externos. En lugar de una submáquina, un estado puede contener una actividad hacer —esto es, un cómputo o actividad continua que requiere un tiempo para su realización y que puede ser interrumpida por eventos. La Figura 7.7 muestra una referencia a submáquina.

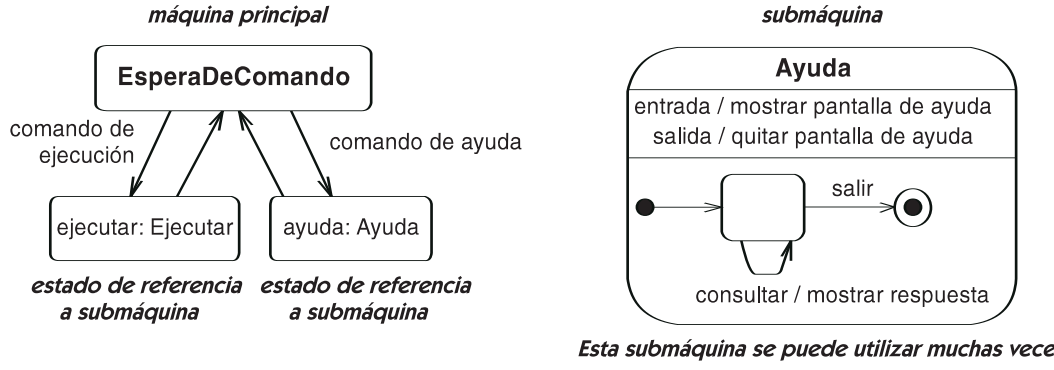


Figura 7.7 Submáquina de estados

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 puede referenciar un punto de entrada. Un punto de entrada identifica un estado en la submáquina sin exponer el contenido de la submáquina.

8

La vista de actividades

Descripción

Una actividad es un grafo de nodos y flujos que muestra el flujo de control (y opcionalmente datos) a través de los paso de la computación. Los pasos de ejecución pueden ser, tanto concurrentes, como secuenciales. Una actividad involucra constructores de sincronización y de bifurcación, de forma similar, pero más potente, a los tradicionales diagramas de flujo, que sólo soportaban constructores secuenciales y de bifurcación.

Una definición de una actividad contiene nodos de actividad. Un nodo de actividad representa la ejecución de una sentencia en un procedimiento o la realización de un paso en un flujo de trabajo. Los nodos se conectan con flujos de control y flujos de datos. Normalmente, un nodo de actividad comienza su ejecución cuando hay tokens (indicadores de control) en cada uno de sus flujos de entrada. Un nodo de actividad espera a que finalice su cómputo. Cuando se completa la ejecución de un nodo, la ejecución sigue a los nodos que se encuentran en sus flujos de salida. Los flujos de actividad son como transiciones de finalización —ocurren cuando finaliza la ejecución— pero se pueden incluir acciones que esperen eventos específicos.

Los nodos de actividad se pueden anidar. Un diagrama de actividad puede contener bifurcaciones, así como divisiones del control en hilos concurrentes, Los hilos concurrentes representan actividades que se pueden realizar concurrentemente por diferentes objetos o personas en una organización. Con frecuencia, la concurrencia surge de la agregación, en la que cada objeto tiene su propio hilo concurrente. Las actividades concurrentes se pueden realizar simultáneamente o en cualquier orden. Un grafo de actividad es como un diagrama de flujo tradicional excepto por el hecho de que permite el control concurrente además del control secuencial —lo que es una gran diferencia.

También hay nodos de control predefinidos que soportan varias formas de control, como decisiones (bifurcaciones) y fusiones. La ejecución concurrente se modela utilizando divisiones y uniones. También hay constructores de control que soportan el manejo de excepciones y la aplicación paralela de una actividad a los elementos de un conjunto.

En última instancia, las hojas de un grafo de actividad son las acciones. Una acción es una actividad básica predefinida, como acceder o modificar atributos o valores de enlace, crear o destruir objetos, invocar operaciones y enviar señales.

Actividad

La Figura 8.1 muestra una definición de una actividad.

Un nodo de actividad se muestra como una caja con las esquinas redondeadas que contiene una descripción de la actividad. Un flujo de control se muestra como una flecha. Las bifurcaciones se muestran como condiciones de guarda sobre los flujos de control o como rombos con múltiples flechas de salida etiquetadas. Una división o unión del control se muestra mediante múltiples flechas saliendo o entrando en una barra gruesa de sincronización. La Figura 8.1 muestra un diagrama de actividad para procesar un pedido por la taquilla.

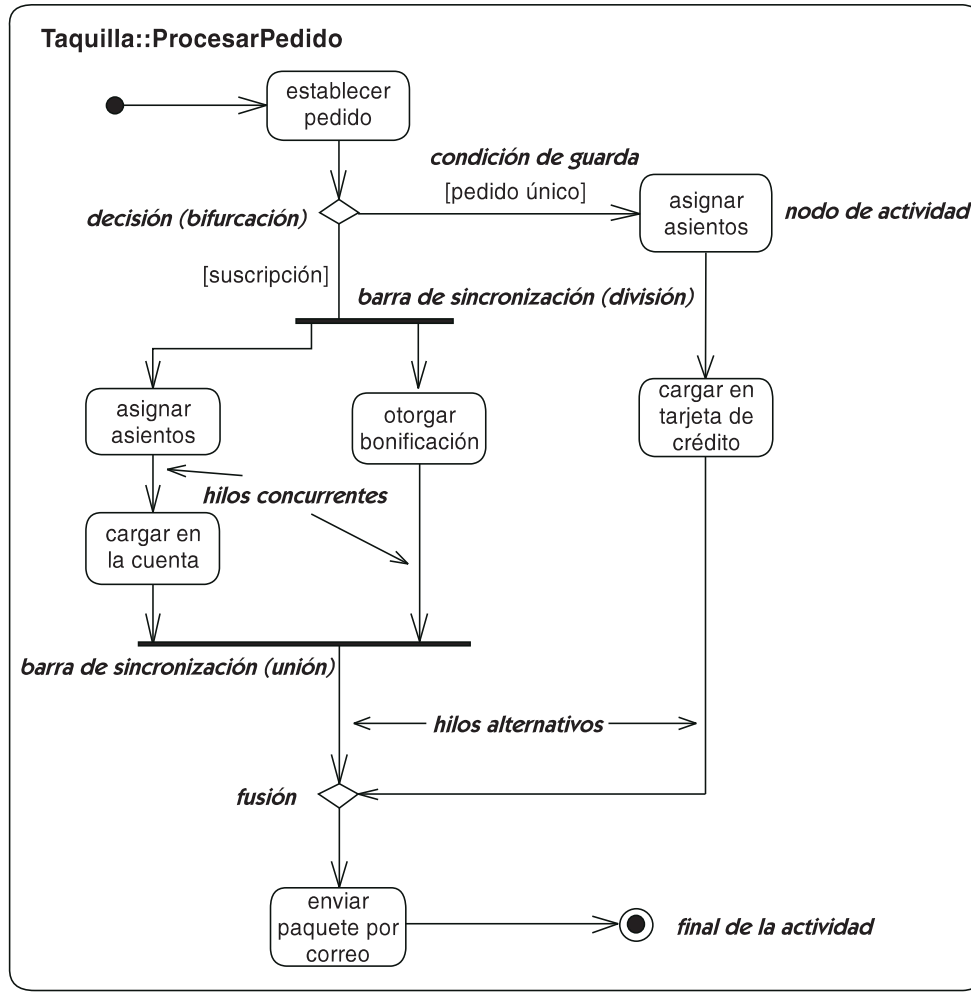


Figura 8.1 Diagrama de actividades

Para aquellas situaciones en las que se deben incluir eventos externos, la recepción de un evento se puede mostrar como una acción que indica la espera de una señal. Una notación similar indica el envío de una señal.

Particiones. A menudo es útil organizar las actividades de un modelo de acuerdo con su responsabilidad —por ejemplo, agrupando juntas todas las actividades manejadas por una organización de negocio. Este tipo de asignación se puede mostrar organizando las actividades en distintas regiones (denominadas particiones) separadas por líneas en el diagrama. Debido a su apariencia, una región a veces recibe el nombre de calle[1]. La Figura 8.2 muestra particiones.

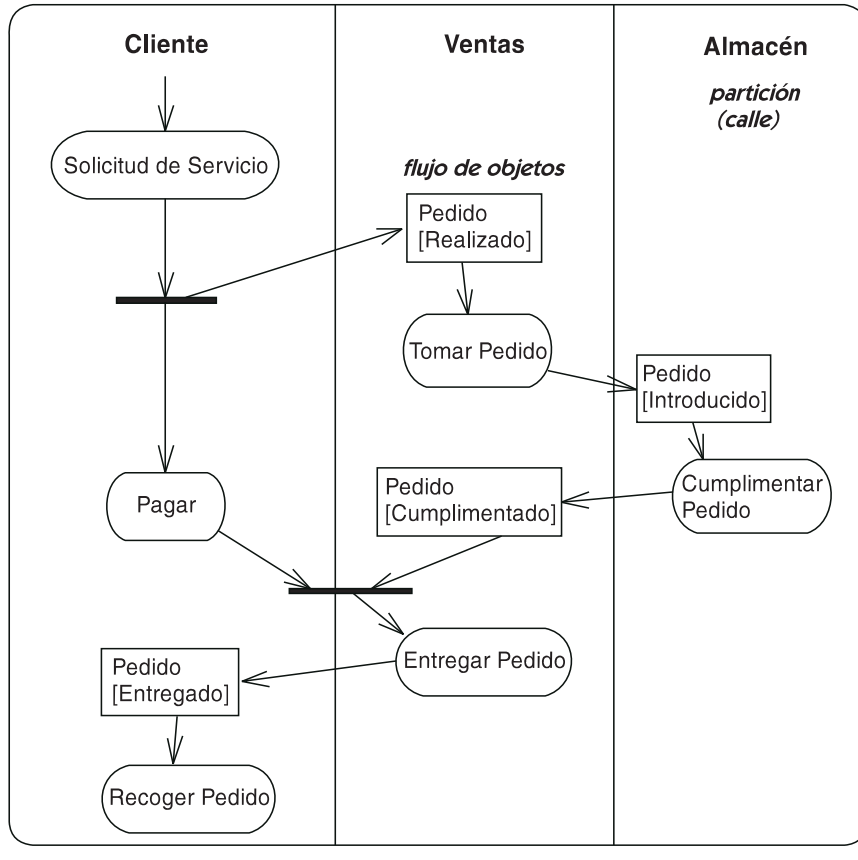


Figura 8.2 Particiones y flujos de objetos

Flujos de objetos. Un diagrama de actividad puede mostrar el flujo de valores de objetos, además del flujo de control. Un flujo de objeto representa un objeto que es la entrada o salida de una actividad. Para un valor de salida, se dibuja una flecha de línea continua desde la actividad al flujo de objeto. Para un valor de entrada se dibuja una flecha de línea continua del flujo de objeto a la actividad. Si una actividad tiene más de un valor de salida o más de un flujo de control sucesor, las flechas se dibujan desde un símbolo de división. De igual forma, las entradas múltiples se dibujan hacia un símbolo de fusión.

[1] Se alude al término ingles *swimlane*, que hace referencia a las calles de una piscina. (*N. del T.*)

La Figura 8.2 muestra un diagrama de actividad en el que se han asignado actividades y flujos de objetos a particiones.

Actividades y otras vistas

Los grafos de actividad no muestran todos los detalles de un cómputo. Muestran el flujo de actividades pero no los objetos que realizan esas actividades, Los grafos de actividad son un punto de comienzo para el diseño. Para completar un diseño, cada actividad se debe implementar como una o más operaciones, cada una de las cuales se asigna a una determinada clase que la implementa. Tal asignación da lugar al diseño de una colaboración que implementa el grafo de actividad.

Acción

UML tiene un conjunto de acciones primitivas que modelan la manipulación de objetos y enlaces, así como los cómputos y la comunicación entre objetos. UML no define una sintaxis para las acciones porque se espera que la mayoría de los modelos utilicen un lenguaje de acción o un lenguaje de programación existente. La Tabla 8.1 muestra los distintos tipos de acciones.

Tabla 8.1 Tipos de acciones

Categoría	*Acciones*	*Propósito*
clasificación	leerEsObjetoClasificado reclasificarObjeto probarIdentidad	probar clasificación cambiar clasificación probar identidad del objeto
comunicación	difundirSeñal llamarOperación responder retornar (implícito) enviarObjeto enviar señal	difundir llamada normal responder después de aceptación explícita acción implícita al final de actividad enviar señal como objeto enviar señal como lista de argumentos
cómputo	aceptación de llamada aceptación de evento añadirValorVariable aplicarFunción llamarComportamiento limpiarVariable leerMismo leerVariable eliminarValorVariable escribirVariable	espera en línea por una llamada espera en línea por un evento añadir un valor adicional al conjunto cálculo matemático comportamiento anidado reiniciar valor en un procedimiento obtener la identidad del propio objeto obtener valor en un procedimiento eliminar valor del conjunto establecer valor en un procedimiento
control	iniciarComportamientoPropio	control explícito
creación	CrearEnlaceObjeto crearObjeto	crear objeto desde una asociación crear objeto normal

Tabla 8.1 Tipos de acciones *(continuación)*

Categoría	*Acciones*	*Propósito*
destrucción	destruirObjeto	destruir objeto
excepción	ElevarExcepción	elevar una excepción en un procedimiento
lectura	leerExtensión leerEnlace leerExtremoEnlaceObjeto leerCalificadorExtremoEnlace leerCaracterísticaEstructural	obtener todos los objetos obtener el valor del enlace obtener el valor de la clase de asociación obtener el valor de calificador obtener el valor del atributo
tiempo	observarDuración observarTiempo	medición de intervalo de tiempo obtener tiempo actual
escritura	añadirValorCaracterísticaEstructural limpiarAsociación limpiarCaracterísticaEstructural crearEnlace destruirEnlace eliminarValorCaracterísticaEstructural	establecer valor de atributo limpiar enlaces limpiar valores de atributo añadir un enlace eliminar un enlace eliminar un valor de un conjunto

9
La vista de interacción

Descripción

Los objetos interactúan entre sí para implementar su comportamiento. Esta interacción se puede describir de dos formas complementarias, una de ellas centrada en objetos individuales y la otra centrada 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 una máquina de estados es precisa y lleva inmediatamente al código. Sin embargo, puede ser difícil comprender el funcionamiento completo de un sistema porque una máquina de estados se centra en un solo objeto en cada momento, y se deben combinar los efectos de muchas máquinas de estados para determinar el comportamiento del sistema completo. La vista de interacción proporciona una vista más integral del comportamiento de un conjunto de objetos. Esta vista se modela mediante interacciones que actúan sobre clasificadores estructurados y colaboraciones.

Interacción

Un clasificador estructurado define una relación contextual. Un clasificador estructurado (incluyendo una colaboración), puede tener un conjunto de comportamientos vinculados que se aplican a un conjunto de objetos ligados a una única instancia del contexto. Un tipo de descripción del comportamiento es una secuencia de mensajes intercambiada por los objetos ligados a los roles. Una descripción de las secuencias de mensajes de una clase estructurada o colaboración se denomina interacción. Una clase estructurada o colaboración puede tener cualquier número de interacciones, cada una de las cuales describe una serie de mensajes intercambiados entre los objetos en el contexto de la realización de un objetivo. Con mucha frecuencia, las interacciones describen la ejecución de operaciones. Los parámetros de la operación sirven de roles de una colaboración que representa la ejecución de la operación.

Una interacción es un conjunto de mensajes, dentro de un clasificador estructurado o colaboración, intercambiado por roles a través de conectores. Una instancia de una interacción se corresponde con una instancia en su contexto, con los objetos ligados a roles que intercambian instancias de mensajes a través de enlaces ligados a conectores. A menudo, una interacción modela la ejecución de una operación, de un caso de uso o de otra entidad de comportamiento.

Un mensaje es una comunicación unidireccional entre dos objetos, un flujo de control con información del emisor al receptor. Un mensaje puede tener argumentos que transportan valores

desde el emisor al receptor. Un mensaje puede ser una señal (una comunicación entre objetos explícita, con nombre y asíncrona) o una llamada (la invocación síncrona o asíncrona de una operación con un mecanismo para devolver posteriormente el control al emisor de una llamada síncrona).

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

Los eventos (incluyendo el envío y recepción de mensajes) de un único rol se ordenan temporalmente. Otros eventos son concurrentes a menos que se encuentren relacionados por una cadena de mensajes intermedios o se encuentren ordenados explícitamente. Dos mensajes están ordenados si el evento de recepción de uno precede al evento de envío del otro. De otra forma pueden ser concurrentes.

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 comunicación (que se centra en las relaciones entre los objetos que intercambian mensajes).

Diagrama de secuencia

Un diagrama de secuencia muestra una interacción como un gráfico de dos dimensiones. La dimensión vertical es el eje de tiempo, que avanza hacia el final de la página. La dimensión temporal muestra los roles que representan objetos individuales en la colaboración. Cada rol se representa mediante una columna vertical que contiene un símbolo de cabecera y una línea vertical —línea de vida. Durante el tiempo que existe un objeto, la línea de vida se representa con una línea discontinua. Durante el tiempo que la ejecución de la especificación de un procedimiento sobre el objeto está activo, la línea de vida se representa con una línea doble.

En general, un diagrama de secuencia sólo muestra secuencias de mensajes y no intervalos exactos de tiempo. Se puede utilizar un diagrama de tiempos cuando la métrica del tiempo es importante, pero para la comprensión de la semántica de la interacción, normalmente los diagramas de secuencia son suficientes.

Un mensaje se muestra como una flecha desde la línea de vida de un objeto a la línea de vida del otro. Las flechas se organizan cronológicamente hacia abajo en el diagrama. Los mensajes asíncronos se muestran mediante flechas de punta abierta.

La Figura 9.1 muestra un diagrama de secuencia típico con mensajes asíncronos.

Especificación de ejecución

Una especificación de ejecución (activación) es la ejecución de un procedimiento, incluyendo el tiempo que necesite para ejecutar los procedimientos anidados. Se muestra mediante una línea doble reemplazando parte de la línea de vida en un diagrama de secuencia.

Una llamada se muestra con una flecha que apunta al comienzo de la ejecución de la especificación que la llamada inicia. Una llamada síncrona se muestra con una punta de flecha triangular rellena.

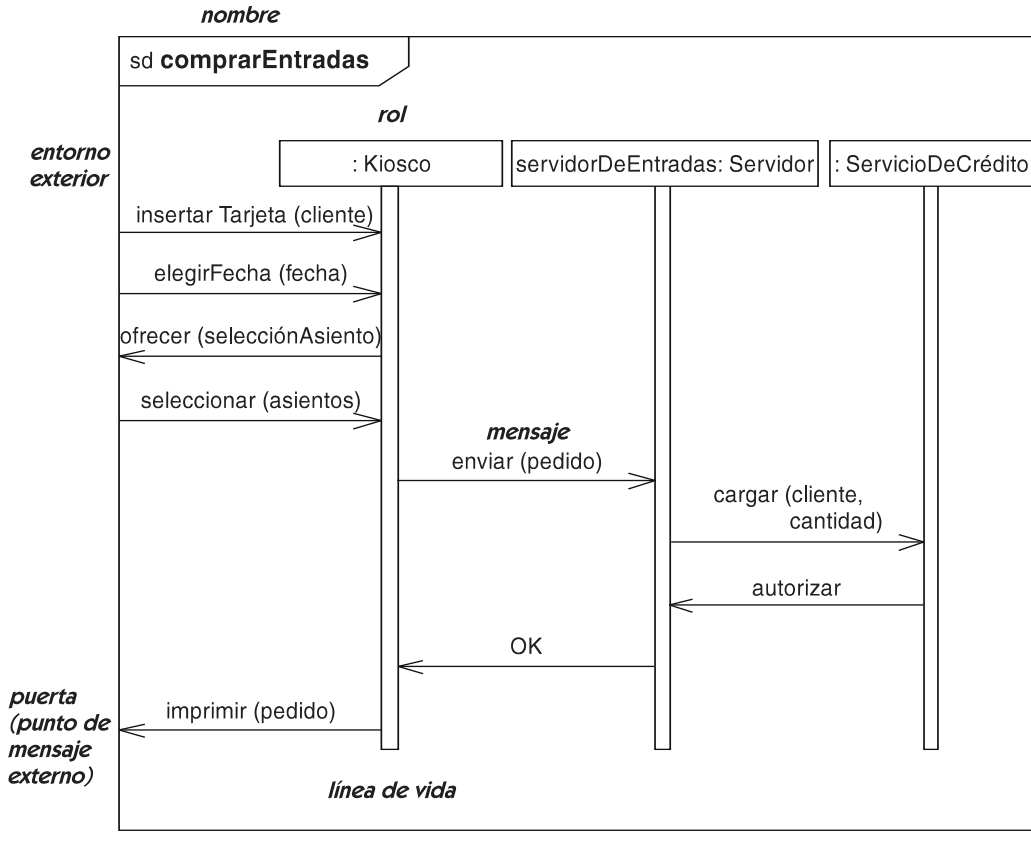


Figura 9.1 Diagrama de secuencia

Una llamada recursiva se da cuando el control vuelve a iniciar una operación sobre el mismo objeto, pero la segunda llamada es una especificación de ejecución distinta de la primera. La recursividad o una llamada anidada a otra operación sobre el mismo objeto se muestran en el diagrama de secuencia apilando las líneas de activación.

El retorno del control desde una llamada se muestra con una flecha de línea discontinua con la punta de flecha abierta. En un entorno procedimental, las flechas de retorno se pueden omitir, puesto que están implícitas al final de una especificación de ejecución, pero es mucho más claro mostrarlas.

La Figura 9.2 muestra un diagrama de secuencia con un flujo de control procedimental, e incluye una segunda llamada a un objeto anidado con otra llamada y la creación de un objeto durante el cómputo.

Un objeto activo es aquél que contiene la raíz de una pila de ejecuciones. Cada objeto activo tiene su propio hilo de control dirigido por eventos que se ejecuta en paralelo con otros objetos activos. Un objeto activo se muestra con una doble línea en cada lado del símbolo de cabecera. Esta indicación se omite a menudo, dado que realmente no significa demasiado. Los objetos que reciben llamadas de un objeto activo son objetos pasivos; reciben el control sólo cuando se les llama y lo ceden cuando retornan.

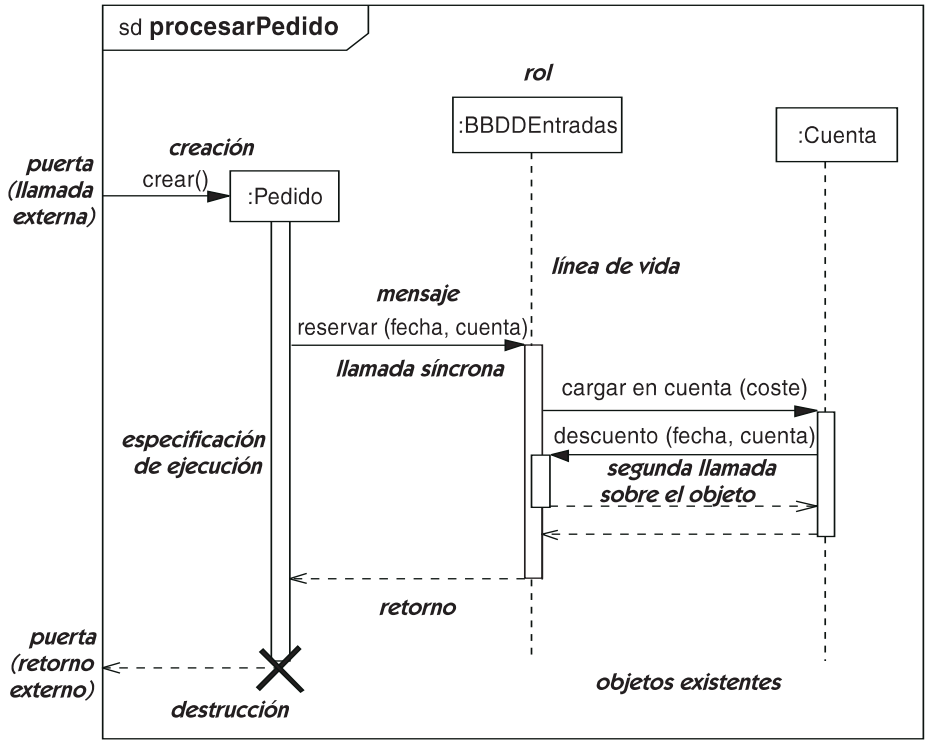


Figura 9.2 Diagrama de secuencias con especificación de ejecución

Construcciones estructuradas de control

Los flujos de control secuenciales se muestran con facilidad en un diagrama de secuencia mediante una secuencia de mensajes (ésta es la razón por la que se denomina *diagrama de secuencia*). Los flujos de control más complejos se pueden mostrar utilizando fragmentos combinados. Un fragmento combinado tiene una palabra clave y uno o más subfragmentos (denominados operandos de interacción). El número y el significado de los subfragmentos dependen de la palabra clave. *Véase* Figura 9.3.

Un uso de una interacción es una referencia a otra interacción, la cual se define normalmente con su propio diagrama de secuencia.

Un bucle tiene un subfragmento, que se ejecuta mientras la primera condición de guarda del subfragmento es verdadera.

Un fragmento condicional (palabra clave alt) tiene dos o más subfragmentos, cada uno de los cuales tiene una condición de guarda inicial. Cuando se alcanza el fragmento condicional, se ejecuta el subfragmento cuya condición de guarda sea verdadera. Si más de una fueran verdaderas, se selecciona una para su ejecución de forma no determinista. Si ninguna es verdadera, ninguna ejecución es consistente con la especificación. El fragmento opcional (palabra clave opt) es un caso especial con un único subfragmento que se ejecuta si su condición de guarda es verdadera, y se omite si es falsa.

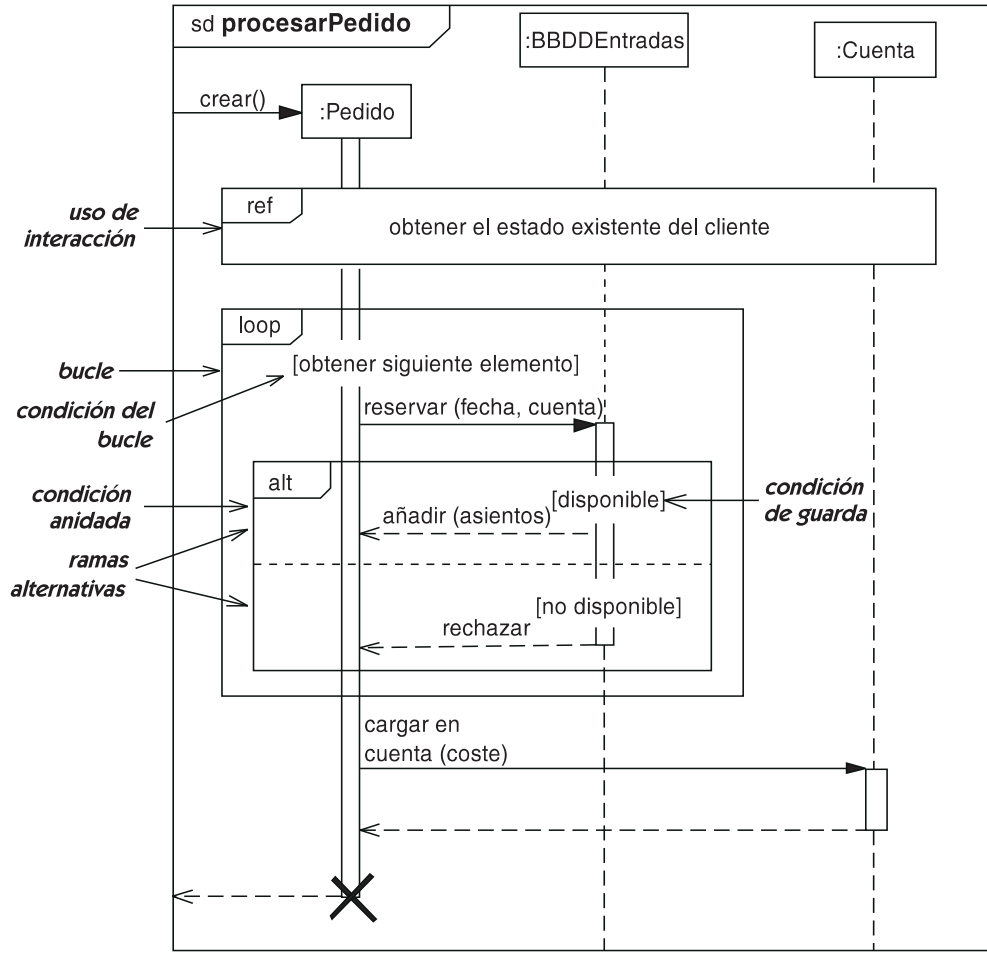


Figura 9.3 Construcciones estructuradas de control

Un fragmento paralelo (palabra clave par) tiene dos o más subfragmentos. Cuando se alcanza el fragmento, todos los subfragmentos se ejecutan concurrentemente. La secuencia relativa de los mensajes en diferentes subfragmentos es indeterminada y los mensajes se pueden entremezclar en cualquier orden posible. Cuando todos los subfragmentos han finalizado su ejecución, las ejecuciones concurrentes se vuelven a unir en un único flujo.

Existen más construcciones estructuradas especializadas.

La Figura 9.3 muestra un diagrama de secuencia que contiene un bucle con una condición anidada.

Diagrama de comunicación

Un diagrama de comunicación está basado en el contexto proporcionado por un clasificador estructurado (incluyendo una colaboración). Los roles y conectores describen la configuración de los objetos y enlaces que pueden ocurrir cuando se ejecuta una instancia del contexto. Cuan-

do se instancia el contexto, los objetos se ligan a los roles y los enlaces se ligan a los conectores. Los conectores también se pueden ligar a varios tipos de enlaces temporales, como argumentos de procedimiento o variables locales de procedimiento. Sólo se modelan los objetos involucrados en el contexto, aunque puede haber otros en el sistema completo. En otras palabras, un diagrama de comunicación modela los objetos y enlaces involucrados en la implementación de una interacción e ignora al resto.

Los mensajes entre roles se muestran mediante flechas etiquetadas vinculadas a los conectores. Cada mensaje tiene un número de secuencia, una condición de guarda opcional, un nombre y una lista de argumentos, y un nombre opcional de valor de retorno. El número de secuencia incluye opcionalmente el nombre del hilo. Todos los mensajes en el mismo hilo están ordenados secuencialmente. Los mensajes en diferentes hilos son concurrentes a menos que haya una dependencia explícita en la secuenciación. Se pueden añadir diversos detalles de implementación, como la distinción entre mensajes síncronos y asíncronos.

La Figura 9.4 muestra un diagrama de comunicación.

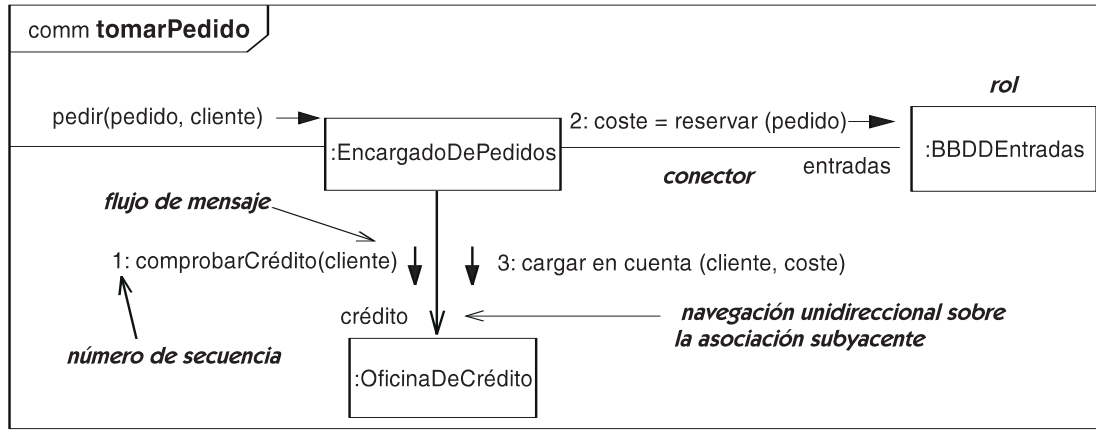


Figura 9.4 Diagrama de comunicación

Diagramas de comunicación y secuencia. Tanto los diagramas de comunicación, como los diagramas de secuencia, muestran interacciones, pero cada uno hace énfasis en aspectos distintos. Los diagramas de secuencia muestran claramente secuencias temporales pero no muestran explícitamente relaciones entre objetos. Los diagramas de comunicación muestran relaciones entre objetos con claridad, pero las secuencias temporales se obtienen de los números de secuencia. A menudo, los diagramas de secuencia son más útiles para mostrar el diseño detallado de procedimientos. Sin embargo, una vez que se ha definido la estructura de un procedimiento, los diagramas de comunicación pueden ser más útiles para planificar los pequeños detalles del control.

10
La vista de despliegue

Descripción

La vista de despliegue muestra la disposición física de los nodos. Un nodo es un recurso computacional de ejecución, como computadoras u otros dispositivos. Durante la ejecución, los nodos pueden contener artefactos, entidades físicas como los archivos. La relación de manifestación muestra la relación entre un elemento de diseño, como un componente, y los artefactos que los plasman en el sistema software. La vista de despliegue puede resaltar los cuellos de botella en el rendimiento debidos a la ubicación de los artefactos que manifiestan componentes independientes en diferentes nodos.

Nodo

Un nodo modela un recurso computacional de tiempo de ejecución, que generalmente tiene menos memoria y a menudo también capacidad de procesamiento. Los nodos pueden tener estereotipos para distinguir los diferentes tipos de recursos, como UCP, dispositivos y memorias. Los nodos pueden albergar artefactos.

Un tipo de nodo se muestra como un cubo estilizado con su nombre en el interior. Una instancia de un nodo se representa mediante una cadena subrayada con el nombre y el tipo de nodo. (Figura 10.1).

Las asociaciones entre nodos representan caminos de comunicación. Las asociaciones pueden tener estereotipos para distinguir los diferentes tipos de comunicación.

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

Artefacto

Un artefacto modela una entidad física como un archivo. Un artefacto se muestra mediante un rectángulo con la palabra clave **«artifact»**. La presencia de un artefacto en un nodo se representa anidando físicamente el símbolo del artefacto dentro del símbolo del nodo.

Se pueden marcar con estereotipos distintos tipos de artefactos, como las bases de datos, las páginas Web, los ejecutables o los guiones.

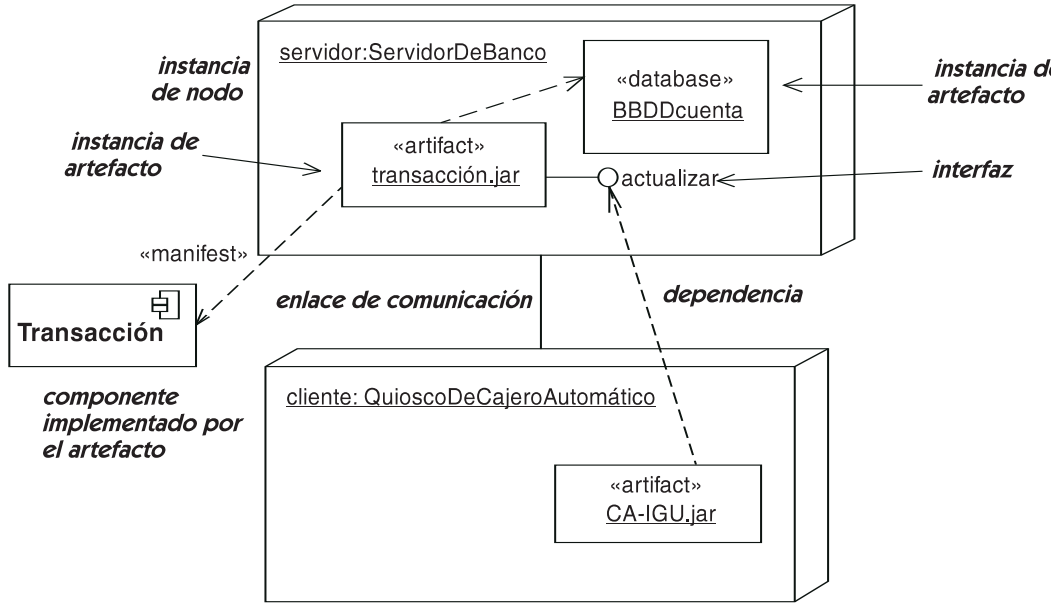


Figura 10.1 Diagrama de despliegue

Si un artefacto implementa un componente u otra clase, se dibuja una línea discontinua, con la palabra clave **«manifest»**, desde el símbolo del artefacto al símbolo del componente que implementa. Esta relación se denomina manifestación.

11

La vista de gestión del modelo

Descripción

Cualquier sistema grande debe ser dividido en unidades más pequeñas, de forma que las personas puedan trabajar con una cantidad limitada de información en cada momento, a la vez que los equipos de trabajo no interfieren entre ellos. La gestión del modelo consiste en paquetes (incluyendo tipos especiales de paquetes) y relaciones de dependencia entre ellos.

Paquete

Un paquete es una pieza de un modelo. Cada parte de un modelo debe pertenecer a un paquete. El modelador puede asignar los contenidos de un modelo a un conjunto de paquetes. Pero, para que sea funcional, la asignación debe seguir algún principio racional, como una funcionalidad común, implementación fuertemente acoplada y un punto de vista común. UML no impone una regla para la formación de paquetes, pero una buena descomposición en paquetes mejora enormemente la capacidad de mantenimiento del modelo.

Los paquetes contienen elementos de alto nivel del modelo, como las clases y sus relaciones, máquinas de estados, diagramas de casos de uso, interacciones y colaboraciones —cualquier cosa que no esté contenida en otro elemento. Los elementos, como atributos, operaciones, estados, líneas de vida y mensajes, están contenidos en otros elementos y no aparecen como contenidos directos de los paquetes. Cada elemento de alto nivel se declara en un paquete. Este es su paquete de origen. Puede ser referenciado en otros paquetes, pero los contenidos del elemento pertenecen al paquete origen. En un sistema de control de la configuración, un modelador debe tener acceso al paquete origen para modificar los contenidos del elemento. Esto proporciona un mecanismo de control de acceso para el trabajo con modelos grandes. Los paquetes también son unidades para cualquier mecanismo de control de versiones.

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

Si los paquetes son elegidos correctamente, de forma que las estructuras de organización (paquetes) se correspondan con las estructuras de diseño (componentes), reflejan la arquitectura de alto nivel de un sistema —su descomposición en subsistemas y sus dependencias.

Los paquetes que cortan a través de varios subsistemas suelen crear problemas entre los equipos de diseñadores. Una dependencia entre paquetes resume las dependencias del contenido del paquete.

Dependencias en los paquetes

Las dependencias se presentan entre elementos individuales, pero en un sistema de cualquier tamaño, se deben ver en un nivel más alto. Las dependencias entre paquetes resumen las dependencias entre los elementos que contiene —es decir, las dependencias entre paquetes son derivables de las dependencias de los 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 sobre el futuro diseño), al menos un elemento de relación de un tipo de dependencia dado entre los elementos individuales dentro del correspondiente paquete. 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 a nivel de paquete no contiene la información adicional; sólo es un resumen.

La Figura 11.1 muestra la estructura de paquetes para el subsistema de pedido de entradas. Tiene dependencias sobre paquetes externos

La aproximación descendente refleja la arquitectura global del sistema. La aproximación ascendente se puede generar automáticamente a partir de los elementos individuales. Ambas aproximaciones tienen su lugar dentro del modelado, incluso dentro de un único sistema.

Las dependencias múltiples del mismo tipo entre elementos individuales se agregan en una única dependencia a nivel de paquete entre los paquetes que contienen los elementos. Si las dependencias entre elementos individuales contienen estereotipos (como los diferentes tipos de

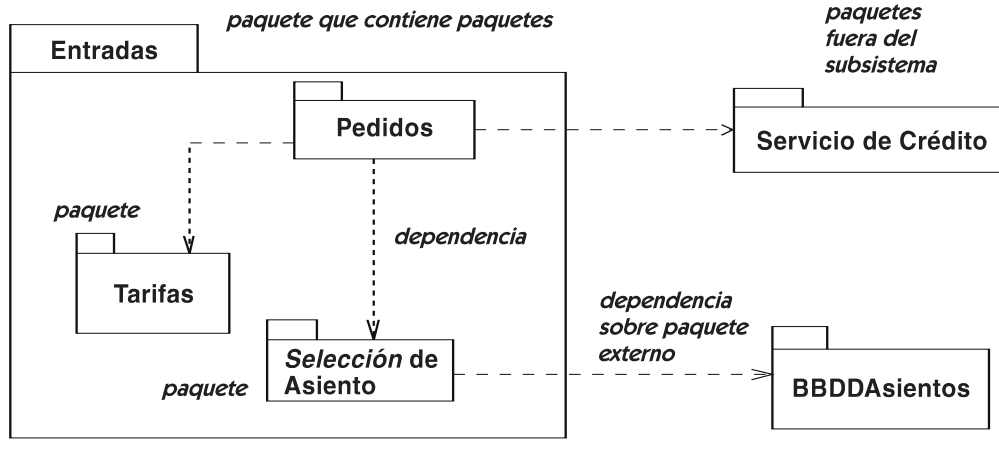


Figura 11.1 Paquetes y sus relaciones

uso), se pueden omitir los estereotipos en la dependencia a nivel de paquete a fin de producir una única dependencia de alto nivel.

Los paquetes se representan como rectángulos con lengüetas (iconos “carpeta”). Las dependencias se muestran como flechas de línea discontinua.

Visibilidad

Un paquete es un espacio de nombres para sus elementos. Establece la visibilidad de sus elementos. Los elementos que poseen directamente los paquetes pueden tener visibilidad pública o privada. Un paquete sólo puede ver los elementos de otros paquetes cuando se les ha dado visibilidad pública a través del paquete que los contiene. Los elementos con visibilidad privada sólo son visibles en el paquete que los contiene y en los paquetes anidados dentro de él.

La visibilidad también se aplica a las características de las clases (atributos y métodos). Los elementos con visibilidad pública son accesibles por otras clases, y por sus descendientes, dentro del mismo paquete o dentro de otros paquetes que puedan ver la clase. Una característica con visibilidad privada sólo es accesible por su propia clase. Una característica con visibilidad protegida es accesible desde su clase y desde los descendientes de la misma. Una característica con visibilidad de paquete es accesible desde otras clases dentro del mismo paquete, pero no por clases de otros paquetes, incluso si pueden ver la clase.

Un paquete anidado dentro de otro paquete es parte del contenedor y tiene acceso completo a su contenido. Sin embargo, un contenedor no puede ver dentro de sus paquetes anidados sin la visibilidad adecuada; los contenidos están encapsulados.

Importación

Un paquete puede importar elementos de otro paquete para añadir sus nombres a su espacio de nombres (Figura 11.2). La importación no añade una mayor potencia o capacidad de modelado, pero simplifica las expresiones de texto que aparecen en las restricciones y finalmente en el código. Se puede importar un elemento si es visible para el paquete que lo importa. Un paquete puede también importar un paquete completo, lo que es equivalente a importar todos los elementos visibles dentro de él.

Un elemento importado se convierte en un elemento visible del paquete que lo importa bajo su nombre importado (el cual puede diferir del nombre original, aunque suele ser el mismo). La

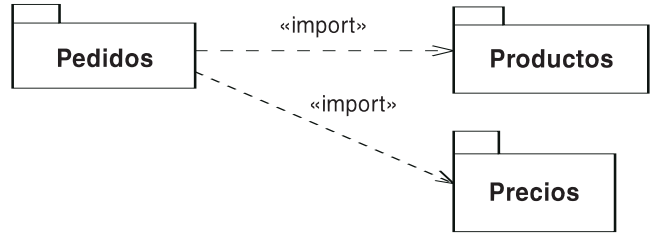


Figura 11.2 Importación de paquetes

relación de acceso añade un elemento al espacio de nombres sin hacerlo público en el paquete que lo importa.

Modelo

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

Normalmente, 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 modelo se puede mostrar como un paquete con un triángulo como adorno, pero normalmente hay poco acuerdo a la hora de mostrar los modelos como símbolos.

12
Perfiles

Descripción

UML proporciona varios mecanismos de extensión que permiten a los modeladores realizar algunas extensiones comunes sin tener que modificar el lenguaje de modelado subyacente. Estos mecanismos de extensión se han diseñado de forma que las herramientas puedan almacenar y manipular las extensiones sin tener que entender su semántica completa o su propósito. 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 para sus extensiones que sólo ellas necesitan entender.

Las extensiones se organizan en perfiles. Un perfil es un conjunto coherente de extensiones aplicable a un dominio o propósito dado. Dada su naturaleza, los perfiles no son aplicables en todas las circunstancias, y diferentes perfiles pueden ser compatibles o no con otros.

Esta aproximación a las extensiones probablemente no satisfará todas las necesidades que surjan, pero complacerá una gran parte de las necesidades de adaptación de la mayoría de los modeladores de una manera sencilla que es fácil de implementar.

Los mecanismos de extensión son los estereotipos, los valores etiquetados y las restricciones.

Hay que tener presente que, por definición, una extensión se desvía de la forma estándar de UML y, por tanto, puede llevar a problemas de interoperatividad. El modelador debería sopesar cuidadosamente los beneficios y los costes antes de utilizar las extensiones, especialmente cuando existen mecanismos que trabajarán razonablemente bien. Típicamente, las extensiones están pensadas para dominios de aplicación o entornos de programación específicos, pero dan lugar a un dialecto de UML, con las ventajas y desventajas de todos los dialectos.

Estereotipo

Muchos modeladores desean adaptar un lenguaje de modelado a un dominio de aplicación específico. Esto entraña algunos riesgos porque el lenguaje adaptado no será universalmente comprensible, pero, sin embargo, la gente intenta hacerlo.

Un estereotipo es un tipo de elemento del modelo definido en el propio modelo. La información que contiene y la forma de un estereotipo es la misma que las de los tipos existentes de elementos del modelo, pero su significado y su uso es diferente. Por ejemplo, los modeladores en el área del modelado de negocio a menudo desean distinguir los objetos de negocio y los proce-

sos de negocio como tipos especiales de elementos de modelado cuyo uso es distinto dentro de un proceso de desarrollo dado. Pueden ser tratados 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 está basado en un elemento de modelado existente. La información que contiene el elemento estereotipado es el mismo que el del elemento de modelado existente. Esto permite a una herramienta almacenar y manipular el elemento nuevo de la misma forma que lo hace con el elemento existente. El elemento estereotipado puede tener su propio icono —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 tener también una lista de restricciones que se aplican a su uso. Por ejemplo, quizá una "organización de negocios" sólo se pueda asociar con otra "organización de negocios" y no con una clase. No todas las restricciones se pueden comprobar automáticamente por una herramienta de propósito general, pero se puede hacer cumplir manualmente o ser verificadas por una herramienta adicional que entienda el estereotipo.

Los estereotipos pueden definir valores etiquetados para almacenar propiedades adicionales que no están soportadas por los elementos base.

Los estereotipos se muestran como cadenas de texto rodeadas por comillas («») situadas en el símbolo del elemento base o cerca de él; el modelador también puede crear un icono para un determinado estereotipo, el cual reemplaza el símbolo del elemento base (Figura 12.1).

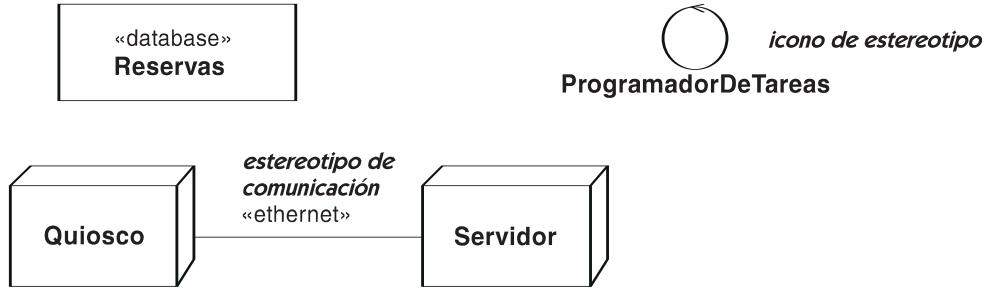


Figura 12.1 Estereotipo

Valor etiquetado

Una definición de etiqueta es una definición de un atributo para el propio elemento de modelado, es decir, la definición de un metaatributo. Define propiedades de elementos en los modelos de usuario, en lugar de propiedades de objetos en ejecución. Tiene un nombre y un tipo. La definición de una etiqueta es propiedad del estereotipo.

Cuando se aplica un estereotipo a un elemento del modelo, el elemento del modelo adquiere las etiquetas definidas en el estereotipo (Figura 12.2). Para cada etiqueta, el modelador puede especificar un valor etiquetado. Los valores etiquetados se muestran como cadenas, dentro de notas vinculadas con el elemento del modelo, con el nombre de la etiqueta, un signo igual y el valor (Figura 12.2). A menudo se omiten en los diagramas, pero se muestran en las listas desplegables y formularios.

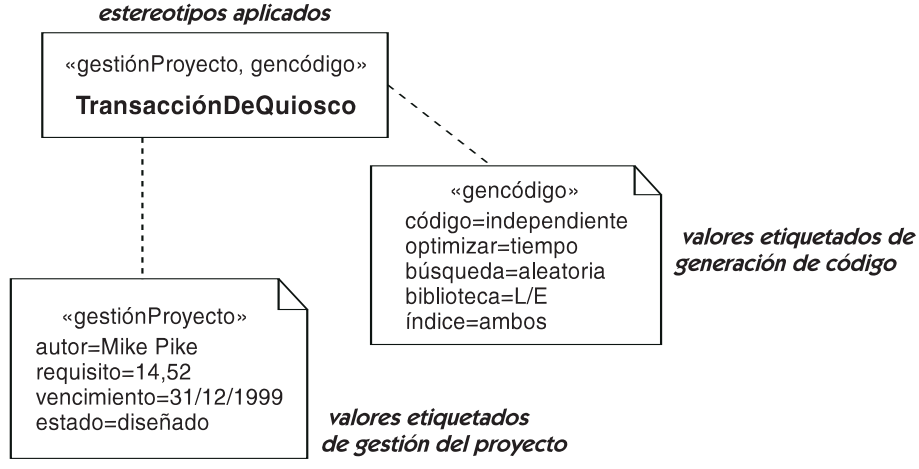


Figura 12.2 Estereotipos y valores etiquetados

En la definición de un estereotipo, cada etiqueta es el nombre de alguna propiedad que el desarrollador quiere guardar, y el valor en un elemento del modelo es el valor de esa propiedad para el elemento dado. Por ejemplo, el nombre podría ser **autor** y el valor podria ser el nombre de la persona responsable del elemento, como **Charles Babbage**.

Se pueden definir etiquetas para almacenar información arbitraria sobre los elementos. Son particularmente útiles para almacenar información para la gestión de proyectos, como la fecha de creación de un elemento, su estado de desarrollo, fechas de vencimiento y estado de las pruebas. Se puede utilizar cualquier cadena como nombre de etiqueta, excepto los nombres de atributos incorporados en el metamodelo (porque las etiquetas y los atributos juntos se pueden considerar propiedades de un elemento y se puede acceder a ellos uniformemente en una herramienta), y el número de nombres de etiquetas está predefinido.

Los valores etiquetados también proporcionan una forma de vincular información adicional dependiente de la implementación a los elementos. Por ejemplo, un generador de código puede necesitar información adicional sobre el tipo de código a generar a partir del modelo. A menudo, hay varias formas posibles de implementar correctamente un modelo; el modelador puede proporcionar una orientación sobre las decisiones a tomar. Algunas etiquetas se pueden utilizar como indicadores para decirle al generador de código qué implementación utilizar. Otras etiquetas se pueden utilizar para otros tipos de herramientas adicionales, como los planificadores de proyectos y los generadores de informes.

Perfil

Un perfil es un paquete que identifica un subconjunto de un metamodelo base existente (posiblemente incluyéndolo por completo) y define estereotipos y restricciones que se pueden aplicar al subconjunto del metamodelo seleccionado (Figura 12.3). Está pensado para realizar extensiones limitadas de UML para adaptarlo a un dominio, tecnología o implementación específicos.

Un perfil se hace disponible para un modelo de uso mediante la aplicación del perfil en un paquete. Las restricciones en el perfil se aplican a los elementos del paquete y los elementos del

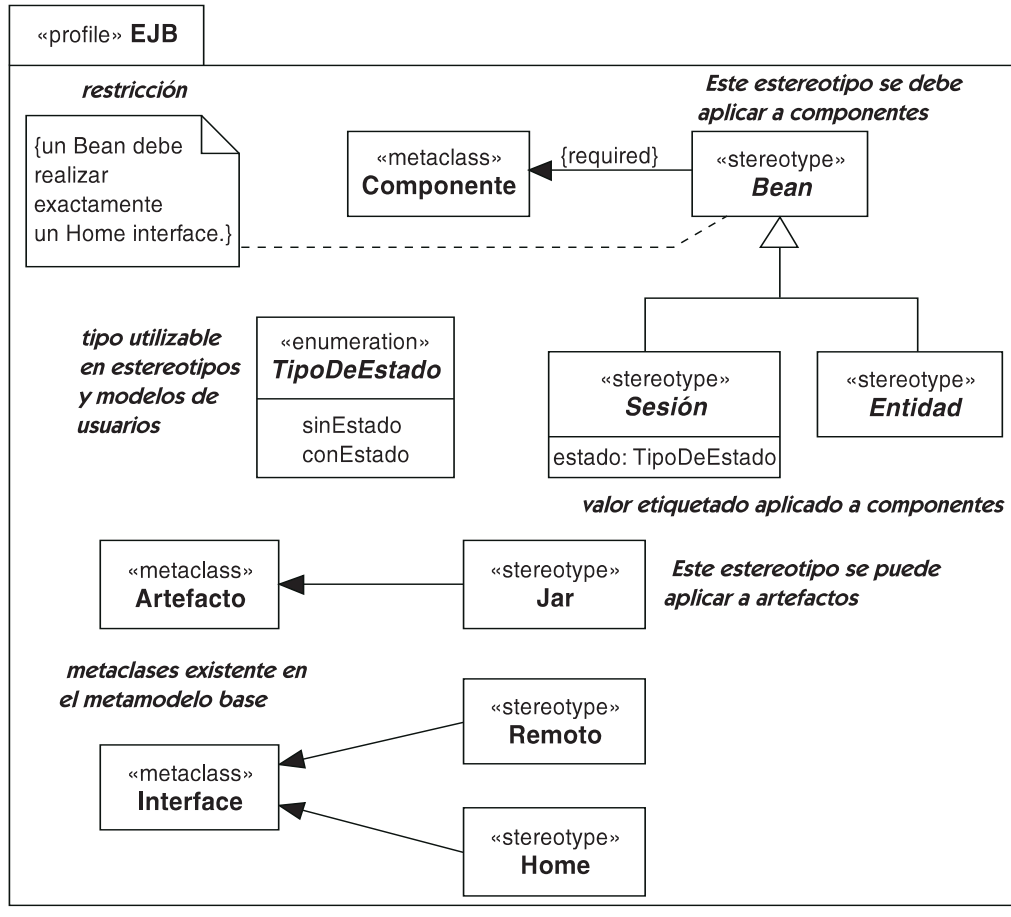


Figura 12.3 Estereotipos y valores etiquetados

paquete pueden utilizar los estereotipos definidos en el perfil. La Figura 12.4 muestra la aplicación de perfiles en un paquete.

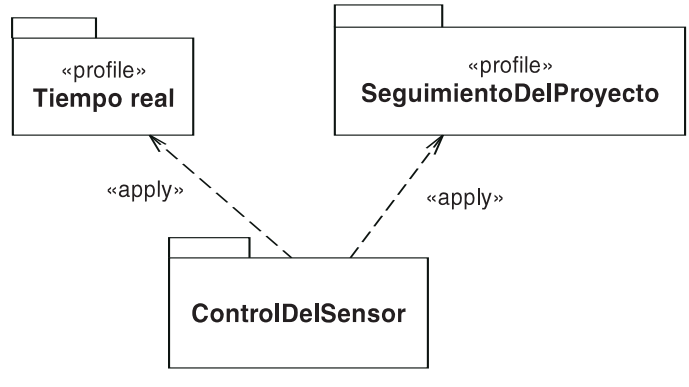


Figura 12.4 Aplicación de perfiles

13
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 el desarrollo de buenos sistemas— y no un fin en sí mismo. El propósito y la interpretación del modelo se ven afectados por el resto del entorno. Otras facilidades en un entorno más amplio incluyen metamodelos que abarcan muchos lenguajes, herramientas de edición del modelo, lenguajes de programación, sistemas operativos y los principales componentes 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 e implementar sus objetivos 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 diferentes herramientas y lenguajes de implementación. Cada nivel de utilización 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 mediante un metamodelo. Un metamodelo intenta hacer que un lenguaje sea preciso mediante la definición de su semántica, pero hay una tensión para permitir extensiones para situaciones nuevas. La forma real del metamodelo es importante para la implementación de herramientas y el intercambio de modelos, pero no es muy importante para la mayoría de los usuarios. Por lo tanto, no lo hemos incluido en este libro. Aquellos que estén interesados pueden consultar la documentación original del estándar disponible en el sitio Web del OMG (www.omg.org).

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, probablemente es erróneo considerar tal opción. En cambio, uno puede pensar en las diferentes 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 ejecución, pero que es ortogonal a otros aspectos del sistema. Por ejemplo, un entorno puede elegir dar soporte, o no, a la clasificación dinámica, la capacidad de un objeto de cambiar de clase durante la ejecución. En la actualidad,

la mayoría de los lenguajes de programación no las permiten, principalmente por razones de implementación de los lenguajes de programación, pero algunos lo hacen. La diferencia es inapreciable en la semántica estática. La elección de clasificación estática o 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, a menudo, argumenta sobre cuál es la interpretación correcta. En cambio, observe que esto es una elección y asígnele un nombre para que se pueda utilizar.

Un metamodelo describe los contenidos de un modelo que está bien formados, de la misma forma que un lenguaje de programación describe un programa bien formado. Sólo un modelo bien formado tiene significado y una semántica apropiada; no tiene sentido pedir el significado de un modelo mal formado. Sin embargo, la mayor parte del tiempo, los modelos en desarrollo están mal formados. Son incompletos y posiblemente inconsistentes. Pero esto es lo que deben contemplar las herramientas de edición de modelos —modelos incompletos, no sólo modelos acabados. El metamodelo de UML describe modelos correctos bien formados. Un metamodelo distinto podría describir posibles fragmentos de un modelo. Dejamos a los fabricantes de herramientas la decisión sobre dónde se encuentra el límite en los fragmentos del modelo admitidos y qué semántica dar a los modelos mal formados.

UNL incluye perfiles para adaptar su uso en dominios especializados. El mecanismo de los perfiles incluye la capacidad de definir estereotipos con valores etiquetados. Los perfiles se pueden utilizar para confeccionar una variante de UML mediante la definición de un conjunto de estereotipos y etiquetas, y adoptando acuerdos para su uso a la hora de construir un modelo. Por ejemplo, las variantes se pueden desarrollar de forma que se centren en las semánticas de implementación de varios lenguajes de programación La adición de extensiones puede ser poderosa, pero conlleva algunos peligros inherentes. Dado que su semántica no está definida dentro de UML, UML no puede proporcionar su significado; la interpretación es una cuestión del modelador. Además, si no se tiene cuidado, algunos significados pueden ser ambiguos o incluso inconsistentes. Las herramientas de modelado pueden proporcionar soporte automático para los estereotipos y etiquetas definidos por las herramientas, pero no para las extensiones definidas por los usuarios. A pesar del soporte para las extensiones, cualquier extensión aleja al usuario de centro común que proporciona el lenguaje estándar y socava las metas de facilidad de intercambio y comprensión de los modelos. Algunos perfiles pueden ser estandarizados por el OMG o grupos independientes, lo que reduce el peligro de la incompatibilidad. Por supuesto, siempre que se utilice una biblioteca de clases particular, se divergirá de la perfecta facilidad de intercambio de algo sin importancia. Así que no se preocupe por ello en abstracto. Utilice los perfiles cuando sean de ayuda, pero evítelos cuando no sean necesarios.

Responsabilidades de la notación

La notación no añade significado a un modelo, pero ayuda al usuario a comprender su significado: La notación no tiene semántica, pero a menudo añade connotaciones para un usuario, como la percepción de la afinidad de dos conceptos basados en su proximidad en un diagrama.

Las especificaciones de UML y este libro definen una notación canónica de UML, que podría ser denominado formato de publicación para modelos. Esto es parecido a muchos lenguajes de programación en los que los programas dentro de artículos de revistas se imprimen en un formato atractivo con una disposición cuidadosa, las palabras reservadas en negrita y figuras sepa-

radas para cada procedimiento. Los auténticos compiladores tienen que aceptar una entrada menos precisa. Esperamos que las herramientas de edición extiendan la notación a un formato de pantalla, incluyendo cosas como el uso de tipografías y el color para resaltar elementos; la capacidad de suprimir fácilmente y de filtrar elementos que no son de interés en un determinado momento, ampliar un diagrama para mostrar elementos anidados, atravesar enlaces a otros modelos o vistas; y animación. Sería imposible tratar de estandarizar todas estas posibilidades, y una locura intentarlo, porque no hay necesidad y limitaría la innovación útil. Este tipo de extensiones en la notación son responsabilidad del constructor de herramientas. En una herramienta interactiva, hay menos peligro de ambigüedad, porque el usuario puede pedir siempre una aclaración. Probablemente esto es más útil que insistir en una notación que sea completamente precisa al primer vistazo. Una herramienta debe ser capaz de producir la notación canónica cuando se le solicita, especialmente en formato impreso, pero se deben esperar extensiones razonables en una herramienta interactiva.

Contamos con que las herramientas también permitirán extender la notación de una forma limitada pero útil. Hemos especificado que los estereotipos pueden tener sus propios iconos. Se podrían permitir otros tipos de extensiones de la notación, pero los usuarios deben aplicar un buen criterio.

Observe que la notación es más que imágenes; incluye 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. UML debería permitir el uso de cualquier (o por lo menos de muchos) lenguaje de programación, tanto para la especificación, como para la generación del código final. El problema es que cada lenguaje de programación tiene muchos elementos semánticos que no deberían ser absorbidos por 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 la concurrencia se maneja de diferentes formas según los lenguajes (si es que se maneja). Los perfiles pueden ser definidos para distintos lenguajes de programación.

Los tipos de datos primitivos no se describen detalladamente en UML. Esto es deliberado, puesto que se quería evitar 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. Esto es un ejemplo de punto de variación semántica que puede ser tratado en un perfil de lenguaje.

La representación de propiedades detalladas del lenguaje para la implementación plantea el problema de la captura de información sobre las propiedades de implementación sin la construcción de su semántica en UML. Una aproximación es capturar las propiedades del lenguaje que van más allá de las capacidades predefinidas de UML mediante la definición de perfiles que contienen estereotipos y valores etiquetados para diversas propiedades del lenguaje de programación. Un editor genérico no necesitaría comprenderlos. Es más, un usuario podría crear un modelo utilizando una herramienta que no permitiera el lenguaje elegido, y transferir el modelo final a otra herramienta, como un generador de código, para su procesamiento final. El generador de código debería entender los estereotipos. Por supuesto, si la herramienta genérica no comprende los estereotipos ni las etiquetas, no puede verificar su

consistencia. Pero esto no es peor que la práctica habitual con los editores de texto y los compiladores.

La generación de código y la ingeniería inversa para el futuro inmediato necesitará la participación del diseñador además de un modelo UML. Las directivas y ayudas para el generador de código se pueden sustituir por estereotipos y valores etiquetados. Por ejemplo, el modelador podría indicar qué tipo de clase contenedora se debería utilizar para implementar una asociación. Por supuesto, esto significa que los ajustes de la generación de código en las herramientas podrían ser incompatibles, pero no creemos que en la actualidad haya un acuerdo suficiente sobre la forma correcta de estandarizar los ajustes de los lenguajes. En cualquier caso, las diferentes herramientas utilizarán sus generadores de código como ventaja competitiva. Finalmente, los ajustes por defecto pueden surgir y llegar a madurar para su estandarización.

Modelado con herramientas

Para modelar sistemas de tamaño real se necesitan herramientas. Las herramientas proporcionan formas interactivas de ver y editar los modelos. Proporcionan un nivel de organización que se encuentra fuera del ámbito del propio UML, pero que aporta comprensión al usuario y le ayuda en el acceso a la información. Las herramientas ayudan a encontrar información en modelos grandes mediante la búsqueda y el filtrado de lo que se presenta.

Algunos aspectos sobre el uso de herramientas

Las herramientas se ocupan de la organización física y del almacenamiento de los modelos. Deben dar soporte a múltiples equipos de trabajo en un único proyecto, así como permitir la reutilización entre proyectos. Los siguientes aspectos se encuentran fuera del ámbito del UML canónico, pero debe ser considerado para el uso real de la herramienta.

Ambigüedades e información sin especificar. En las etapas iniciales, faltan muchas cosas por decir. Las herramientas deben ser capaces de ajustar la precisión de un modelo y no forzar a que cada valor sea específico. Véanse las secciones “Modelos inconsistentes para el trabajo en curso” y “Valores sin especificar y nulos”.

Opciones de presentación. Los usuarios no quieren ver toda la información todo el tiempo. Las herramientas deben permitir el filtrado y ocultación de la información que no desea en un momento dado. Las herramientas también deberán permitir visualización alterna mediante las capacidades del hardware de visualización. Esto se ha descrito ya en la sección “Responsabilidades de la notación”.

Gestión del modelo. El control de la configuración, el control de acceso y el control de versiones de las unidades del modelo se encuentra fuera del ámbito 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 generadores de informes, los motores de ejecución y otras herramientas del extremo final del desarrollo. Es necesario incluir información para otras herramientas en el modelo, pero no es información UML. Los perfiles con estereotipos y valores etiquetados son adecuados para albergar esta información.

Modelos inconsistentes para el trabajo en curso

La máxima meta del modelado es producir una descripción de un sistema a un determinado nivel de detalle. El modelo final debe satisfacer varias restricciones de validez para que tenga significado. Sin embargo, como en cualquier proceso creativo, el resultado no se produce necesariamente de un modo lineal. Los productos intermedios no satisfarán todas las restricciones de validez en cada paso. En la práctica, una herramienta debe manejar, no sólo modelos semánticamente válidos que satisfacen las restricciones de validez, sino también modelos sintácticamente válidos que satisfacen ciertas reglas de construcción pero que pueden violar algunas restricciones de validez. Los modelos semánticamente inválidos no son directamente utilizables. En cambio, se puede pensar en ellos como "trabajo en curso" que representa caminos 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 posibles valores, pero si un valor puede ser nulo, es una parte de la descripción del tipo del atributo; muchos tipos no tienen un valor nulo natural dentro de su rango de valores. Por ejemplo, nulo no tiene ningún sentido como límite superior del tamaño de un conjunto. O el conjunto tiene un límite superior fijo, o no tiene límite, en cuyo caso su tamaño máximo es ilimitado. La posibilidad de ser nulo realmente es sólo un aumento del rango de posibles valores de un tipo de datos.

Por otra parte, durante las primeras etapas del diseño, un desarrollador no se preocupa de los valores de una determinada propiedad. Podría haber un valor que carece de significado en una determinada fase, por ejemplo, la visibilidad cuando se realiza el modelo de dominio. O el valor puede tener significado pero el modelador no lo ha especificado todavía y el desarrollador necesita recordar que todavía debe dársele uno. En este caso, el valor está sin especificar. Esto indica que un valor finalmente será necesario pero no se ha especificado todavía. Esto no es lo mismo que un valor nulo, el cual puede ser un valor legítimo en el modelo final. En muchos casos, especialmente con las cadenas, un valor nulo es una buena forma de indicar un valor no especificado, pero no es lo mismo. Un valor no especificado no tiene sentido en un modelo bien formado. La definición de UML no maneja valores no especificados. Estos son responsabilidad de las herramientas que soportan UML y son considerados parte del "trabajo en curso" del modelo que, por necesidad, no tiene significado semántico.

Parte 3: Referencia

14
Diccionario de términos

abstracción

El acto de identificar las características esenciales de una cosa que la distingue de otros tipos de cosas y omitir los detalles que no son importantes desde un cierto punto de vista. La abstracción implica buscar parecidos entre conjuntos de cosas centrándose en sus características esenciales comunes. Una abstracción siempre involucra la perspectiva y el propósito del que la ve; diferentes propósitos son el resultado de distintas abstracciones de la misma cosa. Cualquier modelado involucra a la abstracción, a menudo a diferentes niveles para varios propósitos. El nivel más alto de abstracción es el que menos detalles incluye.

Un tipo de dependencia que relaciona dos elementos que representan el mismo concepto en diferentes niveles de abstracción.

Véase derivación, realización, refinamiento y dependencia de traza.

Semántica

Una dependencia de abstracción es una relación entre dos elementos en diferentes niveles de abstracción, como las representaciones en diferentes modelos, con diferentes niveles de precisión, con distintos niveles de concreción o con diferentes niveles de optimización. Generalmente, las dos representaciones no se podrían utilizar de forma simultánea. Normalmente, un elemento es más detallado que otro: el elemento más detallado es el cliente y el menos detallado es el proveedor. Si no está claro qué elemento es el más detallado, entonces cualquiera de ellos se puede modelar como cliente.

Las variantes de la dependencia de abstracción son la dependencia de traza, la de refinamiento, la de derivación y la de realización.

Notación

Una dependencia de abstracción se muestra mediante una flecha de línea discontinua desde el elemento cliente al elemento proveedor, con la palabra clave **«trace»**, **«refine»** o **«derive»**. La dependencia de realización tiene su propia notación, con una flecha de línea discontinua y una punta de flecha triangular en el elemento proveedor.

La correspondencia entre elementos se puede vincular a la relación mediante el símbolo como restricción.

Discusión

La frase clase abstracta sigue la primera de las definiciones. Una clase abstracta se centra en unos pocos detalles esenciales que se aplican a un rango de clases concretas. El cambio de punto de vista se consigue subiendo o bajando por la jerarquía de generalización.

abstracto/a

Clase, caso de uso, seña u otro clasificador que no puede ser instanciado. 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 está incompleta (como cuando faltan métodos para una o más operaciones), o bien porque no se tiene la intención de instanciarla aunque su descripción esté completa. Una clase abstracta está pensada para la especialización. Para que sea útil, una clase abstracta debe tener descendientes que puedan tener instancias; una clase abstracta hoja carece de utilidad. (Puede aparecer como hoja en un marco de trabajo, pero finalmente deberá ser especializada.)

Una operación que carece de implementación (un método) es abstracta. Una clase concreta no puede tener ninguna operación abstracta (de otra forma, sería necesariamente abstracta), pero una clase abstracta puede tener operaciones concretas. Las operaciones concretas son aquellas que se pueden implementar una vez y utilizadas en todas las subclases. En su implementación, las operaciones concretas sólo pueden utilizar características (atributos y métodos) conocidos en la clase en la que son declaradas. Uno de los propósitos de la herencia es situar estas operaciones en superclases abstractas, de forma que puedan ser compartidas por todas las subclases. Una operación concreta puede ser polimórfica —es decir, puede ser sobreescrita por un método en una clase descendiente— pero no es necesario que lo sea (puede ser una operación hoja). Una clase, en la que se hayan implementado todas sus operaciones, puede ser abstracta, pero debe ser declarada explícitamente como tal. Una clase con una o más operaciones sin implementar es abstracta automáticamente.

Una semántica similar 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 puede aparecer en la definición de casos de uso concretos mediante las relaciones de generalización, inclusión o extensión. Llevando el comportamiento común a un caso de uso abstracto, el modelo se reduce y es más fácil de comprender.

Notación

El nombre de un clasificador abstracto o de una operación abstracta se muestra en cursiva. También se puede utilizar la palabra reservada **abstract**, situándola en la lista de propiedades debajo o después del nombre, por ejemplo, **Cuenta {abstract}**.

Véase también nombre de clase.

Ejemplo

La Figura 14.1 muestra una clase abstracta **Cuenta** con una operación abstracta, **calcularIntereses,** y una operación concreta, **ingresar**. Se han declarado dos clases concretas. Debido a que las subclases son concretas, cada una de ellas debe implementar la operación **calcularIntereses**. Los atributos siempre son concretos.

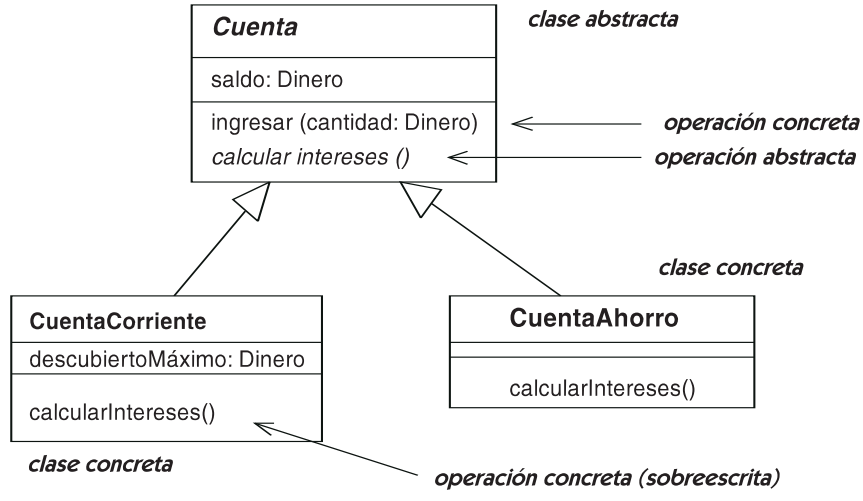


Figura 14.1 Clases abstractas y concretas

Historia

En UML2, se ha restringido la generalización a los clasificadores. No obstante, la declaración de una operación en un clasificador se puede considerar abstracta o concreta aunque las propias operaciones no sean generalizables.

Discusión

La distinción entre modelar una clase abstracta o concreta no es tan importante como podría parecer a primera vista. Es más una decisión de diseño sobre el modelo, que una propiedad inherente. Durante la evolución del diseño, el estado de una clase puede cambiar. Una clase concreta se puede modelar como abstracta si se añaden subclases que incluyan todas sus posibilidades. Una clase abstracta se puede modelar como concreta si se constata que las diferencias entre las subclases son innecesarias y deben ser eliminadas o si se representan mediante atributos en lugar de con subclases distintas.

Una forma de simplificar la decisión, es adoptar el principio de que todas las clases no hoja deben ser abstractas (y todas las clases concretas deben ser, por necesidad, concretas, excepto en el caso de una clase abstracta hoja pensada para una especialización futura). Esto no es una regla de UML; es un estilo que se puede adoptar, o no. La razón de esta regla de “superclases abstractas” es que un método heredable de una superclase y un método de una clase concreta a menudo tienen necesidades distintas y no están bien cubiertas con un único método. El método de la superclase está obliga-

do a hacer dos cosas: definir el caso general que debe ser observado por todos los descendientes e implementar el caso general para la clase específica. Generalmente, estos objetivos entran en conflicto. Por otra parte, cualquier superclase no abstracta se puede separar mecánicamente en una superclase abstracta y en una subclase concreta hoja. La superclase abstracta contiene todos los métodos que se necesitan para la clase instanciable específica. Siguiendo la regla de la superclase abstracta también se puede distinguir claramente entre una variable o parámetro que debe albergar el tipo concreto específico y el que puede albergar cualquier descendiente de la superclase.

En la Figura 14.2, considere la declaración de la clase **Carta** que no sigue la regla de la superclase abstracta. Esta clase tiene una operación, **obtenerSiguienteFrase**, que devuelve el texto de la siguiente frase no leída, así como una operación, **inicializarCursor**, que coloca el cursor al comienzo. Sin embargo, la subclase **CartaCifrada** representa una carta que ha sido cifrada. La operación **obtenerSiguienteFrase** ha sido sobreescrita porque se debe descifrar el texto antes de devolverlo. La implementación de la operación es completamente diferente. Dado que **Carta** es una superclase concreta, es imposible distinguir un parámetro que debe ser una **Carta** ordinaria (no sobreescribible) de otra que podría ser, tanto una **Carta** ordinaria, como una **CartaCifrada**.

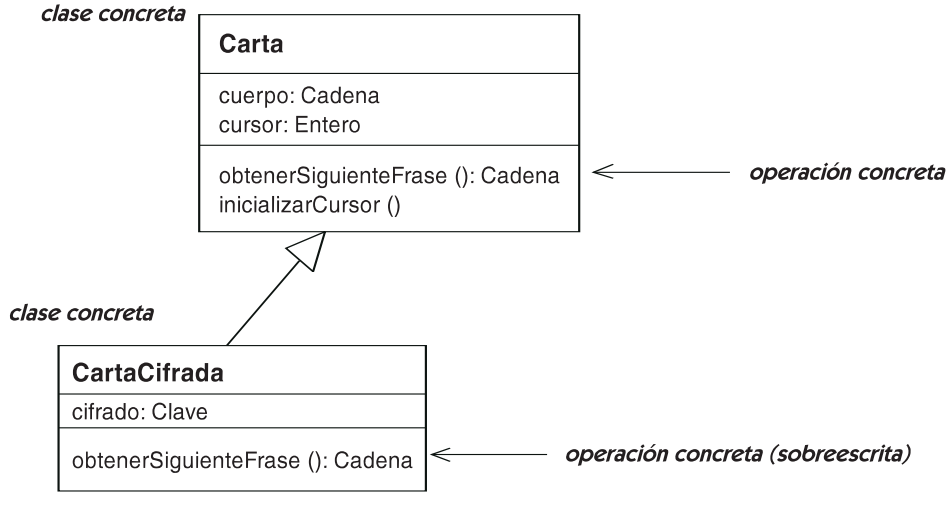


Figura 14.2 Una superclase concreta crea ambigüedad

El enfoque de la superclase abstracta es distinguir la clase abstracta **Carta** (que podría ser una carta cifrada o no cifrada) y añadir la clase **CartaNoCifrada** para representar la clase concreta, como se muestra en la Figura 14.3. En este caso, **obtenerSiguienteFrase** es una operación abstracta que se implementa en cada subclase, mientras que **inicializarCursor** es una operación concreta, idéntica en todas las subclases. El modelo es simétrico.

Si se sigue la regla de la superclase abstracta, se puede determinar automáticamente la declaración de clases abstractas a partir de la jerarquía de clases, y mostrarlo en los diagramas sería redundante.

Hay una excepción a la afirmación de que una clase abstracta hoja es inútil: Una clase abstracta se puede declarar como parte de un espacio de nombres común para un conjunto de atri-

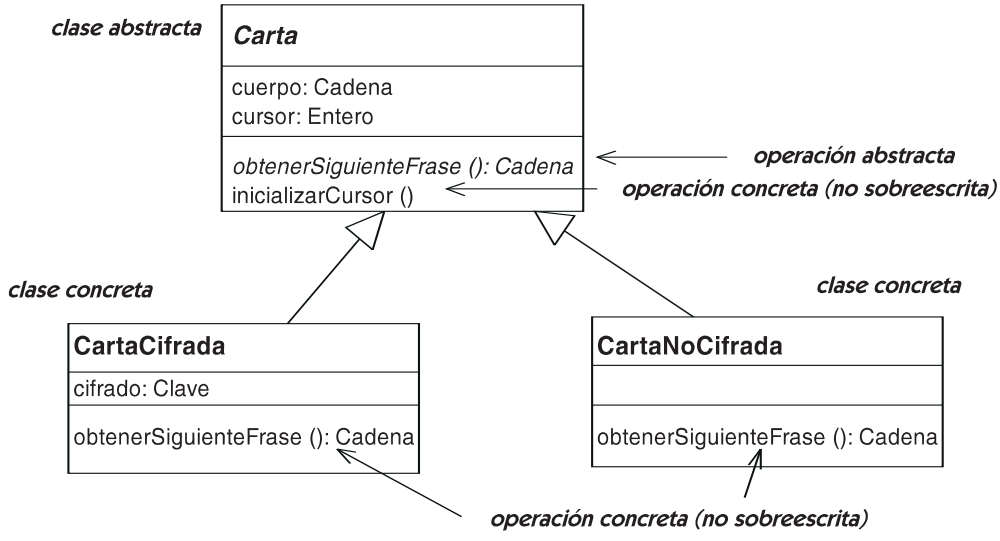


Figura 14.3 Una superclase abstracta evita la ambigüedad

butos y operaciones de ámbito global de la clase. Esta utilización es poco frecuente, pero a veces necesaria cuando se programa con lenguajes no orientados a objetos. Eso sí, los usuarios deben evitarlo siempre que sea posible, ya que las variables globales violan el espíritu del diseño orientado a objetos al introducir dependencias globales. Una clase unitaria (singleton) a menudo puede ofrecer la misma funcionalidad de una forma más amplia. *Véase* [Camma-95], Patrón Unitario (Singleton).

acceso

Una variación de la dependencia de importación que permite que un paquete utilice nombres de otro espacio de nombres para referenciar elementos de ese espacio de nombres. Los nombres importados son privados dentro del paquete que los importa y no pueden ser referenciados ni reimportados por paquetes externos.

Véase importar para más detalles.

acción

Un nodo de actividad atómica cuya ejecución se da en el cambio de estado del sistema o en el retorno de un valor.

Semántica

Una acción es un nodo de actividad atómica —es decir el cómputo más pequeño que se puede expresar en UML. Una acción es un nodo de actividad que *hace* algo al estado del sistema o extrae información de él. En una actividad de alto nivel se ve como un árbol de nodos de actividad anidados en el que las hojas son acciones. Las acciones pueden ser funciones aritméticas y

de cadena, manipulaciones de los objetos y de sus valores, comunicaciones entre objetos, y cosas similares. Observe que los constructores de flujo de control, como las condiciones y los bucles, no se consideran acciones debido a que son actividades que contienen partes más pequeñas. Sin embargo, las llamadas a operaciones se consideran acciones. Aunque la ejecución de una llamada puede ser recursiva, la propia acción de llamada es un nodo hoja dentro de la especificación de una determinada actividad y no tiene una estructura embebida dentro de la actividad. Una acción de llamada referencia a la definición de otra actividad, pero la definición de la actividad es distinta de aquélla que contiene la llamada.

Entre las acciones típicas se encuentran la asignación de valores a atributos, el acceso a los valores de los atributos o de los enlaces, la creación de nuevos objetos o enlaces, la aritmética simple y el envío de señales a otros objetos. Las acciones son los pasos discretos con los que se construye el comportamiento.

Una acción tiene un número de patillas de entrada y de patillas de salida que modelan la entrada y salida de valores. El número y el tipo de las patillas depende del tipo de acción, Por ejemplo, la acción de resta tiene dos patillas de entrada y una de salida, todas de tipo numérico.

Una acción puede comenzar su ejecución cuando los valores de entrada se encuentran disponibles en todas las patillas de entrada y los tokens de control se encuentran disponibles en todos los edges de control. Cuando comienza la ejecución, se consumen los valores de entrada y los tokens de control, por lo que no pueden ser utilizados por otra acción. Cuando se completa la ejecución de una acción, se producen los valores de salida en todas las patillas de salida y los tokens de control se sitúan en todos los edges de control salientes.

Normalmente las acciones se perciben como cómputos “rápidos” que necesitan un tiempo mínimo para su ejecución. En particular, y con pocas excepciones, la ejecución de una acción no tiene una estructura interna relevante que deba ser modelada. Si la ejecución de la acción es interrumpida antes de que finalice, no tiene efecto sobre el estado del sistema; no hay efectos intermedios visibles desde el exterior. La excepción es la acción de llamada. Después de que se ha transmitido la petición de llamada, la ejecución de esta acción se bloquea hasta que se recibe una respuesta de la actividad iniciada; la ejecución de una acción de llamada puede llevar una cantidad de tiempo no definida. El modelo de ejecución podría ser ligeramente más claro si la acción de llamada se dividiera en dos partes, una para realizar la llamada y otra para esperar la respuesta, pero esto introduciría complicaciones innecesarias.

El sistema puede ejecutar varias acciones concurrentes, pero su ejecución no tiene interacción directa; sin embargo, pueden existir dependencias indirectas a través del uso de objetos compartidos. La previsión de la ejecución de acciones concurrentes no restringe la implementación de un modelo. La concurrencia se puede implementar en un único procesador que comparte tiempo entre múltiples ejecuciones o se puede implementar mediante múltiples unidades de ejecución.

Algunos tipos de acciones pueden terminar de forma anormal y elevar excepciones en lugar de su finalización normal. Para una discusión sobre el tema, *véase* excepción.

La ejecución de las acciones puede interrumpirse por la fuerza mediante determinados mecanismos. Para una discusión sobre el tema, *véase* interrupción.

La especificación completa de UML permite la especificación del streaming behavior y de la ejecución de acciones no reentrantes. Estas características avanzadas se deben utilizar con cuidado para evitar modelos mal formados. Se deben evitar en la mayoría de los modelos lógicos.

Estructura

Una acción tiene una lista de patillas de entrada y patillas de salida. El número de patillas depende del tipo específico de acción. La mayoría de los tipos de acciones tienen un número fijo de patillas para albergar valores de tipos especificados; unos pocos tipos de acciones tienen un número variable de patillas con un tipo arbitrario. Durante la ejecución, cada patilla de entrada alberga un valor que es un argumento de la acción. Después de la ejecución, cada patilla de salida alberga un valor que es el resultado de la acción.

Precondiciones y postcondiciones. Una acción puede tener precondiciones y postcondiciones locales. Una precondición es una afirmación realizada por el modelador en la que una determinada condición se supone que debe ser verdadera cuando se inicia una acción. No está pensada como una condición de guarda que puede o no ser verdadera. Si se viola una precondición, el modelo está mal formado. La aplicación de precondiciones es un punto de variación semántica determinado por el entorno de implementación.

Una postcondición es una afirmación de que una determinada condición se supone que debe ser verdadera cuando se completa la ejecución de la acción. Su aplicación es similar a una precondición.

Tipos de acciones

Acción de aceptación de llamada. Es una variante de la acción de aceptación de evento. El disparador es la recepción de una llamada de una operación especificada. La llamada normalmente es síncrona y la ejecución de la llamada se bloquea a la espera de la respuesta. La salida de esta acción proporciona una pieza de información opaca que identifica la ejecución de la acción que llamó al objeto actual. Esta información se puede copiar y pasar a otras acciones, pero sólo se puede utilizar una vez en una acción de respuesta para transmitir una respuesta al llamador. Si la llamada fuera asíncrona, la información de retorno está vacía y cualquier intento posterior de respuesta a la llamada sería un error de programación con consecuencias no determinadas. (Un perfil de implementación particular podría especificar las consecuencias.)

Acción de aceptación de evento. La ejecución de esta acción bloquea la ejecución del hilo de ejecución actual hasta que el objeto que lo posee detecta un tipo especificado de evento, a menudo la recepción de una señal especificada. Los parámetros del evento se incluyen en la salida de la acción. Esta acción está pensada para invocaciones asíncronas. Si se dispara mediante una llamada síncrona sin parámetros de retorno, se realiza un retorno inmediato aunque continúa la ejecución del hilo actual. Es un error semántico si se dispara la acción con una llamada síncrona con parámetros de retorno, porque la acción de aceptación de evento no puede proporcionarlos. La acción de aceptación de llamada está pensada para esta situación.

Acción de aplicar función. La ejecución de esta acción genera una lista de valores de salida como función de una lista de valores de entrada. No hay efectos laterales en la parte del sistema; el único efecto es la generación de los propios valores de salida. La transformación se especifica como una función primitiva. Una función primitiva es una función matemática; es decir, define cada valor de salida en función de la lista de valores de entrada sin ningún efecto interno o efecto lateral. La especificación de funciones primitivas se encuentra fuera del propio ámbito de UML; se debe especificar utilizando algún lenguaje externo, como la notación matemática. La intención es que cualquier implementación particular de un sistema tenga un conjunto predefinido de funciones primitivas que se pueden utilizar. No está pensado para permitir la

definición, por parte del usuario, de nuevas funciones primitivas según le convenga. Como ejemplos de funciones primitivas se encuentran las operaciones aritméticas, operaciones Booleanas y operaciones de cadena. Para determinados sistemas, las operaciones sobre la hora y la fecha también son funciones primitivas.

El número y los tipos de valores de entrada y salida deben ser compatibles con la función especificada. En la mayoría de los casos, una determinada función primitiva tiene un número fijo de patillas de entrada y de salida.

El propósito de proporcionar funciones primitivas en UML es evitar la selección de ciertas operaciones primitivas como "fundamentales". Cada lenguaje de programación selecciona un conjunto de operaciones primitivas diferente.

Las funciones primitivas están pensadas para especificar las acciones de cómputo empotradas y predefinidas de un sistema, como las aritméticas, las funciones Booleanas o el procesamiento básico de cadenas, cuya reducción a llamadas a procedimiento no sería útil. Se espera que sean definidas mediante constructores de perfiles y no por la mayoría de los modeladores. Una función primitiva no está pensada para utilizarse en lugar de la definición de procedimientos para los cómputos algorítmicos normales.

Acción de difusión de evento. La ejecución de esta acción crea una instancia de una señal especificada, utilizando los valores de entrada de la acción para rellenar los valores de los atributos de la señal, y transmitiendo una copia de la instancia de la señal a un conjunto de objetos destino del sistema dependientes de la implementación. La ejecución de esta acción se completa cuando se han transmitido todas las instancias de la señal (y posiblemente antes de que sean recibidos por sus destinos). La transmisión de cada instancia de la señal y el subsiguiente comportamiento de cada objeto destino se realizan de forma concurrente.

Acción de llamada. Existen dos variedades: una acción de llamada a operación y una acción de llamada a comportamiento.

La ejecución de una acción de llamada a operación es el resultado de la invocación de una operación sobre un objeto. Esta acción especifica una operación. Como entradas tiene una lista de valores de argumentos y un objeto destino. Los valores de entrada deben ser compatibles con los parámetros de la operación y con la clase a la que pertenece la operación. Como salidas, la acción tiene una lista de valores resultado. La acción también tiene un indicador que establece si la llamada es síncrona o asíncrona.

Cuando se ejecuta la acción, los valores de entrada y el tipo de operación conforman un mensaje que se transmite al objeto destino. Si la llamada es asíncrona, la acción está completa y el llamador puede proseguir; si la operación invocada posteriormente intenta devolver valores de retorno, son ignorados. Si la llamada es síncrona, el llamador se bloquea durante la ejecución de la operación.

La recepción del mensaje por parte del objeto destino provoca un efecto sobre el objeto destino, normalmente la ejecución de un método, pero a menudo los efectos se pueden especificar mediante el modelo, como el disparo de una transición de una máquina de estados. El mecanismo que determina qué comportamiento se realiza en base a una operación y a un objeto destino se denomina resolución.

Si la llamada es síncrona, el sistema guarda información suficiente para devolver el control a la ejecución de la acción que realizó la llamada. Esta información de retorno es opaca para el

modelo de usuario y no se puede manipular de ninguna forma. Cuando se completa la ejecución del método llamado, los valores de salida del comportamiento (normalmente una actividad) se incorporan al mensaje de respuesta que se transmite a la ejecución del evento que realizó la llamada.

La acción de llamada a comportamiento es una variante que referencia a una actividad directamente, en lugar de a una operación. Esta variante es similar a una llamada a procedimiento convencional en la que no se necesita una búsqueda de método; la actividad referenciada se ejecuta utilizando los valores de entrada de la acción como argumentos. En otros aspectos, es similar a la acción de llamada a operación.

Acción de clasificación. La acción clasificada determina si un objeto se clasifica mediante un clasificador dado. La acción de reclasificación añade y/o elimina clasificadores de un objeto dado. Estas acciones están pensadas para utilizarse en sistemas que permitan la clasificación dinámica.

Acción de creación. La ejecución de una acción de creación es el resultado de la instanciación de un objeto (*véase* creación). La clase que se instancia es un parámetro de la acción. No hay patillas de entrada. La ejecución de la acción crea una nueva instancia de la clase. La identidad de la instancia se sitúa en la patilla de salida de la acción. El objeto que se crea es un objeto "sin procesar" que no ha sido inicializado; la inicialización se debe modelar independientemente. Sin embargo, se espera que los perfiles de implantación se definan de forma que incluyan varios protocolos de inicialización.

Acción de destrucción. La ejecución de una acción de destrucción es el resultado de la destrucción de un objeto destino. La identidad del objeto es una entrada de la acción. No existen salidas. El resultado de la ejecución de la acción es la destrucción del objeto. Los enlaces implicados en el objeto no son válidos y se deben destruir, pero pueden ser, o no, destruidos automáticamente por una implementación. Hay un indicador en la acción que señala que los enlaces participantes y las partes compuestas del objeto (*véase* composición) son destruidos como parte de la acción; de otra forma su destrucción se debe modelar explícitamente utilizando acciones adicionales. Se espera que se definan perfiles de implementación que incluyan varios protocolos de destrucción.

También hay una acción que explícitamente destruye enlaces y objetos de enlace.

Acción de elevar excepción. La ejecución de una acción de elevar excepción provoca la ocurrencia de una excepción. El valor de la única patilla de entrada es un objeto que se convierte en el valor de la excepción. La elevación de una excepción provoca que la ejecución normal de la acción finalice y comience una búsqueda de la acción y de las actividades que comprende por parte del manejador de excepción, cuyo tipo de excepción incluye a la clase del objeto de excepción. Para más detalles, *véase* manejador de excepción.

Acción de lectura. Hay una familia de acciones de lectura. La ejecución de una acción de lectura lee el valor de una ranura y coloca el valor en la salida de la acción. Dependiendo del tipo específico de acción, la ranura puede ser una variable local de la actividad que la contiene, un atributo de un objeto proporcionado como valor de entrada o el valor de un enlace o de uno de sus atributos. La acción tiene como entradas una expresión para la variable, objeto o enlace destino, así como el nombre de una propiedad del objeto (si es aplicable). Como acción de lectura se modela una declaración de navegación.

Acción de respuesta. Una acción de respuesta transmite un mensaje en respuesta a la recepción previa de una acción de aceptación de llamada. La salida de la acción de aceptación de lla-

mada proporciona la información necesaria para descifrar el envió de la acción de aceptación de llamada; esa información no se puede utilizar de ninguna otra forma. La acción de respuesta puede tener una lista de valores de respuesta que se transmiten como parte del mensaje de respuesta. Después de ejecutar una acción de respuesta, continúa la ejecución del hilo actual. Mientras tanto, se transmite el mensaje de respuesta a la ejecución de la acción de llamada que disparó la acción de aceptación de llamada por el objeto actual. Normalmente esta ejecución pertenece a diferentes objetos. Cuando el llamador recibe el mensaje de respuesta, se trata como un retorno normal desde una operación llamada, y continúa la ejecución del llamador.

Se debe realizar una respuesta a una llamada aceptada. Intentar responder dos veces a la misma llamada es un error semántico. Incluso si no se realiza una respuesta, el llamador se colgaría para siempre (asumiendo que no se produjeran interrupciones).

Acción de lectura de colección. La ejecución de una acción de lectura de colección produce un conjunto que contiene todas las instancias de un clasificador especificado en una única patilla de salida. El conjunto de instancias producidas puede depender de consideraciones de implementación, y una colección puede soportar múltiples colecciones. Algunas implementaciones pueden elegir no soportarlas. Por su naturaleza, esta acción depende de la suposición de que el sistema tiene una frontera bien definida, dentro de la cual los objetos se pueden recuperar sistemáticamente. Es más apropiado en sistemas cerrados en los que hay un registro central de instancias, como en sistemas de bases de datos. Es menos probable que sea útil en sistemas altamente distribuidos con fronteras abiertas.

Acción de retorno. No hay una acción explícita para el retorno de una llamada normal. Cuando se completa la ejecución de una actividad a la que se ha llamado, sus valores de salida se empaquetan en un mensaje de respuesta que se transmiten a la ejecución que llamó a la actividad. La recepción de este mensaje desbloquea a la ejecución llamante. La declaración de retorno, en los lenguajes convencionales, se modela mediante el uso de construcciones de ruptura para devolver el control al nivel superior de la actividad en el momento en el que se alcanza el final de la actividad.

Para el retorno desde una llamada que se ha capturado asíncronamente mediante una acción de aceptación, se utiliza acción de respuesta explícita. En UML, tanto la recepción de una llamada, como su respuesta, son ambas implícitas (como en una actividad ordinaria que sirve como método) o ambas explícitas.

Acción de envío. Una acción de envío crea un mensaje de objeto y lo transmite a un objeto destino, donde se puede disparar un comportamiento. Hay dos variantes: La acción envío de objeto tiene dos patillas de entrada, una para el objeto destino y otra para el mensaje de objeto. Se transmite una copia del objeto en la entrada del mensaje de objeto al objeto destino. La acción de envío de señal especifica una clase como tipo de señal. La acción tiene una patilla de entrada para el objeto destino más una lista de patillas de entrada con el mismo número de patillas que el número de atributos del tipo de señal. La ejecución de esta acción crea un mensaje de objeto del tipo de la señal dada, utilizando los valores de las patillas de entrada como valores de los atributos. Por lo demás, ambas variantes se comportan igual.

Se transmite una copia del mensaje de objeto al objeto destino. La identidad del mensaje de objeto no se conserva en la transmisión. La ejecución del emisor mantiene su propio hilo de control y sigue concurrentemente con la transmisión del mensaje y con cualquier efecto provocado; el envío de un mensaje es asíncrono. Si se dispara una actividad síncrona por la recepción de un mensaje, cualquier intento de responder, devolver el control o devolver valores se ignora silenciosamente. El emisor no tiene posteriores conexiones con la señal enviada o con sus efectos.

Acción de inicio de comportamiento propio. Esta acción tiene como único valor de entrada el objeto destino. La ejecución de esta acción causa que comience el comportamiento propio del objeto destino. El comportamiento propio puede ser una máquina de estados o una actividad. Esta es una acción de muy bajo nivel que expone el motor de implementación. Muchos modelos evitarán esta acción, asumiendo simplemente que el comportamiento propio comienza automáticamente cuando se crea el objeto. Esta elección es un punto de variación semántica. Los lenguajes de programación tienen una amplia variedad de semánticas de inicialización, por lo que no se incluye en UML una semántica de inicialización de alto nivel.

Acción de prueba de identidad. Esta acción compara la identidad de dos objetos de entrada y da como salida un valor Booleano que indica si son el mismo objeto.

Acción de tiempo. Hay acciones que devuelven el momento actual y devuelven la duración entre el envío y la recepción de un mensaje.

Acción de escritura. Una acción de escritura establece el valor de una ranura con un valor de ejecución recibido como entrada de la acción. Dependiendo del tipo específico de la acción, la ranura se puede corresponder con una variable local de la actividad que la contiene, un atributo de la clase de un objeto proporcionado como valor de entrada o el valor de un enlace o de uno de sus atributos. La acción tiene como entradas una expresión para una variable, objeto o enlace destino; el nombre de la propiedad del objeto (si es aplicable); y un valor que se asigna a la ranura. Una acción de escritura se modela como una declaración de asignación. Para ranuras con múltiples valores, las acciones de escritura tienen variaciones para añadir o eliminar valores individuales, y para reemplazar o limpiar todos los valores.

Notación

Una acción se muestra como parte de un diagrama de actividad. Una acción se representa como un rectángulo con las esquinas redondeadas (Figura 14.4). El nombre de la acción o su descripción de texto aparece dentro del rectángulo redondeado. Además de los tipos predefinidos de acciones, los modelos del mundo real y los modelos de alto nivel pueden utilizar acciones para representar efectos del mundo real o el comportamiento informal del sistema, simplemente utilizando palabras para nombrar a la acción.

Las flechas de flujo se dibujan hacia y desde el borde del rectángulo. Las patillas de entrada y de salida se pueden mostrar mediante cuadrados pequeños sobre el borde del rectángulo, en cuyo caso, las flechas de flujo conectan los cuadrados que representan las patillas.

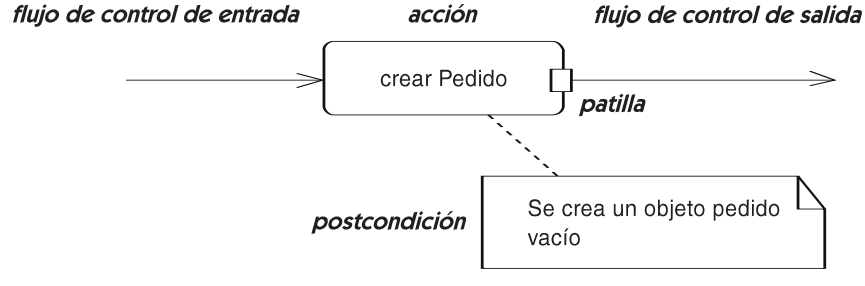


Figura 14.4 Acción

Si una implementación utiliza un determinado lenguaje de programación o lenguaje de acción, las acciones se pueden expresar mediante la sintaxis del lenguaje, como se especifica en un perfil de implementación. La especificación de UML no incluye una sintaxis de texto predefinida para los diversos tipos de acciones.

Las precondiciones y las postcondiciones se pueden mostrar mediante símbolos de nota (rectángulos con una esquina doblada) conectados al símbolo de acción mediante una línea discontinua.

Algunas acciones de comunicación tienen una notación especial; como la acción de aceptación de evento, la acción de aceptación de tiempo de evento y la acción de envío de señal. La Figura 14.5 muestra algunas de ellas.

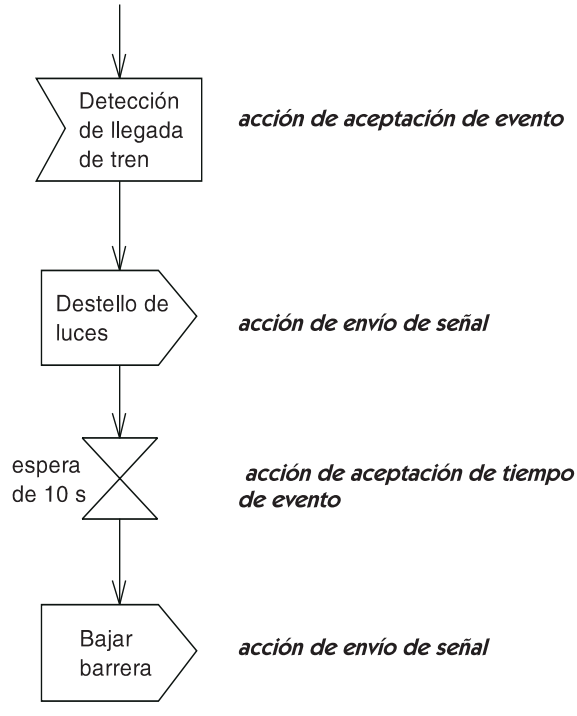


Figura 14.5 Notación especial para las acciones de comunicación

UML no define una notación para la mayoría de las acciones. El modelador especifica una notación para una acción utilizando una caja redondeada, en la que se escribe la acción con la notación de algún lenguaje o pseudolenguaje elegido.

Historia

En UML1, las acciones incluían en su estructura algunos mecanismos de flujo de control, como expresiones de repetición y expresiones de conjunto de objetos, incrementando enormemente su complejidad. En UML2, las acciones se han vuelto a posicionar como nodos de actividad primitiva, en los que el flujo de control y los mecanismos de alto nivel se han movido a los modelos de activi-

dad. En UML 1.1 y UML 1.3, todas las acciones se consideraban como invocaciones enviadas a objetos. En UML 1.5 se introdujo un modelo completo de acción, pero no se utilizó ampliamente porque se iba a sustituir pronto por UML 2.0. En UML2 las invocaciones se han distinguido de otros tipos de acciones, simplificando la especificación de cada tipo de acción. UML 1.1 y UML 1.3 incluyeron una amplia selección de acciones con algunas omisiones curiosas. UML 1.5 incluye un conjunto de acciones más balanceado. UML2 intenta incluir un completo conjunto base de acciones.

Discusión

Todos los lenguajes de programación tienen diferentes conjuntos de acciones base construidos dentro de sus definiciones y estilos de uso, y es imposible incluirlas todas sin conflictos en la semántica. La especificación de UML define un conjunto de acciones de bastante bajo nivel suficiente para describir el comportamiento. La semántica de alto nivel de efectos complejos, como la inicialización, la destrucción en cascada y los mecanismos de control reflectivo, fueron evitados en la semántica de UML y abandonados en los perfiles de implementación. Esta decisión fue el resultado de un compromiso entre la precisión deseada y la necesidad de los desarrolladores de trabajar con varios lenguajes destino, los cuales tienen un amplio rengo de conceptos semánticos. Hay mucha más variación en la semántica de ejecución entre lenguajes de programación, que entre estructuras de datos o en el conjunto de construcciones de control disponibles. Las pequeñas diferencias entre lenguajes son difíciles de plasmar de una forma práctica, a pesar de que, en teoría, sea posible. Por tanto, la selección de un lenguaje de programación como base para un lenguaje de acción tendría el efecto de desanimar a los otros, cosa que los diseñadores no querían hacer. Por lo tanto, la semántica de las acciones se ha dejado a bajo nivel y libre de preocupaciones de implementación dentro del propio UML. Para muchos usos prácticos, como la generación de código, UML debe ser aumentado con el lenguaje de acción (a menudo un lenguaje de programación estándar) que se está utilizando. Algunos críticos se han quejado de que UML es impreciso debido a su libertad, pero es impreciso sólo en la medida en la que el lenguaje de acción elegido sea impreciso. El defecto real es que UML no impone una lengua franca de acciones y otras expresiones, pero esto es muy difícil en el actual mundo políglota de la informática, a pesar de la emocional súplica de dicha unidad.

Hay una irregularidad en la selección de las acciones de UML. Algunas acciones, como acción de comienzo de comportamiento propio, se podrían haber dejado mejor para los perfiles de implementación, dado que tienen un aroma indiscutible a implementación.

acción asíncrona

Una petición en la cual el objeto emisor no realiza una pausa en espera de los resultados. Un envío.

Véase enviar, acción síncrona.

acción de aceptación

Una acción cuya ejecución se bloquea hasta que el objeto en ejecución reconoce un tipo de evento especificado. Una versión especializada espera la recepción de una llamada de una operación especificada.

Véase acción para más detalles.

Historia

Esta acción es nueva en UML2.

acción de aplicar función

Una acción cuya ejecución evalúa una función matemática predefinida para calcular uno o más valores de salida. Es una función de evaluación pura y no tiene efectos laterales o interacción con el estado del sistema, excepto a través de los valores de entrada y salida.

Véase acción para más detalles.

acción de clasificación

Una acción cuya ejecución determina si un objeto se clasifica mediante una clase dada.

Véase acción.

acción de creación

Acción cuya ejecución crea un nuevo objeto de un tipo dado. Las propiedades del objeto no son inicializadas por la acción.

Véase acción.

acción de elevar excepción

Una acción cuya ejecución produce una excepción de un tipo dado. Como resultado de la excepción, la ejecución de la acción se termina y se inicia el mecanismo manejo de la excepción.

Véase acción, excepción, manejador de la excepción.

acción de escritura

Hay una familia de acciones que modifican los valores de varias ranuras, incluyendo atributos, extremos de asociación, calificadores y variables.

Véase acción.

acción de inicio de comportamiento propio

Una acción cuya ejecución inicia la ejecución del comportamiento principal anexo a un objeto. *Véase* acción.

acción de lectura

Una familia completa de acciones cuya ejecución completa va encaminada a la salida del valor de un atributo, extremo de asociación, calificador, magnitud, u otro valor.

Véase acción.

acción de prueba de identidad

Una acción cuya ejecución prueba si dos valores de entrada tienen la misma identidad, eso es, si son el mismo objeto.

Véase acción.

acción de recepción

Véase acción de aceptación.

acción de respuesta

Una acción cuya ejecución pasa los valores y restaura el control a la ejecución de una acción de llamada anterior.

Véase acción.

acción de tiempo

Hay varias acciones relacionadas con modelo de tiempo proporcionado por UML. Modelos más complicados de tiempo, como los de tiempo real, requieren perfiles de UML.

Véase acción.

acción de vigilancia de duración

Acción que devuelve el valor de una duración en tiempo de ejecución.

Notación

La duración de la transmisión de un mensaje se puede representar en un mensaje:

nombre = duración

La Figura 14.6 muestra un ejemplo.

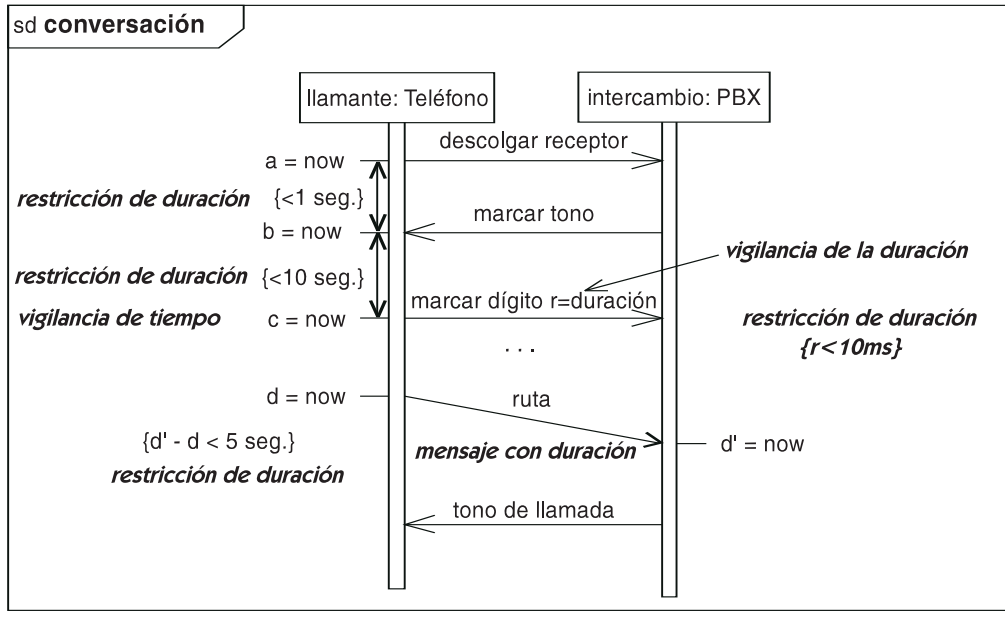


Figura 14.6 Restricción de duración

Discusión

Aunque en la especificación de UML está definida como una acción, en la mayoría de los casos los modeladores querrán hacer aserciones sobre tiempo sin ejecutar acciones en tiempo de ejecución. Probablemente es mejor considerarla simplemente como la definición de un nombre que puede ser utilizado en una expresión de restricción, más que como una acción real.

El modelo básico de interacción asume que el tiempo en líneas de vida diferentes es independiente (una asunción "Einsteniana"). Sin embargo, medir la duración de la transmisión de un mensaje requiere que la escala de tiempo de las dos líneas de vida sea la misma. En la mayoría de los casos en que la expedición de dicha transmisión del mensaje es un tema de modelado, se puede asumir algún tipo de escala de tiempo global. En aquellos casos en los que es importante la utilización de tiempos procedentes de relojes distintos, el modelo sencillo de tiempos de UML no es adecuado y se debe usar un modelo de tiempos más elaborado. Se espera que los perfiles de tiempo real proporcionen varios modelos de tiempo.

acción de vigilancia de tiempo

Acción que devuelve el tiempo actual.

Semántica

La acción devuelve el tiempo actual.

Notación

Se puede utilizar la siguiente sintaxis en un diagrama de secuencia:

`nombre` = now

La Figura 14.264 muestra un ejemplo.

Discusión

A diferencia de la acción de vigilancia de duración, la acción de vigilancia de tiempo toma sentido como una acción que podría implementarse y utilizarse en programas. Si embargo, en la especificación de UML, se formula de una forma inconsistente con otras acciones, en la que incluye tanto una observación de tiempo como una acción de asignación. Es probablemente mejor para los implementadores considerar esto como un error y hacer que simplemente produzca como salida un valor que se puede asignar utilizando las acciones de escritura normales.

En un sistema sencillo, todos los relojes se pueden considerar idénticos. En un sistema de tiempo real más realista, los relojes diferirán y una vigilancia de tiempo debe contener la identidad del reloj utilizado. Para modelar dichos sistemas, el modelo simple de UML no es adecuado y se debe usar un modelo de tiempo real más robusto.

La especificación de UML no define una vigilancia de tiempo para modelar propósitos, es decir, una que no esté pensada para ser ejecutada por un programa de usuario, sino que esté pensada para ser utilizada para hacer aserciones por el modelador. Recomendamos la utilización de marcas de tiempo para adjuntar nombres a los tiempos en que sucede un evento. Esto no es parte de la especificación oficial de UML, pero debería serlo.

Historia

El modelo de tiempos de UML2 ha perdido algunas capacidades respecto al modelo de UML1, como las marcas de tiempo. Con un poco de suerte esta capacidad se restaurará en el futuro.

acción síncrona

Una solicitud, en la que el objeto llamante se detiene para esperar la respuesta; una llamada.

Contraste: acción asíncrona.

activación

Término de UML1 reemplazado por especificación de una ejecución.

actividad

Una especificación de un comportamiento ejecutable como la ejecución coordinada secuencial y concurrente de unidades subordinadas, incluyendo actividades anidadas y, en última instancia,

acciones individuales conectadas por flujos desde las salidas de un nodo a las entradas de otro nodo. Las actividades pueden ser invocadas por acciones y como integrantes de otros comportamientos, como transiciones de máquinas de estados. Las actividades se muestran como diagramas de actividad.

Véase también máquina de estados.

Semántica

Una actividad es una especificación de comportamiento que describe los pasos secuenciales y concurrentes de un procedimiento de cómputo. Los flujos de trabajo, algoritmos y el código de computadora son ejemplos de procedimientos que a menudo se modelan como actividades. Las actividades se centran en el proceso de cómputo en lugar de en los objetos realizando un cómputo o en los valores de los datos involucrados, aunque estos se pueden representar como parte de una actividad. Las máquinas de estados y las actividades son similares, en tanto que ambas describen secuencias de estados que ocurren a lo largo del tiempo, y las condiciones que causan los cambios entre los estados. La diferencia es que una máquina de estados se preocupa de los estados de un objeto que realizan o se someten a un cómputo, mientras que una actividad se preocupa de los propios estados de cómputo, posiblemente a lo largo de varios objetos, y modela explícitamente el flujo de control y la información entre nodos de actividad.

Estructura. Una actividad se modela como un grafo de nodos de actividad conectados por flujos de control y de objetos. Los nodos de actividad representan actividades anidadas, acciones, ubicación de datos y construcciones de control. Las acciones modelan efectos sobre el sistema. Los flujos entre nodos modelan el flujo de control y datos dentro de la actividad. Los nodos de construcción de control proporcionan formas más complicadas de especificar el flujo de control. El flujo de control se puede implementar de varias formas, incluyendo los cambios de estado de los objetos y los mensajes transmitidos entre objetos.

Una actividad tiene parámetros de entrada y de salida que representan los valores que, en una ejecución, se le proporcionan y produce una actividad. Una actividad se puede vincular a varios comportamientos cono una pieza de comportamiento parametrizado. Por ejemplo, una actividad se puede vincular a una transición de una máquina de estados como un efecto; como entrada, salida o presencia en un estado; como un método de una operación; como implementación de un caso de uso; y como una invocación dentro de otra actividad. Cada construcción específica proporciona una sintaxis para proporcionar los parámetros de entrada y de salida que se ligan a cada uno de los parámetros de la actividad. Cuando se ejecuta uno de los constructores que invoca a una actividad, se crea una ejecución de la actividad y se inicializa mediante la copia de los argumentos de entrada. Cuando se completa la ejecución de la actividad, los valores resultantes de la actividad se convierten en los valores de salida del constructor que la invocó.

Una actividad se puede marcar como de sólo lectura. Esto es una afirmación de que su ejecución no modificará cualquier variable u objeto fuera de la propia actividad, excepto para posibles variables temporales que desaparecen antes de la finalización de la ejecución. Este tipo de afirmación puede ser útil para los generadores de código, pero puede ser difícil o imposible realizar la comprobación automáticamente, por lo que el modelador debe tener cuidado.

Semántica de ejecución. La estructura de grafo de las actividades y su semántica está basada en parte en las redes de Petri, un área importante en la teoría de la computación desde los años sesenta. Cada actividad es un grafo de nodos conectados por flujos dirigidos. Los nodos incluyen

acciones, actividades anidadas, ubicación de los datos y varios constructores de control. Los flujos representan el flujo de información dentro de la ejecución de una actividad, incluyendo el flujo de control y el flujo de datos.

Una ejecución de una actividad se puede imaginar como el proceso de una copia del grafo de actividad que contiene tokens. Un token es una marca que indica la presencia de un control o un dato en el punto del cómputo representado por un nodo o flujo. Los tokens de control no tienen estructura interna. Los tokens de datos contienen valores que representan resultados intermedios en ese punto del cómputo. El modelo es similar a los diagramas de flujo tradicionales, pero los grafos de actividad pueden contener subcómputos concurrentes, por lo que pueden tener múltiples tokens en un momento dado.

La regla de cómputo básico es sencilla: Se permite la ejecución de un nodo de acción si tiene presentes tokens en sus flujos de entrada (o en nodos de datos en los extremos del origen de los flujos de entrada). En otras palabras, un flujo de entrada indica una dependencia en la finalización de un paso previo de la actividad, y un token en el flujo de entrada indica que el paso previo se ha completado. Cuando se permite que un nodo se ejecute, puede empezar su ejecución. Si también se permite la ejecución de otros nodos que involucran a los mismos tokens, sólo se ejecuta uno de ellos en una elección no determinista. Cuando un nodo comienza su ejecución, consume sus tokens de entrada; es decir, son eliminados de los flujos de entrada y sus valores se vuelven disponibles dentro de la ejecución. El consumo de los tokens evita que otro nodo en ejecución pueda utilizar los mismos tokens. Cada tipo de nodo tiene sus propias reglas de ejecución, descritas en la entrada del glosario para ese nodo. Cuando se completa la ejecución de un nodo, se producen los tokens de salida y se colocan en los flujos de salida del nodo. Los tokens pueden permitir la ejecución de otros nodos.

Las acciones necesitan tokens en todas sus entradas para producir tokens en todas las salidas. Otros elementos de actividad proporcionan varios comportamientos del flujo de control, como las decisiones, bucles, divisiones y uniones del control, ejecución paralela, etcétera. En general un nodo de actividad comienza su ejecución cuando los tokens están presentes en un conjunto especificado de sus nodos de entrada, se ejecuta concurrentemente con otros nodos y produce tokens en un conjunto especificado de sus nodos de salida. Muchos tipos de nodos de actividad necesitan tokens en un subconjunto de entradas para producir tokens en un subconjunto de salidas. También hay tipos avanzados de acciones que permiten la ejecución basada en un subconjunto de entradas.

Las reglas de ejecución son fundamentalmente concurrentes. Si dos nodos de actividad son habilitados por diferentes conjuntos de tokens, se pueden ejecutar concurrentemente. Esto significa que no hay un orden inherente de ejecución entre nodos. Esto no significa que la implementación deba ser paralela o que involucre a varias unidades de cómputo, aunque podría ser así. Tampoco significa que la implementación deba ser una secuencia de ejecuciones en un determinado orden, incluido en paralelo, sin alterar la corrección del cómputo. La ejecución secuencial es fundamentalmente determinista, pero la ejecución concurrente puede introducir indeterminación, lo que puede ser una parte importante en algunos modelos.

Contexto. Una actividad se ejecuta dentro de un contexto. El contexto incluye sus valores de los parámetros de entrada, el propio objeto sobre el que se ejecuta y su lugar en la jerarquía de invocación de ejecución. Los nodos en la actividad tienen acceso a los parámetros de entrada y a los atributos y asociaciones del propio objeto. Si una actividad se vincula directamente a un clasificador, comienza su ejecución cuando se crea una instancia del clasificador y termina cuan-

do se destruye. Si una actividad se vincula a un método de una operación, comienza su ejecución cuando se invoca a la operación.

Excepciones. Una excepción es un tipo de ocurrencia que para la secuencia de ejecución normal e inicia una búsqueda de un manejador de excepción en el nodo en ejecución actual o en un nodo dentro del que se encuentre la unidad. El manejador de ejecución se debe declarar para que maneje el tipo de excepción. El nodo que posee el manejador de ejecución se denomina nodo protegido. Toda actividad contenida dentro del nodo protegido termina, y el manejador de ejecución se ejecuta utilizando los parámetros de la excepción como entradas. Cuando el manejador de ejecución finaliza su ejecución, la ejecución se reanuda como si el nodo protegido hubiera completado su ejecución. En la mayoría de los casos, el nodo que eleva la excepción está anidado (en algún nivel) dentro del nodo protegido.

Interrupciones. Una región de actividad interrumpible es una región de la que sale una o más transiciones de actividad interrumpibles. Si una transición de actividad interrumpible pasa un token, terminan todas las actividades dentro de la región.

Múltiples tokens. La forma más simple y habitual de actividad permite al menos un único token en cualquier transición. Una variante más complicada permite múltiples tokens en transiciones y buffers (nodo objeto que recopila y posibilita secuencias de tokens). También es posible distinguir actividades en las que una única ejecución maneja todas las invocaciones de las actividades más habituales en las que cada invocación crea ejecuciones independientes. Una actividad de ejecución única es parecida a las funciones estáticas de la familia del lenguaje C.

Flujo de objetos. Algunas veces es útil ver las relaciones entre un nodo y los objetos que son sus argumentos o resultados. Cada grupo representa alguna partición con significado de las responsabilidades de las actividades —por ejemplo, la organización responsable de un determinado paso de un flujo de trabajo o el objeto que ejecuta nodos dentro de una actividad. Debido a su notación gráfica, los grupos se denominan calles[1].

Notación

Una actividad se representa en un diagrama de actividad. La definición de una actividad se muestra como un gran borde con las esquinas redondeadas que contiene un grafo con símbolos de nodo y flechas de flujo que representan la descomposición de una actividad en sus componentes. El nombre de la actividad normalmente se representa en la esquina superior izquierda. Las precondiciones y postcondiciones se muestran como cadenas de texto precedidas de las palabras clave **«precondition»** y **«postcondition»**. Los parámetros de entrada y de salida de los nodos se dibujan con pequeños rectángulos sobre el borde; el nombre de los parámetros se puede colocar dentro de los rectángulos. Un parámetro también se puede listar bajo el nombre de la actividad con el formato **nombre: Tipo**.

Los diagramas de actividad incluyen varios símbolos de nodo: acciones, decisiones, divisiones, uniones, fusiones y nodos objeto. Una flecha representa tanto los flujos de control como los flujos de objetos; un flujo de objetos debe conectar un nodo objeto a uno o dos extremos, por lo que la distinción está clara por el contexto. El interior de una definición de actividad se puede dividir en particiones mediante líneas gruesas. *Véanse* varias entradas para más detalles.

---

[1] Se alude al término inglés *swimlane*, que hace referencia a las calles de una piscina. (*N. del T.*)

Véase nodo de control para ver algunos símbolos opcionales que pueden ser útiles en diagramas de actividad.

La palabra clave **«singleExecution»** se coloca dentro de una actividad que tiene una única ejecución compartida.

Ejemplo

La Figura 14.7 muestra un flujo de actividades involucradas en el procesamiento de un pedido en una taquilla de un teatro. Incluye una bifurcación y la subsiguiente fusión basada en si el pedido es para una suscripción o para una entrada individual. La división inicia actividades concurrentes que lógicamente ocurren al mismo tiempo. Su ejecución real puede estar, o no, solapada. La concurrencia termina con la correspondiente unión. Si sólo está involucrada una persona, las actividades concurrentes se pueden realizar en cualquier orden (asumiendo que no se pueden hacer simultáneamente, lo que está permitido en el modelo, pero que puede ser difícil en la prác-

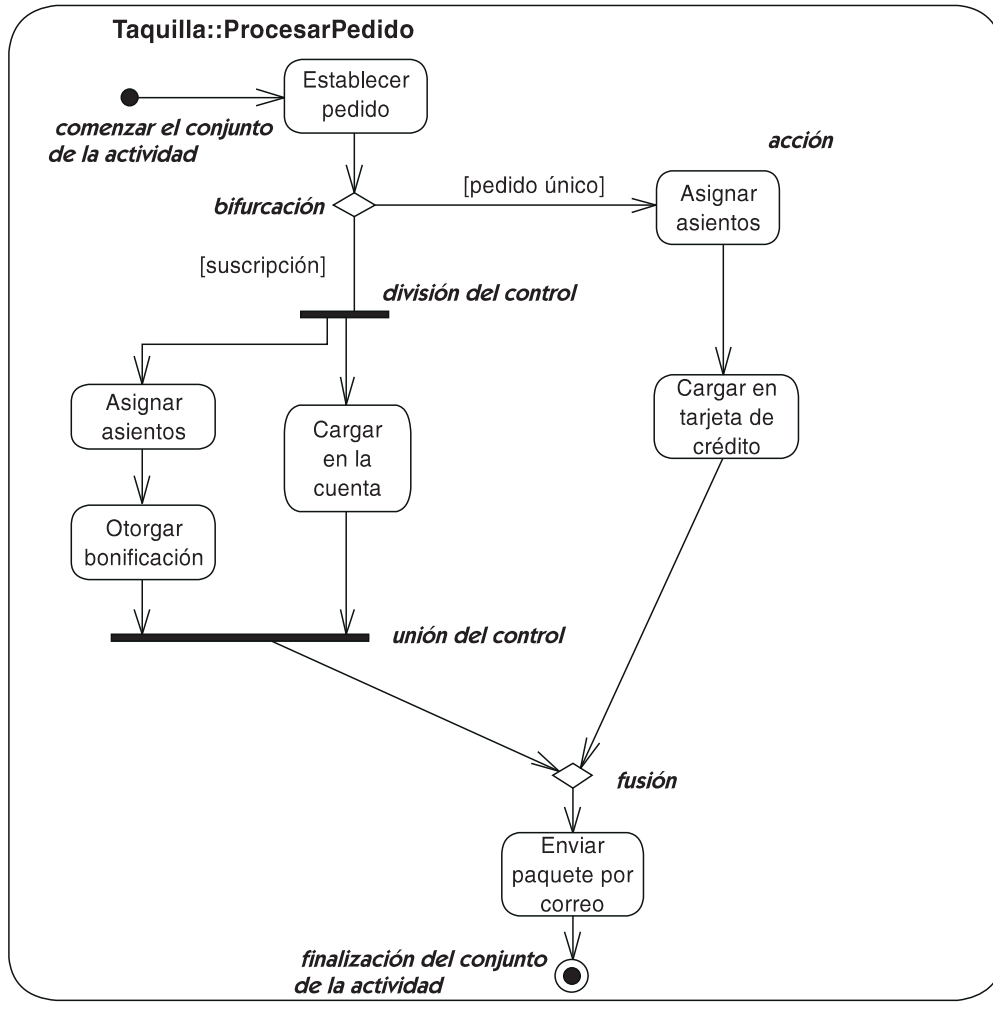


Figura 14.7 Diagrama de actividad

tica). Por ejemplo, el personal de la taquilla puede asignar asientos, otorgar la bonificación y cargar en la cuenta; o podrían otorgar la bonificación, asignar asientos y cargar en la cuenta —pero no pueden cargar en la cuenta hasta después de que se hayan asignado los asientos.

Particiones. Las actividades en un grafo de actividad se pueden dividir en regiones en un diagrama, que se denominan calles por su apariencia visual, separadas por líneas continuas. Una partición es una unidad organizacional para los contenidos de un grafo de actividad. A menudo cada partición representa una unidad organizacional dentro de una organización del mundo real. Las particiones también se pueden utilizar para especificar las clases responsables de implementar las acciones incluidas dentro de cada partición.

Ejemplo

En la Figura 14.8, las actividades están divididas en tres particiones, en la que cada una se corresponde con un responsable distinto. No hay un requisito de UML que indique que cada partición se corresponde con un objeto, aunque en este ejemplo hay clases obvias que podrían recaer en cada partición, y estas clases podrían realizar las operaciones para implementar cada actividad en el modelo final.

La figura también muestra el uso del símbolo del flujo de objetos. Los flujos de objetos se corresponden cono los distintos estados de un objeto pedido en su funcionamiento por el camino a través de la red de actividades. Por ejemplo, el símbolo **Pedido[Realizado]**, significa que en ese lugar en el cómputo, un pedido ha avanzado hasta el estado **Realizado** dentro de la actividad **Solicitud de Servicio**, pero todavía no ha sido consumido por la actividad **Tomar Pedido**. Después de que se completa la actividad **Tomar Pedido** el pedido está en el estado **Introducido**, como muestra el símbolo del flujo de objetos en la salida de la actividad **Tomar Pedido**. Todos los flujos de objetos de este ejemplo representan el mismo objeto en diferentes momentos de su vida. Dado que representan el mismo objeto, no pueden existir al mismo tiempo. Se puede representar un camino de control a través de todos ellos, como queda claro en el diagrama.

Flujo de objetos. Los objetos que son entradas a una acción o salidas de una acción se pueden mostrar mediante símbolos de objetos. El símbolo representa al objeto en un momento del cómputo en el que es apropiado como entrada o acaba de ser producido como salida (normalmente un objeto hace ambos). Una flecha de flujo de objeto se representa desde un nodo objeto a un nodo de actividad que utiliza el objeto como una de sus entradas. El mismo objeto puede ser (y normalmente lo es) la salida de un nodo y la entrada de uno o más nodos.

Las flechas de control de flujo se pueden omitir cuando las flechas de flujo de objetos proporcionan restricciones redundantes. En otras palabras, cuando una acción produce una salida que es la entrada de la siguiente acción, esa relación de flujo de objetos implica una restricción de control.

Clase en un estado. Con frecuencia, el mismo objeto es manipulado por varias actividades consecutivas que cambian su estado. Para una mayor precisión, el objeto se puede representar varias veces en un diagrama, en el que cada apariencia denota un estado diferente durante su vida. Para distinguir las distintas apariciones del mismo objeto, el estado del objeto en cada punto se puede colocar entre corchetes a continuación del nombre de la clase —**OrdenDeCompra[aprobada]**.

Véase también nodo de control para comprobar que otros símbolos se pueden utilizar en un diagrama de actividad.

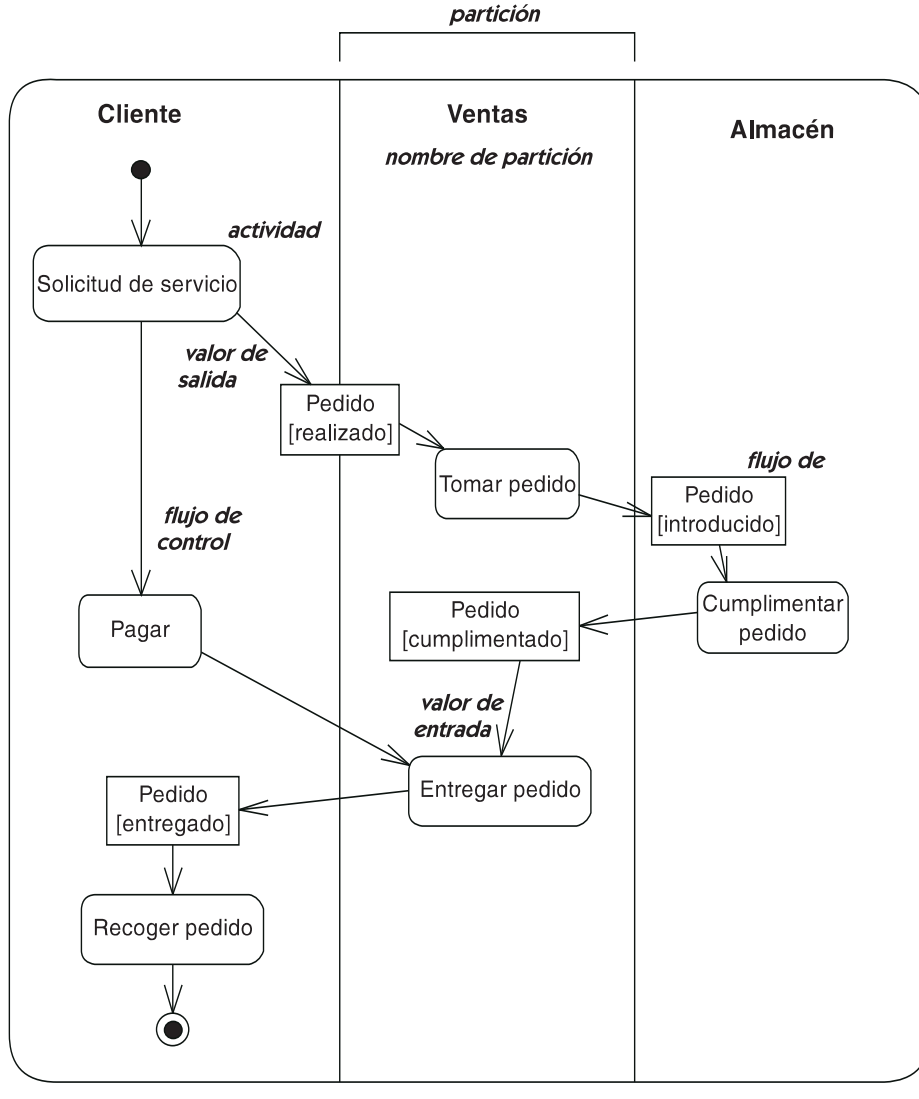


Figura 14.8 Diagrama de actividad con calles

Región de expansión. Una región de expansión es la expansión de un cómputo que contiene un valor múltiple dentro de un conjunto de cómputos que se ejecutan en paralelo (Figura 14.9). Esto indica que múltiples copias de una actividad pueden darse concurrentemente. Cada entrada a la región de expansión recibe una colección de valores, mostrados mediante un icono de una caja segmentada. Si hay múltiples entradas, todas las colecciones deben ser del mismo tamaño. Las entradas escalares también están permitidas. La ejecución de una región de expansión se realiza por cada elemento de la colección. Para cada posición de salida de la región de expansión, los valores de salida de todas las ejecuciones se agrupan en una única colección de valores. Una región de expansión representa al constructor “para-todos”.

Como se muestra en el ejemplo, además de la ejecución paralela, la ejecución puede ser iterativa (secuencial) o de flujo (múltiples tokens en una tubería en una única ejecución).

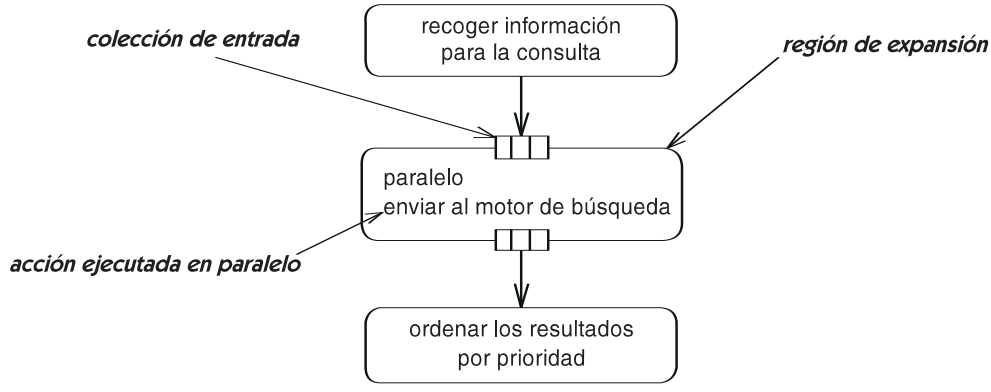


Figura 14.9 Región de expansión

Historia

El modelado de actividad ha experimentado una gran expansión en UML2, con muchos constructores adicionales y la incorporación de opciones cómodas motivadas por la comunidad del modelado de negocio. En UML1, las actividades se han considerado variantes de las máquinas de estados, lo que asegura una buena formación, pero impone severas restricciones en su formato. En UML2, se separaron los metamodelos y la semántica de la actividad se basó (débilmente) en la semántica de las redes de Petri. Además, la sintaxis de la composición de actividad se ha suavizado bastante, permitiendo un formato mucho más libre pero que permite construir con facilidad modelos mal formados.

Discusión

Mucha de la notación opcional que se ha añadido en UML2 está basada en la notación utilizada en varias aproximaciones de modelado de procesos de negocio (aunque no hay un único estándar). Un modelador debe entender a la audiencia a la que se destina el modelo antes de decidir si utiliza ésta y otra notación opcional orientada hacia una comunidad particular.

actividad entrar

Acción realizada cuando se entra en un estado.

Véase también actividad salir, ejecutar hasta finalizar, máquina de estados, transición.

Semántica

Un estado puede tener opcionalmente una actividad entrar asociada. Siempre que se entre en el estado de la forma que sea, se ejecuta la actividad entrar después de cualesquiera actividades asociadas a los estados o transiciones externos y antes de cualesquiera actividades asociadas a los estados internos. La actividad entrar no puede ser eludida de ninguna forma. Se garantiza que será ejecutada siempre que se active el estado que la posee o cualquier estado anidado dentro de él.

Orden de ejecución. En una transición entre dos estados con actividades entrar y salir en la que la transición también tiene una actividad adjunta, el orden de ejecución es: Se ejecutan cualesquiera actividades salir en el estado origen y en los estados que lo engloban, pero sin incluir el estado que engloba tanto al estado origen como al estado destino. Después se ejecuta la actividad adjunta a la transición, después de la cual se ejecutan las actividades entrar (las más extremas primero) en los estados englobados dentro del estado común, hasta el estado destino, incluido. La Figura 14.82 muestra algunas transiciones que tienen varias actividades.

Notación

Una actividad entrar se codifica utilizando la sintaxis de una transición interna con el evidente nombre de evento **entry** (que por tanto es una palabra reservada y no se puede utilizar como un nombre de evento real).

entry / **`actividad`**

Un estado sólo puede tener asociada una actividad entrar, pero la actividad puede ser una secuencia de forma que no se pierde generalidad.

Discusión

Las actividades entrar y salir no son semánticamente esenciales (la acción entrar podría estar adjunta a todas las transiciones de entrada) pero facilitan la encapsulación de un estado de forma que su uso externo se puede separar desde su construcción interna. Eso hace posible definir actividades de inicialización y de terminación, sin importar que se pudieran evitar. Son particularmente útiles con excepciones, puesto que definen actividades que deben realizarse incluso si se produce una excepción.

Una actividad entrar es útil para realizar una inicialización que deba hacerse cuando se entra por primera vez en un estado. Una de sus utilidades es inicializar variables que capturen información acumulada durante un estado. Por ejemplo, una interfaz de usuario que permita la entrada por teclado de un número de teléfono o de un número de cuenta podría borrar el número a su entrada. Restablecer un contador de error, como el número de errores en la introducción de una contraseña podría ser otro ejemplo. Otro uso para una actividad entrar sería asignar almacenamiento temporal necesitado durante el estado.

A menudo las actividades entrar y salir son utilizadas juntas. La actividad entrar asigna recursos, y la actividad salir los libera. Incluso si se produce una transición externa, los recursos se liberan. Esta es una buena forma de manejar errores de usuario y excepciones. Los errores a nivel de usuario desencadenan transiciones de nivel superior que abortan estados anidados, pero los estados anidados tienen una oportunidad de limpieza antes de que pierdan el control.

Historia

Se ha renombrado de *acción* a *actividad*, con la interpretación de que su ejecución puede requerir tiempo, pero por lo demás no ha cambiado mucho desde UML1.

actividad hacer

Ejecución en curso de un comportamiento que se produce dentro de un estado. Contrastar con: efecto, actividad entrar, actividad salir.

Véase también transición de finalización, estado.

Semántica

Una actividad hacer es la ejecución de comportamiento, dentro de una máquina de estados, basado en mantener un estado determinado. La actividad empieza cuando se entra en el estado y continúa hasta que la actividad se completa o se sale del estado. Una transición que fuerce la salida del estado aborta la actividad hacer. Una actividad hacer no se termina por el disparo de una transición interna, puesto que no hay cambio de estado. La acción de la transición interna puede terminarla de forma explícita.

Una actividad hacer es una excepción a la semántica normal de ejecutar hasta finalizar de una máquina de estados, puesto que puede continuar la ejecución incluso cuando la ejecución directa de un efecto se ha completado y la máquina de estados maneja otro evento.

Notación

Una actividad hacer utiliza la notación de una transición interna con la palabra reservada **do** en lugar del nombre de evento:

do/ **`expresión de actividad`**

Ejemplo

La Figura 14.10 muestra un sistema de alarma que ilustra la diferencia entre el efecto de la transición y una actividad hacer. Cuando se produce el evento **detectar intrusión**, el sistema dispara una transición. Como parte del disparo de la transición, se produce el efecto **llamar a la policía.**

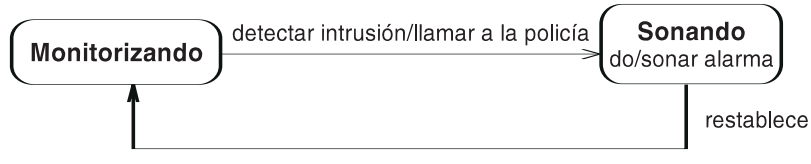


Figura 14.10 Actividad hacer

No se puede aceptar ningún evento mientras la acción se esté ejecutando. Después de que se haya realizado la acción, el sistema entra en el estado **Sonando**. Mientras el sistema se encuentra en ese estado, realiza la actividad **sonar alarma**. A una actividad hacer le lleva un cierto tiempo completarse, tiempo durante el que pueden producirse eventos que interrumpan la actividad hacer. En este caso, la actividad **sonar alarma** no termina por sí misma; continúa durante todo el tiempo que el sistema esté en el estado **Sonando**. Cuando se produce el evento **restablecer**, la transición se dispara y lleva al sistema de vuelta al estado **Monitorizando**. Cuando el estado **Sonando** deja de estar activo, su actividad **sonar alarma** se termina.

actividad interna

Actividad adjunta a un estado que se ejecuta cuando se entra en el estado, cuando se activa el estado o cuando se sale de él.

Véase también transición interna.

Semántica

Las actividades internas están adjuntas a un estado. Una actividad interna puede ser una actividad entrar, una actividad salir o una actividad hacer. Una actividad entrar se ejecuta cuando se entra en un estado. Se ejecuta después de cualesquiera actividades entrar de los estados que la engloban y antes de cualesquiera actividades entrar de los estados anidados. Una actividad salir se ejecuta cuando se sale de un estado. Se ejecuta después de cualesquiera actividades salir de los estados anidados y antes de cualesquiera actividades salir de los estados que la engloban.

Una actividad hacer se ejecuta mientras el estado esté activo. Se inicia cuando se entra en el estado, pero después de cualesquiera actividades entrar adjuntas al estado. A diferencia de otras actividades adjuntas a estados o transiciones, una actividad hacer puede continuar su ejecución durante un tiempo extendido. Si una transición causa una salida del estado y la actividad hacer está todavía en ejecución, es terminada a la fuerza antes de que se ejecute cualquier actividad salir. Si la actividad hacer termina por sí misma mientras el estado esté activo todavía, el estado satisface su condición de finalización y puede desencadenar una transición de finalización. La actividad hacer no se reinicia a no ser que se salga del estado y se vuelva a entrar en él.

Notación

Una actividad interna se representa como una cadena de texto en un compartimento en el símbolo del estado con la sintaxis:

`palabra-clave` / **`expresión-de-actividad`**

La palabra clave es **entry**, **exit** o **do**. La expresión de actividad es dependiente de la implementación, pero en la mayoría de los casos será suficiente el nombre de la actividad definida con una lista de argumentos.

Ejemplo

La Figura 14.11 muestra un estado al que se entra cuando se solicita una contraseña. La actividad entrar se ejecuta cuando se entra en el estado; desactiva el eco del teclado antes de que comience la introducción de la contraseña. La actividad salir se ejecuta cuando se sale del estado; restaura el eco del teclado después de que se haya escrito la contraseña. La otra entrada es una transición interna que se ejecuta si se pide la función de ayuda mientras el estado está activo.

Discusión

El documento UML2 estipula que las actividades internas y las transiciones internas se pueden colocar en compartimentos separados, pero dos compartimentos son superfluos e inconsistentes con los ejemplos del documento.

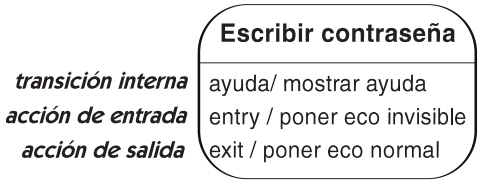


Figura 14.11 Transición interna y sintaxis de actividad interna

actividad salir

Actividad realizada cuando se sale de un estado.

Véase también actividad entrar, ejecutar hasta finalizar, máquina de estados, transición.

Semántica

Un estado opcionalmente puede tener una actividad salir adjunta. Siempre que se salga del estado de cualquier forma, se ejecuta la actividad salir después de cualesquiera actividades adjuntas a estados internos y antes de cualesquiera actividades adjuntas a estados externos. La actividad salir no se puede evitar por ningún medio. Se garantiza su ejecución antes de que el control abandone el estado propietario.

Las actividades entrar y salir no son semánticamente esenciales; la actividad salir podría ser adjuntada a todas las transiciones salientes. Sin embargo, facilitan la encapsulación de un estado de forma que su uso externo se puede separar de su construcción interna. Hacen posible la definición de actividades de inicialización y de terminación, sin importar que pudieran ser evitadas. Son particularmente útiles con las excepciones, puesto que definen efectos que deben realizarse incluso si se produce una excepción.

Notación

Una actividad salir se codifica usando la sintaxis de una transición interna con el evidente nombre de evento **exit** (que es, por tanto, una palabra reservada y no se puede utilizar como un nombre de evento real).

```
exit / actividad
```

Un estado sólo puede tener adjunta una actividad salir. Pero la actividad puede ser una secuencia, de forma que no se pierde generalidad.

Discusión

Una actividad salir es útil para realizar una limpieza que deba hacerse cuando se sale de un estado. El uso más significativo de las actividades salir es liberar almacenamiento temporal y otros recursos asignados durante la ejecución del estado (normalmente, un estado con detalle anidado).

A menudo, las actividades entrar y salir se utilizan juntas. La actividad entrar asigna recursos, y la actividad salir los libera. Incluso si se produce una excepción, los recursos se liberan.

activo/a

Un 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 se convierte en activo cuando se dispara una transición entrante. Un estado activo deja de estar activo cuando se dispara una transición saliente. Si un objeto está activo, al menos un estado también está activo. (En un caso extremo, una clase puede tener sólo un único estado. En ese caso, la respuesta a un evento siempre es 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.

Un objeto puede mantener múltiples estados simultáneamente. El conjunto de estados activos se denomina configuración del estado activo. Si está activo un estado anidado, entonces todos los estados que lo contienen están activos. Si el objeto permite la concurrencia, entonces puede estar activa más de una región ortogonal. Cada transición afecta, al menos, a unos pocos estados en la configuración del estado activo. Ante una transición, los estados activos que no se ven afectados continúan estando activos.

Un estado compuesto puede ser secuencial u ortogonal. Si es secuencial y está activo, entonces exactamente uno de sus subestados cercanos está activo. Si es ortogonal y está activo, entonces exactamente un subestado en cada una de sus regiones cercanas está activo. En otras palabras, un estado compuesto se expande en un árbol Y-O de subestados activos; en cada nivel ciertos estados están activos.

Una transición que atraviesa la frontera de un estado compuesto debe estructurarse para que mantenga las restricciones de concurrencia. Una transición en un estado compuesto secuencial, normalmente tiene un estado origen y un estado destino. Disparar una transición de este tipo no cambia el número de estados activos. Una transición en un estado compuesto ortogonal, normalmente tiene un estado origen y un estado destino por cada región del estado compuesto ortogonal. Una transición de este tipo se denomina división. Si se omiten como destino una o más regiones, el estado inicial de cada región omitida está presente de forma implícita como destino; si una de las regiones carece de estado inicial, el modelo está mal formado. Disparar una transición de división incrementa el número de estados activos. La situación es la inversa si se sale de un estado ortogonal compuesto.

Véase máquina de estados, que contiene una discusión completa sobre la semántica de estados ortogonales y transiciones complejas.

Ejemplo

La parte superior de la Figura 14.12 muestra en ejemplo de máquina de estados, con estados compuestos secuenciales y ortogonales. Se han omitido las transiciones para centrar la atención en los estados. En la parte inferior de la figura se muestran varias configuraciones de estados que pueden estar activos concurrentemente. En este ejemplo, hay cuatro posibles configuraciones de estados activos. Sólo los estados hoja son concretos; los estados superiores son abstractos —es decir, un objeto no puede estar en alguno de ellos sin estar también en un estado hoja anidado. Por ejemplo, el objeto no puede estar en el estado **Q** sin estar en los subestados de **Q**. Debido a que **Q** es ortogonal, tanto **C** como **D** deben estar activos si **Q** está activo. Cada estado hoja se corresponde con un

hilo de control. En un ejemplo más amplio, el número de posibles configuraciones crecería de forma exponencial y puede ser imposible mostrarlos todos, de ahí la ventaja de la notación.

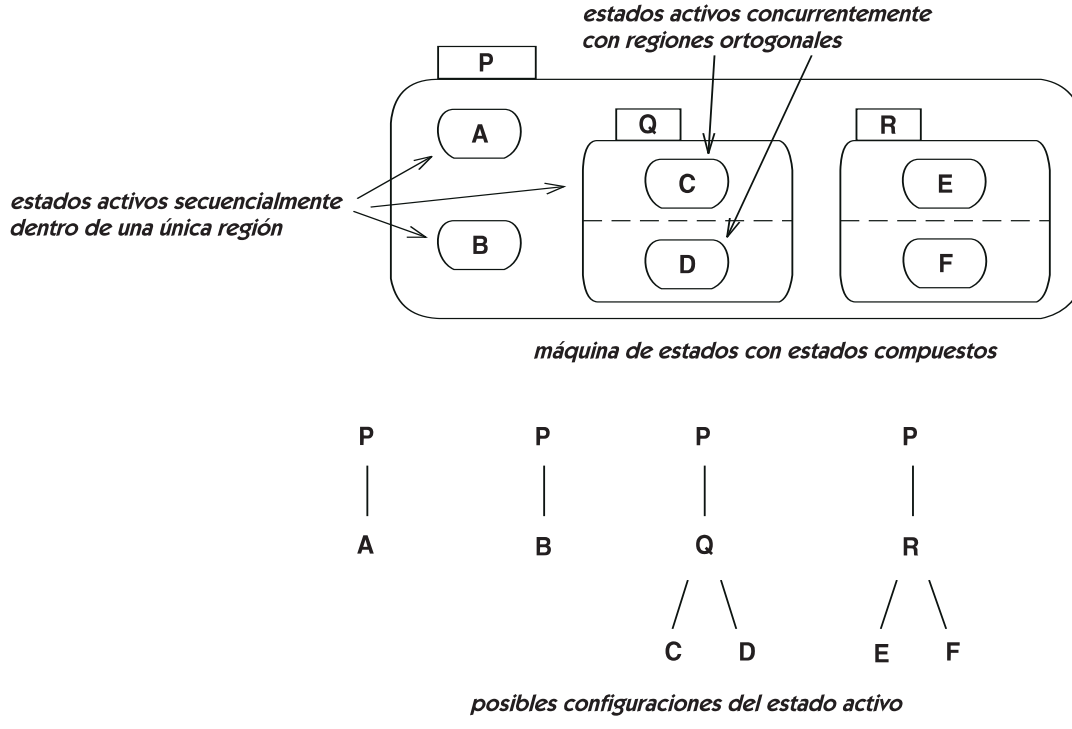


Figura 14.12 Estados activos concurrentemente

actor

Un clasificador para las entidades externas al entorno y que obran recíproca y directamente con él. Un actor participa en un caso del uso o en un conjunto coherente de casos del uso para lograr un objetivo global. *Véase* también caso de uso.

Semántica

Un actor es un clasificador que caracteriza un rol de un usuario o usuarios exteriores o un sistema relacionado con un entorno. El entorno también es un clasificador. Un actor es una idealización con un propósito y un significado concretos y no puede corresponderse exactamente a objetos físicos. Un objeto físico puede combinar propósitos dispares y por lo tanto puede ser caracterizado por varios actores. Diversos objetos físicos pueden tener el mismo propósito, y por lo tanto serían modelados como un mismo actor. El objeto del actor puede ser un ser humano, un sistema informático (de computación), un dispositivo, otro subsistema, u otra clase de objeto. Por ejemplo, los actores en una red de computadoras pudieran incluir al operador, al administrador del sistema, al administrador de la base de datos, y al usuario común. También puede haber actores no humanos, tales como un Cliente Remoto, un *Secuenciador*, y una Impresora de Red.

Cada actor define un rol que pueden asumir los usuarios de un entorno al operar con él. El conjunto completo de actores describe todas las formas en las cuales los usuarios exteriores se comunican con el entorno. Cuando se implementa un sistema, los objetos físicos implementan a los actores. Un objeto físico puede implementar más de un actor para desempeñar todos sus papeles. Por ejemplo, una persona puede ser dependiente y cliente de un almacén a la vez. Estos actores no están intrínsecamente relacionados, sino que puede que ambos sean puestos en ejecución por una misma persona. Cuando el diseño del entorno se implementa, las clases del diseño realizan a los distintos actores dentro del sistema (*véase* la realización).

Las distintas interacciones de los actores con un entorno son encapsuladas en casos de uso. Un caso de uso es un trozo coherente de funcionalidad que implica a un entorno y a sus actores para lograr algo significativo para los actores. Un caso de uso puede implicar a unos o más actores. Un actor puede participar en uno o más casos de uso. En última instancia, los roles determinan a los actores de los casos de uso y los actores pueden participar en varios casos de uso. Un actor que no participa en ningún caso de uso es insustancial.

Las instancias del actor se comunican con el sistema enviando y recibiendo instancias de mensajes (las señales y las llamadas) a y desde casos de uso, en el nivel de la realización, y a y desde los objetos que implementan el caso de uso. Esto se expresa por asociaciones entre el actor y el caso de uso.

Un actor es externo a su entorno y por lo tanto no le pertenece ni se define dentro del entorno. Un actor normalmente se define en el mismo paquete que su entorno. No es adecuado mostrar asociaciones entre actores en un modelo de casos de uso; los actores pueden tener asociaciones a sus entornos y a los casos del uso.

Los actores pueden tener asociaciones con los componentes, y las clases del caso de uso.

Generalización

Dos o más actores pueden tener semejanzas; es decir, pueden comunicarse con el mismo conjunto de casos del uso de la misma forma. Esta semejanza se expresa con la generalización a otro actor abstracto (posiblemente), que modela los aspectos comunes de los actores. Los actores descendientes heredan los roles y las relaciones a los casos de uso que tiene el actor del antepasado. Una instancia de un actor descendiente se puede utilizar siempre en los casos en los que se espera una instancia del antepasado (principio de la sustitución). Un descendiente incluye las cualidades y las operaciones de sus antepasados.

Notación

Un actor puede mostrarse como símbolo de clase (rectángulo) con el “actor estereotipado”. El icono estereotipado estándar para un actor es “la figura lineal de un humano”, con el nombre del actor debajo de la figura. El actor puede tener compartimentos que muestren eventos que recibe, atributos, y también pueden tener dependencias para mostrar los eventos que envía. Éstas son las capacidades que posee un clasificador normal (Figura 14.13).

Se pueden definir iconos especiales para los actores o conjuntos de actores. El nombre de un actor abstracto se muestra usando letra cursiva.

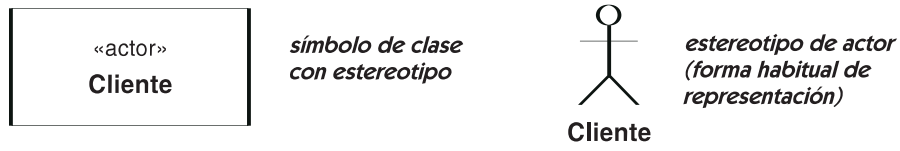


Figura 14.13 Símbolo de actor

afirmación

Un fragmento combinado en una interacción, que indica que el comportamiento descrito por el contenido es el único comportamiento válido en ese punto de la ejecución. Cualquier otro comportamiento contradice el significado.

Semántica

En general, cualquier interacción es una especificación de comportamiento admisible, el cual no incluye todos los posibles comportamientos. Una afirmación es una declaración de que el comportamiento especificado en la región de afirmación es el único comportamiento posible en ese punto de la ejecución. Sin embargo, hay que tener en cuenta que los operadores ignore y consider (ignorar y considerar respectivamente) se pueden utilizar para filtrar el comportamiento bajo consideración. Si cierto tipo de eventos se ignoran explícitamente, su ocurrencia o la falta de ella no afectará a una afirmación.

Por ejemplo, utilice un fragmento afirmación que contenga un fragmento condicional para indicar que las alternativas proporcionadas explícitamente en el fragmento condicional son los únicos resultados posibles y no se pueden dar otros.

Notación

Una afirmación se muestra como una región etiquetada en un diagrama de secuencia con la palabra clave **assert**. La Figura 14.14 muestra una afirmación.

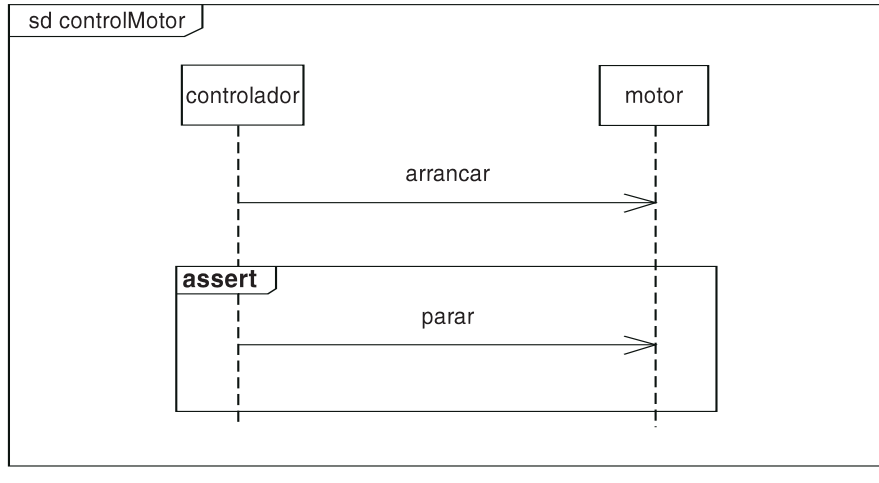


Figura 14.14 Afirmación

Este ejemplo, establece que después de que el motor recibe un mensaje de arrancar del controlador, el siguiente mensaje debe ser un mensaje de parar. La afirmación no realiza ninguna declaración sobre los mensajes que involucran a cualquier otro objeto.

agregación

Una forma de asociación que especifica una relación todo-parte entre un agregado (todo) y una parte que lo constituye. También es aplicable a atributos y parámetros.

Véase también composición.

Semántica

Una asociación binaria se puede declarar como una agregación —es decir, una relación todo-parte. Un extremo de la asociación se designa como agregado mientras que la otra queda sin marcar. Ambos extremos no pueden ser agregados (o compuestos), pero ambos extremos pueden quedar sin marcar (en cuyo caso no sería una agregación).

Los enlaces instanciados de las agregaciones obedecen ciertas reglas. La relación de agregación es transitiva y antisimétrica a través de todos los enlaces de la agregación, incluso a través de aquellos de diferentes asociaciones de agregación. La transitividad significa que tiene sentido decir que "B es parte de A" si existe una ruta de enlaces de agregación de B a A en sentido directo (en este ejemplo, de la parte al todo). Antisimétrico significa que no hay ciclos en las rutas dirigidas de enlaces de agregación. Es decir, un objeto no puede ser directa o indirectamente parte de sí mismo. Al poner las dos reglas juntas, el grafo de enlaces de agregación desde todas las asociaciones de agregación forma un grafo parcialmente ordenado, un grafo sin bucles (un árbol es un caso especial y común de ordenación parcial). La Figura 14.15 muestra un ejemplo.

Una ruta dirigida de enlaces desde el objeto B al objeto A implica que existe una ruta dirigida de asociaciones de agregación de la clase de B a la clase de A, pero la ruta de asociaciones puede implicar ciclos en los que la misma clase aparece más de una vez. Una ruta dirigida de asociaciones de agregación de una clase a sí misma es un *montaje recursivo*.

Existe una forma más estricta de agregación denominada composición. Un compuesto es un agregado con la restricción adicional de que un objeto sólo puede ser parte de un compuesto y que el objeto compuesto tiene la responsabilidad de la disposición de todas sus partes —es decir, de su creación y destrucción. Una agregación que no es una composición se denomina agregación compartida, porque sus partes pueden ser compartidas entre más de un todo.

Véase composición para más detalles.

En una agregación compartida, una parte puede pertenecer a más de un agregado y puede existir con independencia del agregado. A menudo, el agregado "necesita" a las partes en el sentido de que se puede considerar una colección de partes. Pero las partes pueden existir por sí mismas, sin ser consideradas sólo partes. Por ejemplo, un camino es poco más que una colección de segmentos. Pero un segmento puede existir por sí mismo tanto si es parte de un camino como si no, y el mismo segmento puede aparecer en varios caminos.

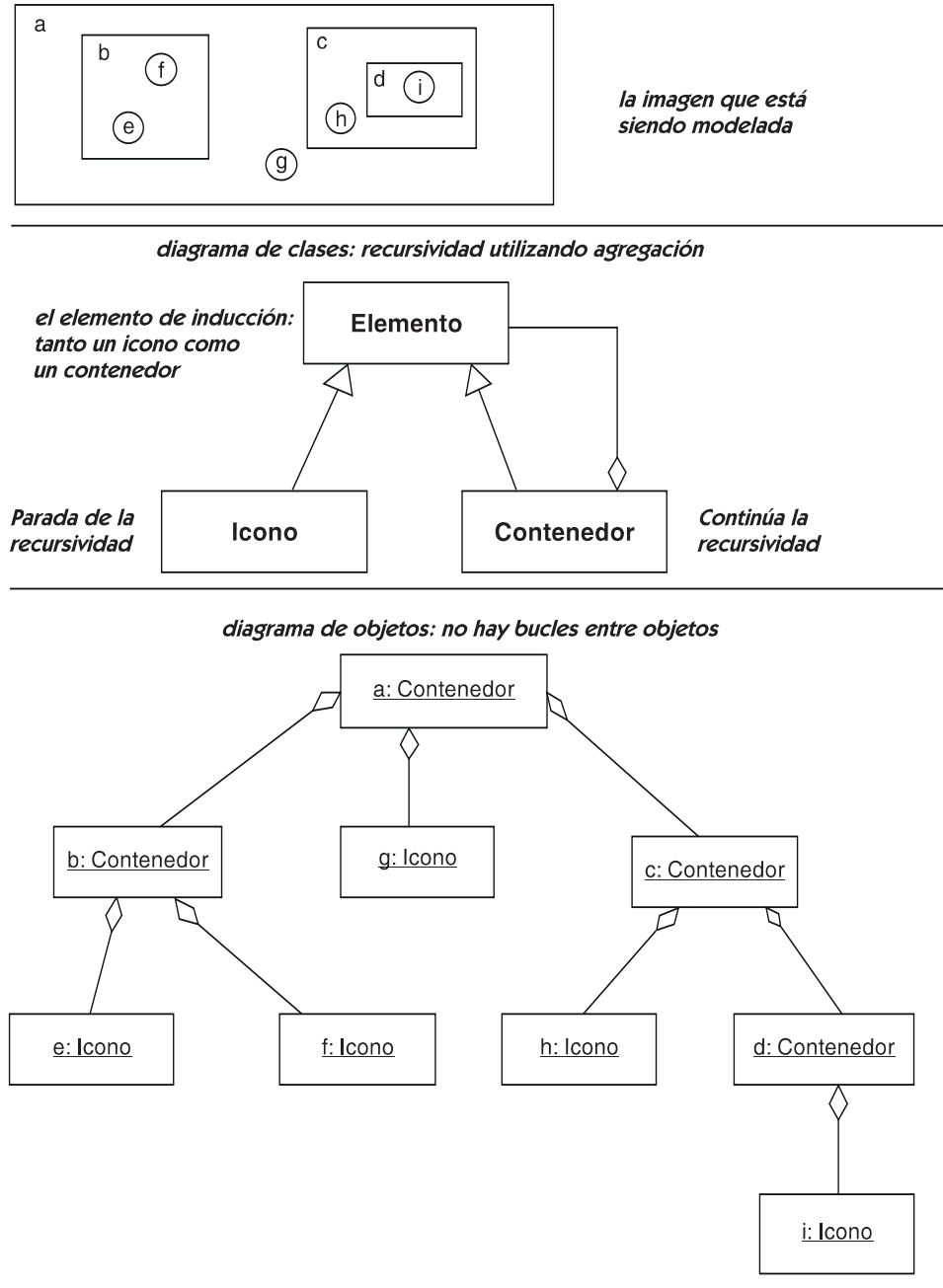


Figura 14.15 Las agregaciones de objetos son acíclicas

La composición también se puede aplicar a los atributos y a los parámetros. En esta situación el objeto al que pertenece el atributo o el parámetro es el todo, y los valores del atributo o parámetro son las partes.

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

Notación

Una agregación compartida se muestra como un rombo o diamante hueco en el extremo de la línea de una asociación que se conecta con la clase agregada (Figura 14.16). Si la agregación es una composición, el rombo está relleno (Figura 14.70). Ambos extremos de la asociación no pueden tener el indicador de agregación.

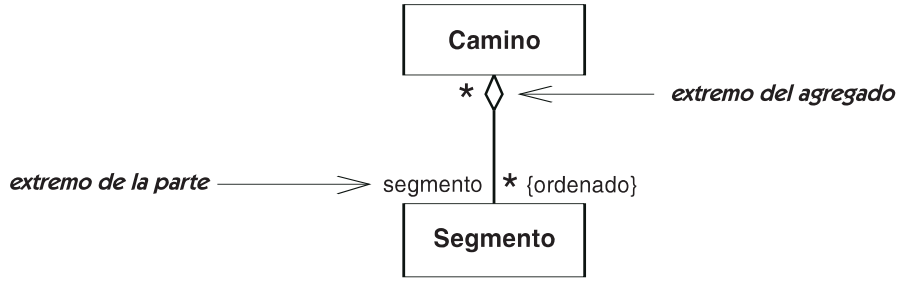


Figura 14.16 Notación de la agregación

Una clase agreado puede tener múltiples partes. La relación entre la clase agregado y cada parte es una asociación independiente (Figura 14.17).

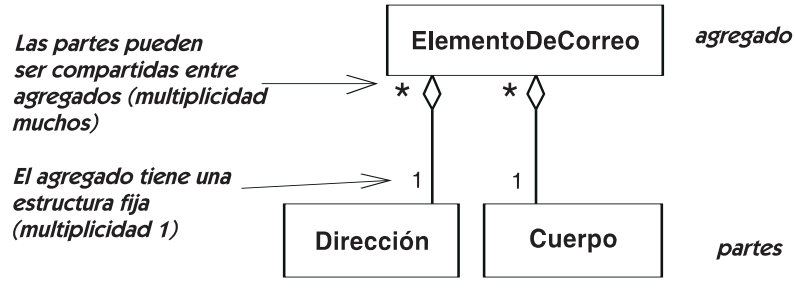


Figura 14.17 Un agregado con varias partes

Si hay dos o más asociaciones de agregaciones sobre la misma clase agregada se pueden representar como un árbol combinando el extremo de la agregación en un único extremo (Figura 14.18). Esto obliga a que todos los adornos de los extremos de las agregaciones sean consistentes; por ejemplo, todos deben tener la misma multiplicidad. Dibujar las agregaciones como un árbol es simplemente una opción de presentación; no tiene ningún tipo de semántica adicional.

Para indicar un atributo compuesto, hay que utilizar la cadena de propiedad **{composite}** a continuación del resto de los atributos. No hay una notación especial para la agregación compartida en un atributo. Si la distinción es importante, la relación se puede mostrar como una asociación.

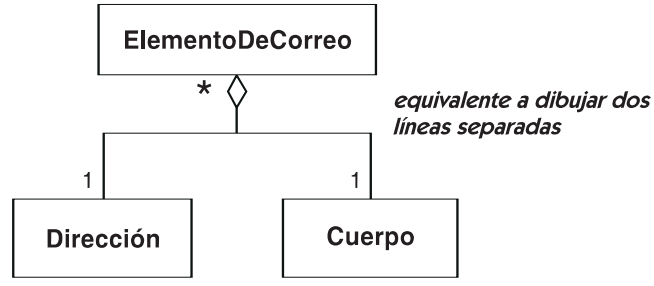


Figura 14.18 Notación con forma de árbol para agregaciones múltiples sobre la misma clase

Discusión

La distinción entre agregación y asociación es, a menudo, cuestión de gustos más que una diferencia de semántica. Hay que tener en mente que la agregación es una asociación. La agregación transmite la idea de que el agregado es inherentemente la suma de sus partes. De hecho, la única semántica real que añade a la asociación es la restricción de que las cadenas de enlaces agregados no pueden formar bucles, lo cual es importante saber. Otras restricciones, como la dependencia de existencia, se especifican mediante la multiplicidad, no con el indicador de agregación. A pesar de la reducida semántica vinculada a la agregación, todo el mundo piensa que es necesaria (por diferentes razones). Se puede pensar en esto como un placebo de modelado.

Varias propiedades secundarias están conectadas con la agregación, pero no de forma los suficientemente fiable como para hacerlas parte de su definición. Entre estas propiedades se incluye la propagación de operaciones del agregado a las partes (como la operación de mover) y la asignación de memoria compacta (de forma que el agregado y sus partes recursivas pueden ser cargadas eficientemente con una transferencia de memoria). Algunos autores han distinguido varios tipos de agregación, pero las distinciones son demasiado ligeras e innecesarias para el modelado general.

La agregación es una propiedad que transciende a 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 la asociación de composición) de que no puede haber bucles en los enlaces de agregación, incluyendo enlaces de diferentes asociaciones. En cierto modo, la agregación es un tipo de generalización de asociación en la que las restricciones y algunas operaciones se aplican a asociaciones de muchos tipos específicos.

La composición tiene una semántica más específica que se corresponde con el contenido físico y diversas nociones de propiedad. La composición es apropiada cuando cada parte es propiedad de un objeto y cuando las partes no tienen vida independiente fuera de su propietario. Es más útil cuando las partes tienen que ser asignadas e inicializadas cuando se crea al propietario, y las partes no sobreviven a la destrucción de su propietario. Normalmente se asume que los atributos de una clase tengan estas propiedades. Se debe evitar, cuando se utiliza la composición, la sobrecarga de la gestión de memoria y el peligro de los punteros perdidos o los objetos huérfanos. También es apropiado en situaciones en las que grupos de atributos han sido aislados en distintas clases por razones de encapsulamiento y manipulación, pero los atributos realmente se aplican a la clase principal. Las clases contenedoras utilizadas para implementar las asociaciones son candidatas claras a ser partes compuestas, aunque normalmente serían generados por un

generador de código y no modelados explícitamente. Obsérvese que una parte compuesta, como una clase contenedora, puede contener referencias (punteros) a partes no compuestas, pero los objetos referenciados no son destruidos cuando el objeto que los referencia es destruido.

La distinción entre asociaciones y atributos es más una cuestión de gusto o del lenguaje de implementación que de semántica. Los atributos inherentemente tienen ambiente de agregación alrededor de ellos.

agregación compartida

Una agregación que no representa propiedad.

Semántica

Un extremo de una asociación binaria puede marcarse como el inicio de una agregación, mientras que una instancia de clase ligada al extremo se considera que es compuesta de partes que son instancias de la clase en el otro extremo. Una agregación puede ser compuesta o compartida. Una agregación compuesta lleva el sentido de propiedad. Un objeto puede ser parte de a lo sumo una agregación compuesta. Una agregación compartida no tiene esta restricción. Un objeto puede pertenecer a muchas agregaciones compartidas.

No hay ninguna semántica precisa para la agregación compartida salvo la restricción que cambia en cadena de eslabones de agregación no puede formar un ciclo. Diseñadores usan la agregación compartida de varias maneras para representar los varios conceptos semánticos.

Notación

La agregación compartida es mostrada por un diamante sin contenido en el fin de una línea de asociación que se conecta a la clase que representa el objeto agregado (como opuesto a sus partes).

Véase asociación para detalles.

agregación compuesta

Véase composición.

agregado

Una clase que representa al todo en una asociación de agregación (todo-parte).

alcance

El concepto de UML1 de alcance del dueño y alcance del destino se ha simplificado en UML2.

El alcance del dueño es planeado por la marca de característica estática en los atributos y operaciones.

El alcance destino ha sido abandonado como concepto.

alcance de destino

Este concepto de UML1 ha sido eliminado de UML2. No demostró ser muy útil, y no ha sido reemplazado.

alcance de propietario

Este término de UML1 se ha anulado. *Véase* característica estática para capacidad equivalente.

alcance original

Este concepto de UML1 ha sido reemplazado por característica estática.

alt

Palabra clave que indica una construcción condicional en una interacción, como por ejemplo, en un diagrama de secuencia. *Véase* condicional.

alternativa

Una construcción condicional en una construcción, como por ejemplo un diagrama de secuencia. *Véase* condicional.

amigo/a

En UML2 se ha eliminado ésta dependencia de UML1.

análisis

Etapa del desarrollo de un sistema que captura los requisitos y el dominio del problema. El análisis se centra en qué hacer; el diseño se centra en cómo hacerlo. En un proceso iterativo, no se necesita que las etapas se hagan de forma secuencial. Los resultados de esta etapa se representan mediante modelos a nivel de análisis, especialmente la vista de caso de uso y la vista estática. Contrastar análisis, diseño, implementación y despliegue (fase).

Véase etapas de modelado, proceso de desarrollo.

antepasado

Un elemento que se encuentra siguiendo un camino de una o más relaciones de padres.

Véase generalización, padre.

aplicación

Véase aplicación del perfil.

aplicación del perfil

La especificación que los estereotipos y restricciones declarados en un perfil dado, puede aplicarse a elementos del modelo dentro de un paquete dado.

Semántica

Un perfil define los estereotipos y restricciones de los que se refieren a un subconjunto designado de un metamodelo base de UML (por defecto, el metamodelo estándar completo de UML). Pueden definirse muchos perfiles diferentes, posiblemente con definiciones contradictorias. La aplicación del perfil hace un perfil dado disponible para el uso dentro de un paquete dado por un usuario del modelo.

Pueden aplicarse uno o más perfiles a un paquete dado. Las restricciones del perfil tienen efecto en los elementos modelo del metatipo dado dentro del usuario del paquete. Pueden aplicarse perfiles múltiples al mismo paquete proporcionadas sus restricciones para evitar conflictos. Si un perfil aplicado depende de otro perfil, ambos perfiles deben aplicarse al paquete del usuario del modelo.

El aplicar un perfil a un paquete hace disponibles los estereotipos declarados en el perfil para el uso dentro del paquete en elementos del modelo en los que el estereotipo está definido. Un estereotipo debe aplicarse explícitamente a un elemento modelo. Si la restricción contiene el estereotipo fuerte {**required**}, debe aplicarse a todos los elementos de los datos del metatipo, pero todavía es preferible ser explícito sobre eso.

Notación

La aplicación del perfil se muestra por una flecha punteada de la dependencia con una punta de flecha del palo del paquete del usuario al símbolo del perfil. La palabra clave **«apply»** se pone adelante la flecha. Pueden aplicarse perfiles múltiples al mismo paquete y el mismo paquete puede aplicarse a los paquetes múltiples.

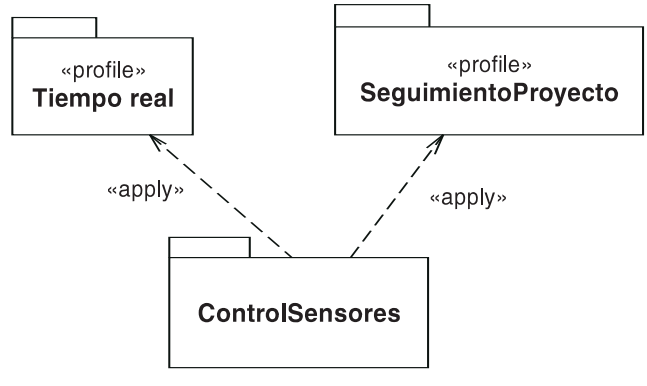


Figura 14.19 Aplicación del perfil

apply

Palabra clave sobre una flecha de línea discontinua entre un paquete y un perfil, que indica la aplicación del perfil sobre el paquete. *Véase* aplicación del perfil.

argumento

Valor específico de un parámetro.

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

Semántica

Una instancia en tiempo de ejecución de un elemento parametrizado, como una operación o una acción, tiene una lista de valores de los argumentos, cada uno de los cuales es un valor que tiene un tipo que debe ser consistente con el tipo declarado del parámetro con el que se corresponde. Un valor es consistente si su clase o tipo de datos es el mismo o es un descendiente del tipo declarado del parámetro. Mediante el principio de capacidad de sustitución, el valor de un descendiente se puede utilizar en cualquier sitio donde se haya declarado un antecesor como tipo del parámetro.

Una ocurrencia de una acción de difusión, de una acción de llamada o de una acción de envío necesita de argumentos para los parámetros, a menos que la señal o la operación carezca de atributos o parámetros.

Sin embargo, en la ligadura de una plantilla, los argumentos aparecen dentro de un modelo UML en el momento del modelado. Los argumentos de una plantilla pueden incluir, además de los valores de datos ordinarios, los objetos y las expresiones, clasificadores. En este caso, el tipo del parámetro correspondiente debe ser **Classifier** (clasificador) o algún otro metatipo. El valor de un argumento de una plantilla se debe fijar durante el modelado y no se puede utilizar para representar un argumento de tiempo de ejecución. No utilice plantillas si no se han fijado los parámetros durante el modelado.

argumento formal

Véase parámetro.

arquitectura

La estructura organizativa de un sistema, incluyendo su descomposición en partes, su conectividad, mecanismos de interacción y los principios rectores que caracterizan el diseño de un sistema.

Véase también paquete.

Semántica

La arquitectura es un conjunto de decisiones significativas sobre la organización de un sistema software. Incluye la selección de elementos estructurales y de las interfaces que permiten su

interconexión, la organización a gran escala de elementos estructurales y la topología de sus conexiones, su comportamiento tal y como se especifica en las colaboraciones entre esos elementos, los mecanismos importantes que se encuentran disponibles a lo largo del sistema, y el estilo arquitectónico que guía su organización. Por ejemplo, la decisión de construir un sistema a partir de dos capas, cada una de las cuales contiene un número pequeño de subsistemas que se comunican de una forma específica es una decisión arquitectónica. La arquitectura software no sólo tiene que ver con la estructura y el comportamiento, también se encarga del uso, las funcionalidades, el rendimiento, la resistencia, la reutilización, la facilidad de comprensión, las restricciones y compromisos tecnológicos y económicos, y la estética.

Discusión

Las decisiones de diseño sobre la descomposición de un sistema en partes se pueden capturar mediante los modelos, subsistemas, paquetes y componentes. La dependencia entre estos elementos son indicadores clave de la flexibilidad de la arquitectura y de la dificultad de modelar el sistema en el futuro.

artefacto

La especificación de una pieza física de información que se utiliza o se produce en un proceso de desarrollo de software, como un documento externo o un producto del trabajo, o mediante el desarrollo y manipulación de un sistema. Un artefacto puede ser un modelo, una descripción, o software.

Semántica

Un artefacto se corresponde con un elemento concreto del mundo real. Una instancia de un artefacto se despliega en una instancia de nodo. A menudo, los artefactos representan archivos o documentos. Se encuentran inherentemente conectados con la implementación de un sistema. Un artefacto se puede asociar con un componente.

Los perfiles están pensados para definir varios tipos de artefactos adecuados para distintos entornos de implementación. Los estereotipos estándar de los artefactos incluyen los fuentes y los ejecutables.

Estructura

Despliegue. Un artefacto se puede desplegar en un nodo. Esto indica que la instancias del artefacto se pueden desplegar en las instancias del nodo.

Artefactos anidados. Un artefacto puede contener otros artefactos. Los artefactos anidados se despliegan en la misma instancia de nodo que el artefacto contenedor o en una instancia de nodo contenedor.

Manifestación. Un artefacto puede manifestar, es decir, puede originarse de un conjunto de elementos del modelo e implementarlos. Esta relación captura los enlaces entre los elementos de diseño y su manifestación física. Por ejemplo, una clase de diseño puede generar como artefactos un archivo fuente, un archivo ejecutable y un archivo de documentación.

Notación

Un artefacto se muestra como un rectángulo con la palabra clave **«artifact»** (artefacto) sobre el nombre del artefacto. Una instancia de un artefacto se muestra con su nombre subrayado. La relación de manifestación se representa mediante una flecha de línea discontinua desde el artefacto a los elementos que manifiesta, colocando junto a la flecha la palabra clave **«manifest»**. El despliegue de un artefacto en un nodo (o en instancias de él) se muestra colocando el símbolo del artefacto dentro de los límites del símbolo del nodo.

La Figura 14.20 muestra un artefacto situado en un nodo manifestando un componente.

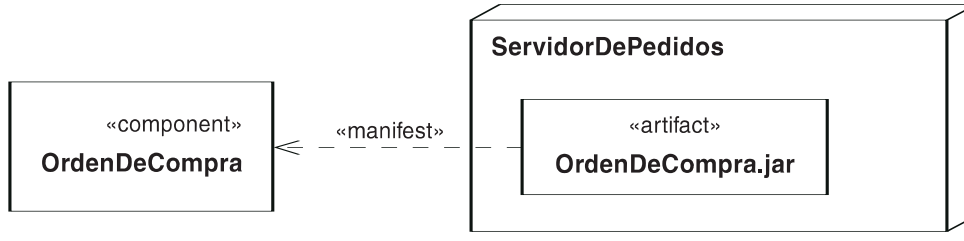


Figura 14.20 Notación de artefacto

Historia

Los artefactos en UML2 pueden manifestar cualquier tipo de elemento, no sólo componentes como en UML1. Diversas cosas que eran consideradas componentes en UML1 han sido reclasificadas como artefactos.

asociación

La relación semántica entre dos o más clasificadores que involucra conexiones entre sus instancias.

Véase también clase de asociación, extremo de la asociación, generalización de asociaciones, asociación binaria, multiplicidad (asociación), asociación *n*-aria.

Semántica

Una asociación es una relación entre dos o más clasificadores especificados que describe las conexiones entre sus instancias. Los clasificadores participantes tienen posiciones ordenadas dentro de la asociación. La misma clase puede aparecer en más de una posición en la asociación. Cada instancia de una asociación (un enlace) es una tupla (una lista ordenada) de referencias a objetos. La extensión de la asociación es una colección de esos enlaces. La colección puede ser un conjunto (sin entradas duplicadas) o una bolsa (en la que se permiten elementos duplicados), y los elementos pueden no tener orden (conjunto) o estar ordenados (lista). Un objeto dado puede aparecer más de una vez dentro del conjunto de enlaces o incluso más de una vez dentro del

mismo enlace (en diferente posición) si la definición de la asociación lo permite. Las asociaciones son el "pegamento" que mantiene junto un modelo de un sistema. Sin las asociaciones, sólo hay un conjunto de clases aisladas.

Estructura

Una asociación tiene un nombre opcional, pero la mayoría de sus descripciones se encuentran en una lista de extremos de la asociación, cada uno de los cuales describe la participación de objetos de una clase en la asociación.Tenga en cuenta que un extremo de la asociación es una parte de la descripción de una asociación y no una semántica individual o concepto notacional.

Nombre. Una asociación tiene un nombre opcional, una cadena que debe ser única entre las asociaciones y clases dentro del paquete que las alberga. (Una clase de asociación es, al mismo tiempo, una asociación y una clase, por lo que las asociaciones y las clases comparten el mismo espacio de nombres.) No es obligatorio que una clase tenga nombre; los nombres de roles proporcionan una forma alternativa de distinguir varias asociaciones entre las mismas clases. Por acuerdo, el nombre se lee en el orden en el que las clases participantes aparecen en la lista: TrabajaPara (Persona, Empresa) = una Persona trabaja para una Empresa; Vende (Vendedor, Coche, Cliente) = un Vendedor vende un Coche a un Cliente.

Extremos de la asociación. Una asociación contiene una lista ordenada de dos o más extremos de la asociación. (Por ordenada, entendemos que los extremos se distinguen entre sí y no son intercambiables.) Cada extremo de la 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, no pudiéndose, en general, intercambiar las posiciones. Cada extremo de la asociación especifica propiedades que se aplican a la participación de los objetos correspondientes, como cuantas veces puede aparecer un objeto individual en los enlaces de la asociación (multiplicidad). Ciertas propiedades, como la navegabilidad, se aplican sólo a las asociaciones binarias, pero la mayoría se aplican, tanto a las asociaciones binarias como a las *n*-arias.

Los extremos de la asociación y los atributos son formas alternativas de especificar las mismas relaciones subyacentes. Los atributos son ligeramente más restrictivos que las asociaciones y siempre son propiedad de sus clases.

Véase extremo de la asociación para ver los detalles completos.

Derivación. Una asociación se puede especificar como una derivación de otros elementos, como asociaciones, atributos o restricciones.

Especialización. Se puede especializar una asociación. Cada enlace de la asociación más específica también puede aparecer en una asociación más general.

Las asociaciones también se pueden redefinir. *Véase* redefinición.

Instanciación

Un enlace es una instancia de una asociación. 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 correspondiente al extremo de la asociación. Un enlace no tiene identidad independiente fuera de la lista de objetos que alberga. Los enlaces en la extensión de una asociación forman una colección que puede ser un conjunto, una bolsa, un conjunto ordenado o una secuencia, dependiendo de los ajustes de unicidad y ordenación en la multiplicidad. El número de apariciones de un objeto en el conjunto de enlaces debe ser compatible con la multiplicidad

en cada extremo de la asociación. Por ejemplo, si la asociación **EntradasVendidas** conecta muchas entradas con una representación, entonces cada entrada puede aparecer sólo una vez en un enlace, pero cada representación puede aparecer muchas veces, cada una con una entrada diferente. (Presumiblemente, pueden aparecer múltiples copias del mismo enlace en bolsas y secuencias. Esto no necesita de identidades diferentes, pero la especificación de UML es algo vaga sobre este punto.)

Los enlaces se pueden crear y destruir durante la ejecución de un sistema, sujetos a las restricciones de cambio de cada extremo de la asociación. En algunos casos, se puede crear o cambiar un enlace con un objeto en un extremo, pero no en el otro. Un enlace se crea de una lista de referencias de objetos. Un enlace no tiene identidad propia. Esto hace que no tenga sentido hablar de cambio en su valor. Sin embargo, se puede destruir y se puede crear un nuevo enlace para que ocupe su lugar. El enlace de una clase de asociación tiene uno o más valores de atributos además de la lista de objetos que definen su identidad, y los valores de los atributos se pueden modificar mediante operaciones mientras se preservan las referencias a los objetos participantes.

Notación

Una asociación binaria se muestra con una ruta que conecta los bordes de dos clases (Figura 14.21). Una asociación *n*-aria se muestra como un rombo o diamante que conecta líneas continuas a las clases participantes (14.asociación n-ariaX). (En la asociación binaria el rombo se suprime por resultar extraño.) Se puede conectar más de un extremo a una única clase.

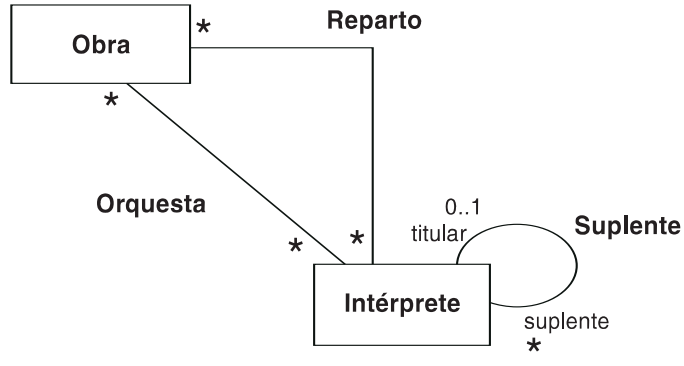


Figura 14.21 Asociaciones

Una ruta consiste en uno o más segmentos de línea sólida, normalmente segmentos de línea continua, aunque también se permiten arcos y otras curvas, especialmente para mostrar una autoasociación (una asociación en la que una clase aparece más de una vez). El segmento individual no tiene significado semántico. El usuario puede elegir el estilo de línea a utilizar. *Véase* ruta.

Para evitar la ambigüedad, un cruce de líneas se puede dibujar utilizando un pequeño semicírculo para pasar una línea sobre la otra que se cruza (Figura 14.22). Esta notación es opcional.

Los extremos de las rutas tienen adornos que describen la participación de una clase en la asociación. Algunos adornos se muestran en el extremo de la ruta, entre el segmento de línea y la caja

Figura 14.22 Notación para el cruce de líneas

de la clase. Si hay varios adornos, se sitúan en secuencia desde el final de la línea hacia el símbolo de clase —flecha de navegación, rombo de agregación/composición y calificador (Figura 14.23).

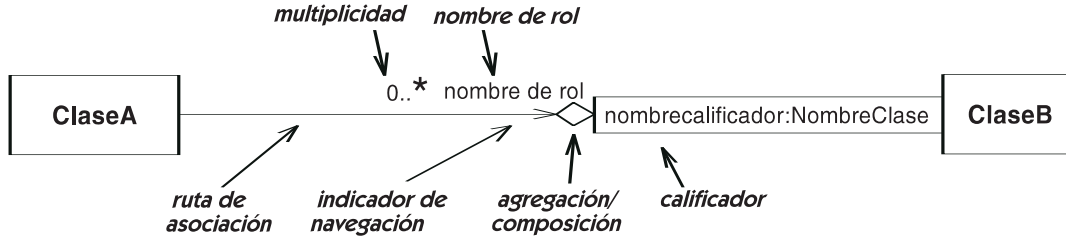


Figura 14.23 Orden de los adornos en el extremo de la asociación

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

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

Nombre de asociación

El nombre de la asociación se coloca cerca de la ruta, pero lo suficientemente lejos de los extremos para que no haya peligro de confusión. (El peligro de confusión es puramente visual para una persona. Dentro de una herramienta gráfica, los símbolos relacionados se pueden conectar con hiperenlaces internos que no permiten ambigüedad. Es responsabilidad de la herramienta determinar cómo de lejos es suficientemente lejos.) El nombre de asociación se puede arrastrar de segmento en segmento de una asociación con múltiples segmentos sin que haya un impacto semántico. El nombre de asociación puede tener un pequeño triángulo relleno cerca de él para mostrar el orden de las clases en la lista. Intuitivamente, la flecha del nombre muestra la forma de "leer" el nombre. En la Figura 14.24, la asociación **TrabajaPara** entre la clase **Persona** y la clase **Empresa** tendría el triángulo del nombre apuntando de **Persona** a **Empresa**, y se leería "Persona trabaja para Empresa". Tenga en cuenta que el triángulo de ordenación en el nombre es una herramienta puramente notacional que indica el orden de los extremos de la asociación. En el propio modelo, los extremos están inherentemente ordenados, por lo que el nombre en el modelo no necesita o tiene una propiedad de ordenación.

Figura 14.24 Nombre de asociación

Un estereotipo de la asociación se indica mostrando el nombre del estereotipo entre comillas (« ») en frente o en lugar del nombre de asociación. Se puede colocar una cadena de propiedad después o debajo del nombre de asociación.

Para identificar una asociación derivada se coloca una barra inclinada (/) en frente del nombre.

Clase de asociación

Una clase de asociación se muestra vinculando un símbolo de clase a una ruta de asociación con una línea discontinua. Para una asociación *n*-aria, la línea discontinua se conecta al diamante o rombo de la asociación. Las propiedades "de clase" de la asociación se muestran en el símbolo de clases, mientras que las propiedades "de asociación" de la asociación se muestran en la ruta. Sin embargo, obsérvese que la estructura de modelado subyacente es un único elemento, aunque la imagen que se muestra utiliza dos construcciones gráficas.

Véase clase de asociación para más detralles.

Restricción xor

La restricción {**xor**} conecta dos o más asociaciones que están conectadas a una única clase (la clase base) en uno de los extremos. Una instancia de la clase base puede participar únicamente en una de las asociaciones conectadas mediante esta restricción. Se debe cumplir la multiplicidad de la asociación elegida. Si cualquier multiplicidad de la asociación incluye la cardinalidad 0, una instancia de la clase base podría no tener enlace de la asociación; de otra forma debe tener uno.

Una restricción xor se muestra mediante una línea discontinua que conecta dos o más asociaciones, que tienen una clase en común, con la cadena de restricción {**xor**} etiquetando la línea discontinua (Figura 14.25). Los nombres de rol en los extremos opuestos a la clase común deben ser distintos.

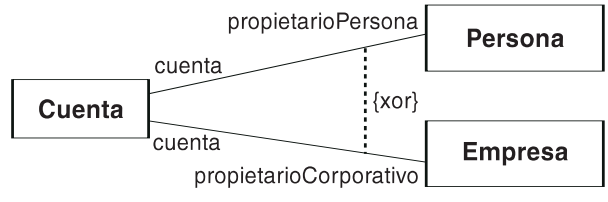


Figura 14.25 Asociación xor

Generalización

La generalización entre asociaciones se muestra con una flecha de generalización entre las líneas de ruta de las asociaciones (grandes puntas de flecha cerradas en el extremo más general). *Véase* generalización de la asociación.

Historia

En UML1, la extensión de una asociación siempre era un conjunto, es decir, no había elementos duplicados. Esto creaba problemas con el uso normal de listas, porque los modeladores espera-

ban (y necesitaban) que fuera posible duplicar entradas. UML2 permite incluir enlaces duplicados. Con mayor frecuencia, los modelos incluirán conjuntos (sin orden y sin duplicados) y listas (ordenados con duplicados), siendo posible disponer de bolsas (sin orden y con duplicados) y conjuntos ordenados (ordenados sin duplicados).

La distinción entre atributos y extremos de asociación se ha debilitado en UML2, por lo que es más sencillo cambiar puntos de vista un modelo. Los extremos de asociación pueden pertenecer a asociaciones o a los clasificadores participantes, lo que permite formas alternativas de empaquetamiento.

Discusión

Una asociación no necesita tener nombre. Normalmente, los nombres de rol son más convenientes porque proporcionan nombres para la navegación y la generación de código y evitan los problemas sobre la forma de leer el nombre. Si tiene un nombre, éste debe ser único dentro de su paquete. Si no tiene nombre y hay más de una asociación entre una pareja (o conjunto) de clases, los nombres de rol deben estar presentes para distinguir las asociaciones. Si sólo hay una asociación entre un par de clases, los nombres de las clases son suficientes para identificar la asociación.

Se puede argumentar que los nombres de asociación son más útiles cuando el concepto del mundo real tiene un nombre, como **Matrimonio** o **Trabajo**. Cuando un nombre de asociación está "dirigido" para ser leído en una determinada dirección, normalmente es mejor utilizar simplemente nombres de rol, que nunca son ambiguos con independencia de la dirección en la 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 composición para un ejemplo de generalización en el que intervienen dos asociaciones.

asociación binaria

Una asociación que se produce exactamente entre dos clases.

Véase también asociación, asociación *n*-aria.

Semántica

Una asociación binaria es una asociación que tiene exactamente dos extremos de asociación, siendo, con mucha diferencia, el tipo más común de asociación. Dado que en una asociación binaria un extremo tiene otro único extremo, las asociaciones binarias son especialmente útiles para especificar rutas de navegación entre objetos. Una asociación es navegable si se puede atravesar en esa dirección. Otras propiedades, como la multiplicidad, están definidas para las asociaciones *n*-arias, pero son más intuitivas y útiles para las asociaciones binarias.

Notación

Una asociación binaria se muestra como una línea continua que conecta dos símbolos de clase. Los adornos se pueden vincular a cada extremo, y se puede colocar un nombre de la asociación cerca de la línea, pero lo suficientemente lejos de los extremos de forma que no se pueda con-

fundir con el nombre de rol. La notación para asociaciones binarias es la misma que para las asociaciones *n*-arias, excepto por el hecho de que se elimina el rombo central. Sin embargo, las asociaciones binarias pueden tener adornos que no son aplicables a las asociaciones *n*-arias, como la agregación. *Véase* asociación para más detalles.

asociación *n*-aria

Una asociación entre tres o más clases. Contraste: asociación binaria.

Semántica

Cada instancia de la asociación es un valor de una tupla de orden *n*, uno de cada una de las respectivas clases. Una clase puede aparecer en más de una posición en la asociación pero los valores en las diferentes posiciones son independientes y no necesitan ser el mismo objeto. Una asociación binaria es un caso especial con su propia notación simplificada y ciertas propiedades adicionales que son sin sentido (o por lo menos absurdamente complicado) para una asociación *n*-aria.

Es posible especificar la multiplicidad para las asociaciones *n*-arias pero puede ser menos obvio que la multiplicidad binaria. La multiplicidad en un extremo de la asociación representa el número potencial de valores en el extremo, cuando los valores en el otro extremo *n*-1 son fijos. Note que esta definición es compatible con multiplicidad binaria.

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

No hay ninguna diferencia semántica entre una asociación binaria y una asociación *n*-aria con dos extremos, excepto la representación. Una asociación con dos extremos debería ser asociación binaria y una con más de dos extremos debería ser una asociación *n*-aria.

Notación

Una asociación *n*-aria se representa como un diamante grande (es decir, grande comparado con un terminador en un camino), con un camino desde el diamante a cada clase participante. En el nombre de la asociación (si lo hay) se representa cerca del diamante. Los adornos pueden aparecer en el extremo de cada camino como ocurre con una asociación binaria. Es posible indicar la multiplicidad, pero no se permiten calificadores ni agregación.

Se puede conectar un símbolo de clase de asociación al diamante mediante una línea punteada. Esto indica una asociación *n*-aria que tiene atributos, operaciones, y/o asociaciones.

Ejemplo

La Figura 14.26 muestra el registro de un equipo en cada temporada con un goleador concreto. Se asume que los goleadores pueden ser transferidos durante la temporada y podrían tener un registro con equipos diferentes. En un libro de registro, cada enlace estaría en una línea separada.

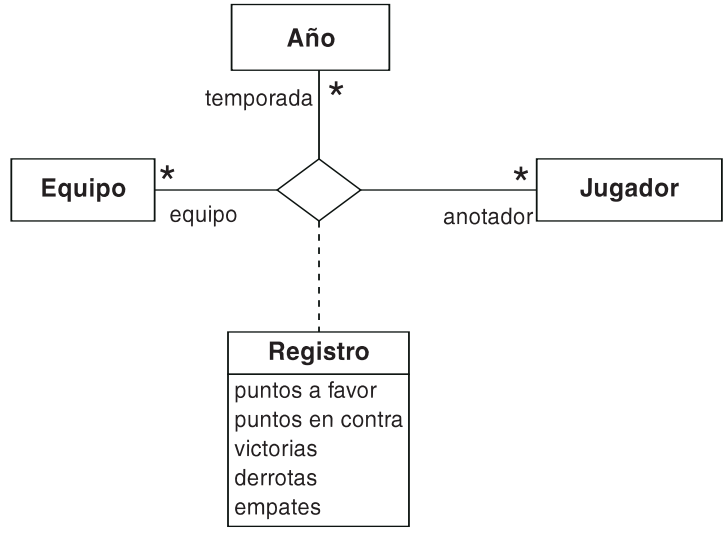


Figura 14.26 Asociación ternaria que también es una clase de asociación

Guía de estilo

Normalmente, las líneas se dibujan desde los vértices del diamante o en el punto medio de un lado.

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 las clases (A, B, C), la multiplicidad de C establece cuántos objetos de C pueden aparecer en asociación con un par particular de objetos de A y B. Si la multiplicidad de esta asociación es (muchos muchos, uno), hay un único valor de C entonces para cada posible pareja (A, B). Para un par dado (B, C), puede haber muchos valores A, sin embargo, e individualmente muchos valores de A, B y C puede participar en la asociación. En una asociación binaria, esta regla reduce a la multiplicidad de cada extremo definida con respecto al otro extremo.

No hay problema en definir la multiplicidad con respecto a un solo extremo (tal y como han propuesto algunos autores) porque la multiplicidad sería muchos para cualquier asociación *n*-aria con sentido. Si no, la asociación podría dividirse en una asociación binaria entre la clase de asociación que incluyen todas las clases restantes, con una ganancia en precisión y eficacia de la implementación.

En general, es mejor evitar las asociaciones *n*-aria, porque las asociaciones binarias son más simples de implementar y permiten navegación. Generalmente, las asociaciones *n*-aria sólo son útiles cuando todos los valores se necesitan para determinar de manera única un enlace. Una asociación *n*-aria se implementará casi siempre como una clase cuyos atributos incluyen punteros a los objetos participantes. La ventaja de modelarlo como una asociación es la restricción de que no puede haber ningún enlace doble dentro de una asociación.

Considere el ejemplo de un estudiante recibiendo un curso de un profesor durante un periodo (Figura 14.27). Un estudiante no tomará el mismo curso de más de un profesor, pero un estu-

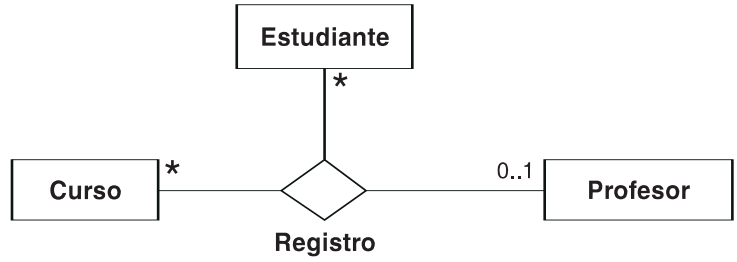


Figura 14.27 Multiplicidad de una asociación *n*-aria

diante puede tomar más de un curso de un solo profesor y un profesor puede enseñar más de un curso. Las multiplicidades se presentan en el diagrama. La multiplicidad en **Profesor** es opcional (0..1); las otras multiplicidades son muchos (0..*).

Cada valor de multiplicidad es relativo a un par de objetos de otros extremos. Para un (curso, estudiante) aparece, hay cero o un profesor. Para una pareja (estudiante, profesor), hay muchos cursos. Para una pareja (curso, profesor), hay muchos estudiantes.

Nótese que si esta asociación se refina en una clase, entonces sería posible tener más de una copia de la misma (estudiante, curso, profesor,) combinación que no es deseable.

assert

Palabra clave en un diagrama de secuencia que indica una afirmación.

atómico/a

Una acción u operación cuya ejecución se debe completar como una unidad; una acción que no se puede ejecutar o terminar parcialmente por un evento externo o interrupción. Normalmente, las operaciones atómicas son pequeñas y sencillas, como asignaciones y cálculos aritméticos o de cadena sencillos. Un cómputo atómico ocurre en un punto definido de la secuencia de ejecución.

No es un término preciso en UML.

Véase también acción, actividad hacer, ejecutar hasta finalizar.

Semántica

Conceptualmente, las acciones atómicas que no acceden a recursos compartidos simplemente necesitan ejecución concurrente. Hay muchas formas de trasladar estas acciones a recursos compartidos del procesador como parte de una implementación.

Si las acciones acceden a recursos compartidos (memoria, dispositivos o interacciones externas) difícilmente pueden ser consideradas atómicas. Este término se emplea algunas veces con el significado de no intercalable a nivel global, pero este tipo de interacciones globales necesitan de un modelo de implementación específico y más detallado.

atributo

Un atributo es la descripción de un elemento 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 un elemento con nombre, dentro de un clasificador, que describe los valores que pueden guardar las instancias del clasificador. Cada instancia del clasificador o uno de sus descendientes tiene una ranura que guarda un valor de un tipo dado. Todas las ranuras son distintas e independientes entre sí (excepto para los atributos de ámbito de clase, los cuales se describen posteriormente). A medida que avanza la ejecución, el valor que guarda una ranura dentro de una instancia puede ser reemplazado por otro valor distinto del mismo tipo, ya que los atributos son modificables.

Un clasificador forma un espacio de nombres para sus atributos. También se incluyen en el espacio de nombres otras propiedades, como los nombres de rol de las asociaciones que salen del clasificador y partes internas.

Un atributo se puede redefinir. *Véase* redefinición (de propiedad).

Estructura

Los atributos y los extremos de la asociación contienen la misma información y se pueden intercambiar con relativa facilidad. Un atributo se puede considerar como un extremo de la asociación que pertenece al clasificador y que es navegable desde el clasificador al valor del atributo. Una pieza de información se puede modelar como un atributo, un extremo de la asociación, o ambos.

Véase propiedad para una lista configuraciones de modelado de un atributo.

Notación

Un atributo se muestra como una cadena de texto que puede ser analizada mediante varias propiedades. La sintaxis por defecto es:

⌊«**estereotipo**»⌋$_{opc}$ **visibilidad**$_{opc}$ /$_{opc}$ **nombre** ⌊:**tipo**⌋$_{opc}$ **multiplicidad**$_{opc}$
⌊=**valor-inicial**⌋$_{opc}$ ⌊{**cadena-de-propiedades**}⌋$_{opc}$

Visibilidad. La visibilidad se muestra mediante una marca de puntuación. De forma alternativa, se puede mostrar como una palabra clave dentro de la cadena de propiedades. Se debe utilizar esta última forma para 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 clase o sus descendientes pueden ver los atributos.
– (private)	Sólo la propia clase puede ver los atributos.
~ (package)	Cualquier clase en el mismo paquete puede ver los atributos.

Nombre. El nombre se muestra como un identificador de tipo cadena.

Tipo. El tipo se muestra mediante una expresión de cadena que denota un clasificador. El nombre de la clase o tipo de dato es una expresión de cadena legítima que indica que los valores del atributo deben ser del tipo especificado. La sintaxis adicional del tipo depende del lenguaje de la expresión. Cada lenguaje tiene una sintaxis para construir nuevos tipos de datos a partir de tipos simples. Por ejemplo, C++ tiene sintaxis para punteros, arrays y funciones. Ada también tiene sintaxis para subrangos. El lenguaje de una expresión es parte del modelo interno, pero normalmente no suele mostrar en un diagrama. Se asume que es conocido para el diagrama completo o que es obvio a partir de su sintaxis.

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

Multiplicidad. La multiplicidad se muestra mediante una expresión de multiplicidad encerrada entre corchetes ([]) y situada después del nombre del tipo. Si la multiplicidad es "exactamente uno" la expresión, incluidos los corchetes, se puede omitir. Esto indica que cada objeto tiene exactamente una ranura para albergar un valor de un tipo dado (este es el caso más común). De lo contrario, es obligatorio mostrar la multiplicidad. *Véase* multiplicidad, donde hay una discusión completa sobre su sintaxis. Por ejemplo:

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

Obsérvese que una multiplicidad de 0..1 abre la posibilidad de valores nulos, la ausencia de un valor como opuesto a un determinado valor del rango. Un valor nulo no es un valor dentro del domino de la mayoría de los tipos de datos; extiende dicho dominio con un valor ajeno al mismo. Sin embargo, para los punteros el valor nulo es a menudo parte de la implementación (aunque incluso en ese caso se utiliza por convenio; por ejemplo, el valor 0 en C o C++ es un convenio de direccionamiento de memoria). La siguiente declaración permite una distinción entre el valor *nulo* y la cadena vacía, una distinción presente en C++ y en otros lenguajes.

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

Si el límite superior es mayor que uno, se pueden especificar las opciones de ordenación y unicidad como cadenas de propiedades. Las opciones de palabras claves incluyen **ordered** (**ordenado**), **bag** (**bolsa**), **sequence** (**secuencia**), **list** (**lista**). *Véase* propiedad para más detalles.

Derivación. Un atributo derivado se indica mediante una barra inclinada (/) antes del nombre.

Valor inicial. El valor inicial por defecto se muestra mediante una cadena. El lenguaje de evaluación no se suele mostrar explícitamente (pero se encuentra presente en el modelo). Si no hay valor por defecto, tanto el signo igual, como la cadena, se omiten. Si el atributo de multiplicidad incluye el valor 0 (es decir, es opcional) y no se proporciona explícitamente un valor inicial por defecto, entonces el atributo empieza con un valor vacío (cero repeticiones).

Redefinición y establecimiento de subconjuntos. Estas opciones se pueden especificar mediante cadenas de propiedad. *Véase* propiedad y redefinición (de propiedad) para más detalles.

Valor etiquetado. Se pueden vincular cero o más valores etiquetados a un atributo (como a cualquier elemento del modelo). Cada valor etiquetado se muestra con el formato **`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.

Estático. Un atributo estático (ámbito de clase) se representa subrayando el nombre y la expresión de cadena del tipo; de lo contrario, el atributo tiene ámbito de instancia.

`atributo-estático`

La Figura 14.28 muestra la declaración de algunos atributos.

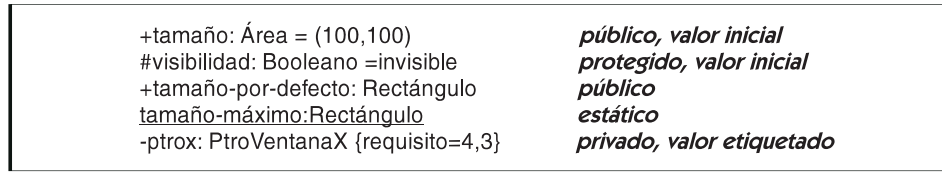


Figura 14.28 Atributos

Opciones de presentación

Sintaxis del lenguaje de programación. La sintaxis de la cadena de un atributo puede ser la del lenguaje de programación, como C++ o Smalltalk. Se pueden incluir las 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 especificar calificadores, parámetros de plantillas, parámetros de operaciones, etcétera (aunque algunos omiten ciertos términos).

Obsérvese que un atributo es semánticamente equivalente a una asociación de composición. (Puede haber algunas dudas sobre esto. La especificación incluye una notación para la composición de atributos. Sin embargo, el uso de atributos en el metamodelo incluido en la especificación indica que esto no significa que se deba tomar en serio.) Sin embargo, la intención y el uso de atributos y asociaciones son, normalmente, diferentes. Utilice los atributos para los tipos de datos, es decir, para valores sin identidad. Utilice las asociaciones para clases, es decir, para valores con identidad. La razón es que para objetos con identidad es importante ver la relación en ambas direcciones, mientras que para los tipos de datos, el tipo de datos normalmente está subordinado al objeto y no tiene conocimiento de él.

Historia

Atributos y extremos de la asociación han sido unificados semánticamente en UML2, por lo que son mucho más sencillas las conversiones entre ellos en un modelo.

La opción de la multiplicidad aparece ahora después del nombre de tipo, en lugar de después del nombre del atributo. Esto hace posible tratar la multiplicidad como parte de la especificación global del tipo. Se ha añadido la redefinición y el establecimiento de subconjuntos. El concepto de ámbito destino se ha perdido.

atributo de clase

Un atributo cuyo valor se comparte entre todas las instancias de una clase. También se denomina característica estática.

Semántica

Un atributo de clase (o atributo estático para los usuarios de C++ y Java) es un atributo para el que se comparte una ranura entre todas las instancias de una clase (pero no es parte de ninguna de ellas en particular). Por tanto, tal atributo no es una propiedad de una instancia, si no que es una propiedad de la clase. El acceso a un atributo de clase no necesita de una instancia, aunque algunos lenguajes de programación proporcionan una sintaxis que utiliza, por conveniencia, una instancia. Para la mayoría de los propósitos, un atributo de clase es una variable global en el espacio de nombres de la clase.

Un valor por defecto de un atributo de clase representa el valor del atributo cuando se inicializa la propia clase. El momento en el que esto sucede no está definido en UML.

Un atributo de clase se modela como un atributo en el que el indicador **isStatic** está a verdadero.

Notación

Un atributo de clase se muestra en la sección de atributos del icono de clase utilizando la sintaxis de un atributo normal con la cadena completa del atributo subrayada. *Véase* la Figura 14.29.

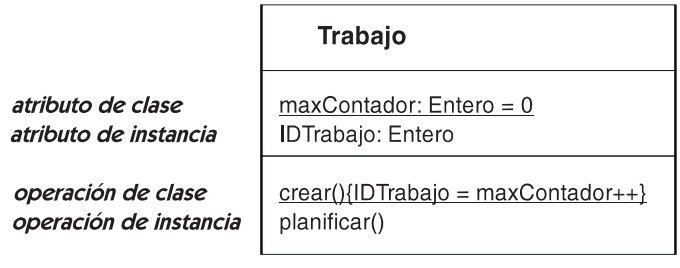


Figura 14.29 Características de clase

autotransición

Una transición en que el estado origen y el estado destino es el mismo. Es considerado un cambio de estado. Cuando se dispara, el estado origen se termina y vuelve a entrar, por consiguiente las acciones y acciones de entrada y de salida se invocan. No es equivalente a una transición interna en que no ocurre ningún cambio de estado.

auxiliary (estereotipo de Clase)

Una clase que soporta a otra clase con un enfoque más centrado, típicamente mediante la provisión de mecanismo de control.

Véase focus.

become (se convierte en)

Esta dependencia de UML1 ha sido eliminada en UML2.

bien formado

Designa un modelo que está construido correctamente, uno que satisface todas las reglas y restricciones predefinidas y especificadas por el modelo. Dicho modelo tiene semántica significativa. Un modelo que no está bien formado se dice que está mal formado.

bifurcación

Una situación en una máquina de estados en la cual un único disparador lleva a más de una posible salida, cada una de las cuales tiene su propia condición de guarda. Las bifurcaciones pueden ser estáticas o dinámicas.

Una situación en una actividad en la cual un flujo de entrada puede llevar a un flujo de varios posibles flujos de salida, cada uno de los cuales tiene su propia condición de guarda.

Véase también conjunción, división, elección, fusión, nodo de decisión, unión.

Semántica

Máquina de estados. Si el mismo evento puede tener diferentes efectos que dependen de diferentes condiciones de guarda, se pueden modelar como transiciones separadas con el mismo disparador de evento. Sin embargo, y en la práctica, es conveniente permitir que un único disparador conduzca a varias transiciones. Esto es especialmente correcto en el caso en el que las condiciones de guarda cubren todas las posibilidades, de forma que se garantice que una ocurrencia del evento dispare alguna de las transiciones. Una bifurcación es una parte de una transición que divide la ruta de la transición en dos o más segmentos, cada uno con una condición de guarda distinta. El disparador del evento se sitúa en el primer segmento común de la transición. La salida de un segmento de bifurcación se puede conectar a la entrada de otra bifurcación para formar un árbol. Cada camino a través del árbol representa una transición distinta.

La bifurcación puede ser estática o dinámica. En el caso de que sea estática, el punto de bifurcación se modela mediante un vértice de conjunción. La conjunción de todas las condiciones de una ruta en una transición es equivalente a una única condición que se evalúa conceptualmente antes de que se dispare la transición. Si no hay una ruta para la cual todas las condiciones sean verdaderas, no se dispara ninguna transición. La ruta elegida no se ve afectada por los cambios de los valores causados por las acciones que se ejecutan durante la transición. Una transición se dispara en un único paso, a pesar de su apariencia de árbol de bifurcaciones. El árbol es simplemente una facilidad de modelado. Una ruta que sale de cada vértice se puede etiquetar con la pseudocondición else. Se permite que esta ruta se dispare si, y sólo si, no se habilita ninguna otra ruta.

En el caso de que sea dinámica, el punto de bifurcación se modela utilizando un vértice de elección. Un vértice de elección representa un estado temporal. Si todas las condiciones de

una ruta que llevan a un vértice de elección son verdaderas, la transición puede disparar el vértice de elección. Se ejecuta cualquier efecto sobre la ruta, que afectará a las condiciones en el resto de la transición más allá del punto de elección. Las condiciones de los subsiguientes segmentos se evalúan y se selecciona una ruta para continuar la transición (que puede llevar a otra elección). Si no se habilita ninguna ruta, la máquina de estados está mal formada (es imposible deshacer los efectos de una ruta parcialmente ejecutada). Para evitar esta situación, es recomendable que se incluya una condición else, a menos que sea cierto que se habilitará una ruta.

Un vértice de elección se puede considerar un estado "real", en el cual se debe habilitar una transición inmediata a otro estado, mientras que un vértice de conjunción es simplemente una conveniencia sintáctica para la organización de las rutas que comparten algunas condiciones.

Actividad. Las bifurcaciones se modelan como nodos de decisión. Un nodo de decisión tiene un flujo de entrada y varios posibles flujos de salida, cada uno de los cuales con su propia condición de guarda. Los nodos de decisión siempre se encuentran activos. Cualquier efecto de un comportamiento en un flujo de entrada puede afectar a las condiciones de guarda de los flujos de salida. Es importante que las condiciones de guarda cubran todas las posibilidades, de forma similar a una elección en una máquina de estados, ya que de otra forma, la actividad está mal formada. Se puede utilizar una condición else en un grafo de actividad para garantizar que una condición de guarda será verdadera.

Notación

Máquina de estados. Una bifurcación estática se puede mostrar representando un disparador de evento en varios arcos de transición con diferentes condiciones de guarda. También se puede mostrar mediante transiciones de finalización, como en un diagrama de actividad.

Sin embargo, para una mayor facilidad, se puede conectar la punta de una flecha de transición con un pequeño círculo negro que indica una conjunción. La flecha de transición se etiqueta con el evento de disparo si lo hay, pero no debería tener ninguna acción asociada. Cualquier acción se sitúa en el segmento final de la transición (a menos que sean compartidas por todas las rutas).

El efecto de un árbol de bifurcaciones es el mismo que expandir el árbol en arcos de transición independientes para cada ruta a través del árbol, todas compartiendo el mismo evento de disparo de evento, pero cada uno con su propia unión de condiciones de guarda, acción y estado destino. La Figura 14.30 muestra dos formas de representar la misma situación. Obsérvese que la condición puede incluir los parámetros del evento de disparo.

Una bifurcación dinámica se puede mostrar conectando la punta de la flecha de transición a un rombo o diamante, el cual indica una elección. La flecha entrante puede tener condiciones y acciones, las cuales se ejecutan antes de que se realice la elección.

Pueden salir dos o más flechas del símbolo del rombo. Cada flecha se etiqueta con una condición de guarda. Se puede utilizar la palabra reservada **else** como condición de guarda. Su valor es verdadero si todas las demás condiciones de guarda (explícitas) son falsas. La punta de cada flecha se puede conectar a otra bifurcación o a un estado. Una flecha conectada a un estado puede tener una etiqueta de acción vinculada. Ninguna flecha que salga de una elección puede tener un disparador de evento.

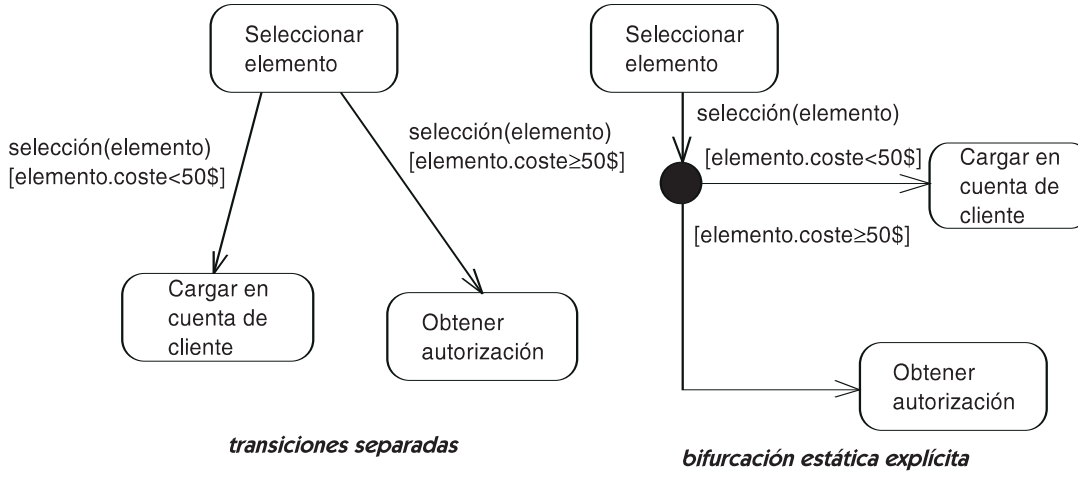


Figura 14.30 Dos formas de mostrar una bifurcación estática

La Figura 14.31 muestra una variación del ejemplo anterior, utilizando esta vez una elección. En este caso, la acción precioDelElemento se realiza como parte de la transición para calcular el precio. Dado que el precio no se conoce antes de que el segmento inicial de la transición se dispare, la bifurcación debe ser dinámica, por lo que se utiliza un vértice de elección. Es fácil verificar que las condiciones cubren todas las posibilidades, por lo cual una ruta else es innecesaria.

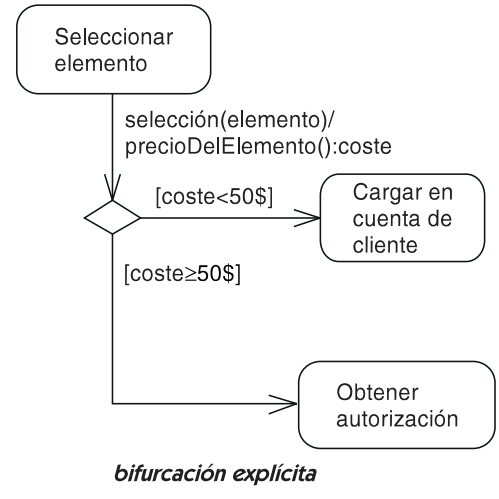


Figura 14.31 Bifurcación dinámica

Diagrama de actividad. Tanto las decisiones (bifurcaciones dinámicas), como las fusiones (el inverso de una bifurcación), se pueden mostrar como rombos o diamantes, como se muestra en la Figura 14.32. En el caso de una fusión, hay dos o más flechas entrantes y una única flecha saliente. No son necesarias las condiciones de guarda en una fusión.

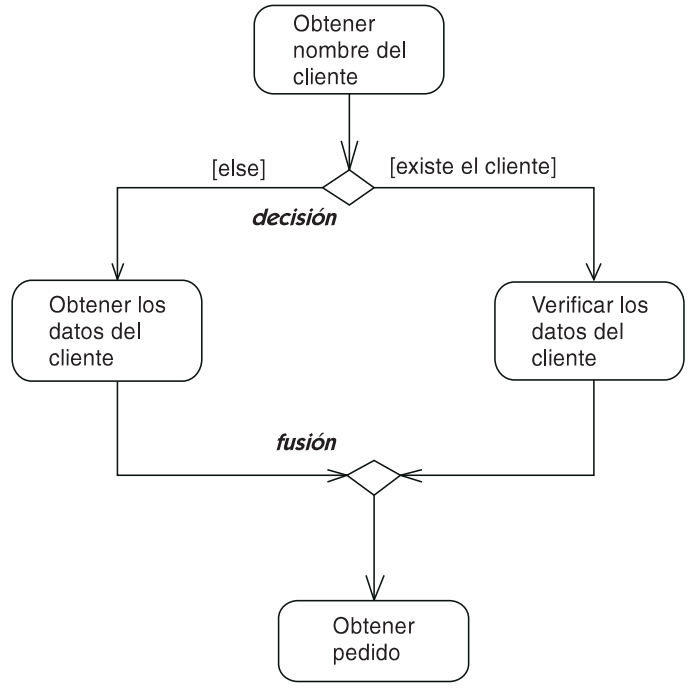


Figura 14.32 Decisión y fusión en un diagrama de actividad

bind (ligar)

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

Véase ligadura.

bolsa

Una colección de elementos en la que puede haber varias copias de elementos idénticos, como lo opuesto a un conjunto, en el que cada elemento debe ser único.

Semántica

Las bolsas se pueden indicar en las especificaciones de la multiplicidad utilizando el indicador de unicidad con el valor falso. En combinación con el indicador de ordenación, los modeladores pueden especificar conjuntos no ordenados, conjuntos ordenados, bolsas no ordenadas o bolsas ordenadas. Las bolsas ordenadas se denominan secuencias o listas. La multiplicidad es aplicable a valores de atributos, extremos de las asociaciones, parámetros y otros elementos.

Historia

El soporte para las bolsas en la multiplicidad es nuevo en UML2.

break

Un tipo de operador de interacción en el que el subfragmento operando se realiza y el resto del fragmento que lo encierra no se realiza.

Semántica

El operador break tiene un subfragmento operando con una condición de guarda. Si la condición de guarda se evalúa como verdadera, se ejecuta el operando y se omite el resto del fragmento que lo encierra. Si la condición de guarda se evalúa como falsa, el operando no se ejecuta y se ejecuta el resto del fragmento que lo encierra. Este operador debe ser considerado como una versión reducida de un operador alternativo en el cual el resto del fragmento que lo encierra no necesita ser un subfragmento explícito.

El operador break debe cubrir todas las líneas de vida del fragmento que lo encierra.

Notación

La palabra clave **break** aparece como la etiqueta del operando de interacción.

La Figura 14.33 muestra un ejemplo de break. Si el usuario presiona el boton cancelar después de recibir una petición de identificación, se ignora el resto de la secuencia. De lo contrario, el usuario proporciona la identificación y la secuencia de validación continúa.

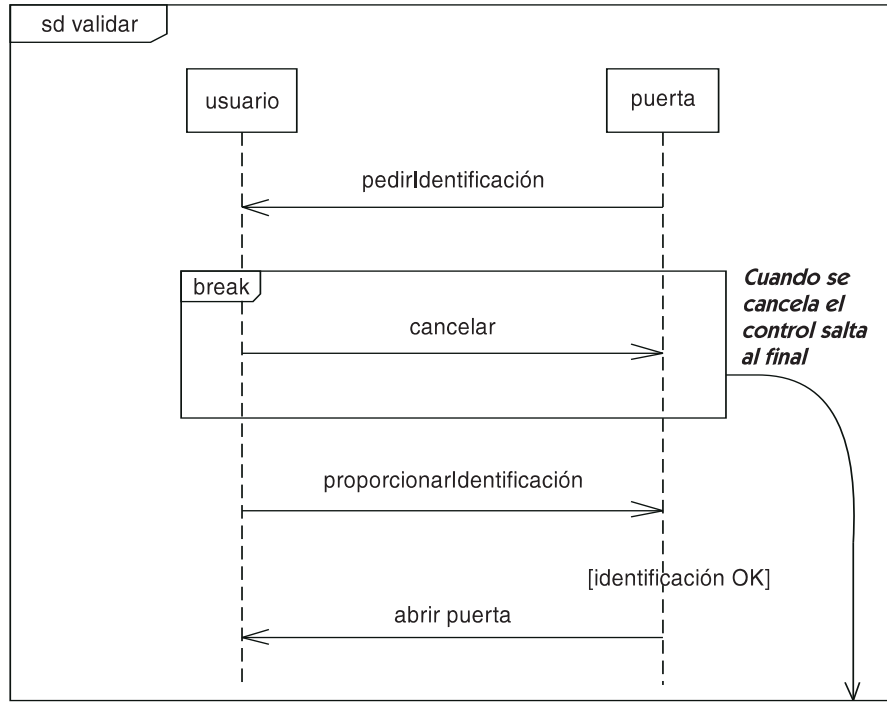


Figura 14.33 Break

bucle

Construcción de comportamiento en la que se ejecuta de forma repetida una pieza de comportamiento mientras una condición especificada permanezca siendo verdadera. Este artículo cubre bucles en modelos de interacción.

Véase también nodo repetitivo para ver los bucles en los modelos de actividad.

Semántica

En una interacción, un bucle es una variedad de fragmento combinado en el que el único cuerpo del fragmento se ejecuta mientras que una condición de guarda sea verdadera. Un fragmento de bucle puede tener un límite inferior y superior sobre el número de repeticiones, así como una condición lógica que utiliza valores de una de las líneas de vida del fragmento. El cuerpo del bucle se ejecuta de forma repetida mientras que la condición lógica se evalúe a verdadera al principio de cada repetición, pero siempre se ejecuta al menos el límite inferior y nunca se ejecuta más que el límite superior. Si el límite inferior no está presente, se entiende que es cero. Si no está presente el límite superior, se entiende que es ilimitado.

Observe que un fragmento de bucle en una interacción es algo más sencillo que un nodo repetitivo en una actividad, que tiene secciones de inicialización, actualización y cuerpo.

Notación

Un bucle se representa como un rectángulo con la etiqueta loop en la esquina superior izquierda. A continuación de la palabra clave loop pueden ir los límites sobre el número de iteraciones del bucle con la sintaxis:

loop	Mínimo = 0, máximo ilimitado
loop (**`repetir`**)	Mínimo = máximo = repetir
loop (**`mínimo`**, **`máximo`**)	Límites mínimo y máximo explícitos, con mínimo ≤ máximo

Además de los límites, se puede incluir una expresión lógica como guarda en una línea de vida. El bucle continuará iterando si la expresión es verdadera, pero iterará al menos el límite mínimo y no más del límite máximo, independientemente de la expresión de guarda.

Las líneas de vida que atraviesan el rectángulo están disponibles para el bucle. Aquellas que están fuera del rectángulo son inaccesibles.

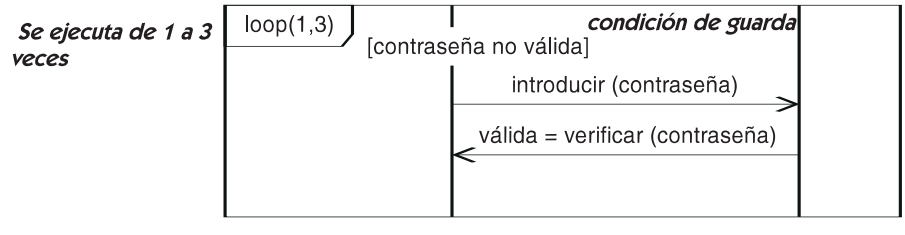


Figura 14.34 Bucle en un diagrama de secuencia

La Figura 14.34 muestra un bucle con límites y una condición de guarda. Este bucle debe ejecutarse al menos una vez, y por tanto la condición no se evalúa la primera vez. El bucle termina con el primer éxito o después de tres fallos, y el estado final está disponible como resultado.

buffer

Véase nodo buffer central.

Semántica

La capacidad de buffering se asume en el manejo de mensajes recibidos por los objetos, pero los detalles de esta capacidad dependen de la implementación. El diseño cuidadoso de modelos puede evitar la necesidad de buffers en muchos casos comunes, pero hay situaciones en las que es semánticamente necesario.

La capacidad de buffering es implícita en el manejo de tokens dentro de una actividad, particularmente si se incluyen tokens de streaming. En muchos casos comunes, sólo puede existir un token en cada momento en una posición, por lo que no siempre es necesario el buffering. Un buffer explícitamente compartido se puede modelar utilizando un nodo buffer central.

buildComponent (estereotipo de Componente)

Componente que define un conjunto de componentes por motivos de organización o de desarrollo a nivel de sistema.

cadena

Una secuencia de caracteres de texto. El detalle de la representación de las cadenas depende de la implementación y pueden incluir conjuntos de caracteres especiales que soporten caracteres internacionales y gráficos.

Semántica

Muchas propiedades semánticas, especialmente nombres, tienen cadenas en sus valores. Una cadena es una secuencia de caracteres ordenados en un conjunto determinado para mostrar información sobre el modelo. Los juegos de caracteres pueden incluir caracteres no románicos. UML no especifica la codificación de una cadena, pero asume que la codificación es lo suficientemente general para permitir un uso razonable. En principio, la longitud de una cadena debería ser ilimitada; cualquier límite real debe ser lo suficientemente largo para ser no restrictivo. Las cadenas también deben incluir la posibilidad de caracteres en varios idiomas humanos. Los identificadores (nombres) deben consistir completamente es caracteres de un juego finito. Los comentarios y elementos similares son cadenas descriptivas sin semántica directa como la que pueden contener otro tipo de elementos como diagramas, gráficos, fotografías, videos y otros tipos de documentos embebidos.

Notación

Una cadena gráfica es una notación primitiva con poca flexibilidad para su implementación. Se asume que es una secuencia lineal de caracteres en algún lenguaje, con la posibilidad de incluir documentos embebidos de varios tipos. Es deseable soportar varios idiomas humanos, pero los detalles de implementación se dejan a las herramientas de edición. Las cadenas gráficas pueden estar formadas por una línea, una lista de líneas, o pueden ser etiquetas asociadas a otros símbolos.

Las cadenas se utilizan para mostrar propiedades semánticas que tienen valores de cadena y también para codificar los valores de otras propiedades semánticas que se desee mostrar. La correspondencia entre las cadenas semánticas y la notación de las cadenas es directa. La correspondencia de otras propiedades con respecto a las cadenas de notación están gobernadas por las gramáticas, descritas en los artículos de varios elementos. Por ejemplo, la notación para mostrar un atributo codifica su nombre, tipo, valor inicial, visibilidad y ámbito en una sola cadena.

Es posible la extensión no canónica de la codificación, por ejemplo se puede mostrar un atributo usando la notación de C++. Algunas de esas codificaciones pueden perder parte de la información del modelo, sin embargo las herramientas deben soportarlo como una opción elegible por el usuario a la vez que mantienen el soporte a la notación UML canónica.

El tipo y tamaño de la fuente son marcadores gráficos que son usualmente independientes de la propia cadena. Se pueden codificar para varias propiedades del modelo, algunas de las cuales se sugieren en este documento y otras queden a la elección de la herramienta o del usuario. Por ejemplo las letras cursivas muestran clases y operaciones abstractas y el subrayado muestra las características estáticas.

Las herramientas pueden tratar las cadenas largas de varias formas, tales como truncado a un tamaño fijo, correspondencia automática e inserción de barras de desplazamiento. Se asume que existe un medio para obtener la cadena completa cuando se quiera.

cadena de propiedad

Una sintaxis de texto para mostrar un valor o un elemento unido a un valor, sobre todo valores etiquetados, pero también incluyendo atributos incorporados de elementos del modelo.

Notación

Una o más especificaciones de propiedad separadas por comas y encerradas entre llaves ({}).

Cada declaración de propiedad tiene la forma

`propiedad-nombre` = **`valor`**

o

`propiedad-literal`

donde la propiedad literal es un único valor enumerado cuya apariencia implica un único nombre de propiedad.

Ejemplo

{abstract, author=Joe, visibility=private}

Opciones de presentación

Una herramienta puede presentar las especificaciones de propiedad en líneas separadas con o sin las llaves abrazándola, con tal de que se marquen para distinguirlos apropiadamente de otra información. Por ejemplo, las propiedades para una clase podrían listarse bajo el nombre de la clase en una tipografía distintiva, tal como cursivas o una familia de fuente diferente.

Éste es un problema de la herramienta.

Note que las cadenas de propiedad pueden usarse para desplegar los atributos incorporados, así como valores etiquetados, pero tal uso debe evitarse si la forma canónica es simple.

calificador

Una sección para un atributo o lista de atributos, en una asociación binaria en que los valores de los atributos seleccionan un único objeto relacionado o un conjunto de objetos relacionados con el conjunto entero de objetos relacionados con un objeto por la asociación. Es un índice delante de la línea transversal de una asociación.

Véase clase de asociación, fin de la asociación.

Semántica

Una asociación binaria mapea un objeto para un conjunto de objetos relacionados. A veces es deseable para seleccionar un objeto del conjunto proporcionando un valor que distingue los objetos en el conjunto. Este valor podría ser un atributo de la clase designada. En general, sin embargo, el valor del seleccionador puede ser parte de la propia asociación, un atributo de la asociación de quien el valor es proporcionado por el creador cuando un nuevo eslabón se agrega a la clase de asociación. Un atributo tal en una asociación binaria se llama calificador. Un objeto, junto con un valor del calificador, determina un único objeto relacionado o (algo a menudo) un subconjunto de objetos relacionados. El valor califica la asociación. En un contexto de implementación, tal atributo se denomina valor de índice.

Un calificador se usa para seleccionar un objeto u objetos del conjunto de objetos relacionados con un objeto (llamado *qualified objet*) por una asociación (Figura 14.35). El objeto seleccionado por el valor del calificador se llama *target objet*. Un calificador siempre actúa sobre una asociación cuya multiplicidad es *many* en la dirección designada. En el caso más simple, cada valor del calificador selecciona un solo objeto del conjunto designado de objetos relacionados. En otras palabras, un objeto calificado y un valor calificado producen un único objeto destino relacionado

Muchos tipos de nombres son calificadores. Tal como un nombre dentro de un contexto que se corresponde a un único valor. El objeto calificado suple el contexto, el calificador es el nombre y el objeto designado es el resultado. Cualquier ID u otro código único es un calificador; su propósito es seleccionar un valor singularmente. Una serie puede diseñarse como una asociación calificada.

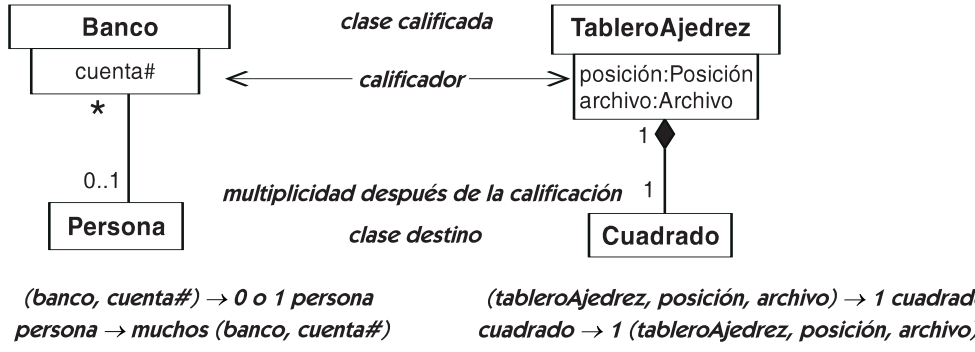


Figura 14.35 Asociación calificada

La serie es el objeto calificado, el índice de la serie es el calificador y el elemento arreglo es el objeto designado. Para una serie, el tipo del calificador es un rango de enteros.

Un calificador puede usarse en una expresión de navegación para seleccionar un subconjunto de objetos relacionados con un objeto por una asociación —a saber, aquéllos que llevan un particular valor para el valor de atributo del calificador o lista de valores. El calificador es un seleccionador dentro del conjunto de objetos relacionados por la asociación. Parte el conjunto en subconjuntos a través del valor del calificador. En la mayoría de los casos, el propósito de un calificador es seleccionar un único objeto del conjunto de objetos relacionados, para que una asociación calificada se comporte como una mesa de operaciones de búsqueda.

Estructura

Calificador. Un atributo del calificador es una parte optativa de un fin de asociación binario. El calificador califica la clase unida al fin de la asociación. Un objeto de la clase y un valor del calificador, que selecciona un objeto o conjunto de objetos de la clase al otro fin de la asociación binaria. Es posible para ambos fines de una asociación binaria que tenga calificadores, pero es raro.

Un calificador es un atributo de la asociación o lista de atributos. Cada atributo tiene un nombre y un tipo pero ningún valor inicial, cuando los calificadores no son objetos autoestables, y cada valor del calificador debe ser explícito cuando un eslabón se agrega a la asociación.

No se usan calificadores con asociaciones *n*-ary.

Multiplicidad. Se pone la multiplicidad de la relación calificada en el opuesto al fin de la asociación binaria del calificador. (El código mnemónico es el de la clase calificada y junto al calificador forma un valor compuesto que está relacionado con la clase designada.) En otros términos, el calificador se enlaza al “near end” de la asociación y la multiplicidad y nombre de rol se juntan al “far end”.

La multiplicidad unida al fin de la asociación designada denota cuántos objetos podrían ser seleccionados por un (objeto de la fuente, valor del calificador) par.

Los valores de multiplicidad común incluyen 0..1 (un único valor puede seleccionarse, pero cada posible valor del calificador necesariamente no selecciona un valor), 1 (cada posible valor del calificador selecciona un único objeto designado, por consiguiente el dominio de valores del

calificador debe ser finito) y * (el valor del calificador es un índice que particiona los objetos designados en subconjuntos).

En la mayoría de los casos, la multiplicidad es poner cero-o-uno. Esta opción significa que un objeto y valor del calificador pueden rendir, a lo sumo, un objeto relacionado. Una multiplicidad de uno significa que los medios que cada posible valor del calificador rinden un objeto exactamente. Esto obviamente exige al tipo del calificador ser un dominio finito (en una aplicación de la computadora sin embargo). Esta multiplicidad puede ser útil para mapear tipos enumerados finitos —por ejemplo, un **Pixel** calificado por **Color Primario** (la enumeración de rojo, verde y azul) produciría la terna de valor rojo-verde-azul para cada pixel en una imagen.

La multiplicidad de la asociación inhábil no se declara explícitamente. Pero es normalmente asumido ser muchos o por lo menos más de uno. No habría por otra parte ninguna necesidad para un calificador.

Una multiplicidad de muchos en una asociación calificada no tiene ningún significante semántico de impacto, porque el calificador no reduce la multiplicidad del conjunto designado.

Tal multiplicidad representa una declaración del diseño que un índice para cruzar la asociación debe proporcionarse. En ese caso, el calificador parte el conjunto de objetos designados en los subconjuntos. Semánticamente, esto no agrega nada más allá de tener un atributo de la asociación que también (implícitamente) parte los eslabones. La connotación del diseño de un calificador en un modelo de diseño es que el transversal debe ser eficiente. Esto es, debe no requerir una búsqueda lineal entre todos los valores designados. Normalmente se implementa por alguna clase de mesa de ordenamiento. Un índice en una base de datos o la estructura de los datos es propiamente diseñado como un calificador.

En sentido inverso a través de una asociación calificada (es decir, yendo de la clase designada al objeto calificado), la multiplicidad indica el número de (objeto calificado, calificador) pares que mapean en el mismo objeto designado, no el número de objetos calificados. En otros términos, si varios (objeto calificado, calificador) pares mapean entonces la multiplicidad inversa es mapeada. Una multiplicidad inversa de uno identifica el sentido del calificador que es exactamente uno apareando objetos calificados y valor del calificador, que relacionan al objeto designado.

Notación

Un calificador se muestra como un rectángulo pequeño unido al fin de un camino de la asociación entre el último segmento del camino y el símbolo de la clase calificada. El rectángulo calificador es la parte del camino de la asociación, no la parte de la clase. El calificador se une a la clase que él califica —esto es, un objeto de la clase calificada, junto con un valor del calificador, únicamente selecciona un conjunto de clase designada en el otro fin de asociación.

Los atributos del calificador se listan dentro de la caja del calificador. Puede haber uno o más atributos en la lista. Los atributos del calificador tienen la misma anotación como atributos de clase sólo que las expresiones de valor iniciales no son significantes.

Opciones de la presentación

Un calificador no puede suprimirse. Proporciona los detalles esenciales, la omisión de los cuales modificaría el carácter inherente de la relación.

Una herramienta puede usar una línea más delgada para los rectángulos del calificador que para los rectángulos de la clase para distinguirlos claramente.

El rectángulo del calificador, preferentemente, debe ser más pequeño que el rectángulo de la clase a que se une, aunque esto no siempre es práctico.

Discusión

Las multiplicidades en una asociación calificada se tratan como si el objeto calificado y el calificador fuesen una sola entidad, una llave compuesta. En la dirección delantera, la multiplicidad en el fin designado representa el número de objetos relacionados con el valor compuesto (objeto calificado + valor del calificador). En sentido inverso, la multiplicidad describe el número de valores compuestos (objeto calificado + el calificador) relacionado con cada objeto designado, no el número de objetos calificados relacionados con cada objeto designado. Esto es porque el calificador se pone en el mismo fin de la asociación camino al símbolo de la clase —puede pensar en el path de la asociación conectando el valor compuesto a la clase designada.

No hay ninguna provisión para especificar la multiplicidad de la relación inhábil.

En la práctica, sin embargo, es normalmente muchos, en la dirección delantera. No hay apunte para tener una asociación calificada a menos que se relacionen muchos objetos del blanco con un objeto calificado. Para un modelo lógico, el propósito del calificador es reducir la multiplicidad a uno, agregando el calificador para que una pregunta pueda asegurarse de volver un solo valor en lugar de un conjunto de valores. La singularidad del valor del calificador frecuentemente es una condición semántica crucial que es difícil de capturar sin los calificadores.

Casi todas las aplicaciones tienen muchas asociaciones calificadas. Muchos nombres realmente son calificadores. Si un nombre es único dentro de algún contexto, es un calificador y el contexto debe identificarse y debe diseñarse apropiadamente. No todos los nombres son calificadores.

Por ejemplo, los nombres de personas no son únicos. Porque los nombres personales son ambiguos, la mayoría de los datos que procesan las aplicaciones usan alguna clase de número de identificación, como un número de cliente, un número de Seguro social o un número de empleado. Si una aplicación requiere el lookup de información o la recuperación de los datos basados en llaves de búsqueda, el modelo generalmente debe usar las asociaciones calificadas.

Cualquier contexto en el que nombres o códigos de identificación se definen para seleccionar las cosas fuera de conjuntos normalmente debe diseñarse como una asociación calificada.

Observe que el valor del calificador es una propiedad del eslabón, no del objeto designado.

Considere un sistema de archivo Unix en el que cada directorio es una lista de entradas cuyos nombres son únicos dentro del directorio, aunque los mismos nombres puedan usarse en otros directorios. Cada entrada apunta a un archivo que puede ser un archivo de datos u otro directorio. Más de una entrada puede apuntar al mismo archivo. Si esto pasa, el archivo tiene varios seudónimos. El sistema de directorio Unix está diseñado como una asociación muchos-a-uno, en que el directorio calificado por los rendimientos del nombre de fichero es un fichero. Note que el nombre de fichero no es parte del fichero; es parte de la relación entre un directorio y un archivo. Un fichero no tiene un solo nombre. Puede tener muchos nombres en muchos directorios (o incluso varios nombres en el mismo directorio). El fichero no es un atributo del fichero.

Una motivación mayor para las asociaciones calificadas es la necesidad de diseñar una importante situación semántica que tiene una importante estructura de datos de aplicación natural.

En la dirección delantera, una asociación calificada es una tabla de búsqueda. Para un objeto calificado, cada valor del calificador rinde un solo objeto designado (o un valor nulo si el valor del calificador está ausente en el conjunto de valores). Las mesas de búsqueda son implementables por la estructura de datos, como mesas de desmenuzar, b-árboles, y listas ordenadas que proporcionan mucha mayor eficacia que listas sin ordenar que deben buscarse linealmente. En casi todos los casos, es diseño pobre para usar una lista unida u otras estructuras de datos sin ordenar para búsquedas en nombres o códigos, aunque, tristemente, muchos programadores los usan. Diseñando situaciones apropiadas con asociaciones calificadas y usando estructuras de datos eficaces para implementarlos es crucial para la buena programación.

Para un modelo lógico, hay un punto pequeño teniendo una asociación calificada con una multiplicidad de muchos en la dirección delantera, porque el calificador no agrega cualquier información semántica que un atributo de la asociación no podría mostrar. En un modelo pensado para el diseño de algoritmos y estructuras datos de, sin embargo, un calificador lleva una connotación-namely adicional, el intento de que la selección sea eficaz.

En otras palabras, una asociación calificada denota una estructura de datos puesta en un índice perfeccionada para la búsqueda en el valor del calificador. En este caso, una multiplicidad de muchos puede ser usada para representar un conjunto de valores que deben ser accesibles juntos bajo un valor de índice común, sin tener que buscar otros valores.

Generalmente, un atributo del calificador no debe ser incluido como un atributo de clase designada, con su presencia en la asociación es suficiente. En caso de un valor del índice sin embargo, puede ser necesario para tomar un valor del que es inherentemente un atributo la clase designada y le hace un valor del calificador redundante. Los valores del índice son inherentemente redundantes.

Restricciones

Algunas situaciones complicadas no son claras para diseñar con cualquier conjunto de relaciones no redundantes. Son mejores las asociaciones calificadas usando diseño para capturar los caminos de acceso básicos con restricciones adicionales declaradas explícitamente. Porque estas situaciones son raras, se sentía que intentando incluirlos en una anotación, eso podría capturar todas las posibles restricciones de multiplicidad, directamente no merecía la pena la complejidad agregada.

Por ejemplo, considere un directorio en el que cada nombre de fichero identifica un único fichero. Un fichero puede corresponder a pares de directorio-nombre de archivo múltiples. Éste es el modelo básico que hemos visto antes. Este modelo se muestra en la Figura 14.36.

Ahora, sin embargo, deseamos agregar las restricciones adicionales. Suponga que cada fichero (archivo) deba estar en un solo directorio, pero dentro de ese directorio podría tener muchos nombres, esto es, hay más de una manera de nombrar el mismo fichero. Esto puede ser diseñado con una asociación redundante entre **Fichero** y **Directorio**, con multiplicidad uno en la dirección del **Directorio** (Figura 14.37). La redundancia de las dos asociaciones es indicada por la restricción **{same}** que implica que los dos elementos son el mismo pero a niveles diferentes de detalle. Porque estas asociaciones son redundantes, sólo la asociación calificada se llevaría a cabo; la otra habría de tratarse como una restricción tiempo-ejecución en sus contenidos.

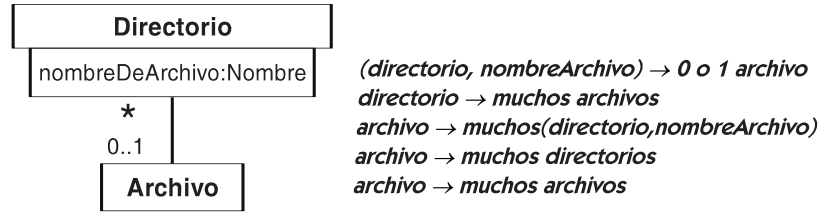


Figura 14.36 Calificador simple

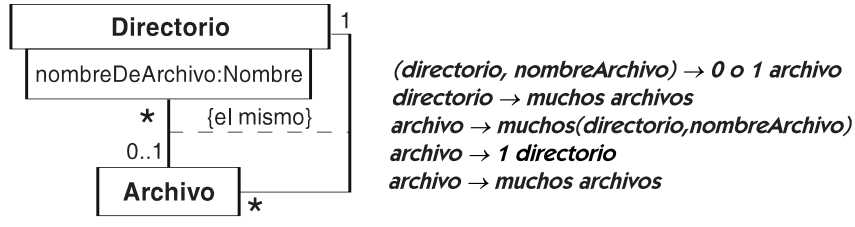


Figura 14.37 Archivo con varios nombres en un directorio

Una restricción similar es que cada fichero puede aparecer en directorios múltiples, pero él siempre tiene el mismo nombre que aparece dondequiera que sea. Otros ficheros pueden tener el mismo nombre pero deben estar en directorios diferentes. Esto puede ser diseñado haciendo el **nombre de archivo** un atributo de **Fichero** pero reprimiendo el atributo de la clase y el calificador para ser el mismo (Figura 14.38). Este modelo frecuentemente ocurre como un índice de la búsqueda, aunque en un índice general la multiplicidad del blanco calificado sería muchos. Esta situación por consiguiente, tiene el contenido más semántico que un índice que es un dispositivo de implementación.

Figura 14.38 Archivo con el mismo nombre en todos los directorios

Un tercer caso permitiría un fichero para aparecer en directorios múltiples bajo varios nombres, pero el fichero sólo podría aparecer una vez dentro de un solo directorio. Esto podría ser diseñado con asociación calificada redundante y clase de asociación que la porción del mismo nombre de fichero del atributo (Figura 14.39).

Estos ejemplos se han mostrado con relaciones redundantes para ilustrar la naturaleza de las restricciones. En la práctica, sin embargo, es normalmente satisfactorio declarar la restricción textualmente, con la asociación calificada mostrada gráficamente.

Figura 14.39 Archivo con un nombre en cualquier directorio como mucho

call (estereotipo de Dependencia de uso)

Una dependencia estereotipada, cuyo origen es una operación y cuyo destino es una operación. Una dependencia de llamada especifica que el origen invoca la operación destino. Una dependencia de llamada puede conectar una operación origen con cualquier operación destino que está dentro del ámbito, incluyendo, pero no limitado a, operaciones del clasificador que las engloba y operaciones de otros clasificadores visibles.

Véase llamada, uso.

Discusión

Este estereotipo está mal concebido. Las operaciones no llaman a operaciones; los métodos (implementación de las operaciones) llaman a operaciones. Una operación es una especificación de un comportamiento y no implica ninguna implementación en particular. Este estereotipo combina erróneamente especificación e implementación. Evítelo.

calle

Véase partición.

camino de la comunicación

Tipo de asociación que permite a los nodos intercambiar señales y mensajes.

característica

Propiedad, como una operación o atributo que está encapsulada como parte de una lista en un clasificador, como una interfaz, una clase o un tipo de datos.

característica de clase

Una característica de la propia clase, en lugar de serlo de las instancias individuales. *Véase* atributo de clase y operación de clase. También recibe el nombre de característica estática.

Notación

Una característica de clase es una cadena de texto con la misma sintaxis que la correspondiente característica de instancia, excepto porque se subrayada la cadena. *Véase* la Figura 14.29.

característica de comportamiento

Un elemento con nombre del modelo que especifica un comportamiento dinámico, como una operación o una recepción, y que es parte de un clasificador. También tiene una lista de excepciones que se podrían elevar durante la ejecución del comportamiento especificado por la característica.

Véase operación, recepción.

Semántica

Una característica de comportamiento tiene una lista de parámetros, incluyendo cero o más parámetros de retorno. También puede tener una especificación del tipo de concurrencia que especifica si están permitidas las invocaciones concurrentes sobre la misma instancia pasiva. La especificación de la concurrencia en una característica de comportamiento se ignora en una instancia activa, la cual controla su propia concurrencia.

Una característica de comportamiento es propiedad de un clasificador. Un comportamiento se puede vincular a una característica de comportamiento como un método dentro de dicho clasificador o cualquier número de sus descendientes. Un método es una implementación de una característica de comportamiento. El método declarado en un clasificador para una característica de comportamiento se hereda por sus clasificadores descendientes a menos que sea reemplazado por un nuevo método.

Una característica de comportamiento puede ser abstracta o concreta con respecto a un determinado clasificador. Si es abstracta, no tiene un método dentro del clasificador, y no debe ser invocado en una instancia directa del clasificador. Si es concreta, debe tener una implementación en el clasificador o en una clase antecesora. Una característica de comportamiento abstracta en un clasificador se convierte en concreta en un subclasificador si se especifica un método en la subclase. De forma excepcional, una característica de comportamiento concreta en un clasificador se puede declarar como abstracta en un subclasificador, en cuyo caso no puede ser utilizada hasta que en un clasificador descendiente de éste la vuelve a hacer concreta de nuevo proporcionandole un método.

característica estática

Una característica que es compartida por una clase, y no aplicable a un solo objeto.

Semántica

La característica estática indica si hay un espacio distinto para guardar la cualidad para cada instancia de una clase (no estática) o si hay un espacio para la clase en sí (estático). Para una operación, la característica estática indica si una operación tiene acceso a una instancia (no estático)

o a la clase en sí misma, tal como un operador de creación (estático). A veces simplemente llamado alcance. Los valores posibles son:

No estático	Cada instancia del clasificador tiene su propia copia distinta de la cualidad. Los valores en una ranura son independientes de los valores en otras ranuras. Éste es el caso por defecto. Para un operador, el operador se aplica a un objeto individual.
Estático	El propio clasificador tiene una copia de la cualidad. Todas las instancias del clasificador tienen acceso a la única cualidad. Si el lenguaje lo permite trabajar con clases como objetos, ésta es una cualidad de la propia clase que se comportará como un objeto. Para un operador, el operador se aplica a la clase entera, tal como un operador de creación o un operador que devuelve estadística sobre las instancias del sistema completo.

Notación

Una cualidad estática o un operador se subraya (Figura 14.40). Los elementos no estáticos no se subrayan (defecto).

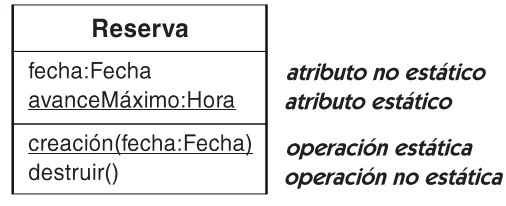


Figura 14.40 Atributo y operación estática

Historia

Los conceptos de UML1 de alcance origen y de alcance de destino han sido substituidos por el concepto más familiar de característica estática, probablemente con poco coste práctico.

Discusión

Las cualidades estáticas proporcionan los valores globales para una clase y se deben utilizar con cuidado o evitarlo completamente, aun cuando son proporcionadas por la mayoría de los lenguajes de programación orientados a objetos. El problema es que implican información global, que viola el principio del diseño orientado a objetos. Por otra parte, la información global llega a ser problemática en un sistema distribuido, ya que fuerza a accesos centrales en una situación en que objetos de una clase se pueden distribuir sobre muchas máquinas. En lugar de utilizar una clase como objeto con el estado, es mejor introducir objetos explícitos para implementar cualquier información compartida que sea necesaria. El modelo y los costes son más evidentes.

Los constructores (operaciones de creación, operaciones de la fabricación) son necesariamente estáticos porque no hay instancias (aún) en el cual pueden funcionar. Éste es un uso nece-

sario y apropiado para las operaciones estáticas. Otras clases de operaciones estáticas tienen las mismas dificultades que los llamados atributos, es decir, ellas implican la información global centralizada sobre las instancias de una clase, lo que no es práctico en un sistema distribuido.

característica estructural

Un rasgo estático de un elemento del modelo, tal como un atributo o un funcionamiento.

La distinción de una propiedad es demasiado sutil para el uso ordinario.

cardinalidad

El número de elementos de un conjunto. Es un número específico. Contrasta con la multiplicidad, que es el rango de posibles cardinalidades que puede poseer un conjunto.

Discusión

Obsérvese que el término *cardinalidad* está mal utilizado por muchos autores que lo utilizan para lo que nosotros llamamos multiplicidad, pero el término cardinalidad tiene una definición matemática de gran prestigio como un número, no un rango de números. Esta es la definición que nosotros utilizamos.

caso de uso

Especificación de secuencias de acciones, incluyendo secuencias variantes y secuencias de error, que pueden realizar un sistema, subsistema o clase interactuando con objetos externos para proporcionar un servicio de valor.

Véase también actor, clasificador, sujeto.

Semántica

Un caso de uso es una unidad coherente de funcionalidad proporcionada por un clasificador (un sistema, subsistema o clase) y manifestada mediante secuencias de mensajes intercambiados entre el sistema y uno o más usuarios externos (representados como actores), junto con acciones realizadas por el sistema.

El propósito de un caso de uso es definir una pieza de comportamiento de un clasificador (incluyendo un subsistema o el sistema entero), sin revelar la estructura interna del clasificador. El clasificador cuyo comportamiento se describe se conoce como sujeto. Cada caso de uso especifica un servicio que el sujeto proporciona a sus usuarios —es decir, una forma específica de utilizar el clasificador que es visible desde fuera. Describe una secuencia completa iniciada por un objeto (modelado por un actor) en términos de la interacción entre usuarios y sujetos así como las respuestas realizadas por el sujeto. La interacción incluye sólo las comunicaciones entre el sujeto y los actores. El comportamiento interno o implementación está oculto.

El conjunto completo de casos de uso de un clasificador o sistema cubre su comportamiento y lo particiona en piezas significativas para los usuarios. Cada caso de uso representa

una pieza significativa de funcionalidad disponible para los usuarios. Observe que usuario incluye a los humanos, así como computadoras y otros objetos. Un actor es una idealización del propósito de un usuario, no una representación de un usuario físico. Un usuario físico puede mapearse a muchos actores, y un actor puede representar el mismo aspecto de varios usuarios físicos. *Véase* actor.

Un caso de uso incluye comportamiento principal normal en respuesta a una petición del usuario, así como posibles variaciones sobre la secuencia normal, como secuencias alternativas, comportamiento excepcional, y manejo de errores. El objetivo es describir una pieza de funcionalidad coherente en todas sus variaciones, incluyendo todas las condiciones de error. El conjunto completo de los casos de uso de un clasificador especifica todas las formas diferentes de utilizar al clasificador. Por comodidad, los casos de uso se pueden agrupar en paquetes.

Un caso de uso es un clasificador; describe comportamiento potencial. Una ejecución de un caso de uso es una instancia del caso de uso. El comportamiento de un caso de uso se puede especificar de varias maneras: mediante una máquina de estados adjunta, una especificación de actividad, una interacción que describa secuencias legales, o mediante precondiciones y postcondiciones. También se puede describir mediante una descripción de texto informal. El comportamiento se puede ilustrar, pero no especificar formalmente, mediante un conjunto de escenarios. Pero en las etapas tempranas del desarrollo esto puede ser suficiente.

Una instancia de un caso de uso es una ejecución de un caso de uso, iniciado por un mensaje desde instancia de un actor. En respuesta al mensaje, la instancia del caso de uso ejecuta una secuencia de acciones especificadas por el caso de uso, como el envío de mensajes a instancias de actor, no necesariamente sólo al actor que inició el caso de uso. Las instancias de actor pueden enviar mensajes a las instancias de los casos de uso, y la interacción continúa hasta que la instancia ha respondido a todas las entradas. Cuando no espera más entradas, termina.

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

Estructura

Un caso de uso normalmente, pero no siempre, pertenece a su clasificador sujeto. Un caso de uso puede tener características de clasificador y relaciones.

Características. Un caso de uso es un clasificador y por tanto tiene atributos y operaciones. Los atributos se utilizan para representar el estado del caso de uso —es decir, el progreso de su ejecución. Una operación representa una pieza de trabajo que el caso de uso puede realizar. Una operación de caso de uso no es directamente llamable desde fuera, pero se puede utilizar para describir el efecto del caso de uso sobre el sistema. La ejecución de una operación se puede asociar con la recepción de un mensaje desde un actor. Las operaciones actúan sobre los atributos del caso de uso e indirectamente sobre el clasificador sujeto al que está adjunto el caso de uso.

Asociaciones con actores. Una asociación entre un actor y un caso de uso indica que la instancia del actor se comunica con la instancia sujeto para llevar a cabo algún resultado de interés para el actor. Los actores modelan usuarios externos de un sujeto. De esta manera, si el sujeto es un sistema, sus actores son los usuarios externos del sistema. Los actores de subsistemas de nivel inferior pueden ser otras clases dentro del sistema global.

El actor del extremo de la asociación puede tener una multiplicidad que indica cuántas instancias del actor pueden participar en una sola ejecución del caso de uso. La multiplicidad por defecto es uno.

Un actor puede comunicarse con varios casos de uso —es decir, el actor puede solicitar varios servicios diferentes al sujeto— y un caso de uso se puede comunicar con uno o más actores cuando provee su servicio. Observe que dos casos de uso que especifican el mismo sujeto no se comunican entre ellos, puesto que cada uno de ellos describe individualmente una utilización completa del sistema. Pueden interactuar indirectamente a través de actores compartidos.

El extremo de la asociación del caso de uso puede tener una multiplicidad que indica con cuántas ejecuciones del caso de uso puede participar una instancia de un actor. La participación en varios casos de uso puede ser simultánea o secuencial.

La interacción entre actores y casos de uso se puede definir con interfaces. Una interfaz define las operaciones que puede soportar o utilizar un actor o un caso de uso. Las distintas interfaces ofrecidas por el mismo caso de uso no tienen por qué 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 un caso de uso más general. El hijo hereda los atributos, operaciones y secuencias de comportamiento del padre y puede añadir atributos y operaciones adicionales. El caso de uso hijo añade comportamiento incremental al caso de uso padre insertando secuencias de acción adicionales en la secuencia del padre en cualesquiera puntos arbitrarios. También puede modificar algunas operaciones y secuencias heredadas, pero esto debe hacerse con cuidado de forma que se preserve la intención del padre. Cualesquiera relaciones de inclusión o de extensión del caso de uso hijo también pueden modificar de forma eficaz el comportamiento heredado del caso de uso padre.

Extender. Una relación de extensión es un tipo de dependencia. El caso de uso cliente añade comportamiento incremental al caso de uso base insertando secuencias de acción adicionales en la secuencia base. El caso de uso cliente contiene uno o más segmentos de secuencia de comportamiento separados. La relación de extensión contiene una lista de nombres de puntos de extensión del caso de uso base, igual en número al número de segmentos en el caso de uso cliente. Un punto de extensión representa una ubicación o conjunto de ubicaciones en el caso de uso base en los que se puede insertar la extensión. Una relación de extensión también puede tener una condición sobre ella, que puede utilizar atributos desde el caso de uso padre. Cuando una instancia del caso de uso padre alcanza una ubicación referenciada por un punto de extensión en una relación de extensión, se evalúa la condición; si la condición es verdadera, se realiza el segmento de comportamiento correspondiente del caso de uso hijo. Si no hay condición, se considera que siempre es verdadera. Si la relación de extensión tiene más de un punto de extensión, la condición se evalúa sólo en el primer punto de extensión anterior a la ejecución del primer segmento.

Una relación de extensión no crea un nuevo caso de uso instanciable. En vez de eso, añade comportamiento implícitamente al caso de uso base original. El caso de uso base implícitamente incluye el comportamiento extendido. El caso de uso base original sin extender no está disponible en su forma inalterada. En otras palabras, si extiende un caso de uso, no puede instanciar explícitamente el caso de uso base sin la posibilidad de las extensiones. Un caso de uso puede

tener varias extensiones que se apliquen al mismo caso de uso base y se puedan insertar en una instancia de un caso de uso si sus condiciones separadas se satisfacen. Por otro lado, un caso de uso de extensión puede extender varios casos de uso base (o el mismo en diferentes puntos de extensión), cada uno en su punto de extensión (o lista de puntos de extensión) propio adecuado. Si hay varias extensiones en el mismo punto de extensión, su orden de ejecución relativo es no determinista.

Observe que el caso de uso de extensión no es para ser instanciado, el caso de uso base debe ser instanciado para obtener el comportamiento combinado base más extensiones. El caso de uso de extensión puede o no ser instanciable, pero en cualquier caso no incluye el comportamiento del caso de uso base.

Incluir. 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 ubicación que el cliente especifica en su descripción. Esto es una dependencia, no una generalización, puesto que el caso de uso proveedor no puede ser sustituido en los lugares en los que el caso de uso cliente aparece. El cliente puede acceder a los atributos del caso de uso base para obtener valores y comunicar resultados. La instancia del caso de uso está ejecutando el caso de uso cliente. Cuando alcanza el punto de inclusión, comienza a ejecutar el caso de uso del proveedor hasta que finaliza. Entonces se reanuda la ejecución del caso de uso cliente más allá de la ubicación de inclusión. Los atributos del caso de uso proveedor no tienen valores que persistan entre ejecuciones.

Un caso de uso puede ser abstracto, lo que significa que no se puede instanciar directamente en una ejecución de un sistema. Define un fragmento de comportamiento que es especializado por casos de uso concretos o incluido en casos de uso concretos, o puede ser una extensión de un caso de uso base. También puede ser concreto si se puede instanciar por sí mismo.

Comportamiento. La secuencia de comportamiento de un caso de uso se puede describir utilizando una máquina de estados, gráfico de actividad, interacción, o código de texto 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.

Se puede describir un caso de uso de forma informal utilizando escenarios en texto plano, pero dichas descripciones son imprecisas y están pensadas sólo para la interpretación humana.

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 llamada 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 mediante objetos en el clasificador que describe el caso de uso. Cada colaboración describe el contexto entre los componentes del sistema dentro del que se producen una o más secuencias de interacción. Las colaboraciones y sus interacciones definen cuantos objetos dentro del sistema interactúan para conseguir el comportamiento externo especificado del caso de uso.

Se puede especificar un sistema con casos de uso a varios niveles de abstracción. Un caso de uso que especifique un sistema, por ejemplo, se puede refinar en un conjunto de casos de uso subordinados, cada uno especificando un servicio de un subsistema. La funcionalidad especificada por el caso de uso principal (de mayor nivel) es completamente trazable con la funcionali-

dad de los casos de uso subordinados (de más bajo nivel). Un caso de uso principal y un conjunto de casos de uso subordinados especifican el mismo comportamiento a dos niveles de abstracción. Los casos de uso subordinados cooperan para proporcionar el comportamiento del caso de uso principal. La cooperación de los casos de uso subordinados se especifica mediante colaboraciones del caso de uso principal y se pueden presentar en diagramas de colaboración. Los actores de un caso de uso principal aparecen como actores de los casos de uso subordinados. Más aún, los casos de uso subordinados son actores entre ellos. Esta realización en capas resulta en un conjunto de casos de uso anidados y colaboraciones que implementan el sistema entero.

Notación

Un caso de uso se representa como una elipse con el nombre del caso de uso dentro o debajo de la elipse. Si los atributos u operaciones del caso de uso se deben mostrar, el caso de uso puede dibujarse como un rectángulo clasificador con la palabra clave **«use case»**. La Figura 14.41 muestra un diagrama de casos de uso. El sujeto se representa mediante un rectángulo que contiene los casos de uso. Los actores se colocan fuera del rectángulo para mostrar que son externos. Las líneas conectan actores con los casos de uso en los que participan.

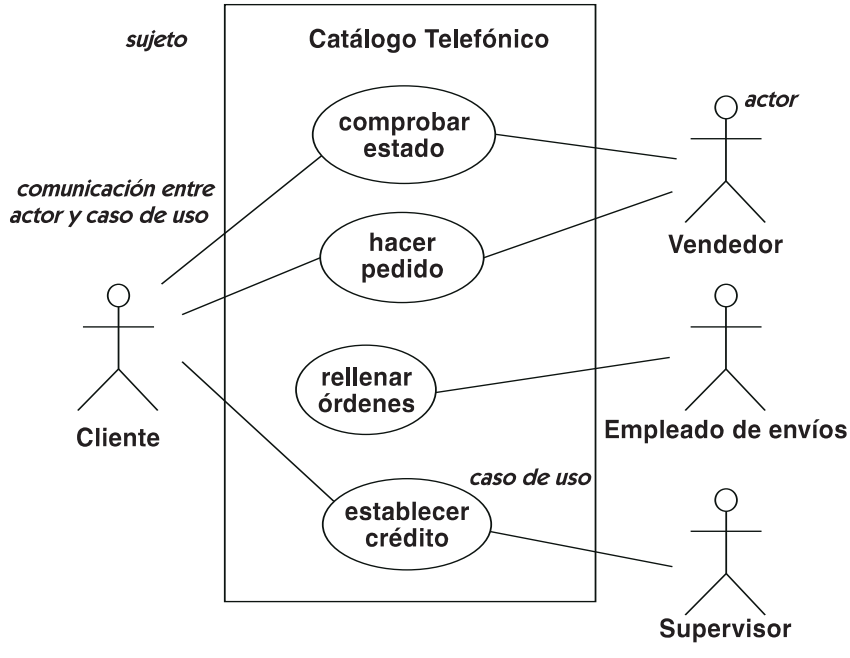


Figura 14.41 Casos de uso, sujeto y actores

Un punto de extensión es una entidad con nombre dentro de un caso de uso que describe ubicaciones en las que se pueden insertar secuencias de acción de otros casos de uso. Proporciona un nivel de indirección entre las extensiones y el texto de secuencia de comportamiento. Un punto de extensión referencia una ubicación o conjunto de ubicaciones dentro de la secuencia de comportamiento del caso de uso. La referencia se puede cambiar independientemente de las relaciones de extensión que utiliza el punto de extensión. Cada punto de extensión debe tener un

nombre único dentro de un caso de uso. Los puntos de extensión se pueden listar en un compartimento del caso de uso con la cabecera **extension points** (Figura 14.42).

Una relación entre un caso de uso y un actor se representa utilizando un símbolo de asociación —una ruta sólida entre los símbolos del caso de uso y del actor. Generalmente, no se coloca ningún nombre o rol en la línea, ya que el actor y el caso de uso definen la relación de manera única.

Una relación de generalización se representa mediante una flecha de generalización —una ruta sólida desde el caso de uso hijo hasta el caso de uso padre, con una cabeza de flecha triangular cerrada en el caso de uso padre.

Una relación de extensión o una relación de inclusión se representan mediante una flecha de dependencia con la palabra clave **«extend»** o **«include»** —una línea discontinua con una cabeza de flecha normal en el caso de uso cliente. Una relación de extensión también tiene una caja de comentario que contiene una condición (sobre la que se ejecuta la extensión) y un punto de extensión o una lista de nombres de puntos de extensión. La caja de comentario se puede suprimir en el diagrama.

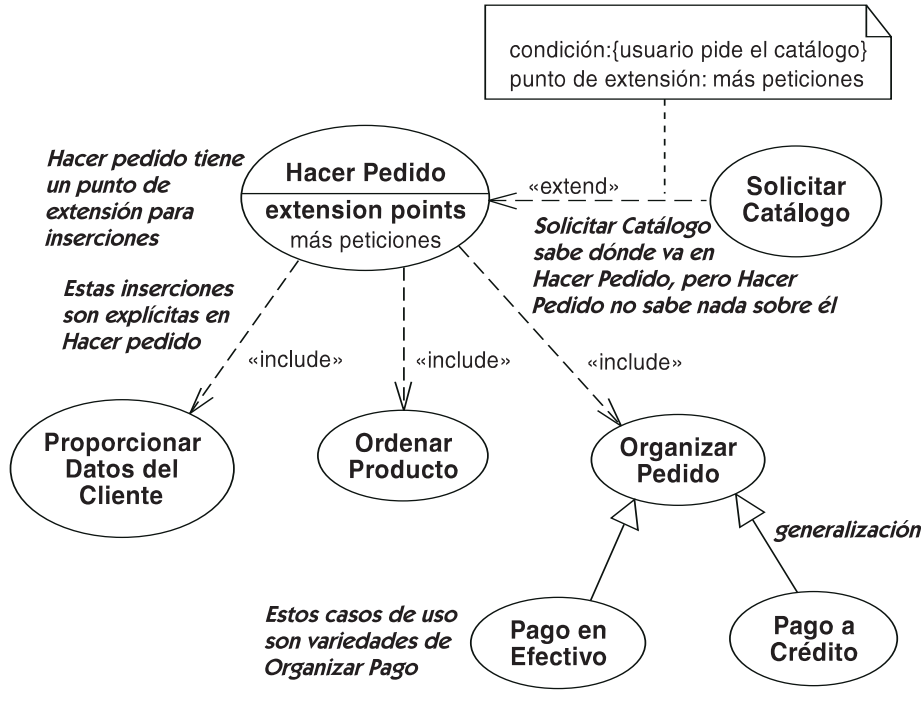


Figura 14.42 Relaciones entre casos de uso

La Figura 14.42 muestra distintos tipos de relaciones entre casos de uso.

Especificación de comportamiento. La relación entre un caso de uso y sus secuencias de interacción externas se representa normalmente mediante un hipervínculo a diagramas de secuencia. El hipervínculo es invisible pero se puede atravesar en un editor. El comportamiento

también puede especificarse mediante una máquina de estados o mediante texto de un lenguaje de programación adjunto al caso de uso. Se puede utilizar texto en lenguaje natural como una especificación informal.

Véase extender para ver un ejemplo de algunas secuencias de comportamiento.

La relación entre un caso de uso y su implementación se puede representar como una relación de realización desde un caso de uso a una colaboración. Pero puesto que estos a menudo están en modelos separados, normalmente se representan mediante un hipervínculo invisible. La esperanza es que una herramienta soportará la capacidad de hacer "zoom" en un caso de uso para ver sus escenarios y/o implementación como una colaboración.

clase

El 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 (para un modelo de análisis), o puede contener también conceptos algorítmicos o de implementación (para un modelo de diseño). Un clasificador es una generalización de una clase que incluye otros elementos parecidos a las clases, como los tipos de datos, actores y componentes.

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 valor de una variable debe tener una clase que es compatible con la clase declarada de la variable, es decir, debe ser de la misma clase que la clase declarada o un descendiente de ella. Una clase también se utiliza para instanciar objetos. Una operación de creación produce un objeto nuevo que es una instancia de la clase dada.

Un objeto instanciado de una clase es una instancia directa de la clase y una instancia indirecta de los antepasados de la clase. El objeto contiene una ranura para albergar un valor para cada atributo; acepta todas las operaciones y señales de su clase, y puede aparecer en enlaces de asociaciones involucrando a la clase o a un antepasado suyo.

Algunas clases pueden no ser instanciadas directamente, pero en su lugar se utilizan sólo para describir estructuras compartidas entre sus descendientes. Tales clases se denominan clases abstractas. Una clase que se puede instanciar es una clase concreta.

Una clase también puede ser considerada como un objeto global. Cualquier atributo de ámbito de clase de la clase es un atributo de este objeto implícito. Estos atributos tienen un ámbito global y cada uno tiene un único valor en todo el sistema. Una operación de ámbito de clase es aquella que se aplica a la propia clase y no a un objeto. Las operaciones de ámbito de clase más comunes son las operaciones de creación.

En UML, una clase es un tipo de clasificador. Los *clasificadores* incluyen a elementos parecidos a las clases, pero encuentran su expresión más completa en las *clases*.

clase abstracta

Una clase que no se puede instanciar.

Véase abstracto/a.

Semántica

Una clase abstracta no puede tener instancias directas. Puede tener instancias indirectas a través de sus descendientes concretos.

Para una discusión sobre el tema, *véase* abstracto/a.

clase activa

Una clase cuyas instancias son objetos activos.

Véase objeto activo para más detalles.

Semántica

Una clase activa es una clase cuyas instancias son objetos activos.

Notación

Una clase activa se muestra como un rectángulo con líneas verticales dobles en los lados izquierdo y derecho. La notación ha cambiado en UML2.

Ejemplo

La Figura 14.43 muestra un diagrama con una clase activa y sus partes pasivas.

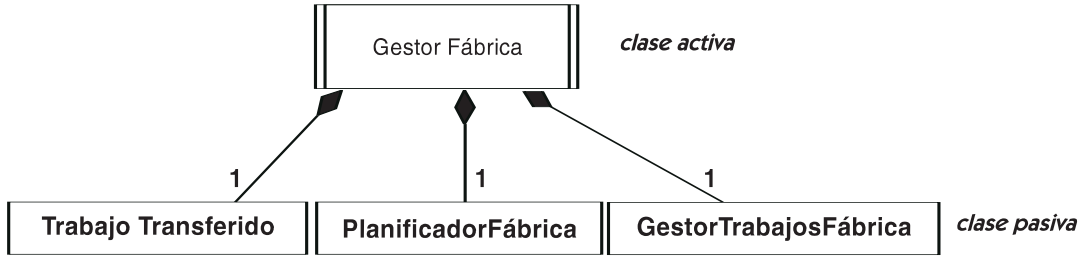


Figura 14.43 Clase activa y partes pasivas

La Figura 14.44 muestra una colaboración que contiene objetos activos que se corresponden con este modelo.

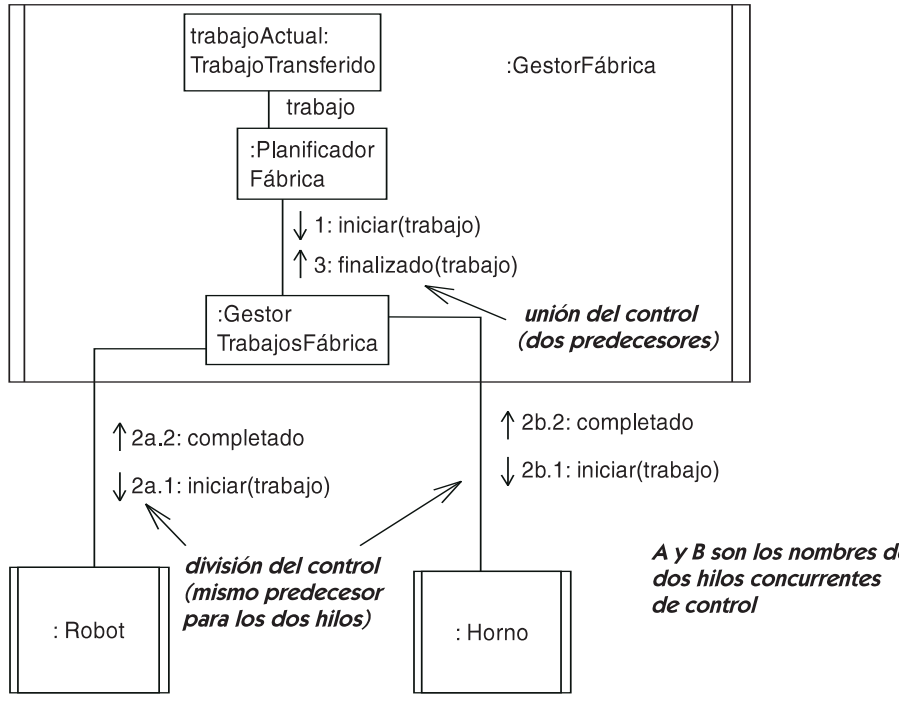


Figura 14.44 Diagrama de comunicación con roles activos y control concurrente

clase compuesta

Clase relacionada con una o más clases por una relación de composición.

Véase composición, clasificador estructurado.

clase de asociación

Una clase de asociación es una asociación que también es una clase. Una clase de asociación tiene propiedades, tanto de asociación, como de clase. Sus instancias son enlaces que tienen valores de atributos así como referencias a otros objetos. Aunque su notación consiste tanto en un símbolo de asociación, como en un símbolo de clase, realmente es un único elemento del modelo.

Véase también asociación, clase.

Semántica

Una clase de asociación tiene las propiedades, tanto de las asociaciones, como de las clases —conecta dos o más clases y tiene también atributos y operaciones. Una clase de asociaciones útil cuando cada enlace debe tener sus propios valores de atributos, operaciones o referencias a objetos. Se puede considerar como una clase con una referencia de clase extra por cada extremo de la asociación, que es la forma obvia y normal de implementarla. Cada instancia de la clase de asociación tiene tanto referencias a objetos, como los valores de atributos especificados por la parte de la clase.

Una clase de asociación C que conecta a las clases A y B no es lo mismo que una clase D con asociaciones binarias a A y B (*véase* la sección de discusión). Como todos los enlaces, un enlace de una clase de asociación como C toma su identidad de las referencias a objetos que contiene. Los valores de los atributos no son los encargados de proporcionarle la identidad. Por lo tanto, dos enlaces de C no deben tener el mismo par de objetos (a, b), incluso si sus valores de atributos son distintos, a menos que la configuración de unicidad de los extremos de la asociación sea no única, es decir, a menos que los enlaces formen una bolsa. *Véase* la sección de discusión.

Las clases de asociación pueden tener operaciones que modifiquen los atributos del enlace o que añadan o eliminen enlaces del propio enlace. Dado que una clase de asociación es una clase, también puede participar en asociaciones.

Una clase de asociación no se puede tener a ella misma cono una de las clases participantes (aunque indudablemente alguien podría encontrar un significado para este tipo de estructura recursiva).

Notación

Una clase de asociación se muestra mediante un símbolo de clase (rectángulo) vinculado, mediante una línea discontinua, a una ruta de asociación (Figura 14.45). El nombre en el símbolo de clase y la cadena de nombre vinculada a la ruta de asociación son redundantes. La ruta de asociación puede tener los adornos de extremo de asociación normales. El símbolo de clase puede tener atributos y operaciones, así como participar por sí misma en asociaciones como clase. No hay adornos en la línea discontinua, ya que no es una relación, sino simplemente parte de símbolo de clase de asociación.

Reglas de estilo

El punto de unión no debería estar tan cerca de uno de los extremos de la asociación como para que pareciera que está vinculado a un extremo de la asociación o a uno de los adornos de rol.

Obsérvese que la ruta de asociación y la clase de asociación son un único elemento del modelo y, por lo tanto, tienen un único nombre. El nombre se puede mostrar en la ruta, en el símbolo de clase o en ambos. Si la clase de asociación sólo tiene atributos y no tiene operaciones u otras asociaciones, el nombre se puede mostrar en la ruta de asociación y omitirlo del símbolo de la clase de asociación para enfatizar su “naturaleza de asociación”. Si tienen operaciones y otras asociaciones, el nombre se puede omitir de la ruta, colocándolo en el rectángulo de clase para enfatizar su “naturaleza de clase”. En ninguno de los dos casos hay una semántica diferente.

Discusión

La Figura 14.45 muestra una clase de asociación que representa el empleo. La relación de empleo entre una empresa y una persona es muchos a muchos. Una persona puede tener más de un trabajo, pero sólo un trabajo para una empresa dada. El sueldo no es un atributo ni de la compañía ni de la persona porque la asociación es muchos a muchos. Debe ser un atributo de la propia relación.

La relación jefe-trabajador no sólo es una relación entre dos personas. Es una relación entre una persona con un empleo y otra persona con otro empleo. Es una asociación (**Gestiona**) entre la clase de asociación y ella misma.

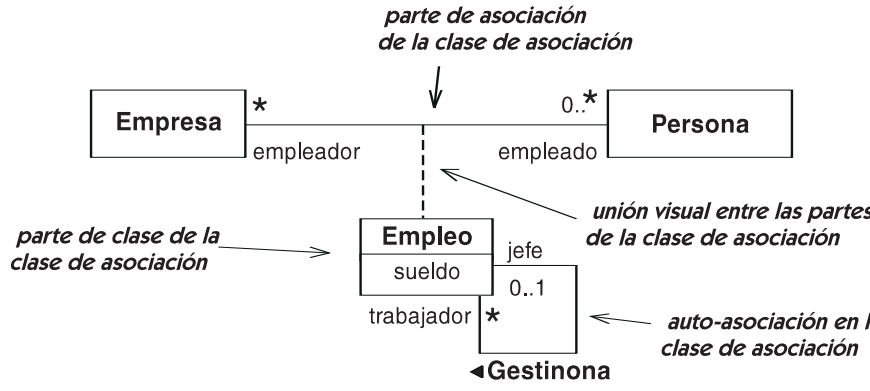


Figura 14.45 Clase de asociación

El siguiente ejemplo muestra la diferencia entre una clase de asociación y la materialización de una relación modelada como una clase. En la Figura 14.46, la propiedad de las acciones se modela como una asociación entre Persona y Empresa. El atributo de la clase de asociación **cantidad** representa la cantidad de acciones que se poseen. Esta relación se modela como una asociación debido a que sólo debe haber una entrada por cada pareja **Persona** y **Empresa**.

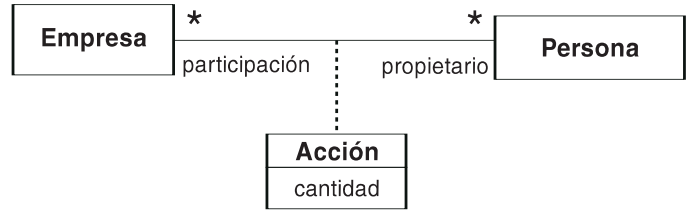


Figura 14.46 Clase de asociación con atributo

Para modelar compras de valores, como se muestra en la Figura 14.47, puede haber múltiples compras con la misma Persona y Empresa. Pero se pueden distinguir porque cada compra es distinta y tiene su propia fecha y coste además de la cantidad. La palabra clave {bag} se coloca en cada extremo de la asociación para indicar que puede haber múltiples enlaces y, por tanto, múltiples objetos enlazados involucrando a las mismas parejas de objetos. De forma alternativa, como se muestra en la Figura 14.48, la relación se puede materializar —es decir, se puede hacer con objetos distintos con su propia identidad. Para modelar este caso se utiliza una clase normal, porque cada compra tiene su propia identidad, con independencia de las clases **Persona** y **Compañía** con las que se relaciona. Aunque los modelos no son idénticos, el modelo tiene la misma información. Por último, se puede modelar como una asociación ternaria, como se muestra en la Figura 14.49. Sin embargo, es una elección peor, porque un lote sólo determina una tupla a partir de la asociación y, por tanto, de la empresa y la persona. Normalmente, las asociaciones ternarias no son útiles a menos que la multiplicidad sea muchos en todos los extremos, ya que de otra forma, una asociación binaria muestra más información sobre las restricciones de la multiplicidad.

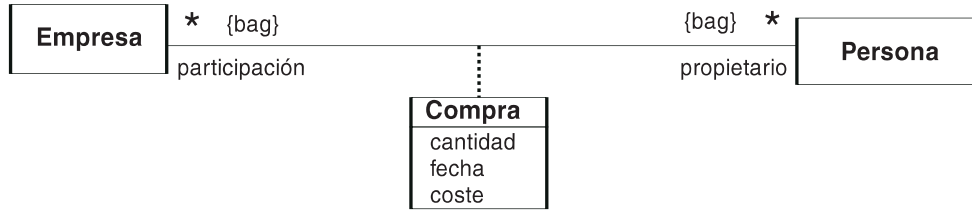


Figura 14.47 Clase de asociación no única

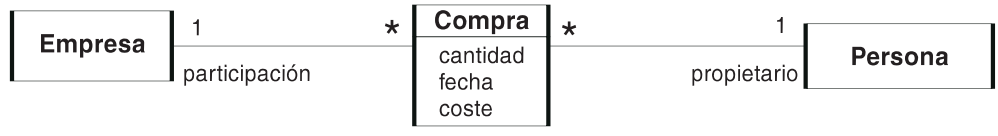


Figura 14.48 Asociación materializada

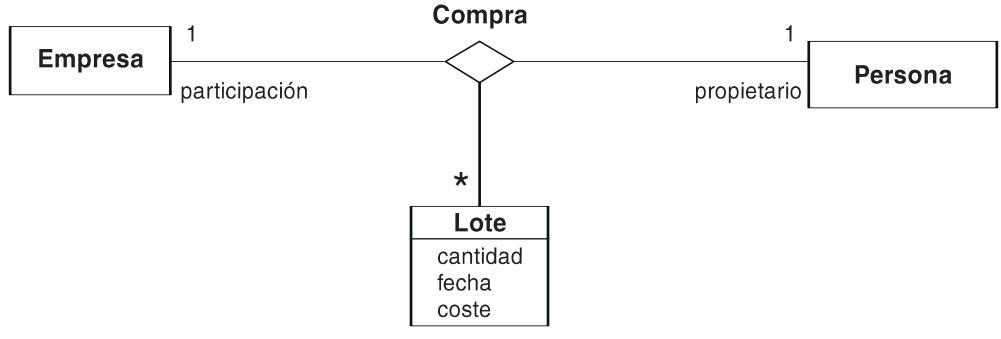


Figura 14.49 Asociación ternaria

clase directa

Clase que en su mayor parte describe totalmente un objeto.

Véase también clase, clasificación múltiple, generalización, herencia, 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 antepasados de dicha clase. La clase directa es la descripción más específica de un objeto, la que lo describe de la forma más completa. Un objeto es una instancia directa de su clase directa y una instancia indirecta de los antepasados de la clase directa. Un objeto no es una instancia de ningún descendiente de la clase directa (por definición).

Si se permite la clasificación múltiple en un sistema, ninguna sola clase directa puede describir completamente 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 contiene parte de su descripción,

siempre que ningún descendiente también describa al objeto. En otras palabras, ninguna de las clases directas de un objeto tiene una relación antepasada con las demás.

Si una clase se instancia para producir un objeto, el objeto es una instancia directa de la clase.

clase-en-un-estado

Una combinación de una clase y un estado aplicable a esa clase.

Véase también actividad.

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 están en el estado. En algunos casos, ciertos atributos, asociaciones u operaciones sólo son válidos cuando el objeto está en un cierto estado o conjunto de estados. 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 algunas veces, es útil tratar la combinación de una clase y un estado como una restricción de un objeto, variable, parámetro o subestado. La combinación de clase y estado, tratada como una modalidad de tipo, es una clase-en-un-estado.

Si la máquina de estados de la clase tiene estados ortogonales, la especificación del estado puede ser un conjunto de subestados que un objeto de la clase puede albergar simultáneamente.

Diagrama de clases. Una clase-en-un-estado es equivalente a una subclase, siempre que esté permitida la clasificación dinámica. Modelándolo como una clase, ésta puede utilizarse como el tipo de una variable o parámetro. Puede participar en asociaciones que son válidas sólo para objetos en el estado dado. Considere, en la Figura 14.50, la asociación **Asignación**

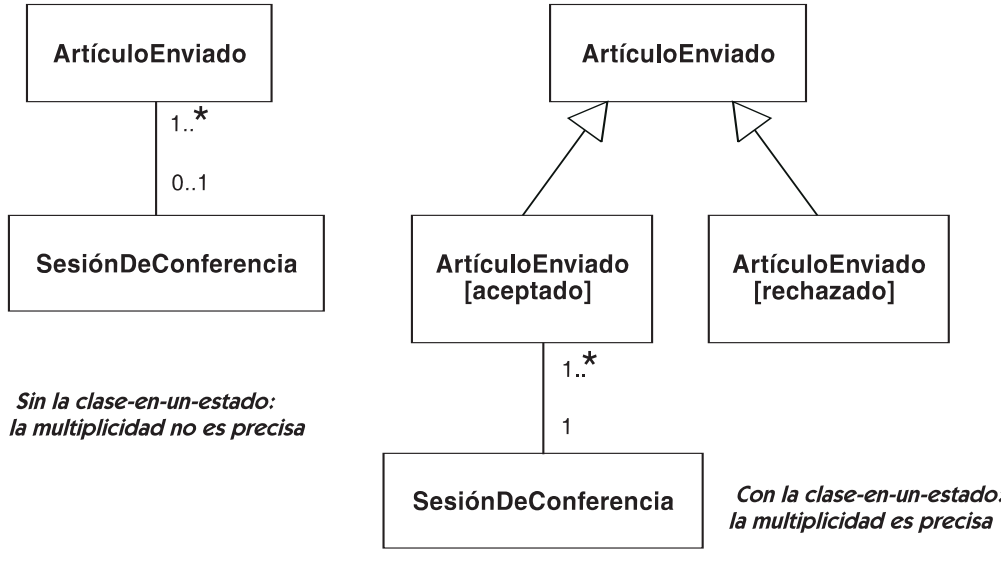


Figura 14.50 Clase-en-un-estado

entre **ArtículoEnviado** y **SesiónDeConferencia**. Esta asociación es válida para un **ArtículoEnviado** en el estado **aceptado** (multiplicidad destino uno) pero no en el estado **rechazado**. Para cualquier **ArtículoEnviado**, la multiplicidad destino es cero o más, porque la clase incluye tanto los artículos **aceptados**, como los **rechazados**. Sin embargo, si la asociación se modela entre **ArtículoEnviado** en el estado **aceptado** y **SesiónDeConferencia**, su multiplicidad destino es exactamente uno.

Diagrama de secuencia. Se puede colocar una afirmación, denominada invariante del estado, en un punto de la línea de vida para especificar una restricción sobre el objeto representado en la línea de vida en ese punto de la ejecución. La restricción puede especificar, en sus valores, el estado del objeto o una expresión Booleana. Esto es una afirmación, no una acción ejecutable, y el modelo estaría mal formado si se viola la afirmación.

Diagrama de actividad. Un nodo de objeto tiene un tipo. También puede especificar un estado, que indica que el correspondiente objeto debe tener el tipo y el estado dados en ese punto de la actividad.

Notación

Diagrama de clases. Una clase-en-un-estado se puede mostrar mediante un símbolo de clase, en el cual el nombre de la clase va seguido de su nombre de estado dentro de corchetes (**`Nombreclase[nombre-estado]`**). Los corchetes también pueden contener una lista de nombres, separados por comas, de los estados concurrentes para indicar que un objeto alberga varios de ellos. Se debería mostrar la clase como una subclase de la clase plana. (Obsérvese que esto es una convención sobre el nombrado, no una notación oficial de UML.) *Véase* la Figura 14.50.

Diagrama de secuencia. Un invariante de estado se muestra como una caja de esquinas redondeadas sobre la línea de vida con una restricción Booleana dentro de la caja. Si la restricción es que el objeto debe estar en un estado dado, se debe mostrar el nombre del estado en la caja (Figura 14.51). Se puede utilizar una lista de nombres de estado separados por comas si el estado del objeto involucra a múltiples estados concurrentes.

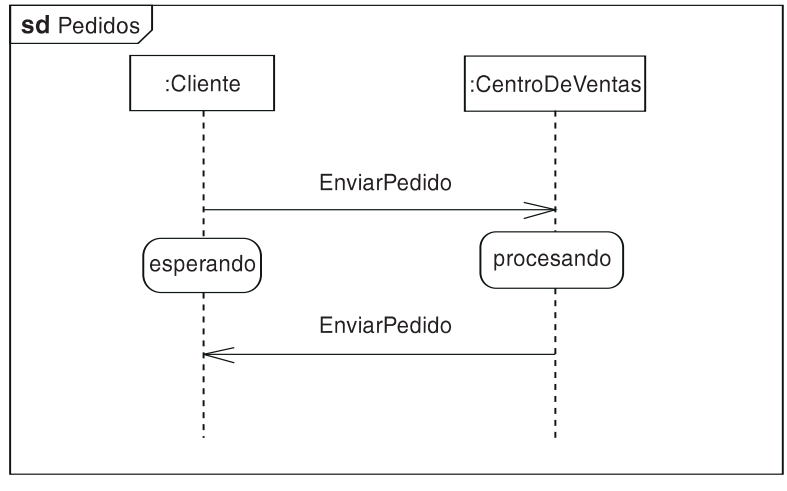


Figura 14.51 Invariantes de estado en un diagrama de secuencia

Diagrama de actividad. Un nodo de objeto se muestra como un rectángulo que contiene el nombre de una clase. El nombre de la clase puede ir seguido opcionalmente de un nombre de estado entre corchetes (**`Nombreclase[nombre-estado]`**). Esto indica que el valor producido en ese punto de la ejecución debe tener el tipo dado y debe estar en el estado dado.

Véase la Figura 14.52.

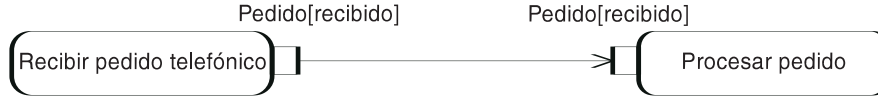


Figura 14.52 Nodo de objeto con restricción de estado

Discusión

La clase-en-un-estado y la clasificación dinámica son dos formas de conseguir el mismo objetivo: permitir cambios en la estructura de un objeto durante su vida. Dependiendo del entorno de implementación, alguno de los dos puede ser el mecanismo más conveniente.

La notación sugerida para la clase-en-un-estado en un diagrama de clases no está definida en la especificación de UML. Quizás sería útil añadir esta habilidad en UML, pero los modeladores que quieran mostrar la clase-en-un-estado pueden utilizar la convención de nombrado sugerida, entendiendo que es una convención.

clasificación dinámica

Variación semántica de la generalización en la que un objeto puede cambiar de tipo o rol. Contrastar con: clasificación estática.

Véase también clasificación múltiple.

Semántica

En muchos lenguajes de programación, un objeto no puede cambiar la clase de la que es instanciado. Esta restricción de clasificación estática simplifica la implementación y la optimización de los compiladores, pero no es una necesidad lógica. Por ejemplo, bajo la asunción de la clasificación estática, un objeto instanciado como círculo debe permanecer como círculo; no se puede escalar a la dimensión *n*, por ejemplo. Bajo la asunción de la clasificación dinámica, un círculo escalado a una dimensión se convierte en una elipse. Esto no se considera un problema.

En un modelo UML se puede utilizar cualquier asunción. Esto es un ejemplo de un punto de variación semántica. La elección, sorprendentemente afecta poco a un modelo, aunque las diferencias son muy importantes cara a la ejecución. Se deben definir las mismas clases en cualquier caso, pero las operaciones que soportan pueden diferir en los dos casos.

clasificación estática

Una variación semántica de la generalización en la cual un objeto no puede cambiar el tipo o no puede cambiar de rol. Ésta es la opinión general en los lenguajes más tradicionales tales como C++ o Java. La opción de la clasificación estática o de la clasificación dinámica es un punto de variación semántica.

clasificación múltiple

Variación semántica de generalización en la que un objeto puede pertenecer directamente a más de una clase.

Semántica

Ésta es un punto de variación semántica bajo la que un objeto puede ser instancia directa de más de una clase. Cuando se utiliza con clasificación dinámica, los objetos pueden adquirir y perder clases durante el tiempo de ejecución. Esto permite que las clases se utilicen para representar los roles temporales que puede tener un objeto.

Aunque la clasificación múltiple encaja bien con el discurso lógico y de todos los días, complica la implementación de un lenguaje de programación y no es soportada por los lenguajes de programación populares.

clasificación simple

Un régimen de ejecución en el que cada objeto tiene exactamente una clase directa. Esto es el modelo de ejecución de la mayoría de los lenguajes de programación orientados a objetos. Si se permite una sola clasificación o la clasificación múltiple, es un punto de variación semántica.

clasificador

Un elemento del modelo que describe características de comportamiento y estructurales. Los tipos de clasificador incluyen a actores, asociaciones, comportamientos, clases, colaboraciones, componentes, tipos de datos, interfaces, nodos, señales, subsistemas (como estereotipos) y casos de uso. Las clases son el tipo más general de clasificador. El resto se pueden entender intuitivamente como similares a las clases, con ciertas restricciones sobre el contenido o el uso, aunque cada tipo de clasificador se representa con su propia clase del metamodelo. En general, la mayoría de las propiedades de las clases se aplican a los clasificadores, con ciertas restricciones para cada tipo de clasificador.

Véase también vista estática.

Semántica

El clasificador es una abstracción que incluye varios tipos de elementos de modelado que tienen descripciones estructurales y de comportamiento. Un clasificador describe un conjunto de ins-

tancias y puede ser el tipo de una variable o parámetro. Los clasificadores se pueden organizar en jerarquías de generalización.

En UML, una clase es un tipo de clasificador. El *clasificador* incluye un número de elementos parecidos a las clases, pero encuentra su expresión plena en la *clase*.

Algunos de los contenidos de un clasificador se pueden redefinir en un descendiente. *Véase* redefinición.

Estructura

Un clasificador tiene un nombre de clase y una lista de operaciones y atributos. Un clasificador puede participar en asociaciones, generalizaciones, dependencias y relaciones de restricción. Un clasificador se declara dentro de un espacio de nombres, como un paquete u otra clase, y tiene varias propiedades dentro del espacio de nombres, como la multiplicidad y la visibilidad. Un clasificador tiene otras propiedades, como si es abstracta o una clase activa. Se puede especificar un clasificador como una hoja, es decir, como una que no se especializa. Puede tener una máquina de estados que especifique su comportamiento reactivo, es decir, su respuesta a la recepción de eventos. Un clasificador puede declarar el conjunto de eventos (incluyendo excepciones) que está preparado para manejar. Puede proporcionar la realización de un comportamiento especificado por cero o más interfaces o tipos proporcionando una implementación para el comportamiento. Una interfaz lista un conjunto de operaciones que un clasificador que realiza la interfaz promete soportar.

Un clasificador contiene una lista de atributos y una lista de operaciones en la que cada una forma un espacio de nombres dentro del clasificador. Los atributos y operaciones heredados también aparecen dentro de los respectivos espacios de nombres. El espacio de nombres de los atributos también incluye otras propiedades, como nombres de rol de las asociaciones que parten de la clase y partes internas. Se debe declarar cada nombre una vez dentro del clasificador y de sus antepasados. De otra forma habría un conflicto, y el modelo estaría mal formado. Los nombres de operaciones se pueden reutilizar con diferentes signaturas. De otra forma representarían la redefinición de la misma operación, por ejemplo, para declarar la operación como abstracta o concreta en una subclase.

Un clasificador puede poseer comportamientos, como métodos e interacciones, que implementan o describen sus operaciones. Un clasificador opcionalmente puede tener vinculado un comportamiento, como una máquina de estados, que describe la dinámica general de una instancia del clasificador. El comportamiento vinculado comienza cuando se crea una instancia y para cuando se destruye. Una instancia responde a los eventos que reconoce. Si un evento se corresponde con un disparador en el comportamiento vinculado, éste se sitúa en un pool de eventos. Por ejemplo, una señal que desencadena una transición de una máquina de estados es un efecto disparado. Los eventos se toman del pool de uno en uno y se permite que los efectos disparados se ejecuten hasta su finalización antes de que se seleccione otro evento. En general, la selección de eventos no tiene orden, aunque los perfiles pueden definir una ordenación por prioridades. Si un evento no se corresponde con un disparador en el comportamiento vinculado, pero se corresponde con una operación resuelve a un comportamiento que se ejecuta inmediata y concurrentemente con otras ejecuciones, independientemente del comportamiento vinculado. Por ejemplo, una ejecución normal de un método es un efecto inmediato.

Un clasificador también es un espacio de nombres y establece el ámbito para las declaraciones anidadas de clasificadores. Los clasificadores anidados no son partes estructurales de las ins-

tancias del clasificador. No hay una relación de datos entre los objetos de una clase y los objetos de las clases anidadas. Una clase anidada es una declaración de una clase que se puede utilizar por los métodos de la clase más externa. Las clases declaradas dentro de una clase son privadas a ella y no son accesibles desde el exterior de la clase a menos que se hagan visibles explícitamente. No hay una notación visual para mostrar la declaración de clases anidadas. Se espera que se puedan hacer accesibles dentro de una herramienta mediante hiperenlaces. Los nombres anidados deben ser referenciados utilizando nombres cualificados.

Notación

Un clasificador se muestra cono un rectángulo de línea continua con tres compartimentos separados por líneas horizontales. El compartimento superior alberga el nombre de la clase y otras propiedades que se aplican a toda la clase. El compartimento intermedio alberga una lista de atributos. El compartimento inferior contiene una lista de operaciones. Los compartimentos intermedio e inferior se pueden suprimir en un símbolo de clase.

Uso. Las clases, interfaces, señales y asociaciones se declaran en un diagrama de clases y se utilizan en muchos otros diagramas. Otros tipos de clasificadores, como los casos de uso y las colaboraciones, aparecen en tipos especiales de diagramas, aunque muchos de ellos también pueden aparecer en diagramas de clases. UML proporciona una notación gráfica para la declaración y el uso de clasificadores, así como una notación de texto para referenciar clasificadores dentro de la descripción de otros elementos del modelo. La declaración de un clasificador en un diagrama de clases define los contenidos del clasificador: sus atributos, operaciones y otras propiedades. Otros diagramas definen relaciones adicionales y vínculos del clasificador.

La Figura 14.53 muestra una declaración básica de clase con atributos y operaciones. Este formato es aplicable a otros tipos de clasificadores, mostrando el tipo de clasificador entre comillas o ángulos dobles («») debajo del nombre del clasificador, como por ejemplo **«controller»**. Ciertos tipos de clasificadores tienen sus propios iconos. Entre ellos se incluyen las asociaciones, colaboraciones, actores y casos de uso.

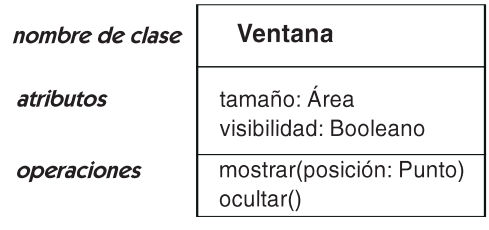


Figura 14.53 Declaración básica de clase

La Figura 14.54 muestra la misma declaración de clase con detalles adicionales, mucha de la información relativa a la implementación, como la visibilidad, las operaciones de creación de nivel de clase y operaciones dependientes de la implementación.

En la Figura 14.55 se ha suprimido toda la información interna sobre la clase. La información está presente en el modelo interno y normalmente se debe mostrar al menos en un diagrama.

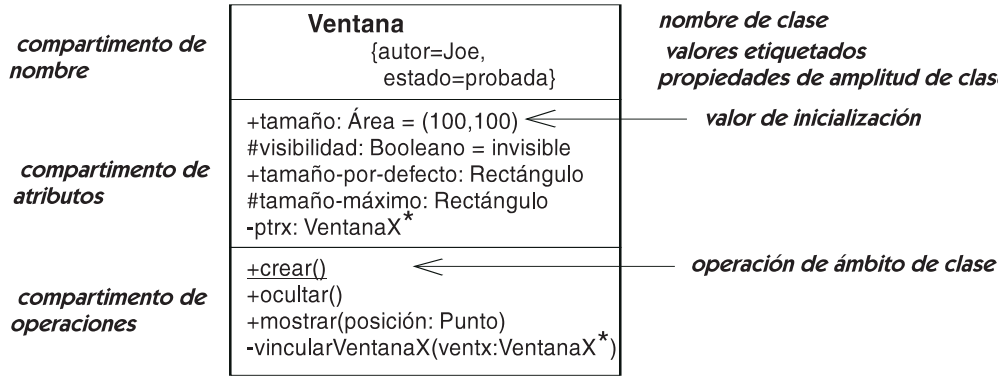


Figura 14.54 Declaración detallada de clase con características de visibilidad

Figura 14.55 Símbolo de clase en el que se han suprimido todos los detalles

Supresión de compartimentos. Se pueden suprimir los compartimentos de los atributos y las operaciones por separado o conjuntamente (Figura 14.56). La línea de separación no se muestra en un compartimento eliminado. Si se suprime un compartimento no se puede hacer ninguna deducción sobre la presencia o ausencia de los elementos en él. Obsérvese que un compartimento vacío (es decir, uno en el que aparezcan las líneas de separación pero sin contenido) implica que no hay elementos en su correspondiente lista. Si se encuentra activo algún tipo de filtrado, entonces no hay elementos que satisfagan el filtro. Por ejemplo, si sólo son visibles las operaciones públicas, la presencia de un compartimento de operaciones vacío indica que no hay operaciones públicas. No se pueden sacar conclusiones acerca de las operaciones privadas.

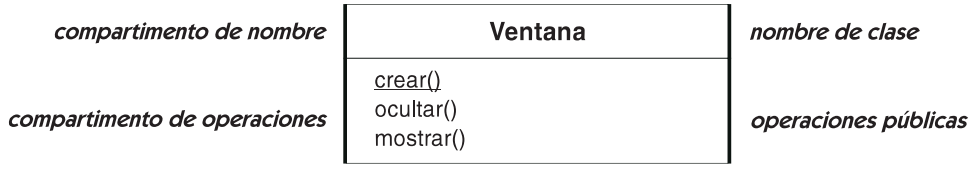


Figura 14.56 Declaración de clase en la que se han suprimido los atributos y las operaciones no públicas

Compartimentos adicionales. Se pueden proporcionar compartimentos adicionales para mostrar propiedades del modelo predefinidas por el usuario, como por ejemplo mostrar reglas de negocio, responsabilidades, variaciones, señales manejadas, excepciones elevadas, etcétera. Un

compartimento adicional se muestra con nombre de compartimento en la parte superior, con un tipo distinto de fuente para identificar su contenido (Figura 14.57).

Los compartimentos estándar (atributos, operaciones) no necesitan de nombre de compartimento, aunque pueden tener nombres para dar énfasis o claridad si sólo está visible un compartimento. La mayoría de los compartimentos son simples listas de cadenas, en las que cada cadena codifica una propiedad. Obsérvese que "cadena" incluye la posibilidad de iconos o documentos embebidos, como hojas de cálculo y gráficos. Son posibles formatos más complicados, aunque UML no los especifica. Hay responsabilidades de usuario y de herramienta. Si la naturaleza del compartimento se puede determinar a partir de la forma de su contenido, el nombre de compartimento se puede omitir.

Véase uso de la tipografía, cadena.

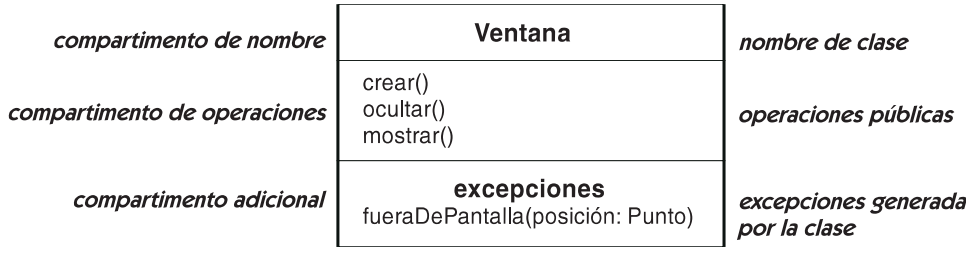


Figura 14.57 Declaración de clase con compartimento adicional con nombre

Estereotipo. Un estereotipo se muestra como una cadena de texto entre comillas o ángulos dobles sobre el nombre de la clase (Figura 14.58). En lugar de la cadena de texto, se puede colocar un icono en la esquina superior derecha del compartimento de nombre. Un símbolo de clase con un icono de estereotipo se puede "colapsar" para mostrar sólo el icono del estereotipo, con el nombre de la clase, tanto dentro del icono, como debajo de él (Figura 14.134). El resto de los contenidos de la clase se suprimen. *Véase* estereotipo.

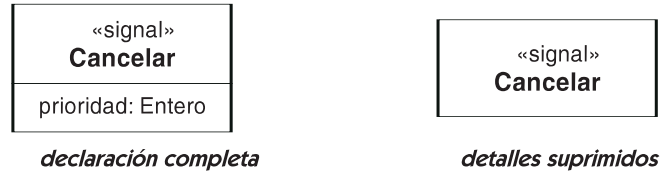


Figura 14.58 Clase con estereotipo

Pautas de estilo

- Centre el nombre de estereotipo con un tipo normal de letra dentro de comillas o ángulos dobles («») sobre el nombre de la clase.
- Centre o justifique a la izquierda el nombre de la clase en negrita.
- Justifique a la izquierda los atributos y operaciones con un tipo normal de letra.

- Muestre en cursiva los nombres de clases abstractas o la signatura de las operaciones abstractas.
- Muestre los compartimentos de los atributos y las operaciones cuando se necesiten (al menos una vez en el conjunto de diagramas) y suprímalos en otros contextos o en referencias. Esto es útil para definir una vez una localización "origen" en el conjunto de diagramas y proporcionar ahí la descripción completa. En otras localizaciones se utiliza la forma mínima.

Discusión

El concepto de clasificador se aplica a un rango de usos en el modelado lógico, así como en la implementación. En UML, bajo ciertos puntos de variación semántica, una instancia puede tener múltiples tipos, así como ser capaz de cambiar su tipo durante la ejecución. Varias nociones más restrictivas de clase encontradas en la mayoría de los lenguajes de programación se pueden plasmar como tipos especiales de clasificadores.

clasificador estructurado

Un clasificador que contiene partes o roles que forman su estructura de datos y realiza su comportamiento.

Semántica

La estructura interna de un clasificador puede definirse en términos de puertos, partes, y conectores. Cada uno de estos elementos se define dentro del contexto proporcionado por la definición de un clasificador estructurado.

Un clasificador estructurado define la aplicación de un clasificador. Las interfaces de un clasificador describen lo que un clasificador debe hacer. Su estructura interna describe cómo hace el trabajo.

Los clasificadores estructurados proporcionan una manera limpia de estructurar la aplicación de un clasificador. Debido a que los tipos de partes dentro de un clasificador estructurado pueden ser estructurados en sí mismos, este mecanismo se extiende jerárquicamente.

Clase estructurada. Una clase estructurada es la descripción de la implementación interna de la estructura de una clase. Una clase estructurada posee sus puertos, partes y conectores. La clase y sus puertos pueden tener las interfaces (incluyendo las interfaces requeridas y proporcionadas) para definir externamente el comportamiento visible. Cuando se crea una instancia de una clase estructurada, también se crean instancias de sus puertos, se crean partes y conectores. Los mensajes en un puerto externo pueden transferirse automáticamente a los puertos en partes internas, de tal forma que una clase estructurada también proporciona alguna descripción del comportamiento. Las clases estructuradas pueden ser jerárquicas en varios niveles.

Colaboración. Una colaboración describe una relación contextual que soporta la interacción entre un conjunto de participantes. Un rol es la descripción de un participante. Al contrario que una clase estructurada, una colaboración no tiene limitadas las instancias a sus roles. Las instancias de los roles existen antes de establecer una instancia de la colaboración pero la colaboración los une y establece los eslabones para conectarlos. Las colaboraciones describen los patrones que estructuran los datos.

Puertos. Un puerto es un punto de interacción individual entre un clasificador y su entorno. Pueden dirigirse mensajes a los objetos (llamadas y señales) mediante una instancia de un puerto en lugar de al objeto entero. Las acciones dentro de un objeto pueden distinguir cuál puerto o llamada o señal llegó antes.

Dentro de un clasificador estructurado, el puerto puede conectarse a las partes internas o a la especificación de comportamiento del objeto total. La implementación de un puerto involucra eventos de la entrada, su reconocimiento y traspaso a la parte apropiada o la ejecución de un comportamiento. No es necesaria ninguna descripción de la implementación si los puertos están correctamente "cableados".

Cuando una instancia de una clase estructurada se crea, sus puertos se crean con él.

Cuando se destruye, sus puertos se destruyen. No se crean instancias de puertos ni se destruyen durante la vida de una instancia.

Partes. Una parte es un pedazo estructural de un clasificador. Una parte describe el papel que la instancia juega dentro de una instancia del clasificador. Una parte tiene un nombre, un tipo y una multiplicidad. Si la multiplicidad máxima de la parte excede uno, una instancia del clasificador puede tener instancias múltiples de una parte pero cada instancia de la parte está separada. Si la multiplicidad es ordenada, las instancias individuales pueden distinguirse por el índice de su posición dentro del objeto.

Cada parte puede tener un tipo. El tipo restringe el tipo de los objetos que pueden ligarse a un rol. El tipo de un objeto asociado a un rol debe ser igual que o un descendiente del tipo declarado del rol.

Un clasificador estructurado no es una relación entre tipos, a diferencia de una asociación. Cada parte es una instancia distinta de un tipo en su propio contexto único. Puede haber múltiples partes con el mismo tipo, cada uno tiene un conjunto diferente de relaciones a otras partes. Una parte no es una instancia, sin embargo, una descripción de todas las instancias que pueden ligarse a la parte. Cada vez que el clasificador estructurado se instancia un conjunto diferente de objetos y enlaces adoptan los diferentes roles.

En el caso de las clases, las partes son poseídas por una clase. Las instancias de las partes se crean y se destruyen conjuntamente con la instancia. Los atributos de una clase pueden ser consideradas partes dentro de la clase.

Una parte en una colaboración representa el uso de una instancia de un tipo dentro de algunos contextos. Las instancias de los roles no son poseídas ni creadas por instancias de una colaboración. Las instancias de la colaboración existen antes de que la colaboración se cree. El crear una instancia de una colaboración implica la ligadura (mediante referencias) de las instancias de los roles de la colaboración. Esto crea una relación transitoria con las instancias de los roles. Cuando la colaboración se destruye, las instancias de sus papeles continúan existiendo, pero su relación contextual transitoria deja de existir.

El mismo objeto puede representar roles diferentes en colaboraciones diferentes. Una colaboración representa una faceta de un objeto. Un solo objeto físico puede combinar diferentes facetas, conectando implícitamente las colaboraciones en las que representa los distintos roles.

Conectores. Un conector es una relación contextual entre los objetos u otras instancias de roles en un clasificador estructurado. Define unas comunicaciones entre los roles. Un conector sólo es significativo dentro del contexto de un clasificador estructurado. Al contrario que una

asociación, es una relación entre roles, no entre los clasificadores que son los tipos declarados de los roles. Una asociación válida entre los tipos de dos roles puede ligarse a un conector. En una clase estructurada, el conector representa más a menudo una "asociación contextual", es decir, una asociación válida sólo dentro de la clase estructurada. Esta asociación puede ser anónima; es decir puede no corresponder a una asociación declarada. Cuando una clase estructurada es instanciada se instancia un enlace por cada uno de sus conectores, tanto si el conector se liga o no, a una asociación declarada.

Dentro de una colaboración, un conector representa a menudo una "asociación transitoria" es decir, una asociación que sólo se manifiesta durante la duración de la colaboración.

Una asociación transitoria normalmente no es una asociación declarada. Tales conectores corresponden a los parámetros de un comportamiento, variables, valores globales o a la relación implícita entre las partes del mismo objeto. Son implícitos dentro de los procedimientos y otros tipos de comportamientos.

Un rol del clasificador tiene una referencia a un clasificador (la base) y una multiplicidad. El clasificador de la base restringe el tipo de objeto que puede representar el rol del clasificador. El objeto de la clase puede ser igual, o un descendiente del clasificador base. La multiplicidad indica cuántos objetos pueden representar el rol una vez en una instancia de la colaboración.

Un rol del clasificador puede tener un nombre o puede ser anónimo. Puede tener múltiples clasificadores base si está permitida la clasificación múltiple. Un rol del clasificador puede conectarse a otros roles del clasificador por conectores.

Semántica de creación. Cuando una clase estructurada es instanciada, sus puertos, partes, y los conectores son instanciados. Si un puerto tiene multiplicidad variable, la cardinalidad del puerto debe especificarse en la instanciación de la clase; los puertos no son creados ni destruidos dinámicamente durante la vida de un objeto. Si una parte tiene multiplicidad variable el número mínimo de partes es instanciado como parte de la instanciación del objeto. Si un conector tiene multiplicidad variable, el número mínimo consecuente de los conectores será la cardinalidad de partes instanciadas. Si la configuración inicial es ambigua los modeladores deben especificarlo. Los modeladores pueden especificar explícitamente la configuración inicial de la instanciación para sobreescribir los valores predeterminados implícitos.

Notación

Una parte o un rol se muestra usando el símbolo de un clasificador (un rectángulo) contenedor etiquetado con una cadena de la forma:

nombreRol$_{opc}$⌊**: NombreTipo**⌋$_{opc}$⌊**[la multiplicidad]**⌋$_{opc}$

Es posible omitir el nombre o el nombre de tipo. La multiplicidad (incluyendo los corchetes) puede ser omitida, y en ese caso la multiplicidad predefinida es uno. Alternativamente, la multiplicidad puede ponerse (sin los corchetes) en la esquina superior derecha del rectángulo.

Una referencia a un objeto externo (es decir, uno que no forma parte del objeto) se muestra como un rectángulo punteado.

La Figura 14.59 muestras varias formas que puede adoptar un rol.

Figura 14.59 Roles o partes

Ejemplo

La Figura 14.60 muestra un ejemplo de una clase estructurada. En la Figura 14.117 se muestra una instancia de esta clase.

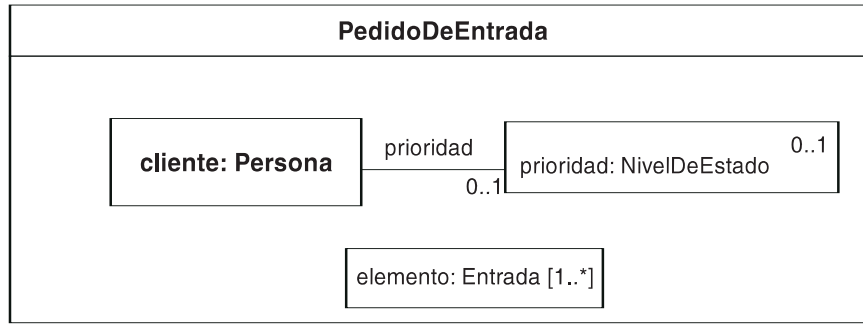


Figura 14.60 Clase estructurada

Historia

Los clasificadores estructurados en UML1 reemplazan los dudosos esfuerzos semánticos de UML1 para marcar las asociaciones para un contexto local. Posiblemente sean el rasgo de modelado más importante agregado en UML2, ya que permiten una expresión clara y multinivel de los modelos.

cliente

Un elemento que solicita un servicio de otro elemento. Este término se utiliza para describir un rol dentro de una dependencia. En la notación, el cliente aparece en el extremo contrario de la flecha de la dependencia. Antónimo: proveedor.

Véase dependencia.

colaboración

Una especificación de una relación contextual entre instancias que interactúan dentro de un contexto para implementar una funcionalidad deseada.

Una colaboración es una descripción de una estructura. La colaboración es una forma de clasificador estructurado. El comportamiento de una colaboración se puede especificar mediante interacciones que muestran flujos de mensajes en la colaboración a lo largo del tiempo.

Véase también conector, rol, interacción, mensaje.

Semántica

Un comportamiento se implementa mediante grupos de objetos que intercambian mensajes dentro de un contexto para conseguir un propósito. Para comprender el mecanismo utilizado en el diseño, es importante centrarse en los patrones de colaboración involucrados en conseguir un propósito o un conjunto relacionado de propósitos, proyectados desde el sistema más grande dentro del cual también satisfacen otros propósitos. Una colección de objetos y enlaces que trabajan juntos para lograr un propósito reciben el nombre de colaboración. Una secuencia de mensajes dentro de una colaboración que implementan un comportamiento se denomina interacción. Una colaboración es una descripción de una "sociedad de objetos". Es un tipo de comentario sobre un fragmento de un modelo de clases que explica cómo trabajan juntos un grupo de objetos para llevar a cabo un propósito particular de forma única para una situación particular.

Por ejemplo, una venta comercial representa un conjunto de objetos que tienen ciertas relaciones entre sí dentro de la transacción. La relación carece de significado fuera de la transacción. Los roles de una venta incluyen comprador, vendedor y corredor. Para realizar una interacción específica, como la venta de una casa, los participantes que desempeñan varios roles intercambian una cierta secuencia de mensajes, como hacer una oferta o firmar un contrato.

Los flujos de mensajes dentro de la colaboración pueden ser opcionalmente especificados mediante interacciones, las cuales especifican secuencias de comportamiento legales. Los mensajes en una interacción se intercambian entre roles sobre conectores dentro de la colaboración.

Una colaboración consiste en roles. Un rol es una descripción de un participante en una interacción. Estructuralmente, es una ranura que se puede vincular a una instancia de un clasificador dentro de una instancia de la colaboración. Un rol puede tener un tipo que restringe las instancias con las que se puede vincular. El mismo clasificador puede jugar diferentes roles en la misma colaboración o en diferentes colaboraciones cada uno de los cuales se ocuparía por un objeto distinto. (También es posible para un objeto, desempeñar múltiples roles en una o muchas instancias de colaboraciones.) Por ejemplo, dentro de una transacción comercial, una parte puede ser el vendedor y la otra el comprador, aunque ambos participantes sean empresas. El **vendedor** y el **comprador** son roles del tipo **Empresa** dentro de la colaboración **Venta**. Los roles sólo tienen significado dentro de la colaboración, fuera de las misma carecen de él. Es más, en otra colaboración los roles pueden estar a la inversa. Un objeto puede ser el **comprador** en una instancia de colaboración y un **vendedor** en otra. El mismo objeto puede desempeñar múltiples roles en diferentes colaboraciones. El ámbito restringido de un rol contrasta con el de una asociación. Una asociación describe una relación que tiene significado global para una clase en todos los contextos, tanto si un objeto realmente participa en la asociación. Una colaboración define relaciones que se restringen a un contexto y carecen de significado fuera de ese contexto.

Un conector es una relación entre dos roles dentro de una determinada colaboración. Es una relación que existe sólo dentro de la colaboración. Los conectores pueden tener tipos, los cuales deben ser asociaciones que restringen al conector a ser implementado como un enlace de la asociación dada. Si no se proporciona un tipo para un conector, representa una conexión transitoria dentro del contexto de la conexión, que debería ser implementado por un parámetro, variable local u otro mecanismo. Por ejemplo, en la transacción inmobiliaria mencionada con anterioridad, el comprador y el vendedor están relacionados en virtud de su participación en la misma venta. No tiene ninguna relación permanente fuera de la colaboración inmobiliaria.

Una colaboración describe el contexto para una operación, un caso de uso u otro tipo de comportamiento. Describe el contexto en el que se ejecuta la implementación de una operación o caso de uso, es decir, la colección de objetos y enlaces que existen cuando comienza la ejecución y las instancias que se crean o destruyen durante la ejecución. Se deben especificar las secuencias de comportamiento en interacciones, que se muestran como diagramas de secuencia o diagramas de comportamiento.

Una colaboración también puede describir la implementación de una clase. Una colaboración para una clase es la unión de las colaboraciones para sus operaciones. Se pueden idear diferentes colaboraciones para la misma clase, sistema o subsistema, de forma que cada colaboración muestre el conjunto de atributos, operadores y objetos relacionados que son relevantes para una vista de la entidad, como la implementación de una determinada operación.

Patrones. Una colaboración parametrizada representa una construcción de diseño que se puede reutilizar en varios diseños. Normalmente, los clasificadores base son parámetros. Tales colaboraciones parametrizadas capturan la estructura de un patrón.

Véase plantilla.

Un patrón de diseño se instancia proporcionando clasificadores reales para los parámetros de los clasificadores base. Cada instanciación produce una colaboración entre un conjunto específico de clasificadores en el modelo. Se puede ligar un patrón más de una vez a diferentes conjuntos de clasificadores dentro del modelo, evitando la necesidad de definir una colaboración para cada uso. Por ejemplo, un patrón modelo-vista define una relación general entre elementos del modelo. Se puede ligar a muchos pares de clases que representan parejas modelo-vista. Cada pareja de clases modelo-vista representa una ligadura del patrón. Una de esas parejas puede ser una casa y una fotografía de la misma, otra pareja puede ser una acción y un gráfico que muestra el precio actual del valor.

Obsérvese que un patrón también involucra directrices para el uso y ventajas y desventajas explícitas. Estas se pueden plasmar en forma de notas o en documentos de texto separados.

Capas de colaboraciones. Una colaboración se puede expresar a diferentes niveles de granularidad. Se puede refinar una colaboración de grano grueso para producir otra colaboración de grano más fino. Esto se consigue expandiendo una o más operaciones de la colaboración de alto nivel en distintas colaboraciones de nivel más bajo, una por operación.

Las colaboraciones se pueden anidar. Se puede implementar una colaboración en términos de colaboraciones subordinadas, cada una de las cuales implementa parte de la funcionalidad global. Las colaboraciones subordinadas se conectan indirectamente mediante su participación en la colaboración más externa, es decir, sus definiciones son independientes pero la salida de una es la entrada de otra.

Ligadura en tiempo de ejecución. En tiempo de ejecución, los objetos y los enlaces se ligan a los roles y conectores de la colaboración. Una instancia de una colaboración no es propietaria de las instancias ligadas a sus roles (a diferencia de una instancia de una clase estructurada, que es propietaria de sus partes). Simplemente las referencia y establece una relación contextual entre los objetos ligados a sus roles durante la duración de la instancia de la colaboración. Los objetos que desempeñan los roles deben existir previamente. Deben ser compatibles con los tipos declarados de los roles con los que se les liga.

Los enlaces que se corresponden con los conectores se crean a menudo como parte de la creación de una instancia de una colaboración, especialmente cuando el conector no tiene una asociación subyacente.

Un objeto se puede ligar a uno o más roles, normalmente en diferentes colaboraciones. Si un objeto se liga a múltiples roles, entonces representa una interacción "accidental" entre roles, es decir, una interacción que no es inherente en los propios roles, si no sólo un efecto lateral de su uso en un contexto más amplio. A menudo, un objeto juega roles en más de una colaboración como parte de una colaboración más grande. Esto es una interacción "inherente" entre los roles. Este solapamiento entre colaboraciones proporciona un flujo de control y de información implícito entre ellos.

Notación

La definición de una colaboración se muestra mediante una elipse de línea discontinua con dos compartimentos: el compartimento superior muestra el nombre de la colaboración, mientras que el inferior muestra su estructura como un conjunto de iconos de rol conectados mediante conectores. Un icono de rol se representa mediante un rectángulo que contiene el nombre del rol y su tipo, con el siguiente formato:

`Nombre`: **`Tipo`** [**`multiplicidad`**]

La multiplicidad es opcional; si se omite, el valor por defecto es uno.

Un conector se muestra como una línea o ruta que conecta dos iconos de rol. *Véase* la Figura 14.61.

Se puede utilizar una notación alternativa cuando no hay conectores. La colaboración se representa como una elipse de línea discontinua con un único compartimento que contiene el nombre de la definición de la colaboración. Para cada rol se dibuja una línea discontinua al rectángulo de clase que representa el tipo. La línea se etiqueta con el nombre del rol en el extremo próximo al rectángulo de tipo. A menudo se vincula una nota a la elipse de la colaboración para mostrar restricciones. *Véase* la Figura 14.62.

Véase diagrama de secuencia para el uso de colaboraciones para establecer las líneas de vida de una interacción.

Historia

Aunque la intención global es más o menos la misma que en UML1, se ha actualizado considerablemente el concepto de colaboración en UML2. Ahora hay una separación entre la estructura estática, la colaboración, y la estructura dinámica, la interacción. Internamente, se trata a la colaboración como un tipo de clasificador estructurado, en el que las partes representan roles, por lo

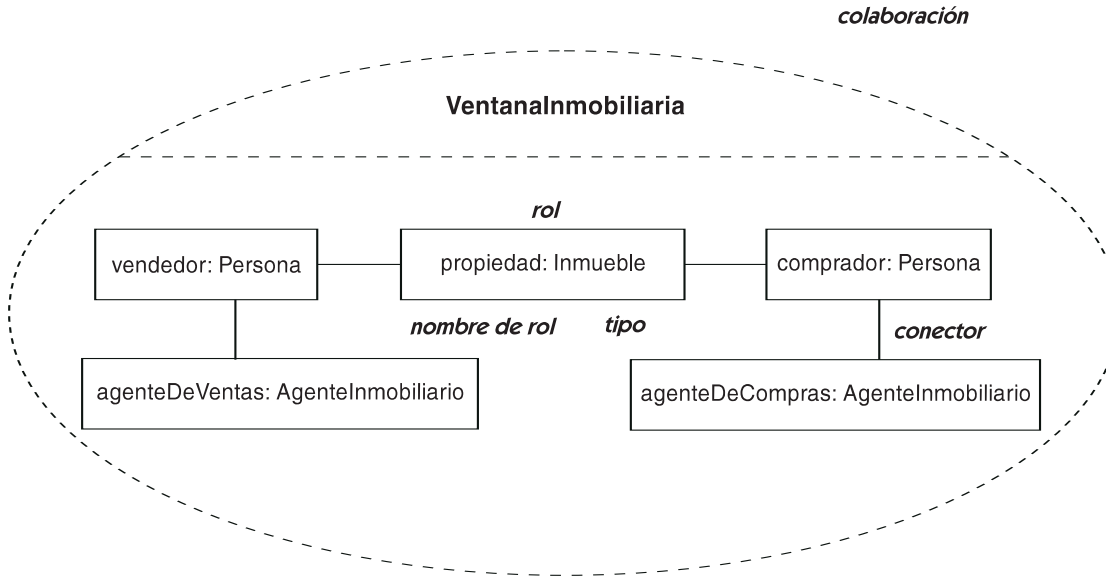


Figura 14.61 Notación de colaboración

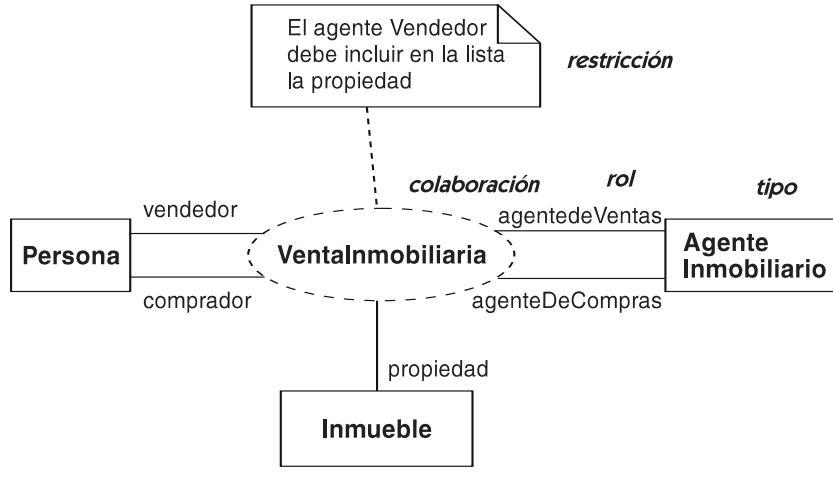


Figura 14.62 Notación alternativa de colaboración

que la parametrización de UML1 ya no es necesaria. Se han eliminados lo términos *rol de clasificador*, *rol de asociación* y *rol del extremo de la asociación*. Tampoco son necesarios ya conceptos como las generalizaciones internas.

Discusión

La especificación de UML no utiliza la palabra *instancia* para las colaboraciones, probablemente por la idea equivocada de que las instancias se corresponden con las clases de C++. Nosotros

utilizamos la palabra porque es la palabra habitual para el concepto, y hay poco peligro de confusión mientras los lectores tengan en mente que la instanciación no implica una implementación específica. En la mayoría de los casos, las colaboraciones no se corresponden con clases de implementación. Son conceptos emergentes, pero eso no significa que no tengan instancias una vez que la palabra es adecuadamente comprendida.

comentario

Anotación vinculada a un elemento o a una colección de elementos. Un comentario no tiene un significado directo, pero puede mostrar información semántica u otra información significativa para el modelador o para una herramienta, como una restricción o el cuerpo de un método.

Véase también nota.

Semántica

Un comentario contiene una cadena, pero también puede incluir documentos incrustados si la herramienta de modelado lo permite. Un comentario puede estar vinculado a un elemento del modelo, a un elemento de presentación o a un conjunto de elementos. Proporciona información textual de información arbitraria, pero no tiene ningún impacto semántico. Los comentarios proporcionan información a los modeladores y se puede utilizar para buscar modelos.

Notación

Los comentarios se muestran en símbolos de nota, que se representan como rectángulos con la esquina superior derecha doblada (como la oreja de un perro), vinculados por medio de una línea o líneas discontinuas al elemento o elementos para los que se aplica el comentario (Figura 14.63). Las herramientas de modelado son libres de proporcionar formatos adicionales para mostrar comentarios y navegar por ellos, como ventanas emergentes, fuentes especiales y demás.

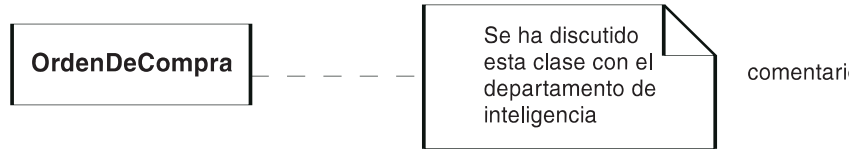


Figura 14.63 Comentario

comillas

Pequeñas marcas de ángulo doble (« ») utilizadas como comillas en Español, Francés, Italiano y otros lenguajes. En la notación UML se utilizan para encerrar palabras clave y nombres de estereotipo. Por ejemplo: «**bind**», «**instanceOf**». Las comillas están disponibles en la mayoría de las tipografías, de forma que no hay ninguna excusa para no usarlas, pero a menudo la tipografía la sustituye por dos paréntesis angulares (**« »**).

Véase también uso de la tipografía.

compartimento

Subdivisión gráfica de un símbolo cerrado, como el rectángulo de una clase dividido verticalmente en rectángulos más pequeños. Cada compartimento muestra 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 es un compartimento de nombre de clase, que contiene un símbolo de estereotipo y/o un nombre de estereotipo, un nombre de clase y una cadena que muestra diversas propiedades de la clase. Se puede suprimir cierta información dependiendo del elemento.

Un *compartimento de lista* contiene una lista de cadenas que codifican componentes del elemento. Un ejemplo es una lista de atributos. La codificación depende del tipo de componente. Los elementos de la lista se pueden mostrar en su orden natural dentro del modelo o bien se pueden ordenar por una o más de sus propiedades (en cuyo caso, el orden natural no será visible). Por ejemplo, una lista de atributos se podría ordenar en primer lugar por su visibilidad y después por nombre. Se pueden mostrar o suprimir las entradas de la lista en base a las propiedades de los elementos del modelo. Por ejemplo, un compartimento de atributos podría mostrar sólo los atributos públicos. Se pueden aplicar estereotipos y palabras clave a entradas individuales anteponiéndolos a dicha entrada. También se puede aplicar un estereotipo o palabra clave a un conjunto de entradas insertándolo como si fuera otra entrada de la lista más. En ese caso, afectan a todas las entradas que aparecen a continuación hasta que acabe la lista o aparezca otro estereotipo o palabra clave. La cadena **«constructor»** colocada en una línea separada en una lista de operaciones estereotiparía las siguientes operaciones como constructores, pero la cadena **«query»** más abajo aún de la lista anularía la primera declaración y la reemplazaría por el estereotipo **«query»**.

Una *región* es un área que contiene una subimagen que muestra la subestructura del elemento, a menudo, potencialmente recursiva. Un ejemplo es una región de estado anidada. La naturaleza de la subimagen es característica del elemento del modelo. Se pueden incluir tanto regiones como compartimentos de texto en un solo símbolo, pero puede ser confuso. Las regiones se utilizan a menudo para elementos recursivos, y el texto para elementos terminales sin subestructura recursiva.

Una clase tiene tres compartimentos predefinidos: nombre, un compartimento fijo; atributos, un compartimento de lista; y operaciones, otro compartimento de lista. Los modeladores pueden añadir otro compartimento al rectángulo y colocar su nombre al principio del compartimento, en una fuente distintiva (por ejemplo, pequeña y en negrita). Si se desea, también se pueden colocar los nombres en los compartimentos de atributos y operaciones.

La sintaxis gráfica depende del elemento y del tipo de compartimento. La Figura 14.64 muestra un compartimento para señales. La Figura 14.261 muestra un compartimento para responsabilidades.

Figura 14.64 Compartimento en una clase

complete

Palabra clave para un conjunto de generalizaciones cuyos subtipos cubren todos los casos posibles del supertipo.

Véase conjunto de generalizaciones.

componente

Parte modular del diseño de un sistema que oculta su implementación tras un conjunto de interfaces externas. Dentro de un sistema, los componentes que satisfacen los mismos interfaces pueden ser sustituidos libremente.

Semántica

Un componente describe una pieza modular de un sistema lógico o físico cuyo comportamiento visible externamente puede describirse de una forma mucho más precisa que su implementación. El comportamiento visible externamente se representa como un conjunto de interfaces, posiblemente con comportamientos adjuntos para más detalle, tales como máquinas de estado de protocolo o casos de uso. Un sistema especifica su implementación utilizando ranuras que tienen componentes que satisfacen interfaces específicos. Se puede sustituir cualquier componente que satisfaga el conjunto de interfaces en una ranura. La separación entre interfaces externas e implementación interna permite que el desarrollo de sistemas y sus partes sean fácilmente separables, permitiendo la sustitución de distintos componentes en una ranura cualquiera del sistema y la utilización de un mismo componente en sistemas distintos. Esta modularidad incrementa la reutilización tanto de las estructuras de los sistemas como de sus componentes.

Los componentes tienen dos aspectos. Definen la apariencia externa de una pieza de un sistema e implementan dicha funcionalidad. Los componentes sin implementación son tipos abstractos. Se utilizan para especificar los requisitos de una ranura del sistema. Los componentes con implementación deben ser subtipos de un componente abstracto. Los componentes concretos pueden sustituir a sus antecesores. (Pueden sustituir en cualquier lugar cuando satisfacen las interfaces, pero se garantiza que un componente hijo satisface las interfaces de su padre.)

Un componente puede ser o no instanciado directamente en tiempo de ejecución. Un componente instanciado indirectamente es implementado, o realizado, por un conjunto de clasificado-

res. El componente en sí mismo no aparece en la implementación; sirve como un diseño que debe seguir la implementación. El conjunto de clasificadores que lo realizan deben cubrir el conjunto entero de operaciones especificado en la interfaz proporcionada por el componente. La forma de implementar el componente es responsabilidad del diseñador.

Un componente directamente instanciado especifica su propia implementación encapsulada. Se puede especificar la implementación de un componente como un clasificador estructurado que contiene partes y conectores. Cuando se instancia el componente, también se instancia su estructura interna. La relación entre el componente y su implementación es explícita. Se pueden especificar las partes por medio de otros componentes, de forma que los componentes pueden formar un ensamblado recursivo.

Se pueden especificar las relaciones entre componentes mediante dependencias entre sus interfaces. Las interfaces de un componente describen la funcionalidad que soporta. Las interfaces pueden ser o bien interfaces proporcionadas o bien interfaces requeridas. Una interfaz proporcionada describe las operaciones que un componente garantiza que tiene disponibles para otros componentes. Finalmente, cada operación de la interfaz debe mapearse a un elemento de implementación soportado por el componente. El componente puede proporcionar operaciones adicionales, pero al menos debe proporcionar todas las operaciones de las interfaces proporcionadas. Una interfaz requerida describe la funcionalidad que necesita de otros componentes. Puede que el componente no utilice siempre todas las operaciones, pero se garantiza que funciona si los componentes que utiliza proporcionan al menos todas las operaciones de la interfaz. Los comportamientos adjuntos a las interfaces pueden especificar restricciones adicionales en la ordenación de mensajes entre componentes.

Para mayor precisión en la especificación e interconexión, se pueden ubicar las interfaces de un componente en puertos. Cada puerto agrupa un conjunto de interfaces proporcionadas y requeridas, e incluye opcionalmente un comportamiento adjunto que especifica las interacciones a través del puerto. Los mensajes hacia un componente con puertos se dirigen a los puertos específicos, y la implementación de los componentes puede determinar a qué puerto llegan los mensajes y de qué puerto parten.

Las “conexiones” internas de un componente se especifican con conectores de ensamblado y conectores de delegación. Dentro de la implementación de un componente, los conectores de ensamblado conectan puertos de diferentes subcomponentes. (Un subcomponente es una parte del componente cuyo tipo es un componente menor.) Un mensaje enviado a un puerto de un componente se recibe en un puerto conectado de otro componente. Pueden estar conectados a través de sus puertos un conjunto de subcomponentes. Un subcomponente no necesita saber nada acerca de otros subcomponentes, excepto que existen y que satisfacen las restricciones en los puertos conectados. En muchos marcos de trabajo no se necesita código adicional si el modelador conecta los componentes de manera correcta. La comunicación entre componentes se modela por medio de sus puertos. Los puertos deben existir en el código físico, o bien pueden ser compilados aparte.

Un conector de delegación conecta un puerto externo de un componente con un puerto de uno de sus subcomponentes internos. Un mensaje recibido por el puerto externo se pasa al puerto en el componente interno; un mensaje enviado por el puerto interno se pasa al puerto externo y de ahí al componente conectado a él. Los conectores de ensamblado permiten el ensamblado de componentes de un cierto nivel en un componente de alto nivel. Los conectores de delegación permiten la implementación de operaciones de alto nivel por medio de componentes de bajo nivel.

Los componentes concretos también pueden tener artefactos, es decir, piezas físicas de implementación, tales como código, guiones, elementos de hipertexto y demás. Son la manifestación física del modelo sobre una plataforma específica. Cada artefacto puede manifestar uno o varios elementos del modelo.

Un componente también es un espacio de nombres que se puede utilizar para organizar elementos de diseño, casos de uso, interacciones y artefactos de código.

Hay varios estereotipos de componente predefinidos, incluyendo subsystem, specification y realization.

Artefactos. Un artefacto es una unidad física de la construcción de un sistema. El término incluye código software (fuente, binario o ejecutable) o equivalentes, como guiones o archivos de comandos. Los artefactos existen en el dominio de la implementación —son unidades físicas, en computadoras, que pueden ser almacenados, movidos y manipulados. Los modelos pueden mostrar dependencias entre componentes y artefactos, como dependencias de compilador y de tiempo de ejecución o dependencias de información en una organización humana. También se pueden usar artefactos para mostrar unidades de implementación que existen en tiempo de ejecución, incluyendo su ubicación en instancias de un nodo. Los artefactos se pueden desplegar en nodos físicos.

Ejemplo

Por ejemplo, un corrector ortográfico puede ser un componente dentro de un editor de texto, en un sistema de correo electrónico y en otros marcos de trabajo. La interfaz proporcionada podría consistir en una operación que comprueba un bloque de texto y una operación que añade palabras al diccionario de excepciones de un documento. El interfaz requerido podría consistir en el acceso a un diccionario estándar y a un diccionario de excepciones. Diferentes correctores ortográficos podrían tener diferentes interfaces de usuario o diferentes características de funcionamiento, pero cualquiera de ellos podría ser sustituido sin afectar a la aplicación que lo contiene.

Estructura

Un componente tiene una interfaz requerida y una interfaz proporcionada.

Un componente puede ser implementado directa o indirectamente.

Un componente tiene un conjunto de referencias a clasificadores que realizan su comportamiento. Este conjunto puede estar vacío si el componente proporciona una implementación directa.

Un componente puede tener operaciones y atributos, que deben cubrir la especificación de sus interfaces proporcionadas.

Un componente puede tener un conjunto de elementos dentro de su espacio de nombres, como un paquete. Esos elementos pueden incluir artefactos.

Un componente, como una clase estructurada, puede tener un conjunto de partes y conectores que componen su implementación directa. También puede tener puertos externos que organizan sus interfaces.

Notación

Un componente se representa como un rectángulo con la palabra clave **«component»**. En vez de la palabra clave, o además de ella, puede contener un icono de componente en su esquina supe-

rior derecha. Este icono es un pequeño rectángulo con dos rectángulos más pequeños aún saliendo de su lado. El nombre del componente (como un tipo) se coloca dentro del rectángulo más externo (Figura 14.65).

Figura 14.65 Notación de componentes (alternativas)

Las operaciones y atributos disponibles a objetos externos se pueden representar directamente en compartimentos del rectángulo. Sin embargo, en la mayoría de los casos se suprimen a favor de las interfaces del componente.

Se pueden listar las interfaces en un compartimento del rectángulo. Se pueden poner las palabras clave «**provided**» y «**required**» en los nombres individuales de los interfaces o en líneas sin nombres de interfaz, en cuyo caso se aplicarán a la lista de nombres que le siguen.

Alternativamente, las interfaces se pueden representar adjuntando iconos de interfaz al borde del rectángulo. Una interfaz proporcionada se representa como un pequeño círculo unido al rectángulo por una línea. *Véase* la Figura 14.66.

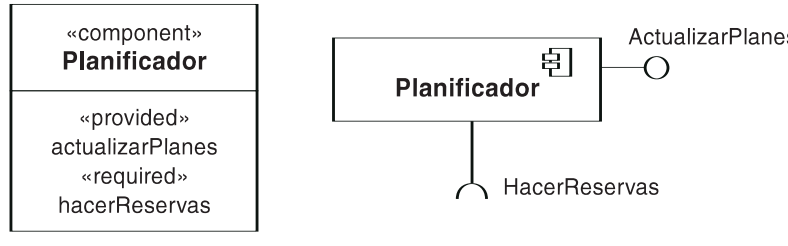


Figura 14.66 Notación de interfaces de componente (alternativas)

También se pueden representar las interfaces en una forma totalmente explícita (Figura 14.67): Una interfaz se representa como un rectángulo con la palabra clave **«interface»**. Un compartimento en el rectángulo contiene una lista de operaciones. Un componente se conecta a una interfaz proporcionada mediante una línea discontinua con una cabeza de flecha triangular. Un componente se conecta a una interfaz requerida mediante una línea discontinua con una cabeza de flecha abierta y la palabra clave **«use»**.

Un puerto se representa como un pequeño rectángulo en el límite del rectángulo del componente (Figura 14.68). El nombre del puerto se coloca cerca del pequeño rectángulo. Se pueden adjuntar al pequeño rectángulo uno o más símbolos de interfaz (proporcionadas y/o requeridas) mediante líneas continuas.

Las clases y los artefactos que realizan la implementación de un componente se pueden conectar al componente mediante flechas de dependencia (una línea discontinua con cabeza de flecha abierta). Alternativamente, se pueden anidar dentro del rectángulo del componente.

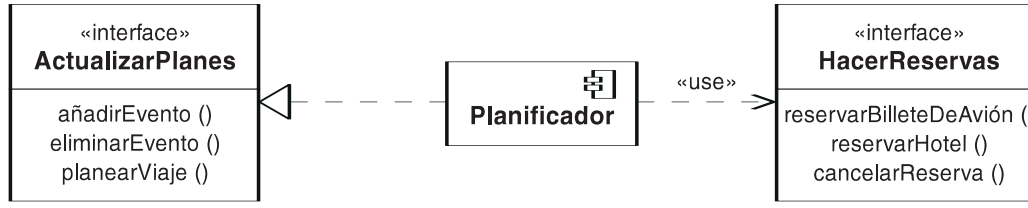


Figura 14.67 Notación explícita de interfaz

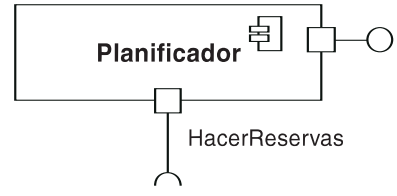


Figura 14.68 Notación de puerto de componente

La implementación directa de un componente se representa como un diagrama de estructura interna de partes y conectores anidados dentro del rectángulo del componente (Figura 14.69). Los tipos de las partes pueden ser clases o subcomponentes.

Se pueden conectar subcomponentes aproximando el semicírculo de una interfaz requerida al círculo de una interfaz proporcionada compatible en una notación de "rótula". Los conectores de

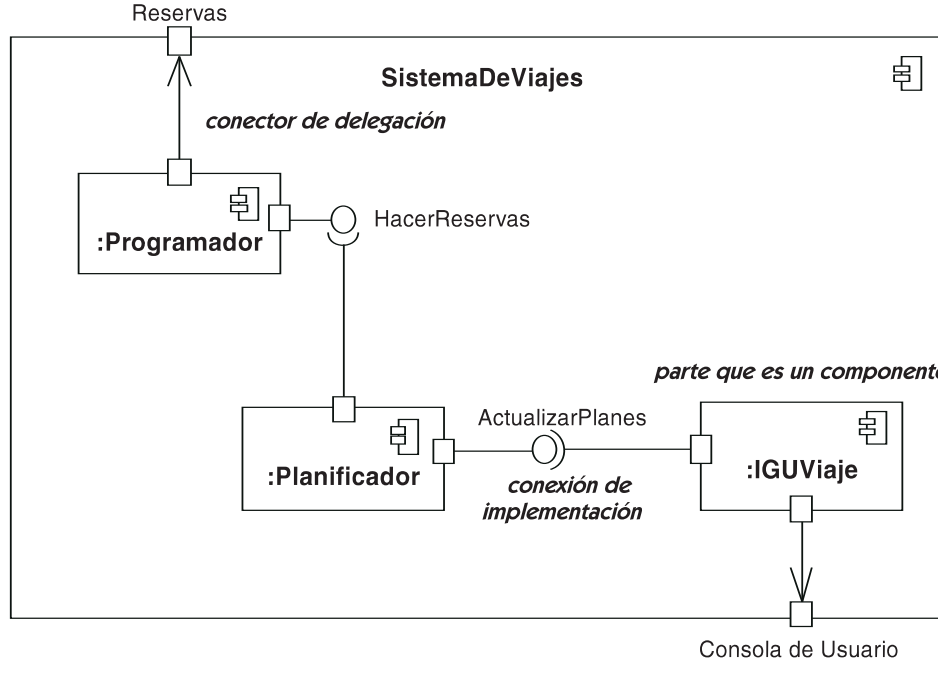


Figura 14.69 Estructura interna de un componente

delegación se representan mediante líneas continuas con cabeza de flecha abierta entre un puerto en el límite del componente global y un símbolo de interfaz o puerto de un subcomponente. Opcionalmente se puede colocar en la flecha la palabra clave «**delegate**».

Historia

Se ha redirigido el enfoque de los componentes desde la vista física de UML1 hacia un concepto más lógico de un componente como una unidad fuertemente encapsulada. Se han separado los componentes del concepto físico de artefacto de manera que se puedan utilizar en modelos conceptuales. La distinción entre una clase estructurada y un componente es un tanto imprecisa y más una cuestión de intención que una semántica firme.

comportamiento

Una especificación de cómo el estado de un clasificador cambia a lo largo del tiempo. El comportamiento está especializado en las actividades, interacciones y máquinas de estados. Un comportamiento describe la dinámica de un clasificador completo como una unidad. Un clasificador también posee un conjunto de características de comportamiento (como las operaciones) que se pueden invocar de forma independiente como parte de su semántica global.

Semántica

Comportamiento es un término genérico que permite especificar cómo el estado de una instancia de un clasificador cambia a lo largo del tiempo en respuesta a eventos externos y a cómputos internos. El clasificador puede ser una entidad concreta (como una clase) o una colaboración. El comportamiento se puede especificar para la instancia como un todo o para operaciones individuales que se pueden ejecutar sobre la instancia. Incluye actividades, interacciones y máquinas de estados. Cada una de ellas tiene su propia estructura detallada, semántica y utilización, las cuales se describen en distintas entradas.

Un comportamiento puede tener parámetros, los cuales se deben corresponder con los parámetros de la operación que implementan. Un comportamiento vinculado a una operación entra en vigor cuando se invoca la operación y continúa hasta que se completa la ejecución de la operación. Un comportamiento vinculado a un clasificador completo entra en vigor cuando se crea una instancia del clasificador y continúa mientras la instancia existe.

Un comportamiento puede ser reentrante y no reentrante. Un comportamiento reentrante se puede invocar dentro de una ejecución existente del mismo comportamiento.

Un comportamiento puede redefinir otro comportamiento. Las reglas para la redefinición se especifican bajo cada tipo de comportamiento.

Las precondiciones y las postcondiciones se pueden vincular a los comportamientos. Son restricciones que se deben satisfacer cuando un comportamiento comienza y termina. Tiene la intención de ser afirmaciones cuyo fallo indica un error de modelado o implementación, no partes ejecutables de la especificación del comportamiento.

Un comportamiento se modela como un clasificador, pero más a menudo describe la ejecución de un clasificador distinto, al que se denomina contexto para el clasificador. Los mensajes no se envían directamente al comportamiento, sino al contexto. Sin embargo, algunas veces un

comportamiento puede servir como su propio contexto. En esta situación, una instancia del comportamiento puede recibir directamente los mensajes de otras instancias. Un comportamiento como contexto se puede utilizar para modelar cosas como procesos de sistemas operativos, tareas en flujos de trabajo y otros tipos de comportamientos materializados.

Se puede redefinir el comportamiento de un clasificador. Un comportamiento se puede especializar como con cualquier clasificador. *Véase* redefinición (de comportamiento).

Semántica de ejecución

En la instanciación de una clasificador, se crea e inicializa una instancia de acuerdo a las reglas por defecto y explícitas de inicialización. Un comportamiento vinculado a un clasificador describe la posterior ejecución de la instancia. Una ejecución del comportamiento correspondiente a la instancia tiene su propio espacio de estado para variables locales y para el estado de la propia ocurrencia del comportamiento, no compartido con cualquier otra ocurrencia de comportamiento. Una ejecución de un comportamiento tiene una referencia a su contexto, es decir, a las instancias cuya ejecución representa. Si la ejecución del comportamiento alcanza un estado final, la instancia del clasificador se destruye y deja de existir.

En la invocación de una operación, se crea una ejecución del comportamiento vinculado a la operación con copias de los argumentos reales de la invocación. La ejecución del comportamiento tiene su propio espacio de estado para variables locales y su propio estado, independiente de cualquier otra invocación de la misma instancia. La ejecución de cada comportamiento se realiza independientemente de otras ejecuciones, excepto para la extensión en la que explícitamente intercambian mensajes o en la que implícitamente intercambian datos de objetos compartidos. Cuando la ejecución de una operación se completa, la ejecución del comportamiento emite valores de retorno (si la operación es síncrona) y deja de existir.

composición

Una forma fuerte de asociación de agregación con una posesión fuerte de las partes por el elemento compuesto y tiempo de vida de las partes coincidente con el elemento compuesto. Una parte sólo puede pertenecer a un elemento compuesto al mismo tiempo. Las partes con multiplicidad variable se pueden crear después del propio elemento compuesto. Pero una vez creadas, viven y mueren con él (es decir, comparten el tiempo de vida). Dichas partes también pueden ser eliminadas explícitamente antes de la muerte del elemento compuesto. La composición es recursiva.

Véase también agregación, asociación, objeto compuesto.

Semántica

Hay una forma fuerte de asociación de agregación llamada composición. Un objeto compuesto es una asociación de agregación con las restricciones adicionales de que un objeto puede ser parte únicamente de un solo objeto compuesto al mismo tiempo, y de que el objeto compuesto es el único responsable de la gestión 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 composición) forman un bosque de árboles formado por objetos y enlaces de composición. Una parte de un objeto compuesto no puede ser compartida por dos objetos compues-

tos. Esto concuerda con la intuición normal de la composición física de partes —una parte no puede ser parte directa de dos objetos (aunque de forma indirecta puede ser parte de varios objetos a diferentes niveles de granularidad en el árbol).

Diciendo que tiene la responsabilidad de la gestión de sus partes, queremos decir que el objeto compuesto es responsable de su creación y destrucción. En términos de implementación, es responsable de su asignación de memoria. En un modelo lógico, el concepto se define de forma menos rígida y debe ser aplicado cuando sea apropiado. Durante su instanciación, un objeto compuesto debe asegurar que todas sus partes se han instanciado y adjuntado a él correctamente. Puede crear una parte por sí mismo o puede asumir la responsabilidad de una parte existente. Pero durante la vida del objeto compuesto, ningún otro objeto puede tener responsabilidad sobre ella. Esto significa que se puede diseñar el comportamiento de una clase compuesta con el conocimiento de que ninguna otra clase destruirá o desasignará las partes. Un objeto compuesto puede añadir partes adicionales durante su vida (si la multiplicidad lo permite), siempre que asuma la exclusiva responsabilidad sobre ellos. Puede eliminar partes, siempre que las multiplicidades lo permitan y otro objeto asuma la responsabilidad sobre ellas. Si el objeto compuesto es destruido, o bien debe destruir todas sus partes o pasar la responsabilidad sobre ellas a otros objetos.

Bajo ciertas circunstancias, una clase puede permitir a otros objetos crear o destruir partes de una de sus instancias, pero reteniendo la responsabilidad final sobre ellos. En particular, debe crear todas las partes necesarias en su inicialización y destruir todas las partes que queden en su destrucción.

Esta definición engloba la mayoría de las intuiciones comunes, tanto lógicas como de implementación, de la composición. Por ejemplo, un registro que contiene una lista de valores es una implementación común de un objeto y sus atributos. Cuando se asigna en memoria el registro, también se asigna automáticamente la memoria necesaria para los atributos, pero puede ser necesario inicializar los valores de los atributos. Mientras existe el registro, no se puede eliminar ningún atributo de él. Cuando se desasigna el registro, también se desasigna la memoria para los atributos. Ningún otro objeto puede afectar a la asignación de un atributo dentro del registro. Las propiedades físicas de un registro imponen las restricciones de un elemento compuesto.

Esta definición de composición funciona bien con recolección de basura. Si se destruye el propio elemento compuesto, se destruye el único puntero a la parte y se vuelve inaccesible y por tanto susceptible a ser recuperado por el recolector de basura. No obstante, la recuperación de partes inaccesibles es sencilla incluso con recolección de basura, lo que es una razón para distinguir la composición de otra agregación.

Observe que una parte no necesita ser implementada como una parte física de un único bloque de memoria con el elemento compuesto. Si la parte es separada del elemento compuesto, entonces dicho elemento compuesto tiene la responsabilidad de asignar y desasignar memoria para la parte cuando sea necesario. En C++, por ejemplo, los constructores y destructores facilitan la implementación de elementos compuestos.

Un objeto sólo puede ser parte de un objeto compuesto a la vez. Esto no excluye que una clase sea parte de más de una clase en diferentes momentos o en diferentes instancias, pero sólo puede existir un enlace de composición con un objeto en un momento dado. En otras palabras, hay una restricción entre los posibles elementos compuestos a los cuales debe pertenecer una parte. Además, un objeto puede ser parte de diferentes objetos compuestos durante su vida, pero sólo de uno a la vez.

Estructura

La propiedad de agregación en un extremo de la agregación o propiedad puede tener los valores siguientes.

ninguno	El clasificador adjunto no es un agregado o elemento compuesto
compartido	El clasificador adjunto es potencialmente un agregado compartido. El otro extremo es una parte.
compuesto	El clasificador adjunto es un elemento compuesto. El otro extremo es una parte

Al menos un extremo de una asociación debe tener el valor **ninguno**.

Notación

La composición se representa por medio de un adorno con forma de rombo negro en el extremo de una asociación, adjunto al elemento compuesto (Figura 14.70). La multiplicidad se puede representar de la manera habitual. Debe ser 1 o 0..1.

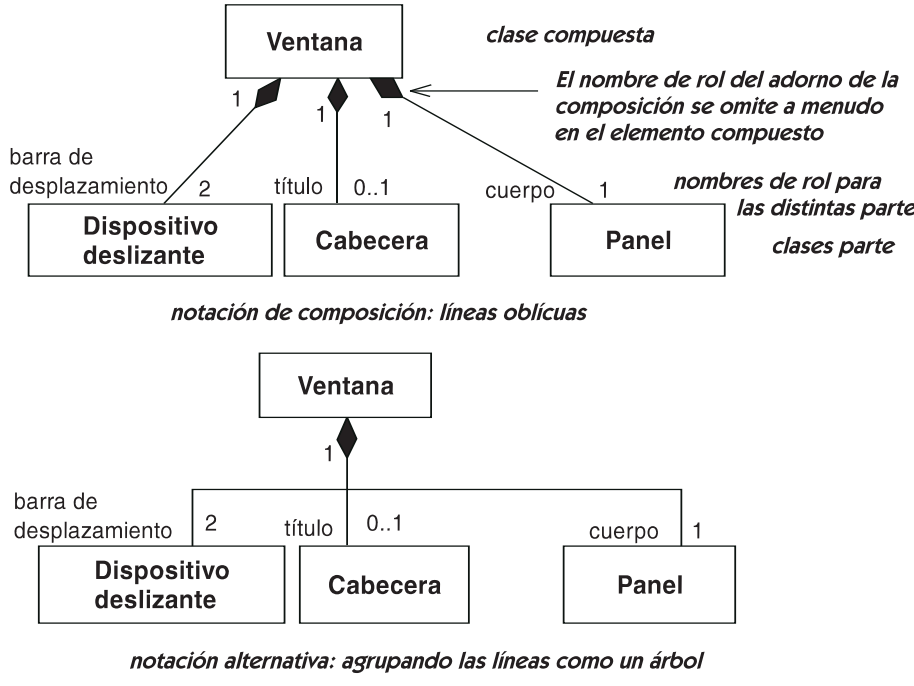


Figura 14.70 Notación de composición

Alternativamente, se puede representar la composición anidando gráficamente los símbolos de las partes dentro del símbolo del objeto compuesto (Figura 14.71). Esta es la notación para un

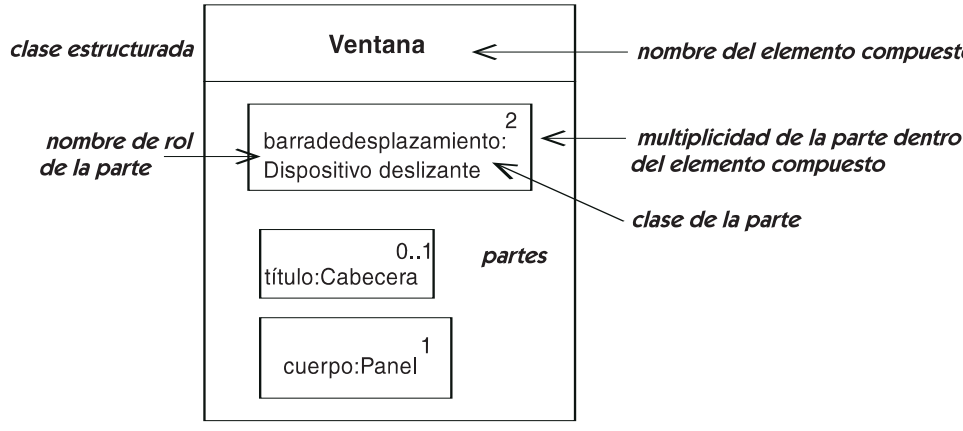


Figura 14.71 Composición de una clase estructurada como un anidamiento gráfico

clasificador estructurado. Un objeto anidado puede tener un nombre de rol dentro de la clase estructurada. El nombre se muestra por delante del tipo en la sintaxis

`nombre de rol` : **`nombre de clase`**

Un clasificador anidado puede tener una multiplicidad dentro de su elemento compuesto. La multiplicidad se muestra mediante una cadena de multiplicidad en la esquina superior derecha del símbolo de la parte o colocando dicha multiplicidad entre corchetes después del nombre del clasificador. Si se omite la multiplicidad, por defecto será uno.

Una línea dibujada por completo dentro del borde de una clase estructurada se considera que forma parte de la composición. Cualesquiera objetos conectados por un solo enlace de la asociación deben pertenecer al mismo elemento compuesto. Una asociación dibujada de forma que su línea atraviese el borde del elemento compuesto no se considera que forme parte de la composición. Representa una asociación normal. Cualesquiera objetos en un solo enlace de la asociación pueden pertenecer al mismo o a diferentes elementos compuestos (Figura 14.72).

Observe que los atributos normalmente son considerados como una relación de composición entre una clase y las clases de sus atributos (Figura 14.73). Pueden representar referencias a otras clases, es decir, asociaciones. Sin embargo, en general los atributos deberían reservarse para valores de datos primitivos (como números, cadenas y fechas) y no para referencias a clases, puesto que cualquier otra relación con las clases referenciadas no se puede ver en la notación de los atributos. Sin embargo, en los casos en que una clase es muy referenciada pero la navegación en la dirección contraria no es importante, puede ser de utilidad el uso de atributos que contengan referencias.

Observe que la notación de la composición se parece a la notación de la colaboración. Una colaboración es un tipo de clasificador estructurado en el que las partes están relacionadas por la colaboración en vez de ser piezas físicas de ella.

La Figura 14.74 muestra una composición multinivel.

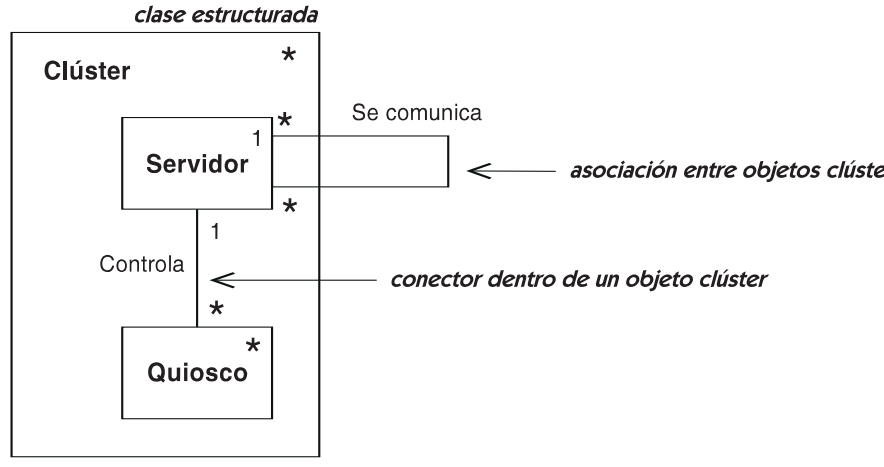


Figura 14.72 Asociación dentro y entre clases estructuradas

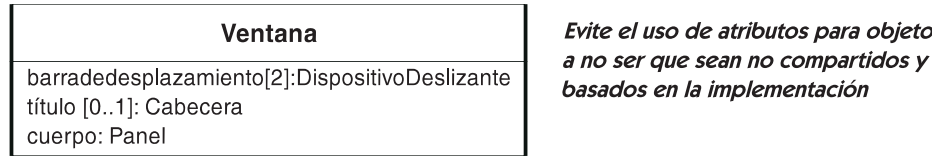


Figura 14.73 Los atributos son una forma de composición

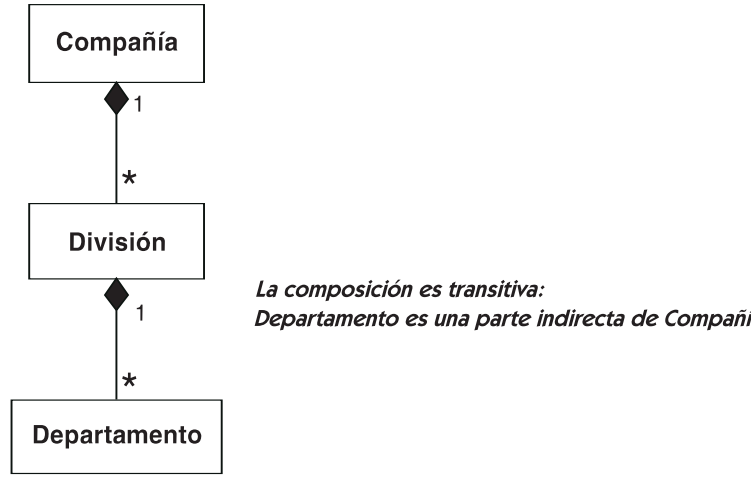


Figura 14.74 Composición multinivel

Discusión

(*Véase también* la discusión de agregación para obtener pautas de cuándo son apropiadas la agregación, composición y asociación simple.)

La composición y la agregación son metarrelaciones —trascienden las asociaciones individuales para imponer restricciones en el conjunto entero de asociaciones. La composición es significativa a través de relaciones de composición. Un objeto puede tener como mucho un enlace de composición (a un elemento compuesto), aunque potencialmente podría venir de más de una asociación de composición. En otras palabras, una clase podría mostrar más de una asociación de composición a otras clases, pero un objeto concreto sólo puede ser parte de otro único objeto, elegido de esas clases, al mismo tiempo. El gráfico completo de objetos y enlaces de composición y asociación debe ser acíclico, incluso si los enlaces provienen de diferentes asociaciones. Observe que dichas restricciones se aplican al dominio de las instancias —las asociaciones de agregación por sí mismas habitualmente forman ciclos, y las estructuras recursivas siempre requieren ciclos de asociaciones.

Considere el modelo de la Figura 14.75. Cada **Autenticación** es una parte de exactamente una **Transacción**, que puede ser o bien una **Compra** o una **Venta**. Sin embargo, cada **Transacción** no necesita tener una **Autenticación**. Con este fragmento, tenemos suficiente información para concluir que una **Autenticación** no tiene otras asociaciones de composición. Cada objeto de autenticación debe formar parte de un objeto de transacción (la multiplicidad es uno); un objeto sólo puede formar parte de un elemento compuesto como mucho (por definición); ahora mismo ya es parte de un elemento compuesto (como se muestra); de esta forma **Autenticación** no puede ser parte de ninguna otra asociación de composición. No hay peligro de que una **Autenticación** pueda tener que manejar su propio almacenamiento. Una **Transacción** siempre puede tomar la responsabilidad, aunque no todas las **Transaccion**es tienen **Autenticacion**es que necesiten gestionar. (Por supuesto, la **Autenticación** se puede gestionar a sí misma si el diseñador así lo decide.)

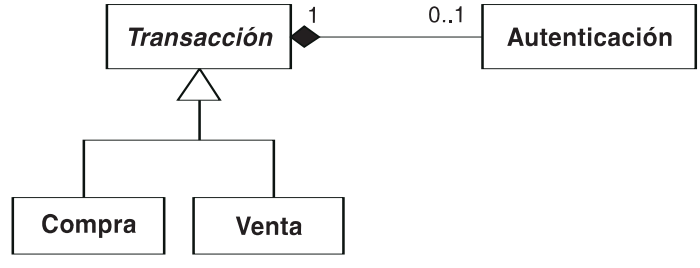


Figura 14.75 Composición a una clase abstracta compuesta

En la Figura 14.76, un **Autógrafo** puede ser opcionalmente parte o bien de una **Transacción** o de una **Carta**. No puede ser parte de las dos al mismo tiempo (por las reglas de la composición). Este modelo no impide que un **Autógrafo** empiece siendo parte de una **Carta** y luego pase a formar parte de una **Transacción** (en cuyo momento debe dejar de formar parte de la **Carta**). De hecho, un **Autógrafo** no necesita formar parte de nada. Además, de este fragmento del modelo no podemos descartar la posibilidad de que un **Autógrafo** sea opcionalmente parte de cualquier otra clase no mostrada en el diagrama o que pudiera añadirse después.

¿Qué ocurre si es necesario afirmar que cada **Autógrafo** debe formar parte o bien de una Carta o bien de una Transacción? Entonces se puede replantear el modelo, como en la Figura 14.75. Se puede añadir una nueva superclase abstracta por encima de **Carta** y **Transacción** (**Documento**)

Figura 14.76 Clase parte compartida

y mover la asociación de composición con **Autógrafo** a ella desde las clases originales. Al mismo tiempo, se convierte a uno la multiplicidad desde **Autógrafo** a **Documento**.

Hay un pequeño problema con este enfoque: la multiplicidad desde **Documento** a **Autógrafo** debe hacerse opcional, lo que debilita la inclusión obligatoria original de **Autógrafo** dentro de **Transacción**. Se puede modelar dicha situación utilizando la generalización de la propia asociación de composición, como se muestra en la Figura 14.77. La asociación de composición entre **Autógrafo** y **Transacción** se modela como un hijo de la asociación de composición entre **Autógrafo** y **Documento**. Pero sus multiplicidades se aclaran para el hijo. (Observe que permanecen consistentes con las heredadas, de forma que el hijo es sustituible por el padre.) Esta situación se puede modelar redefiniendo la asociación del padre por la asociación del hijo. Alternativamente, se puede utilizar el modelo original añadiendo una restricción entre las dos composiciones que una de ellas siempre debe mantener.

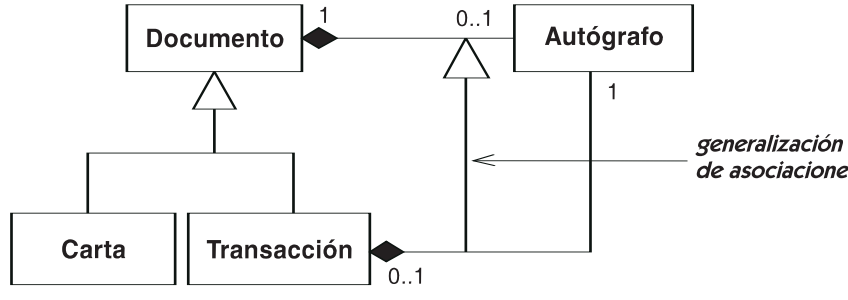


Figura 14.77 Generalización de asociación de composición

comunicación

Los objetos se comunican transmitiendo mensajes a otros objetos

Semántica

La comunicación entre objetos e hilos de ejecución se especifica por un modelo abstracto de transmisión de mensajes entre objetos. Se inicia la comunicación por medio de acciones de difusión, llamada, envío y respuesta. El destino de una transmisión puede ser explícito (acciones de llamada y envío), dependiente del contexto de ejecución (respuesta), o implícito en el entorno de ejecución. En una difusión, llamada o envío, los argumentos de la acción se copian en un mensaje para la transmisión al destino o destinos. Todas las transmisiones cumplen la semántica de copia; es decir, los objetos en sí no son transmitidos, y dos mensajes con argumentos idénticos no tienen identidad individual y no son distinguibles. No obstante, se puede transmitir referencias a objetos. UML no

especifica el mecanismo exacto de transmisión de mensajes, y no es accesible para un programa de usuario (excepto en el grado en que como perfil haga la información de implementación accesible). En particular, la información adicional dependiente de la implementación puede incluirse en la transmisión para utilidad del entorno de ejecución. Además, no se especifica la codificación de la información, de forma que se permiten diversas implementaciones. Por ejemplo, no hay distinción en UML entre una llamada a un procedimiento local y una llamada a un procedimiento remoto; si una implementación requiere serialización y deserialización de argumentos es cosa suya, y no parte del modelo UML (o, con optimismo, del lenguaje de programación a medida que la tecnología evoluciona). Se requiere explícitamente un trozo de información sin especificar una implementación concreta: En una llamada síncrona, se debe codificar suficiente información de forma que una acción de retorno o respuesta posterior pueda transmitir la información de respuesta al hilo de ejecución correcto. Puesto que dicha información es opaca, la actividad receptora no puede manipularla excepto respondiendo. La codificación de esta información es fuertemente dependiente del entorno de ejecución, aunque en un sentido lógico la codificación es irrelevante.

En la especificación de UML no se especifica la velocidad y medio de transmisión de un mensaje. Si hay varios mensajes concurrentes no se puede hacer ninguna suposición sobre su orden de recepción. Si se requiere sincronización debería modelarse de forma explícita. Los perfiles pueden explicitar dicha información, como por ejemplo en un perfil de tiempo real.

Las llamadas ordinarias se comportan como llamadas procedimentales, en el sentido de que realmente no requiere ninguna acción en la parte del objeto de destino; se crea una ejecución cuando se recibe la llamada. La definición de “se recibe” y los recursos con los que se lleva a cabo la ejecución son detalles de implementación, pero no parte del modelo lógico. Una acción de envío o una llamada que desencadena una transición de máquina de estados, sin embargo, debe ser manejada por el hilo de ejecución del objeto destino en sí. Se puede modelar el hilo de ejecución como una actividad entrante con acciones de aceptación o como una máquina de estados con transiciones vinculada. En cualquier caso, los mensajes se deben recopilar de alguna manera, y se deben procesar de uno en uno. La especificación UML obliga a disponer de algún tipo de almacén de eventos que recopile eventos, incluyendo los mensajes recibidos, pero no impone ninguna restricción en el tratamiento de dichos eventos, excepto en que deben ser procesados de uno en uno. Un perfil puede añadir restricciones típicas de un sistema reactivo, como procesamiento FIFO (primero en entrar, primero en salir), prioridades o limitar el tiempo de respuesta.

El modelo básico de comunicación en UML tiene la intención de ser muy general para acomodarse a un amplio rango de decisiones de implementación, pero es sin embargo útil en la identificación de las cualidades esenciales de la comunicación y en obligar a los modelos a exponer de forma explícita las suposiciones subyacentes sobre el entorno de ejecución.

Notación

La notación se discute en otros temas, tales como acción, diagrama de comunicación y diagrama de secuencia.

concreto/a

Elemento generalizable (como una clase) que se puede instanciar directamente. Por necesidad, su implementación debe estar completamente especificada. Para una clase, todas sus

operaciones deben estar implementadas (por la clase o por un antepasado). Antónimo: abstracto/a.

Véase también clase directa, instanciación.

Semántica

Sólo se pueden instanciar los clasificadores concretos. Por tanto, todas las hojas de una jerarquía de generalización deben ser concretas. En otras palabras, todas las operaciones abstractas y otras propiedades abstractas finalmente deben ser implementadas en algún descendiente. (Por supuesto que una clase abstracta podría no tener descendientes concretos si el programa está incompleto, tal como un marco de trabajo pensado para la extensión por los usuarios, pero dicha clase no se puede utilizar en una implementación hasta que se proporcionen los descendientes concretos.)

Notación

El nombre de un elemento concreto aparece en un estilo normal. El nombre de un elemento abstracto aparece en cursiva.

concurrencia

Realización de dos o más actividades en el mismo intervalo de tiempo. No hay ninguna implicación de que las actividades estén sincronizadas. En general, operan de forma independiente excepto en puntos de sincronización explícitos. La concurrencia se puede alcanzar intercalando o ejecutando de forma simultánea dos o más hilos.

Véase transición compleja, estado compuesto, hilo.

Discusión

UML considera la concurrencia como una característica normal inherente a la computación, al contrario que su estatus secundario en muchos lenguajes de programación. Los objetos, mensajes, eventos y ejecuciones son concurrentes por naturaleza. Las acciones y las actividades incluyen la posibilidad de concurrencia a bajo nivel entre diferentes hilos de ejecución. No hay ninguna suposición sobre un reloj universal —la sincronización se lleva a cabo mediante intercambio de mensajes. En su tratamiento de la concurrencia, UML adopta una visión Einsteniana que es consistente con la naturaleza distribuida de las computadoras, sistemas y redes modernos.

concurrencia dinámica

Este concepto de UML1 se ha reemplazado por región de expansión

condición de guarda

Condición que se debe satisfacer con el fin de permitir el disparo de una transición asociada.

Véase también disparador cualquiera, bifurcación, conjunción, transición.

Semántica

Una condición de guarda es una expresión lógica que es parte de la especificación de una transición. Cuando se recibe una señal, se guarda hasta que la máquina de estados haya finalizado cualquier paso de ejecutar hasta finalizar. Cuando se finaliza cualquier paso de ejecutar hasta finalizar, los desencadenadores de las transiciones que abandonan el estado actual (incluyendo los estados que lo contienen) se examinan para elegir aquellos aptos para ser disparados. Las condiciones de guarda de las transiciones aptas se evalúan, no necesariamente en un orden fijo. Si se satisface la condición de al menos una transición apta, la transición está habilitada para su disparo (pero si hay más de una transición habilitada, sólo una se disparará y el evento es consumido, posiblemente eliminando la satisfacción de los otros desencadenadores). La prueba sucede como parte del proceso de evaluación del desencadenador. Si la condición de guarda se evalúa a falso cuando se gestiona el evento, no se reevalúa hasta que el evento desencadenador sucede de nuevo, incluso si más tarde la condición se torna verdadera.

Una condición de guarda debe ser una consulta —es decir, no debe modificar el valor del sistema o su estado; no puede tener efectos laterales.

Una condición de guarda puede aparecer en una transición de finalización. Es ese caso, selecciona una rama de una bifurcación.

Notación

Una condición de guarda es parte de la cadena de una transición. Tiene la forma de una expresión lógica encerrada entre corchetes.

```
[expresión lógica]
```

Los nombres utilizados dentro de la expresión deben estar disponibles para la transición. Son o bien parámetros del evento desencadenador, atributos de los objetos propietarios o nombres de propiedades alcanzables mediante navegación comenzando desde dichos nombres.

condicional

Una de las construcciones básicas de la computación, una elección dinámica entre varias alternativas en base a valores en tiempo de ejecución. Las condicionales aparecen en cada una de las vistas dinámicas en UML:

- En una interacción, un fragmento combinado dentro de una interacción que implica una elección en tiempo de ejecución entre varias alternativas. *Véase* fragmento condicional.
- En una actividad, un nodo que implica una elección en tiempo de ejecución entre varios arcos de salida. *Véase* nodo condicional.
- En una máquina de estados, una transición con una condición de guarda y ningún disparador. *Véase* transición condicional.

En cada una de las vistas, el concepto básico de una condicional es el mismo, pero hay diferencias en la especificación detallada.

conector

Conexión de dos partes estructuradas dentro de un clasificador estructurado o una colaboración; especificación de una asociación contextual que se aplica sólo en un determinado contexto, como los objetos dentro de un clasificador o los objetos que satisfacen una colaboración.

Véase también asociación, colaboración, componente, conector de delegación, clasificador estructurado.

Semántica

Un conector es una asociación contextual que es significativa y definida sólo en el contexto descrito por un clasificador estructurado o una colaboración. Es una relación que es parte del contexto pero no una relación inherente en otras situaciones. Los conectores son la parte estructural clave de las colaboraciones. Permiten la descripción de relaciones contextuales. A menudo representan caminos de comunicación entre las partes de un clasificador estructurado.

Dentro de un clasificador, una parte estructurada denota una aparición individual de otro clasificador, distinto de otras apariciones de ese clasificador y de la misma declaración del clasificador. De forma similar, un conector representa una asociación que se utiliza en un contexto particular, a veces un uso restrictivo de una asociación normal y en otras ocasiones otro mecanismo, como un parámetro o una variable local de un procedimiento. Un conector conecta dos partes. Cuando se instancia una clase estructurada, se crean sus partes y conectores como parte de la instanciación.

Para establecer una instancia de una colaboración, los objetos deben estar ligados a roles (partes en la colaboración) y los enlaces deben estar ligados a los conectores. Los enlaces a menudo se crean como parte de la instanciación de la colaboración. En muchos casos, los enlaces entre los objeto definen una colaboración. Un objeto puede jugar (estar ligado a) más de un rol.

Un conector conecta dos o más partes dentro de un clasificador estructurado o roles dentro de una colaboración (Figura 14.78). Puede incluir una referencia a una asociación que especifica los enlaces que implementa el conector. Si el conector va a ser implementado mediante otros mecanismos, como parámetros, variables locales, valores globales, o relaciones implícitas entre las partes de un solo objeto, la asociación se omite.

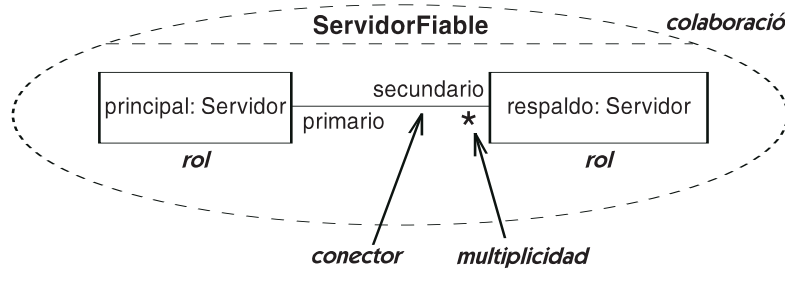


Figura 14.78 Conector en una colaboración

Uno o ambos extremos de un conector pueden estar conectados a puertos en partes internas (Figura 14.79). Un conector entre dos puertos internos es un conector de ensamblado. Los

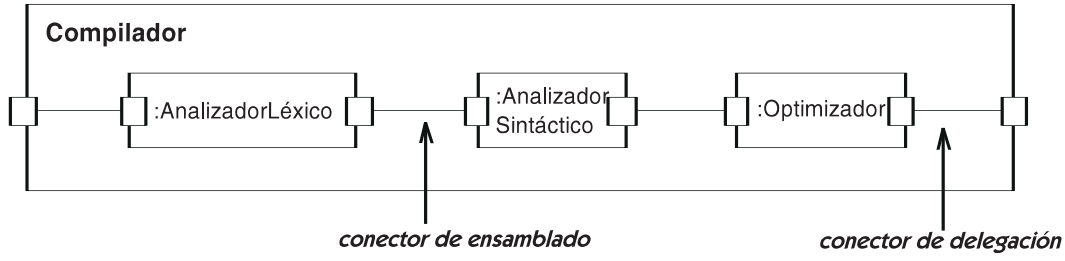


Figura 14.79 Conectores en una clase estructurada

puertos deben ser de tipos complementarios. Un mensaje enviado a un puerto es recibido en el otro. Un extremo de un conector puede estar conectado a un puerto externo del clasificador estructurado. Un conector entre un puerto externo y un puerto interno es un conector de delegación. Los puertos deben ser del mismo tipo. Un mensaje recibido en el puerto externo es recibido por el puerto interno; un mensaje enviado en el puerto interno es enviado sobre el puerto externo.

Un conector tiene una multiplicidad en cada extremo que indica cuántos objetos pueden estar conectados a un solo objeto (incluyendo cualquier restricción de ordenación o de unicidad). Si un conector está conectado a un puerto y no se especifica ninguna multiplicidad, la multiplicidad del conector es la misma que la del puerto.

En algunos casos, un conector puede ser considerado como usos de una asociación general entre las clases participantes. En ese caso, la colaboración muestra una forma de utilizar la asociación general para un propósito particular dentro de la colaboración.

En otros casos, las conexiones entre las partes o roles no tiene validez fuera del contexto del clasificador estructurado o colaboración. Si un conector no tiene una asociación explícita, entonces define una asociación implícita (“transitoria”) válida sólo dentro de la colaboración

Notación

Un conector se muestra de la misma manera que una asociación —a saber, como una línea continua entre dos partes o símbolos de rol (Figura 14.78 y Figura 14.79). El hecho de que es un conector es claro puesto que implica partes o roles. Puede tener una etiqueta con la sintaxis:

`nombre-del-conector : nombre-de-la-Asociación`

El nombre de la asociación (incluyendo los dos puntos) se puede omitir si no hay ninguna asociación subyacente, es decir, si el conector representa una relación contextual transitoria.

Se puede colocar un nombre de extremo y una multiplicidad en el extremo de un conector.

Historia

En UML1, las relaciones contextuales se tenían que modelar o bien como asociaciones o bien como enlaces, ninguna de las cuales era correcta en la mayoría de los casos. Había una idea de que una agregación podía pertenecer tanto a clases como a asociaciones, pero dicha idea no fue

resuelta de forma correcta. El concepto de UML2 para un concepto como la estructura interna de un clasificador modela las relaciones contextuales de manera explícita.

conector de delegación

Conector entre un puerto externo de un clasificador estructurado o componente y una parte interna. Las conexiones al puerto externo se tratan como si fueran al elemento que se encuentra en el otro extremo del conector de delegación.

Semántica

Un conector de delegación conecta un puerto externo de un componente con un puerto en uno de sus subcomponentes internos. Un mensaje de un origen externo recibido por el puerto externo se pasa al puerto en el componente interno; un mensaje desde una fuente interna recibido por un puerto interno se pasa al puerto externo y de ahí al componente conectado a él. Los conectores de delegación permiten la implementación de operaciones de alto nivel mediante componentes de bajo nivel.

Los conectores de delegación deben emparejar elementos de la misma polaridad, es decir, elementos requeridos con elementos requeridos o elementos proporcionados con elementos proporcionados. La interfaz de los elementos que reciben un mensaje delegado debe incluir los tipos de mensaje que puede producir el elemento productor del mensaje delegado. Un puerto externo puede delegar a un conjunto de elementos que juntos cubran los tipos de elementos del puerto externo. En tiempo de ejecución, el mensaje delegado se suministra a todos los elementos de delegación que concuerdan con el tipo de mensaje de tiempo de ejecución.

Notación

Un conector de delegación se representa con una línea entre el puerto externo y el puerto o componente interno. Se puede dibujar una flecha en la dirección de la transmisión del mensaje, es decir, desde un puerto externo proporcionado a un puerto externo requerido. Si un puerto representa una interfaz compleja (interfaces tanto requeridas como proporcionadas), se utiliza una línea sencilla.

Ejemplo

La Figura 14.80 muestra la descomposición interna de una televisión. Tiene tres puertos externos: **poner canal**, **poner volumen** y **mostrar configuración.** El puerto externo **poner canal** delega a un puerto en el componente interno de tipo **Sintonizador**. El puerto externo **poner volumen** delega a

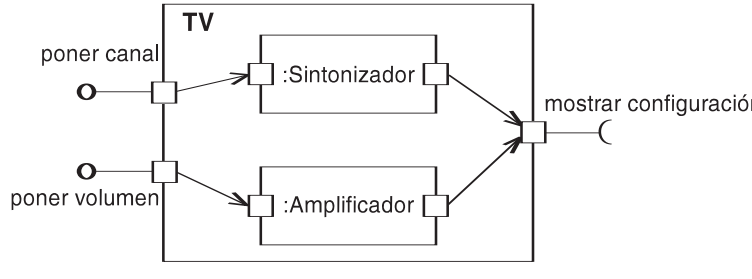


Figura 14.80 Conectores de delegación

un puerto en el componente interno de tipo **Amplificador**. Los puertos internos para **mostrar configuración** tanto en el **Sintonizador** como en el **Amplificador** delegan al puerto externo **mostrar configuración**.

conector de ensamblado

Un conector entre dos elementos (partes o puertos) en la especificación de la implementación interna de un clasificador estructurado o componente.

Véase conector, conector de delegación.

configuración del estado activo

El conjunto de estados que se encuentran activos en determinado momento dentro de una máquina de estados. El disparo de una transición cambia unos pocos estados del conjunto, mientras que el resto permanecen sin cambios.

Véase activo/a, transición compleja, máquina de estados.

Semántica

En general, un sistema puede tener múltiples estados activos concurrentemente. El conjunto completo de los estados activos se denomina *configuración del estado activo*. Las reglas de disparo de la transición de la máquina de estados definen qué estados deben estar activos para una transición que se dispara y qué cambios ocurren en la configuración del estado activo como resultado del disparo de la transición.

Una máquina de estados es un árbol de estados. Si un estado compuesto está activo, exactamente un subestado directo de cada una de sus regiones debe estar activo. Si una submáquina de estados está activa, un estado de alto nivel dentro de su máquina de estados referenciada debe estar activo. Si un estado simple está activo, este no tiene subestructura. Comenzando por las máquinas de estados completas, se pueden aplicar estas reglas recursivamente para determinar la configuración del estado activo legal.

conflicto

Generalmente, una situación en la que se produce alguna potencial ambigüedad entre los diferentes elementos de un modelo. Dependiendo del caso, puede haber reglas para resolver el conflicto o la presencia del conflicto puede crear un modelo mal formado.

Semántica

Un conflicto es una situación de especificaciones contradictorias, por ejemplo, cuando se hereda de más de una clase un atributo u operación con el mismo nombre, o cuando el mismo evento permite más de una transición, o cualquier otra situación similar en la que las reglas normales dan lugar a resultados potencialmente contradictorios. Dependiendo de la semántica para cada tipo de elemento del modelo, se puede resolver un conflicto mediante una regla de resolución de

conflicto, puede ser legal pero producir un resultado no determinista, o puede indicar que el modelo está mal formado.

En UML se pueden producir los siguientes tipos de conflictos:

- Se define el mismo nombre varias veces en el mismo espacio de nombres, a menudo debido a herencia o importación.

Por ejemplo, se puede utilizar el mismo nombre para elementos que se encuentran en diferentes niveles del mismo espacio de nombres o en diferentes espacios de nombres importados en un tercer espacio de nombres. Esto se puede resolver utilizando nombres calificados. Los nombres duplicados para el mismo tipo de elemento definidos en el mismo espacio de nombres están prohibidos puesto que no hay ninguna forma de distinguirlos.

Los nombres duplicados entre atributos en un clasificador están prohibidos y hacen al modelo mal formado. La duplicación puede producirse por medio de la herencia. La duplicación de nombres entre operaciones no causa conflicto si las signaturas son diferentes. Si las signaturas son iguales, un método en una clase descendiente anula el método de una clase antecesora, pero hay grandes restricciones en cualesquiera cambios en la signatura de operaciones de clases descendientes.

- Acciones que acceden al mismo recurso compartido (tal como una variable o dispositivo) pueden ejecutarse concurrentemente.

La concurrencia conlleva la posibilidad de accesos conflictivos a los mismos recursos. Si un hilo concurrente puede modificar un recurso y otro u otros hilos pueden leerlo o modificarlo, el estado de la lectura o el estado final del recurso pueden ser indeterminados. Esto no es necesariamente un problema; en algunas ocasiones la indeterminación es inofensiva o incluso deseable. Si se pueden mezclar diversas modificaciones en un recurso desde diferentes hilos, el estado final del recurso podría ser inconsistente o totalmente sin sentido. Los conflictos de acceso se pueden resolver de diversas formas: asegurando que las acciones concurrentes no comparten recursos; aislando un conjunto de acciones en un recurso para asegurar que el resultado final es consistente, si no determinista; conduciendo todas las acciones a un recurso determinado a través de un solo objeto guardián; o eliminando la concurrencia. Hay varios elementos en UML para realizar las distintas soluciones. *Véase* tipo de concurrencia, *flag* de aislamiento (indicador de aislamiento).

- Especificaciones inconsistentes aplicadas a dos objetos relacionados.

Dos partes del modelo pueden especificar cosas inconsistentes. A veces se aplican reglas de prioridad para resolver el conflicto. Por ejemplo, un subestado puede retrasar un evento mientras un estado compuesto lo consume. Este conflicto se resuelve de acuerdo con las prioridades para disparar transiciones. Por otro lado, si se especifican restricciones conflictivas para el mismo elemento (a menudo a través de la herencia), el modelo está mal formado.

- Están activas varias elecciones.

Las diversas construcciones condicionales permiten que varias ramas estén activas concurrentemente. También especifican que sólo una rama se ejecutará. Si el conflicto no se resuelve mediante reglas de prioridad, la elección es no determinista. Esto no es un error del modelo y de hecho puede modelar comportamiento útil. De forma similar, para una operación se pueden heredar varios métodos. Si todos los métodos están definidos en clases que forman un solo camino a través de la jerarquía de generalización, las reglas de anulación resuelven el conflicto. Sin

embargo, si dos métodos para la misma operación se heredan desde diferentes caminos antecesores, los métodos están en conflicto y el modelo está mal formado.

Discusión

Es posible evitar conflictos definiéndolos aparte junto con reglas de resolución de conflictos, tales como: Si está definida la misma característica por más de una superclase, use la definición encontrada en la superclase más temprana (esto requiere que las superclases estén ordenadas). UML generalmente no especifica reglas para resolver conflictos bajo el principio de que es peligroso contar con dichas reglas. Son fáciles de ser pasadas por alto y frecuentemente son el síntoma de problemas más graves con el modelo. Es mejor forzar al modelador a ser explícito en vez de depender de reglas sutiles y posiblemente confusas. En una herramienta o lenguaje de programación, dichas reglas tienen su lugar sólo para hacer el significado determinista. Pero podría ser de ayuda para las herramientas proporcionar avisos cuando se utilizan las reglas de forma que el modelador esté al tanto del conflicto.

conformidad con el protocolo

Declaración de que una máquina de estado específica conforma al protocolo definido por una máquina de estado de protocolo general.

Semántica

Una máquina de estado protocolar define las reglas en la invocación de operaciones o intercambio de mensajes que una máquina de estado conductual o el procedimiento puede realizar. Una relación de conformidad de protocolo requiere que una máquina de estado conductual o una máquina del estado de protocolo más especializada obedezca cada regla impuesta por la máquina de estados de protocolo general.

conjunción

Pseudoestado que es parte de una sola transición global en una máquina de estados. No divide un solo paso de ejecución hasta la finalización en la ejecución de una transición ni introduce la posibilidad de una elección dinámica basada en acciones ejecutadas durante la transición.

Véase también bifurcación, fusión.

Semántica

Una transición en una máquina de estados puede atravesar varias fronteras de estados compuestos desde el estado origen al estado destino. En la ejecución de tal transición, se pueden invocar una o más actividades entrar o salir. A veces es necesario intercalar una o más acciones en la transición con las actividades entrar y las actividades salir adjuntas a los estados anidados. Esto no es posible con una transición simple, que tiene adjunta una sola acción.

También es conveniente permitir que varios desencadenadores tengan un solo resultado, o permitir que un único desencadenador tenga varios resultados posibles con condiciones de guarda diferentes.

Un estado conjunción es un pseudoestado que hace posible construir una sola transición global desde una serie de fragmentos de transición. Un estado conjunción puede tener uno o más segmentos de transición entrantes y uno o más segmentos de transición salientes. No puede tener una actividad hacer interna, una submáquina, o cualesquiera transiciones salientes con desencadenadores de eventos. Es un estado tonto para estructurar transiciones y no un estado que pueda estar activo durante cualquier tiempo finito.

Un estado conjunción se utiliza para estructurar una transición desde varios segmentos. Sólo el primer segmento en una cadena de estados conjunción puede tener un desencadenador de eventos, pero todos ellos pueden tener condiciones de guarda. Los segmentos subsiguientes no deben tener desencadenadores. La condición de guarda efectiva es la conjunción de todas las condiciones de guarda individuales. La transición no se dispara a no ser que el conjunto entero de condiciones se satisfagan antes de que se dispare la transición. En otras palabras, la máquina de estados no puede permanecer en el estado conjunción.

Si varias transiciones entran en un solo estado conjunción, pueden tener cada una de ellas un desencadenador diferente o pueden no tener ningún desencadenador. Cada camino a través de un conjunto de estados conjunción representa una transición distinta.

Una transición saliente puede tener una condición de guarda. Si hay varias transiciones de salida, cada una debe tener una condición de guarda distinta. Esto es una bifurcación.

Una transición saliente puede tener un efecto adjunto. (El estado conjunción puede tener una acción interna, pero esto es equivalente a adjuntar una acción a la transición saliente, que es el mecanismo preferido.) La acción se ejecuta siempre que todas las condiciones de guarda se satisfagan, incluso aquellas que se encuentren en segmentos subsiguientes. Una transición no puede “dispararse parcialmente” de forma que pare en un estado conjunción. Debe alcanzar un estado normal.

Cuando se dispara una transición entrante, la transición saliente se dispara inmediatamente. Cualquier acción adjunta se ejecuta. La ejecución de las transiciones entrante y saliente es parte de un solo paso atómico (un paso de ejecutar hasta finalizar) —es decir, no son interrumpibles por un evento u otras acciones.

Si una transición compuesta atraviesa un estado, se ejecutan cualesquiera acciones en los segmentos fuera del estado antes de la actividad entrar, y cualesquiera acciones en los segmentos dentro del estado se ejecutan después de la actividad entrar. Una acción en un segmento que cruza el límite se considera que está fuera del estado. Existe una situación similar para las transiciones que rozan un estado.

Observe que un pseudoestado de elección también conecta varios segmentos en una transición compuesta, pero tiene diferentes reglas para las condiciones de guarda. Cuando un camino contiene un vértice de elección, las condiciones de guarda del camino antes del vértice de elección se evalúan antes de que se dispare la transición. Las condiciones de guarda en el camino después del vértice de elección se evalúan dinámicamente después de que se hayan realizado cualesquiera acciones en los segmentos iniciales. El modelador debe garantizar que existirá un camino subsiguiente válido. A no ser que un segmento de cada vértice de elección contenga una condición else o que se garantice que cubre todos los resultados posibles, existe el riesgo de un escenario de ejecución inválida. No hay tal peligro con un vértice de conjunción puesto que todas las condiciones se evalúan por anticipado antes de que la transición se dispare. Tanto los vértices conjunción como los vértices de elección tienen su propio lugar.

Notación

Un pseudoestado conjunción se representa en una máquina de estados como un pequeño círculo relleno. No tiene nombre. Puede tener flechas de transición de entrada y de salida.

Ejemplo

La Figura 14.81 muestra un ejemplo (un tanto artificial) que ilustra los efectos equivalentes de pseudoestados conjunción. Los dos segmentos entrantes y los dos segmentos salientes se multiplican para producir cuatro transiciones completas equivalentes. Observe que en cada caso se evalúan todas las condiciones de guarda antes de que se disparen cualesquiera transiciones o de que se ejecuten cualesquiera acciones. Si los valores de las condiciones de guarda cambian durante la ejecución de las acciones, no tiene efecto en el disparo de la transición.

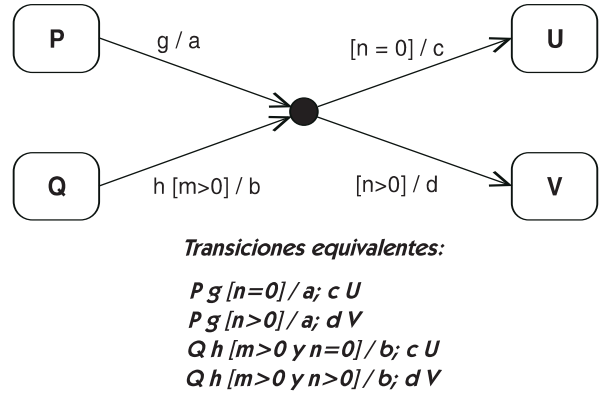


Figura 14.81 Pseudoestados conjunción con varios caminos

La Figura 14.82 muestra dos transiciones finalizadas desde el estado S al estado T —una transición de segmento único desencadenada por el evento f, y una transición multisegmento desencadenada por el evento e, que se estructura utilizando dos estados conjunción. Las anotaciones muestran el intercalado de las acciones de transición con las acciones entrar y salir.

Observe que la colocación de la etiqueta de acción a lo largo de la línea de transición no tiene importancia. Sin embargo, si la acción d se hubiera colocado dentro del estado X, se ejecutaría después de que se saliera del estado X y antes de que se entrara en el estado Y. Por tanto, debería dibujarse en la ubicación más extrema de la transición.

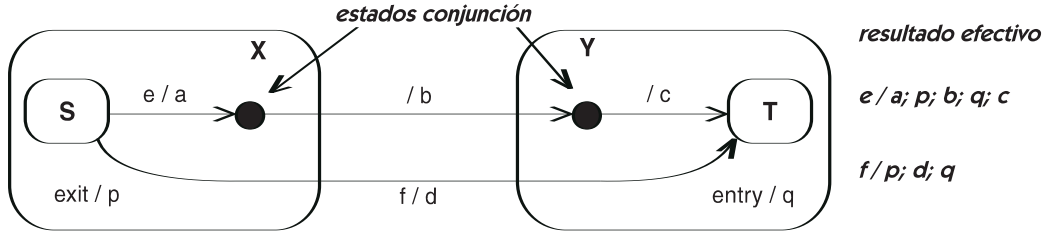


Figura 14.82 Pseudoestados conjunción con varias acciones

Para otros ejemplos, *véase* la Figura 14.274.

Véase también nodo de control para otros símbolos que se pueden incluir en diagramas de estados y en diagramas de actividad.

conjunto de generalizaciones

Conjunto de generalizaciones que compone una dimensión o aspecto de la especialización de un clasificador dado.

Véase también generalización, supratipo.

Semántica

Un clasificador a menudo se puede especializar a lo largo de varias dimensiones, cada una de ellas representando un aspecto diferente de la estructura y semántica del concepto. Puesto que dichas dimensiones son formas ortogonales de ver una jerarquía de clasificación, es útil dividir los hijos de un clasificador de acuerdo al aspecto que representan del padre. Un conjunto de generalizaciones que representan dicha dimensión se conoce como conjunto de generalizaciones. El elemento padre de todas las generalizaciones en el conjunto debe ser el mismo.

Normalmente cada dimensión caracterizada por un conjunto de generalizaciones representa sólo parte de la semántica del clasificador original. Para obtener un clasificador hijo completo, es necesario combinar (utilizando herencia múltiple) un clasificador de cada conjunto de generalizaciones ortogonal.

Por ejemplo, una forma geométrica podría ser especializada de acuerdo a si sus bordes son rectos o curvos y de acuerdo a si su límite tiene intersecciones o no. Una forma curva podría estar especializada en secciones cónicas o curvas. Una subclase completa requeriría una elección para cada dimensión. Por ejemplo, una estrella es una forma que tiene líneas rectas y con intersecciones.

Cada conjunto de generalizaciones representa una cualidad abstracta del padre, una cualidad que es especializada por los elementos en el conjunto. Pero un padre con varios conjuntos de generalizaciones tiene varias dimensiones, cada una de las cuales deben ser especializadas para producir un elemento concreto. Por tanto, los clasificadores dentro de un conjunto de generalizaciones son inherentemente abstractos. Cada uno de ellos es sólo una descripción parcial del padre, una descripción que enfatiza una cualidad e ignora el resto. Por ejemplo, una subclase de forma geométrica que se centra en la rectitud de las líneas ignora las intersecciones. Un elemento concreto requiere especializar todas las dimensiones al mismo tiempo. Esto puede producirse por herencia múltiple del elemento del modelo concreto desde un clasificador en cada uno de los conjuntos de generalizaciones, o mediante clasificación múltiple por un clasificador en cada uno de los conjuntos de generalizaciones. Hasta que se combinen todos los conjuntos de generalizaciones, la descripción permanece abstracta.

Por ejemplo, considere dos conjuntos de generalización sobre un vehículo; medio de propulsión y escenario (dónde viaja). El conjunto de generalizaciones propulsión incluye vehículos propulsados por el viento, propulsados por motor, propulsados por la gravedad y propulsados por músculos. El conjunto de generalizaciones escenario incluye vehículos de tierra, mar y aire. Un vehículo real debe tener una forma de propulsión y un escenario. Un vehículo marítimo pro-

pulsado por el viento es un barco velero. No hay ningún nombre particular para un vehículo aéreo propulsado por animales, pero en la fantasía y mitología existen instancias de dicha combinación.

Un supratipo es un clasificador asociado con un conjunto de generalizaciones. Es una metaclase cuyas instancias son los clasificadores en un conjunto de generalizaciones. Representa la cualidad que se está seleccionando en la generalización.

Restricciones del conjunto de generalizaciones

Se puede aplicar una restricción a los elementos de un conjunto de generalizaciones. Se pueden especificar las siguientes propiedades. *Véase* supratipo.

disjunto	Los clasificadores del conjunto son mutuamente excluyentes. Ninguna instancia puede ser instancia directa o indirecta de más de uno de los elementos.
solapado	Los clasificadores del conjunto no son mutuamente excluyentes. Un elemento puede ser instancia de más de uno de los elementos.
completo	Los clasificadores del conjunto cubren completamente una dimensión de especialización. Cada instancia del supertipo debe ser una instancia de al menos uno de los clasificadores del conjunto.
incompleto	Los clasificadores del conjunto no cubren completamente una dimensión de especialización. Una instancia del supertipo puede no ser una instancia de uno de los clasificadores del conjunto.

Notación

Un conjunto de generalizaciones se representa como una etiqueta de texto a continuación de dos puntos colocada en una flecha de generalización. La Figura 14.83 muestra una especialización de **Empleado** en dos dimensiones: **situación** del empleado y **localidad**. Cada conjunto de generalizaciones tiene un rango de posibilidades representadas por subclases. Pero se requieren ambas dimensiones para producir una subclase instanciable. Por ejemplo, **Enlace** es una clase que es tanto **Supervisor** como **Expatriado**.

Se pueden combinar varias flechas de generalización en un solo árbol con una cabeza de flecha. El nombre del conjunto de generalizaciones se puede colocar en la cabeza de flecha de forma que no necesite repetirse en cada subclase. Las restricciones en los conjuntos de generalizaciones se pueden colocar en una cabeza de flecha que se divide hacia las diversas subclases, o se pueden colocar en una línea discontinua que cruza un conjunto de flechas de generalización que componen el conjunto de generalizaciones. La Figura 14.84 muestra la declaración de restricciones en generalizaciones. Ilustra tanto la notación “estilo árbol”, en la que los caminos de generalización se dibujan en una rejilla ortogonal y comparten una cabeza de flecha común, así como el “estilo binario”, en el que cada relación padre-hijo tiene su propia flecha oblicua.

Historia

Los conjuntos de generalizaciones de UML2 sustituyen los discriminadores de UML1 con una capacidad más o menos equivalente.

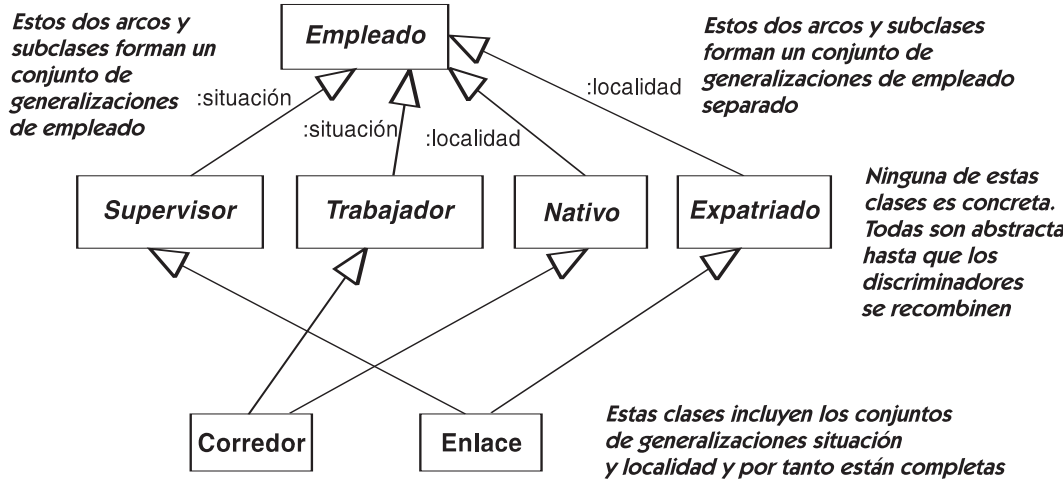


Figura 14.83 Conjuntos de generalizaciones

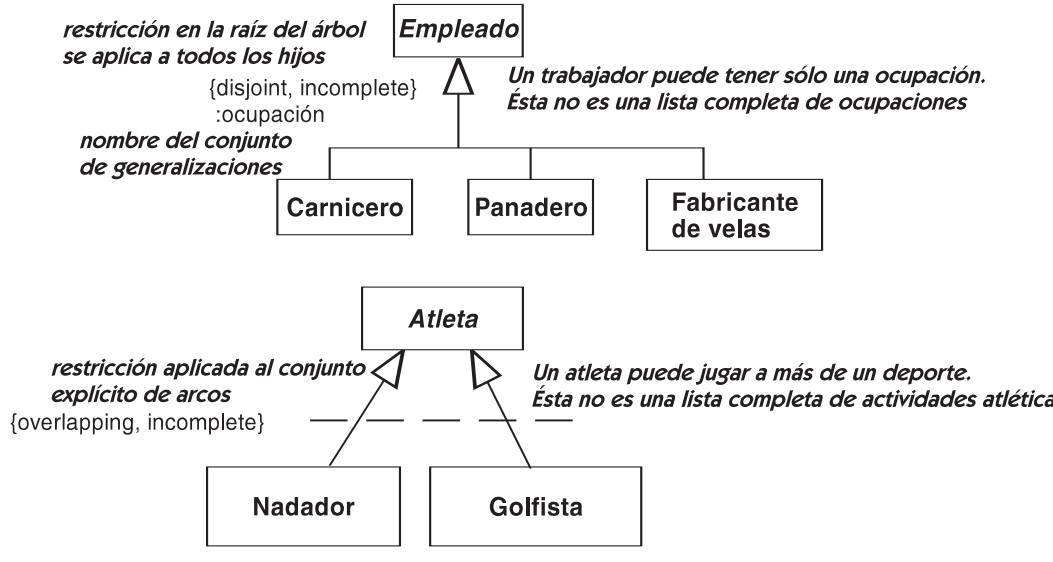


Figura 14.84 Restricciones sobre el conjunto de generalizaciones

conjunto de parámetros

Un conjunto completo de entradas o salidas para un comportamiento, en alternancia con otro conjunto completo de entradas o salidas para el comportamiento.

Semántica

Normalmente todos los parámetros de un comportamiento son requeridos por él (para las entradas) o se producen por él (para las salidas) en cualquier ejecución. En términos de flujo de datos

las condiciones fluyen dentro de una actividad, el control es una unión (un “and”) de entradas y una división de salidas.

A veces, sin embargo, un comportamiento tiene entradas alternativas o salidas. Un conjunto de parámetros es una lista completa de parámetros de entrada o de salida. Cada entrada es un parámetro. Un comportamiento puede tener entradas múltiples o juegos de parámetro de salida. En cualquier ejecución de comportamiento, exactamente un juego completo de entradas y un juego completo de salidas se producirán. En ejecuciones diferentes, pueden usarse juegos de parámetros diferentes.

Lo que pasa si un comportamiento recibe un juego completo de entradas así como las entradas parciales a otros juegos del parámetro no se especifica.

Notación

En un diagrama de actividad, un conjunto de parámetros se muestra por agrupación de uno o más símbolos de parámetros (cuadrados en el límite de un nodo de actividad) dentro de un rectángulo. Un nodo de actividad puede tener un conjunto de múltiples parámetros en su límite.

La Figura 14.85 muestra un ejemplo de juegos de parámetros en un nodo de actividad. En cualquiera ejecución del nodo, cualquiera (i1 and i2), or (i3) or (i4, i5 and i6) valores, habrán entrado. En cualquier ejecución, cualquiera (o1), or (o2, o3, o4 and o5) conseguirá valores de salida. La opción de juego de salida no depende de la opción de juego de entrada (o en cualquier caso, no podemos decirlo, de este modelo).

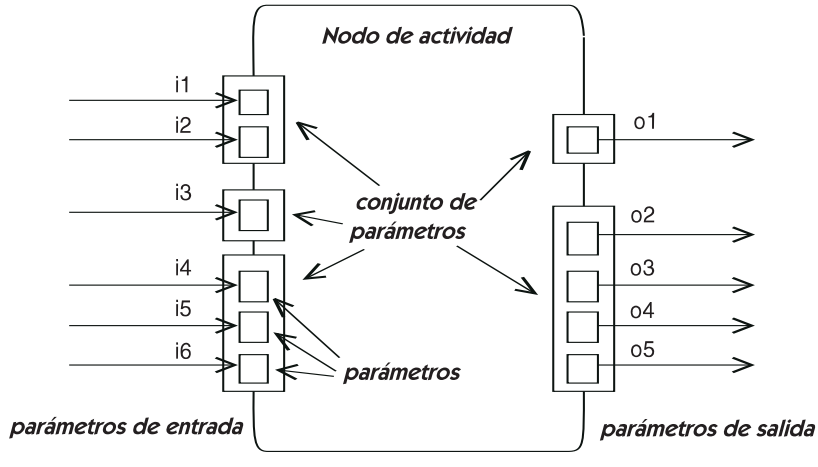


Figura 14.85 Conjunto de parámetros

Discusión

La anotación es un pedazo infortunado, porque usa el anidamiento doble para cambiar una situación “and” en una situación “or”. Podría haber sido mejor tener alguna clase de marca que conecta los conjuntos de entrada alternativos.

consider

Fragmento combinado en una interacción que filtra mensajes de forma que sólo se muestran aquellos tipos de mensajes especificados.

Semántica

Una construcción consider tiene una lista de tipos de mensajes y un subfragmento, que es un fragmento de interacción. Dentro del subfragmento, sólo se representan aquellos mensajes cuyos tipos aparecen en la lista. Esto indica que pueden producirse mensajes de otros tipos, pero no son representados en la interacción, lo que es por tanto una abstracción del sistema real. Esta construcción se combina a menudo con la aserción para indicar que un cierto mensaje debe seguir a otro mensaje.

Notación

Una construcción consider se representa en un diagrama de secuencia como una región con la palabra clave **consider** seguida por una lista de nombres de mensaje.

La Figura 14.86 muestra una construcción consider que contiene una aserción. Dicha construcción estipula que sólo se considerarán los mensajes inicio o parada. Por tanto, la aserción se aplica sólo a los mensajes inicio o parada. Esencialmente significa que un mensaje inicio debe estar seguido por un mensaje parada y no por otro mensaje inicio, pero que pueden producirse mensajes de cualquier otro tipo aunque son ignorados por la aserción.

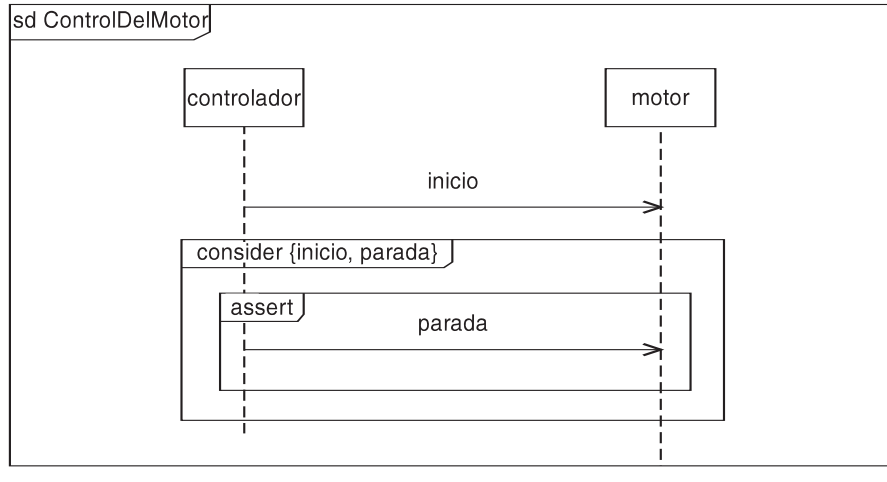


Figura 14.86 Fragmento combinado consider

construcción

Tercera fase de un proceso de desarrollo de software, durante la cual se realiza el diseño detallado y se implementa y prueba el sistema en software, firmware y hardware. Durante esta fase se

completan sustancialmente las vistas de análisis y de diseño, junto con la mayoría de la vista de implementación y parte de la vista de despliegue.

Véase proceso de desarrollo.

constructor

Operación que crea e inicializa una instancia de una clase. Se puede utilizar como un estereotipo de operación.

Véase creación, instanciación.

consulta

Una operación que devuelve un valor pero no altera el estado del sistema; una operación sin efectos colaterales.

Semántica

Una operación (pero no una recepción) puede declararse para ser una consulta. Una consulta no hace que se altere el estado del entorno; sólo devuelve un valor a la llamada. La operación del comportamiento implementada debe obedecer la restricción.

contenedor

Objeto que existe para contener otros objetos, y que proporciona operaciones para acceder o iterar sobre ellos, o una clase que describe dichos objetos. Por ejemplo, colecciones, listas y conjuntos.

Véase también agregación, composición.

Discusión

Normalmente no es necesario modelar contenedores de forma explícita. La mayoría de las veces son la implementación del extremo “muchos” de una asociación. En la mayoría de los modelos, es suficiente una multiplicidad mayor que uno para indicar la semántica correcta. Cuando un modelo de diseño se utiliza para generar código se puede especificar al generador de código la clase contenedora utilizada para implementar la asociación, utilizando valores etiquetados.

contexto

Vista de un conjunto de elementos de modelado que están relacionados para un propósito determinado, como ejecutar una operación o formar un patrón. Un contexto es una pieza de un modelo que restringe o proporciona el entorno para sus elementos. Una colaboración proporciona un contexto para sus contenidos.

Como término específico en el metamodelo, la palabra se utiliza para:

- El espacio de nombres en que se evalúa una restricción.
- El clasificador que directamente posee un comportamiento o que indirectamente posee una acción dentro del comportamiento.

Véase clasificador estructurado.

continuación

Etiqueta en una interacción que permite descomponer las condicionales en dos piezas semánticamente combinadas.

Semántica

Una continuación es una etiqueta que puede aparecer como elemento final en un operando de un fragmento condicional en un diagrama de secuencia. La misma etiqueta puede aparecer como primer elemento de un operando de un fragmento condicional diferente. Intuitivamente, si el flujo de control alcanza la etiqueta al final de la región en la primera condicional, puede continuar en la etiqueta correspondiente al principio de la región en la segunda condicional.

Las etiquetas en las dos condicionales conectan de forma efectiva las dos condicionales en una sola condicional virtual. La construcción es útil si una de las condicionales se encuentra embebida en un subdiagrama referenciado y por tanto no se puede combinar con la segunda condicional.

Notación

Una continuación se representa con el mismo símbolo que un estado, es decir, un rectángulo con esquinas redondeadas que contiene el nombre de la continuación. Un par de continuaciones correspondientes debe cubrir el mismo conjunto de líneas de vida.

Ejemplo

La Figura 14.87 muestra continuaciones al principio y al final de condicionales. La Figura 14.88 expande el ejemplo para mostrar un modelo equivalente que tiene el mismo significado.

copy

El estereotipo copy de UML1 ha sido retirado.

corregión

Conveniencia rotacional en la que un área de una línea de vida puede ser dibujada entre corchetes para indicar que el orden de los eventos en esa área no tiene ninguna restricción.

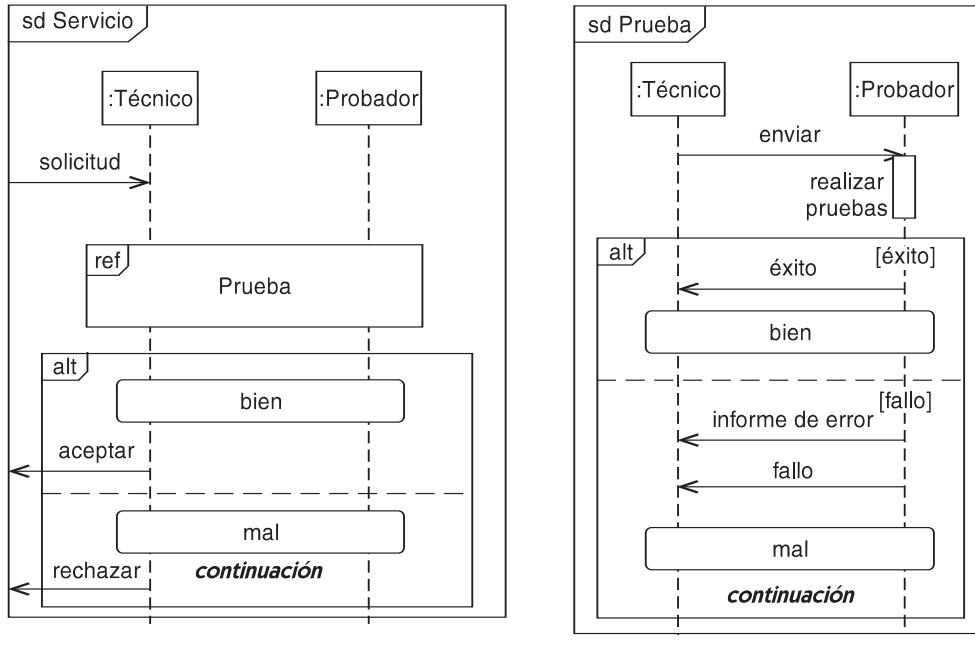


Figura 14.87 Continuaciones

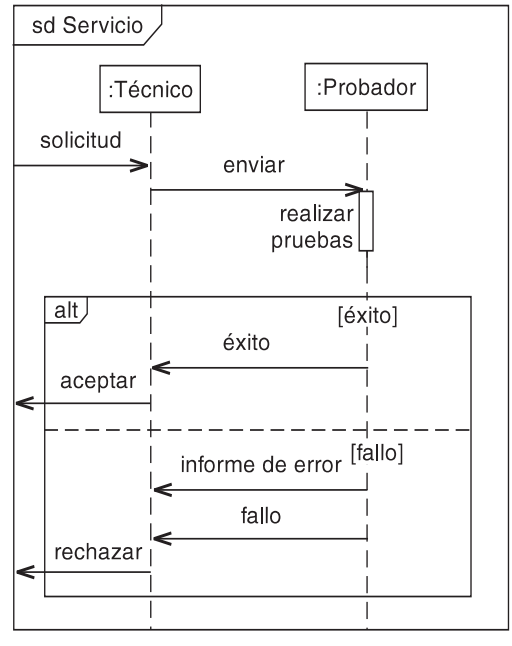


Figura 14.88 Interpretación de continuaciones

Semántica

Una corregión es equivalente a una construcción paralela en la que cada evento de una línea de vida pertenece a una región concurrente separada de la construcción paralela.

Notación

Una corregión se indica por un par de corchetes en una línea de vida, siendo la corregión la sección de la línea de vida entre los corchetes. Los eventos dentro de la región entre corchetes pueden producirse en cualquier orden, aunque estén ordenados a lo largo de la línea.

Ejemplo

La Figura 14.89 muestra un árbitro y dos jugadores. El árbitro notifica a ambos jugadores que se preparen. Cada jugador debe asentir con un mensaje de listo. No importa en qué orden reciba el árbitro los mensajes. Ambos mensajes deben ser recibidos antes de proceder. Por tanto, la recepción de los mensajes de listo se coloca dentro de una corregión. Después de que ambos mensajes se hayan recibido, el árbitro ordena a los jugadores que comiencen.

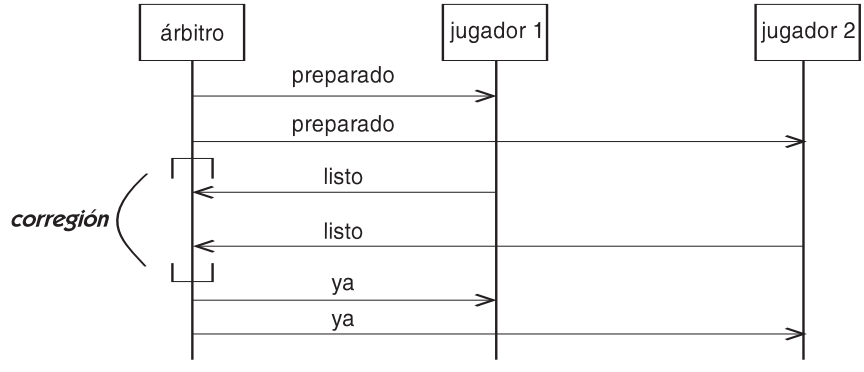


Figura 14.89 Corregión

creación

Instanciación e inicialización de un objeto u 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 una acción que instancia al objeto. La creación se puede modelar a diferentes niveles: una acción de creación en bruto o una operación de más alto nivel que invoca la acción y después inicializa el nuevo objeto. Una operación de creación puede tener parámetros que se utilizan para la inicialización de la nueva instancia. Al final de la operación de creación, el objeto nuevo cumple las restricciones de su clase y puede recibir mensajes.

Se puede declarar una operación de creación, o constructor, como una operación en el ámbito de la clase. El destino de dicha operación es (al menos conceptualmente) la propia clase. En un lenguaje de programación como Smalltalk, una clase se implementa como un

objeto real en tiempo de ejecución, y por tanto la creación se implementa como un mensaje normal a dicho objeto. En un lenguaje como C++, no existe el objeto real en tiempo de ejecución. La operación debe ser entendida como un mensaje conceptual que se ha optimizado en tiempo de ejecución. La aproximación de C++ descarta la oportunidad de calcular la clase a instanciar. Por lo demás, cada aproximación se puede modelar como un mensaje enviado a una clase. (Esto asume que se puede tratar una clase como a un objeto, lo cual es un punto de variación semántica.)

Las expresiones de valor inicial para los atributos de una clase son (conceptualmente) evaluadas en la creación, y los resultados se utilizan para inicializar los atributos. La acción de creación de un objeto no evalúa explícitamente las expresiones, pero ciertos perfiles de UML las invoca automáticamente en la creación. El código para una operación de creación puede reemplazar explícitamente esos valores, de forma que las expresiones de valor inicial podrían ser consideradas como valores por defecto anulables.

La creación se debe tratar a dos niveles: La creación de un objeto en bruto con el tipo correcto y ranuras de propiedades pero sin la inicialización de los valores, y la creación de un objeto completamente formado que satisfaga todas las restricciones. El primer nivel se puede modelar mediante una acción de creación. El segundo se puede modelar con una operación que incluya una acción de creación así como acciones para inicializar los valores de las propiedades del objeto. Aunque ambos niveles podrían estar incluidos en un modelo, la mayoría de los modelos probablemente usarán un nivel u otro.

Dentro de una máquina de estados, los parámetros de la operación de creación que creó un objeto están disponibles como un evento actual implícito en la transición que abandona el estado inicial de nivel superior.

Notación

En un diagrama de clases, la declaración de una operación de creación (constructor) se incluye como una de las operaciones en la lista de operaciones de la clase. Puede tener una lista de parámetros, pero el valor de retorno implícitamente es una instancia de la clase y se puede omitir. Como cualquier operación de ámbito de clase, su nombre debe ir subrayado (Figura 14.90). Si una operación de ámbito de instancia de una clase crea una instancia de otra clase, su nombre no se subraya.

Se puede mostrar un prototipo de la instancia creada como un símbolo de objeto (rectángulo con el nombre subrayado). Desde el símbolo de objeto a la operación de creación se dibuja una línea discontinua con una flecha abierta. La flecha tiene el estereotipo **«create».**

La ejecución de una operación dentro de un diagrama de secuencia se representa dibujando una flecha de mensaje, con la flecha abierta en el rectángulo superior de una línea de vida. Debajo del rectángulo está la línea de vida del objeto (línea discontinua o línea doble sólida, dependiendo de si está activa), que continúa hasta la destrucción del objeto o el final del diagrama. Sin embargo, si una llamada síncrona crea un objeto y después le transfiere el control, se usa una flecha rellena (Figura 14.91).

Véase también diagrama de comunicación y diagrama de secuencia para ver la notación para mostrar creación dentro de la implementación de un procedimiento.

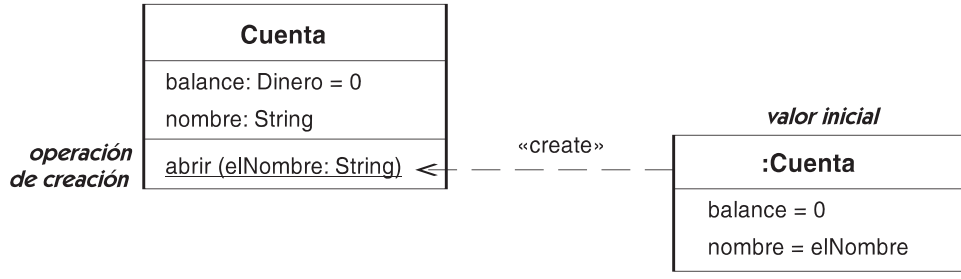


Figura 14.90 Operación de creación

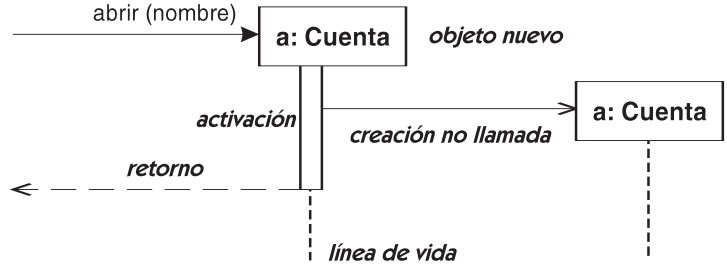


Figura 14.91 Diagrama de secuencia de creación

create (estereotipo de Característica de comportamiento)

Característica de comportamiento que crea una instancia del clasificador que contiene la característica.

Véase creación, uso.

create (estereotipo de Dependencia de uso)

Dependencia que denota que el clasificador cliente crea instancias del clasificador proveedor.

Véase creación, uso.

critical

Palabra clave para una construcción de región crítica.

decisión

Nodo de control en una actividad. *Véase* nodo de decisión.

definición de etiqueta

Una propiedad declarada en un estereotipo. Representa el nombre de una propiedad definida en tiempo de modelado.

Semántica

Una definición de etiqueta, es simplemente un atributo de una declaración de estereotipo.

Notación

La definición de una etiqueta, se muestra como un atributo en un rectángulo, junto con la declaración del estereotipo.

Ejemplo

La Figura 14.92, muestra la declaración de un estereotipo **Autor**. Se aplica a cualquier elemento. Declara las etiquetas **autor**, **estado**, **requisito**, **eliminar**. Los primeros tres son valores de cadena, y el último es un valor lógico.

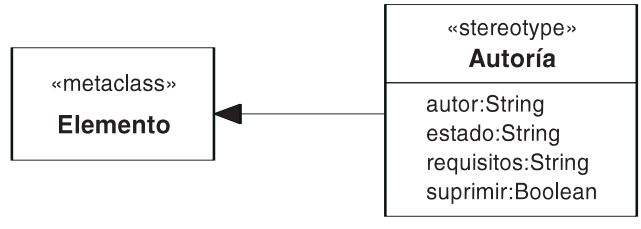


Figura 14.92 Definición de etiquetas en la declaración de un estereotipo

delegación

Capacidad de un objeto para emitir un mensaje a otro objeto en respuesta a un mensaje. La delegación se puede utilizar como una alternativa a la herencia. En algunos lenguajes (como self), se consigue mediante mecanismos de herencia en el propio lenguaje. En la mayoría de los demás lenguajes, como C++ y Smalltalk, se puede implementar mediante una asociación o agregación a otro objeto. Una operación en el primer objeto invoca una operación en el segundo objeto para llevar a cabo su trabajo. Contrastar: herencia.

Aunque la mayoría de modelos asume un punto de vista tradicional respecto a la herencia, el mecanismo de resolución de UML que determina el efecto de la invocación de una operación está escrito de una forma general de modo que si se desea se puede utilizar un comportamiento como la delegación. La inclusión de la delegación en UML sería por tanto un punto de variación semántico.

Véase también conector de delegación.

dependencia

Relación entre dos elementos en la que un cambio a un elemento (el proveedor) puede afectar o proporcionar información necesitada por el otro elemento (el cliente). Es un término de conveniencia que agrupa juntos distintos tipos de relaciones de modelado.

Véase relación (Tabla 14.3) para ver un cuadro completo de las relaciones en UML.

Semántica

Una dependencia es una sentencia de relación entre dos elementos en un modelo o en diferentes modelos. El término, un tanto arbitrario, agrupa juntos varios tipos de relaciones diferentes, como el término biológico *invertebrado* agrupa juntos todos los filos excepto los *vertebrados*.

En el caso de que la relación represente una asimetría de conocimiento, los elementos independientes se conocen como proveedores y los elementos dependientes se conocen como clientes.

Una dependencia puede tener un nombre para indicar su rol en el modelo. Sin embargo, normalmente la presencia de la misma dependencia es suficiente para dejar su significado claro, y el nombre es redundante. Una dependencia puede tener un estereotipo para establecer la naturaleza precisa de la dependencia, y puede tener una descripción de texto para describirse a sí misma en detalle, aunque sea de manera informal.

Una dependencia entre dos paquetes indica la presencia de al menos una dependencia del tipo dado entre un elemento en cada uno de los paquetes (excepto para acceder e importar directamente ese paquete relacionado). Por ejemplo, una dependencia de uso entre dos clases se puede mostrar como una dependencia de uso entre los paquetes que las contienen. Una dependencia entre paquetes no significa que todos los elementos en el paquete tengan la dependencia —de hecho, esa situación suele ser rara. *Véase* paquete.

Las dependencias no son necesariamente transitivas.

Observe que la asociación y la generalización encajan dentro de la definición general de dependencia, pero tienen su propio modelo de representación y notación y no se suelen considerar como dependencias. La ligadura de plantillas también tiene su propia representación. La importación de elementos también tiene su propia representación.

Las dependencias se encuentran en diversas variedades que representan distintos tipos de relaciones: abstracción, 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 diferentes formas. Normalmente uno de los elementos es más abstracto, y el otro es más concreto, aunque son posibles otras situaciones cuando ambos elementos son representaciones alternativas en el mismo nivel de abstracción. Desde relaciones menos específicas a más específicas, la abstracción incluye los estereotipos trace dependency, refinement (palabra clave **refine**), realization (que tiene su propia metaclase y notación especial), derivation (palabra clave **derive**) y substitution (caso especial de realization).

Permiso. Una dependencia de permiso (palabra clave **permit**) relaciona un elemento, como un paquete o clase, a otro elemento al cual se le da permiso para usar los contenidos privados. Por ejemplo, la construcción friend de C++ se podría modelar como una dependencia de permiso entre una operación y una clase.

Uso. Una dependencia de uso (palabra clave **use**) conecta un elemento cliente a un elemento proveedor, cuyo cambio requiere un cambio en el elemento cliente. El uso a menudo representa una dependencia de implementación, en la que un elemento utiliza los servicios de otro elemento para implementar un comportamiento. Los estereotipos de uso incluyen call, creation (palabra clave **create**), instantiation (palabra clave **instantiate**), y send. Esta es una lista abierta. Se pueden producir otros tipos de dependencia de uso en diversos lenguajes de programación.

Notación

Una dependencia se representa con una flecha discontinua entre dos elementos del modelo. El elemento del modelo situado en la cola de la flecha (el cliente) depende del elemento del modelo a la cabeza (el proveedor). La flecha se puede etiquetar con una palabra clave opcional, para indicar el tipo de dependencia y un nombre opcional (Figura 14.93).

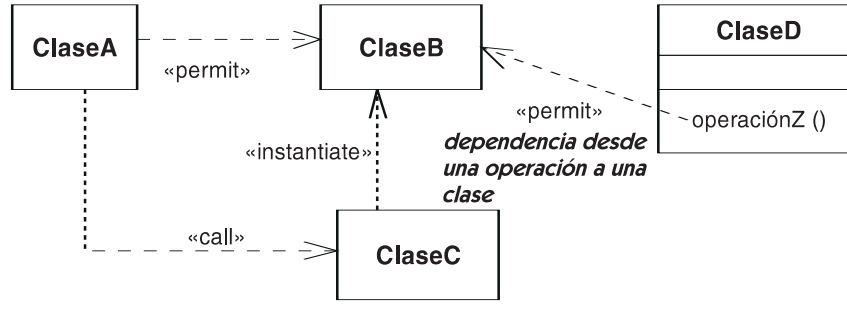


Figura 14.93 Algunas dependencias entre clases

Otros tipos de relaciones utilizan una flecha discontinua con una palabra clave, aunque no encajan en la definición de dependencia. Entre otras tenemos ligadura, extensión de caso de uso e inclusión, y adjuntar una nota o restricción al elemento del modelo que describe. Si uno de los elementos es una nota o restricción, la flecha se puede eliminar puesto que la nota o restricción siempre es el origen de la flecha.

dependencia de implementación

Relación de realización entre una interfaz y una clase en la que la clase se ajusta al contrato especificado por la interfaz.

Semántica

Una interfaz proporcionada es una interfaz que describe las operaciones que una clase hace disponibles externamente. Una dependencia de implementación modela la relación de realización entre la interfaz y la clase. Una clase puede tener varias dependencias de implementación hacia varias interfaces, y varias clases pueden implementar la misma interfaz.

Notación

Se dibuja una línea continua desde una interfaz proporcionada hacia el límite de la clase que la implementa (Figura 14.94).

Figura 14.94 Dependencia de implementación

derivación

Relación entre un elemento y otro elemento que se puede calcular de él. La derivación se modela como un estereotipo de una dependencia de abstracción con la palabra clave **derive**.

Véase elemento derivado.

derive (estereotipo de Dependencia de abstracción)

Dependencia, cuyo origen y destino son normalmente elementos del mismo tipo, pero no necesariamente. Una dependencia de derivación especifica que el origen puede ser calculado desde el destino. Aunque el origen es lógicamente redundante, se puede implementar por razones de diseño, como podría ser la eficiencia.

Véase derivación, elemento derivado

desarrollo incremental

El desarrollo de un modelo y otros artefactos de un sistema como una serie de versiones, cada una completa respecto a algún grado de precisión y funcionalidad, pero añadiendo cada una detalle incremental a la versión previa. La ventaja es que cada versión del modelo se puede evaluar y depurar en base a los relativamente pequeños cambios respecto a la versión anterior, haciendo más fácil hacer los cambios correctamente. El término está fuertemente relacionado con el concepto de desarrollo iterativo.

Véase proceso de desarrollo.

desarrollo iterativo

Desarrollo de un sistema mediante un proceso dividido en una serie de pasos o iteraciones, cada una de las cuales proporciona una aproximación mejor al sistema deseado que la iteración previa. El resultado de cada paso debe ser un sistema ejecutable que se pueda ejecutar, probar y

depurar. El desarrollo iterativo está estrechamente aliado con el concepto del desarrollo incremental. En el desarrollo incremental e iterativo, cada iteración añade funcionalidad incremental a la iteración previa. El orden de la adición de funcionalidad se elige de forma que balancee el tamaño de las iteraciones y ataque pronto las fuentes potenciales de riesgo, antes de que el precio de arreglar los problemas sea mayor.

Véase proceso de desarrollo.

descendiente

Hijo o elemento encontrado por medio de una cadena de relaciones hijas; el cierre transitivo de la relación de especialización. Antónimo: antepasado.

Véase generalización.

descriptor

Elemento del modelo que describe las propiedades comunes de un conjunto de instancias, incluyendo su estructura, relaciones, comportamiento, propósito y demás. Contrasta con: instancia.

Semántica

La palabra *descriptor* caracteriza elementos del modelo que describen conjuntos de elementos individuales, incluyendo instancias en el sentido más amplio. La mayoría de los elementos de un modelo son descriptores —clases, asociaciones, estados, casos de uso, colaboraciones, eventos y demás. A veces se utiliza la palabra *tipo* con este significado, pero dicha palabra se usa normalmente en un sentido más reducido para dar a entender sólo cosas parecidas a clases. La palabra descriptor se supone que incluye todos los tipos de elementos descriptivos. Un descriptor tiene un propósito y una extensión. El propósito está formado por la descripción de la estructura y otras reglas generales. Cada descriptor caracteriza a un conjunto de instancias, que son su extensión. No se puede hacer ninguna suposición de que la extensión sea físicamente accesible en tiempo de ejecución. La dicotomía más grande en un modelo es la distinción entre descriptor e instancia.

Un modelo de instancia muestra elementos que no son descriptores.

Notación

La relación entre un descriptor y sus instancias normalmente se refleja utilizando el mismo símbolo geométrico para ambos, pero subrayando el nombre de una instancia. Un descriptor tiene un nombre, mientras que una instancia tiene tanto un nombre individual como un nombre de descriptor, separados por dos puntos, y además el nombre está subrayado.

descriptor completo

Descripción implícita completa de una instancia directa. El descriptor completo se ensambla implícitamente mediante la herencia desde todos los antecesores.

Véase también clase directa, herencia, clasificación múltiple.

Semántica

Una declaración de una clase o de otro elemento de un modelo es, de hecho, sólo una descripción parcial de sus instancias; llamémoslo el *segmento de clase*. En general, un objeto contiene más estructura que la descrita por el segmento de clase de su clase directa. El resto de la estructura se obtiene por herencia desde sus clases antecesoras. La descripción completa de todos sus atributos, operaciones y asociaciones se conoce como el descriptor completo. El descriptor completo no se manifiesta normalmente en un modelo o programa. El propósito de las reglas de herencia es proporcionar una forma de construir automáticamente el descriptor completo desde los segmentos. En principio, hay varias formas de hacerlo, a menudo conocidas como *protocolos de metaobjetos*. UML define un conjunto de reglas de herencia que cubre los lenguajes de programación más populares y que también son útiles para el modelado conceptual. Sin embargo, debe ser consciente de que existen otras posibilidades —por ejemplo, el lenguaje CLOS.

Discusión

La especificación de UML no define realmente las reglas de herencia. El mapeo real de una operación hacia un método depende del mecanismo de resolución que no está completamente definido en la especificación. La mayoría de modeladores (y la mayoría de herramientas) asumirá reglas de herencia compatibles con Smalltalk o Java, pero la especificación permite definiciones de herencia más amplias, como aquellas que se pueden encontrar en CLOS o self.

despliegue

Asignación de artefactos de software a nodos físicos durante la ejecución.

Semántica

Los artefactos modelan entidades físicas, como archivos, scripts, tablas de bases de datos, documentos de texto y páginas web. Los nodos modelan recursos computacionales, como computadoras y discos. Un despliegue es la asignación de un artefacto o conjunto de artefactos a un nodo para su ejecución.

Una dependencia de manifestación relaciona un artefacto al elemento lógico que implementa.

Los dispositivos y el entorno de ejecución son tipos de nodos. La distinción entre tipos de nodos es bastante vaga y probablemente se podría ignorar.

Un camino de comunicación es una asociación entre nodos que los permite intercambiar mensajes y señales. Las redes se pueden modelar como nodos conectados por caminos de comunicación.

En cuanto a componentes, un despliegue puede poseer un conjunto de especificaciones de despliegue, cada una de las cuales contiene una especificación de cadena de la ubicación de un despliegue en el nodo y una ubicación de ejecución. Este concepto es un gancho que requiere la extensión en perfiles para ser útil.

Notación

Un despliegue se representa mediante el anidamiento gráfico de un símbolo de artefacto dentro de un símbolo de nodo (Figura 14.95). Alternativamente, se puede dibujar una línea discontinua desde el símbolo de artefacto al símbolo de nodo con la palabra clave **«deploy»**. Los despliegues se pueden representar en el nivel de tipo o de instancia mediante la utilización de símbolos de especificación de clasificador o instancia (cadenas con nombre no subrayado o subrayado, respectivamente).

Una especificación de despliegue se representa con un rectángulo con la palabra clave **«deploymentSpec»**. Se pueden listar los valores de parámetros específicos. Se dibuja una flecha discontinua desde el rectángulo hasta el artefacto cuyo despliegue describe.

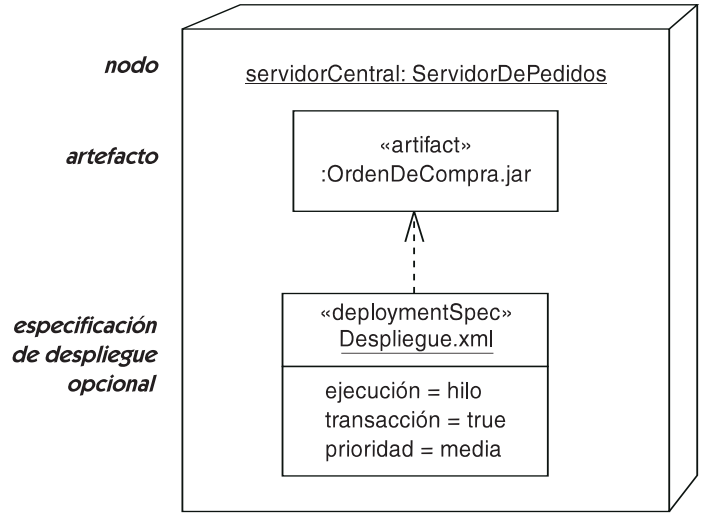


Figura 14.95 Despliegue

Historia

Se ha reposicionado el despliegue en UML2 para aplicarlo a artefactos más que a elementos del modelo. El concepto de manifestación relaciona elementos del modelo con artefactos.

despliegue (fase de)

Fase del desarrollo que describe la configuración del sistema en ejecución en un entorno del mundo real. En cuanto al despliegue, se deben tomar decisiones acerca de los parámetros de configuración, rendimiento, asignación de recursos, distribución y concurrencia. Los resultados de esta fase se capturan tanto en archivos de configuración como en la vista de despliegue. Contraste el análisis, diseño, implementación y despliegue (fase).

Véase proceso de desarrollo, etapas de modelado.

destroy (estereotipo de Característica de comportamiento)

Estereotipo que denota que la característica de comportamiento señalada destruye una instancia del clasificador que contiene la característica.

destrucción

Eliminación de un objeto y reclamación de sus recursos. Conceptualmente, la destrucción de un objeto compuesto lleva a la destrucción de todas sus partes compuestas. La destrucción de un objeto no destruye automáticamente los objetos relacionados por asociación ordinaria o incluso por agregación, pero cualquier enlace que implique a dichos objetos se destruye con el objeto.

Véase también composición, estado final, instanciación.

Semántica

Como la creación, la destrucción se puede modelar a dos niveles: una acción de bajo nivel que destruye la identidad y almacenamiento del objeto, y una operación de mayor nivel que destruye las partes compuestas que pertenecen al objeto y realiza otros tipos de limpieza implicados por la semántica de una clase particular. Ambos niveles pueden existir en un modelo, aunque la mayoría de los modelos probablemente sólo utilicen uno de ellos.

Notación

En un diagrama de secuencia, la destrucción de un objeto se representa mediante una gran X en la línea de vida del objeto (Figura 14.96). Se coloca en el mensaje que causa la destrucción del objeto o en el punto en el que el objeto se finaliza a sí mismo. Un mensaje que destruye un objeto puede representarse con el estereotipo «**destroy**».

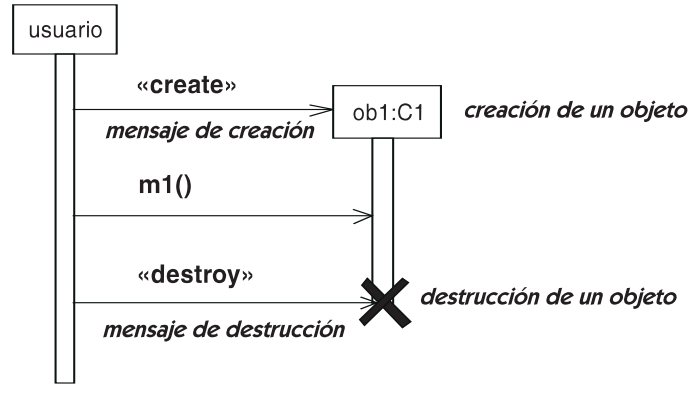


Figura 14.96 Creación y destrucción

Véase diagrama de secuencia (Figura 14.100) para ver la notación utilizada para representar la destrucción dentro de la implementación de un procedimiento.

destruir

Eliminar un objeto y reclamar sus recursos. Normalmente es una acción explícita, aunque puede ser consecuencia de otra acción, o de una restricción, o de la recolección de basura. *Véase* acción, destrucción.

determinista

Característica de un cálculo que produce el mismo valor o efecto cada vez que se ejecuta.

Semántica

A menudo, aunque no siempre, el determinismo es una característica deseable. Tratando con interacciones del mundo real, la indeterminación es a menudo una parte esencial de la semántica. Incluso dentro de cálculos internos, a veces la indeterminación puede ser útil, como por ejemplo en la producción de una aplicación que juegue a juegos con un comportamiento no predictivo. En otras ocasiones la indeterminación puede ser inofensiva puesto que diversas salidas pueden ser igual de buenas y que un programa no determinista puede ser más sencillo y rápido.

La indeterminación está relacionada a comportamiento concurrente en el que los hilos concurrentes comparten el acceso al mismo recurso, con al menos un hilo modificando el recurso. En UML, el comportamiento concurrente puede surgir de máquinas de estados que contienen estados ortogonales, actividades con nodos concurrentes o interacciones con líneas de vida concurrentes. La indeterminación también puede surgir de condiciones de guarda solapadas en transiciones y condicionales, es decir, desde un conjunto de condiciones de guarda en un solo estado o nodo de actividad para el que ciertos valores satisfacen más de una condición. De acuerdo con la semántica de UML, sólo una rama será elegida, pero la elección es indeterminada. La indeterminación se puede eliminar ordenando las condiciones, pero a veces no importa qué rama se elija, y el programa es más sencillo y a menudo más rápido si no se restringe la elección.

A veces hay una indeterminación aparente que no puede suceder en la práctica debido a los valores en tiempo de ejecución reales. En tales casos, el modelador debería anotar el hecho en comentarios o con aserciones para que la intención sea clara.

diagrama

Representación gráfica de una colección de elementos del modelo, habitualmente presentado como un gráfico conectado de arcos (relaciones) y vértices (otros elementos del modelo). UML dispone de cierto número de tipos de diagrama.

Véase también información de fondo, uso de la tipografía, hipervínculo, palabra clave, etiqueta, paquete, ruta, elemento de representación, cadena de propiedad.

Semántica

Un diagrama no es un elemento semántico. Un diagrama muestra representaciones de elementos del modelo semánticos, pero su significado no se ve afectado por la forma en que se representan.

Notación

La mayoría de diagramas UML y algunos símbolos complejos son gráficos que contienen formas conectadas por rutas. La información está mayormente en la topología, no en el tamaño o ubicación de los símbolos (hay algunas excepciones, como en un diagrama de tiempos). Hay tres tipos importantes de relaciones visuales: conexión (normalmente de líneas con formas bidimensionales), contención (de símbolos por formas bidimensionales cerradas) y acoplamiento visual (un símbolo que está "cerca" de otro en un diagrama). Esas relaciones geométricas se mapean en conexiones de nodos en un gráfico en la forma analizada de la notación.

La notación UML está hecha para dibujarse en superficies 2-D. Algunas formas son proyecciones 2-D de formas 3-D, como cubos, pero aun así se presentan como iconos en una superficie 2-D. En un futuro próximo, la disposición y navegación 3-D verdadera puede ser posible en máquinas de escritorio, pero actualmente no es común.

Hay cuatro tipos de construcciones gráficas utilizadas en la notación UML: iconos, símbolos 2-D, rutas y cadenas.

Un icono es una forma gráfica de tamaño y forma fijos. No se expande para albergar contenidos. Los iconos pueden aparecer dentro de símbolos de área, como terminadores en rutas o como símbolos autónomos que pueden o no estar conectados a rutas. Por ejemplo, los símbolos de agregación (un diamante), navegabilidad (una punta de flecha), estado final (ojo de buey) y destrucción de objeto (una gran *X*) son iconos.

Los símbolos bidimensionales tienen altura y anchura variables, y se pueden expandir para albergar otras cosas, como listas de cadenas u otros símbolos. Muchos de ellos están divididos en compartimentos de tipo similar o diferente. Las rutas se conectan a símbolos 2-D terminando la ruta en el límite del símbolo. Arrastrar o borrar un símbolo 2-D afecta a su contenido y a cualesquiera rutas conectadas a él. Por ejemplo, los símbolos de clase (un rectángulo), estado (un rectángulo redondeado) y nota (un rectángulo con la esquina superior izquierda doblada) son símbolos 2-D.

Una ruta es una secuencia de segmentos de línea rectos o curvos cuyos extremos están acoplados. Conceptualmente, una ruta es una única entidad topológica, aunque sus segmentos pueden ser manipulados gráficamente. Un segmento puede no existir fuera de su ruta. Las rutas siempre están acopladas a otros símbolos gráficos en ambos extremos (sin líneas sueltas). Las rutas pueden tener terminadores —es decir, iconos que aparecen en una secuencia al final de la ruta y que califican el significado del símbolo de ruta. Por ejemplo, los símbolos de asociación (líneas continuas), generalización (líneas continuas con un icono de triángulo) y dependencia (líneas discontinuas) son rutas.

Las cadenas presentan diversos tipos de información de una forma no analizada sintácticamente. UML asume que cada uso de una cadena en la notación tiene una sintaxis que se puede analizar en información subyacente del modelo. Por ejemplo, se proporcionan diferentes sintaxis para atributos, operaciones y transiciones. Dichas sintaxis están sujetas a su extensión por las herramientas como opción de presentación. Las cadenas pueden existir como contenido de un compartimento, como elementos en listas (en cuyo caso la posición en la lista proporciona información), como etiquetas acopladas a símbolos o caminos, o como elementos independientes en un diagrama. Por ejemplo, son cadenas nombres de clases, etiquetas de transiciones, indicación de multiplicidad y restricciones.

Un diagrama se puede presentar como un marco que contiene el contenido gráfico. El marco incluye el nombre del diagrama y establece su extensión. Un marco se dibuja como un rectán-

gulo con un pequeño pentágono (conocido como etiqueta de nombre) en la esquina superior izquierda. El pentágono contiene el tipo y nombre del diagrama en el formato:

`etiqueta nombre parámetros`$_{\text{opc}}$

La *etiqueta* es una palabra que indica el tipo de diagrama. El *nombre* es el nombre individual de un diagrama concreto, gracias al cual puede ser localizado en un explorador. Los nombres de diagramas son jerárquicos dentro de su estructura de paquete. Un nombre completamente calificado se representa como una secuencia de nombres separados por dos puntos dobles (::), por ejemplo:

Compañía-Acme::Compras::EntradaDeOrden

Ciertos tipos de diagramas tienen parámetros.

El marco se puede omitir cuando el contexto es claro y no se muestra ninguna información en la frontera. Es una capacidad de presentación opcional.

La Figura 14.97 muestra un diagrama de paquetes que utiliza el marco y notación de etiqueta.

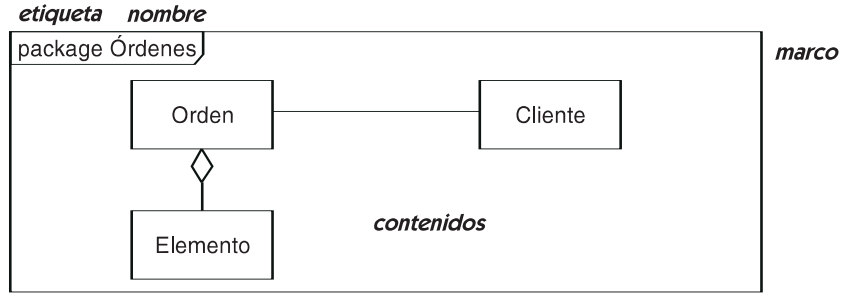


Figura 14.97 Diagrama con marco

La Tabla 14.1 muestra los tipos de diagramas y etiquetas UML.

Discusión

La especificación UML no describe coherentemente las etiquetas de cada tipo de diagrama, de forma que hemos tenido que extrapolar. El capítulo de interacciones utiliza **sd** para todos los tipos de diagramas de interacción, cosa poco útil si los diferentes tipos de diagramas no pueden ser distinguidos de otra forma y sin sentido en el caso de que puedan, de forma que hemos hecho algunos ajustes.

diagrama de actividad

Un diagrama que muestra la descomposición de una actividad en sus componentes.

Véase actividad.

Tabla 14.1 Etiquetas de diagrama y tipos de diagramas

Etiqueta	*Nombre*
activity	diagrama de actividad
class	diagrama de clase
comm	diagrama de comunicación
component	diagrama de componentes
class	diagrama de estructura compuesta
deplyment	diagrama de despliegue
intover	vista general del diagrama de interacción
object	diagrama de objetos
package	diagrama de paquetes
state machine	diagrama de máquina de estados
sd	diagrama de secuencia
timing	diagrama de tiempos
use case	diagrama de casos de uso

diagrama de casos de uso

Diagrama que muestra las relaciones entre actores y casos de uso dentro de un sistema.

Véase actor, caso de uso.

Notación

Un diagrama de casos de uso es un gráfico de actores, un conjunto de casos de uso englobados por un límite de sujeto (un rectángulo), asociaciones entre los actores y los casos de uso, relaciones entre los casos de uso, y generalización entre los actores. Los diagramas de casos de uso muestran elementos 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 puede contener símbolos para los paquetes anidados Un diagrama de clases contiene ciertos elementos de comportamien-

to, como operaciones, pero sus dinámicas se expresan en otros diagramas, como en los diagramas de estados y los diagramas de comunicación.

Véase también clasificador, diagrama de objetos.

Notación

Un diagrama de clases muestra una presentación gráfica de la vista estática. Normalmente se necesitan varios diagramas de clases para mostrar una vista estática completa. Los diagramas de clases individuales no indican necesariamente divisiones en el modelo subyacente, aunque las divisiones lógicas, como los paquetes, son fronteras naturales para formar los diagramas.

Discusión

La especificación de UML distingue entre diagramas de clases y diagramas de componentes, llamados colectivamente diagramas de estructura. La mayoría de los usuarios probablemente los llamarán a todos diagramas de clases sin pérdida de la semántica. En cualquier caso, no hay una línea rígida entre los distintos tipos de diagramas.

El diagrama de clases de UML se encuentra fuertemente influenciado por el diagrama entidad-relación (E-R) de P. P. Chen, a través de las notaciones predecesoras de UML, como Shlaer-Mellor, Booch, OMT, Coad-Yourdon, etcétera. Es el diagrama de modelado de objetos más importante.

diagrama de colaboración

Diagrama que muestra la definición de una colaboración. En UML2, un diagrama de colaboración es un tipo de diagrama de estructura compuesta. Observe que las definiciones y usos de la colaboración también pueden aparecer en diagramas de clases.

diagrama de componentes

Diagrama que representa la definición, estructura interna y dependencias de componentes. No hay una diferencia clara entre diagramas de componentes y diagramas generales de clases.

Véase componente para ver ejemplos de diagramas de componentes.

diagrama de comunicación

Diagrama que muestra interacciones organizadas alrededor de las partes de una estructura compuesta o los roles de una colaboración. A diferencia de un diagrama de secuencia, un diagrama de comunicación muestra explícitamente las relaciones entre los elementos. Por otro lado, un diagrama de comunicación no representa el tiempo como una dimensión separada, por lo que la secuencia de los mensajes y los hilos concurrentes se deben determinar utilizando números de secuencia. Tanto los diagramas de secuencia como los de colaboración expresan interacciones, pero las muestran de diferente forma.

Véase colaboración, patrón, diagrama de secuencia.

Notación

Un diagrama de comunicación es una clase estructurada o una colaboración junto con mensajes que dan forma a una interacción. Se representa como un gráfico cuyos nodos son rectángulos que representan partes de una clase estructurada o roles de una colaboración. Los nodos corresponden a las líneas de vida en una interacción, pero son representados dentro de su contexto estructural. Las líneas entre las partes representan conectores que forman caminos de comunicación. Se puede etiquetar las líneas con su nombre contextual y/o el nombre de la asociación subyacente, si existe. Se pueden representar multiplicidades en los conectores. Los mensajes entre las partes se representan por medio de flechas etiquetadas situadas cerca de los conectores. Típicamente, una interacción representa la implementación de una operación o caso de uso.

Un diagrama de comunicación representa las ranuras para objetos involucrados como líneas de vida en una interacción. Las líneas de vida tienen la sintaxis **`nombredelrol`** : **`nombredelaclase`**. Se pueden omitir tanto el nombre del rol como el nombre de la clase, pero los dos puntos son obligatorios si se omite el nombre del rol. El diagrama también muestra los conectores entre los objetos. Dichos conectores representan asociaciones y enlaces transitorios que representan argumentos de un procedimiento, variables locales y autoenlaces. Una flecha próxima a una línea indica un mensaje pasando por el enlace en la dirección indicada. Una flecha en un conector representa un camino de la comunicación en una sola dirección (notación opcional extrapolada de la especificación UML, pero que no aparece explícitamente en dicha especificación).

Implementación de una operación

Un diagrama de comunicación que muestra la implementación de una operación incluye símbolos para el rol del objeto destino y los roles de otros objetos usados por el objeto destino, de forma directa o indirecta, para realizar la operación. Los mensajes en los conectores representan el flujo de control en una interacción. Cada mensaje representa un paso dentro del método para la operación

Un diagrama de comunicación que describe una operación también incluye símbolos de rol que representan argumentos de la operación y variables locales creadas durante su ejecución. Se pueden etiquetar con notas los objetos creados o destruidos durante la ejecución. Los objetos sin palabra clave existen cuando empieza la operación y perduran cuando se completa.

Los mensajes internos que implementan un método están numerados, empezando con el número 1. Para un flujo de control procedimental, los números de mensajes posteriores usan secuencias separadas por puntos anidadas de acuerdo con el anidamiento de la llamada. Por ejemplo, el segundo paso de primer nivel es el mensaje 2; el primer paso subordinado dentro del mensaje 2 es el mensaje 2.1. Para mensajes asíncronos intercambiados entre objetos concurrentes, todos los números de secuencia están en el mismo nivel (es decir, no están anidados).

Véase mensaje para obtener una descripción completa de la sintaxis de los mensajes, incluyendo su numeración.

Un diagrama de comunicación completo muestra los roles de todos los objetos y enlaces que utiliza la operación. Si no se muestra un objeto, se asume que no se utiliza. Sin embargo, no es seguro asumir que la operación utiliza todos los objetos que aparecen en el diagrama de comunicación.

Ejemplo

En la Figura 14.98, se llama a una operación **refrescar** en un objeto **Controlador**. En el momento en que se llama a la operación, ya tiene un enlace al objeto **Ventana** donde se va a mostrar la imagen. También tiene un enlace a un objeto **Cable**, objeto cuya imagen se va a mostrar en la ventana.

La implementación de alto nivel de la operación **refrescar** tiene un único paso —la llamada a la operación **mostrarPosiciones** en el objeto **Cable**. Esta operación tiene número de secuencia 1 porque es el primer paso del método superior. Este flujo de mensaje pasa junto con una referencia al objeto **Ventana** que será necesario más tarde.

La operación **mostrarPosiciones** llama a la operación **dibujaSegmento** en el mismo objeto **Cable**. La llamada, etiquetada con el número de secuencia 1.1, se envía a través del enlace **self** implícito. El asterisco indica una llamada iterativa a la operación; los detalles se proporcionan en los corchetes.

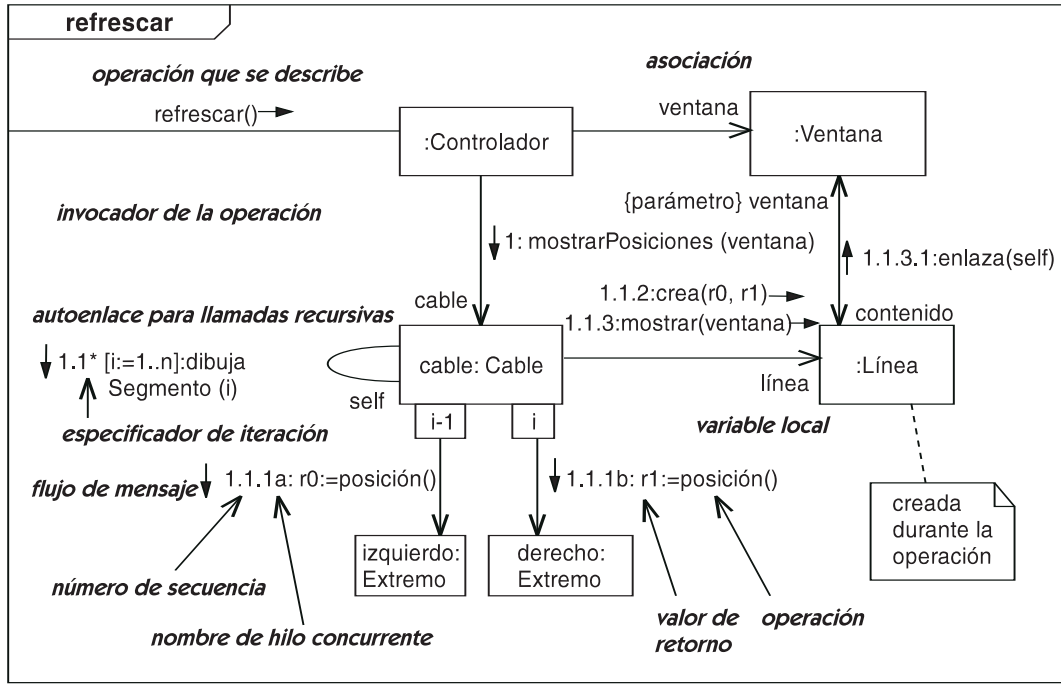


Figura 14.98 Sintaxis de un diagrama de comunicación con flujos de mensaje

Cada operación **dibujaSegmento** accede a dos objetos **Extremo**, uno indexado por el valor **i-1** y otro por el valor **i**. Aunque en el contexto de la operación sólo hay una asociación de **Cable** a **Extremo**, se necesitan dos enlaces a dos objetos **Extremo**. Los objetos están etiquetados **izquierdo** y **derecho** (que son los roles de clasificador en la colaboración). Se envía un mensaje por cada enlace. Los mensajes están etiquetados 1.1.1a y 1.1.1b. Esto indica que son pasos de la operación 1.1; las letras al final indican que los dos mensajes se pueden enviar de forma concurrente. En una implementación normal, probablemente no se ejecutarán en paralelo, pero como se declaran como concurrentes, se pueden ejecutar en cualquier orden secuencial que se considere oportuno.

Cuando son devueltos ambos valores (**r0** y **r1**), puede continuar el paso siguiente a la operación 1.1. El mensaje 1.1.2 es un mensaje de creación enviado a un objeto **Línea**. Realmente, el mensaje va a la propia clase **Línea** (al menos en principio), que crea un nuevo objeto **Línea** enlazado al emisor del mensaje.

El paso 1.1.3 utiliza el enlace recién creado para enviar un mensaje **mostrar** al objeto **Línea** recién creado. El puntero al objeto **ventana** se pasa como argumento, haciéndolo accesible al objeto **Línea** como un enlace. Observe que el objeto **Línea** tiene un enlace al mismo objeto **ventana** que está asociado con el objeto **Controlador** original; esto es importante para la operación y por tanto es representado por el diagrama. En el paso final 1.1.3.1, se le solicita al objeto **Ventana** que cree un enlace al objeto **Línea**.

Se puede observar el estado final del sistema borrando mentalmente todos los enlaces temporales. Hay un enlace desde **Controlador** a **cable**, y desde **cable** a sus partes **Extremo**, desde **Controlador** a **ventana** y desde **ventana** a su **contenido**. Sin embargo, una vez completada la operación, una **Línea** no tiene acceso a la **Ventana** que la contiene. El enlace en esa dirección es transitorio y desaparece cuando se completa la operación. De manera similar, un objeto **Cable** ya no tiene acceso a los objetos **Línea** que se utilizan para mostrarlo.

Discusión

La especificación de UML es un tanto confusa en cuanto a lo que está permitido en diagramas de comunicación. Los hemos interpretado de forma que permitan mensajes superpuestos en la estructura interna, pues de otra forma serían superfluos.

La especificación indica la etiqueta **sd** para todos los diagramas que muestren interacciones. Esto parece confuso e inútil; o bien puede decir qué tipo de diagrama es mirándolo, en cuyo caso no se necesita ninguna etiqueta, o no puede decirlo, en cuyo caso es necesaria una distinción entre diagramas. Por tanto hemos utilizado la etiqueta **comm** en estos diagramas.

Historia

En UML se conocía como diagrama de colaboración, que era confuso, puesto que las colaboraciones eran estructuras estáticas, pero el diagrama incluía mensajes.

diagrama de despliegue

Diagrama que muestra la configuración de los nodos de procesamiento en tiempo de ejecución y los artefactos ubicados en ellos. Un diagrama de despliegue puede ser a nivel de clase o de instancia. *Véase* despliegue.

diagrama de estados

Nombre usado por David Harel para su extensión a la notación básica de la máquina de estados que incluye estados anidados y estados concurrentes. Esta notación sirvió como base para la notación de la máquina de estados de UML. *Véase* diagrama de máquina de estados.

diagrama de estructura

Un diagrama que describe la estructura estática de un sistema, como opuesto a su comportamiento dinámico. Realmente no hay ninguna línea rígida entre los diferentes tipos de diagramas de estructura aunque pueden nombrarse según el tipo de elementos que contienen, como diagrama de clases, diagrama de interfaces o diagrama de paquetes.

diagrama de estructura compuesta

Diagrama que muestra la estructura interna (incluyendo partes y conectores) de un clasificador estructurado o una colaboración. No hay gran diferencia entre un diagrama de estructura compuesta y un diagrama general de clases.

diagrama de interacción

Término genérico que se aplica a diversos tipos de diagramas que enfatizan la interacción de objetos. Incluye a los diagramas de comunicación y a los diagramas de secuencia. Bastante relacionados son los diagramas de actividades.

Variantes más especializadas incluyen los diagramas de tiempos y la vista general del diagrama de interacción.

Véase también colaboración, interacción.

Notación

Un patrón de interacción entre objetos se representa en un diagrama de interacción. Los diagramas de interacción vienen en distintas formas, todas basadas en la misma información subyacente pero cada una enfatizando una vista de dicha información: diagramas de secuencia, diagramas de comunicación y vista general del diagrama de interacción.

Un diagrama de secuencia muestra una interacción centrada en la secuencia de tiempos. En particular, muestra los objetos que participan en la interacción por sus líneas de vida y los mensajes que intercambian, colocados en una secuencia temporal. Un diagrama de secuencia no muestra los enlaces entre los objetos. Los diagramas de secuencia vienen en varios formatos intencionados para diferentes propósitos.

Un diagrama de comunicación muestra una interacción centrada alrededor de los objetos que realizan operaciones. Es similar a un diagrama de objetos que muestra los objetos y los enlaces entre ellos necesarios para implementar una operación de más alto nivel.

La secuencia temporal de los mensajes se indica mediante números de secuencia en las flechas de flujo de mensajes. Se pueden mostrar tanto las secuencias concurrentes como las secuenciales, utilizando la sintaxis apropiada. Los diagramas de secuencia muestran las secuencias temporales utilizando la ordenación geométrica de las flechas en el diagrama. Por tanto, no requieren números de secuencia, aunque se pueden incluir por comodidad o para permitir cambiar a un diagrama de comunicación.

Los diagramas de secuencia y los de colaboración expresan información similar pero mostrada de forma diferente. Los diagramas de secuencia muestran la secuencia de mensajes explí-

cita 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 entender todos los efectos sobre un objeto y para diseño procedimental.

Un diagrama de tiempos permite definir un eje de tiempo numérico más que especificar solamente secuencias de mensajes relativas. También permiten mostrar la representación visual de cambios de estados para cada línea de vida siempre que haya sólo un puñado de estados. Los diagramas de tiempo pueden ser útiles para situaciones de tiempo real, pero no son necesarios en la mayoría del diseño conceptual.

Una vista general del diagrama de interacción es una mezcla extraña de diagramas de interacción y diagramas de actividad cuya utilidad no está demasiado clara.

Discusión

Tanto las interacciones como las actividades modelan comportamiento, pero desde diferentes puntos de vista. Un diagrama de secuencia muestra principalmente secuencias de eventos para cada rol, con mensajes conectando los roles. El centro de atención está en los roles y en el flujo de control entre los roles, no en el procesamiento. Un diagrama de actividad muestra los pasos procedimentales implicados en realizar una operación de alto nivel. Muestra el flujo de control entre pasos procedimentales más que el flujo de control entre objetos. Cierto número de construcciones en los diagramas de actividad y de interacción están estrechamente relacionadas y a veces (pero no siempre) utilizan los mismos iconos.

diagrama de máquina de estados

Un diagrama que demuestra una máquina de estados, incluyendo estados simples, transiciones, y estados compuestos anidados. El concepto original fue inventado por David Harel, que los llamó los diagramas de estados. *Véase* máquina de estados.

diagrama de objetos

Un diagrama que muestra objetos y sus relaciones en un momento del tiempo. Un diagrama de objetos puede ser considerado un caso especial de un diagrama de clases en que se pueden mostrar tanto las instancias como las especificaciones. También están relacionados los diagramas de comunicación que muestran los roles dentro de un contexto.

Véase también diagrama.

Notación

Las herramientas no necesitan soportar un formato separado para los diagramas de objetos. Los diagramas de clases pueden contener especificaciones de objetos, así que un diagrama de clases con objetos y sin clases, es un “diagrama de objetos”. La frase es útil, no obstante, para caracterizar un uso particular de varias maneras.

Discusión

Un diagrama de objetos modela los objetos y enlaces que representan el estado de un sistema en un momento particular. Contiene especificaciones de objetos que pueden mostrar desde objetos aislados hasta una ejecución particular u objetos prototípicos con un rango de valores. Para mostrar un modelo general de objetos y sus relaciones que pueden ser instanciados, use una colaboración.

Un diagrama de objetos no muestra la evolución del sistema con el tiempo. Para ese propósito, use un diagrama de comunicación o un diagrama de secuencia para representar una interacción.

diagrama de paquetes

Un diagrama de estructura cuyo contenido es principalmente paquetes y sus relaciones.

No hay ninguna línea rígida entre los tipos diferentes de diagramas de estructura, así que el nombre es meramente una convención sin importancia semántica.

diagrama de secuencia

Un diagrama que muestra las interacciones del objeto colocadas en secuencia temporal. En particular muestra los objetos que participan en una interacción y las sucesiones de mensajes intercambiados.

Véase también activación, colaboración, línea de vida, mensaje.

Semántica

Un diagrama de secuencia representa una interacción, un conjunto de comunicaciones entre los objetos colocados visualmente en orden de tiempo. Al contrario de un diagrama de comunicación, el diagrama de secuencia muestra las sucesiones de tiempo explícitamente pero no incluye las relaciones de los objetos.

Puede existir en una forma de descriptor (describiendo todos los posibles escenarios) y en forma de instancia (describiendo un escenario actual). Los diagramas de secuencia y colaboración expresan una información similar (aunque no idéntica), pero lo muestran de maneras diferentes.

Notación

Un diagrama de secuencia se muestra en un marco rectangular con la cadena **`sd nombre`** en un pentágono pequeño en la esquina superior izquierda. El nombre es el nombre de la interacción mostrada por el diagrama. Si la interacción tiene parámetros, se muestran como una lista separada por comas entre paréntesis siguiendo al nombre.

Un diagrama de secuencia tiene dos dimensiones: la dimensión vertical que representa el tiempo; la dimensión horizontal representa objetos que participan en la interacción (Figura 14.99 y Figura 14.100). Generalmente, el tiempo avanza página abajo (pueden invertirse los ejes si se desea). A menudo, sólo es importante la secuencia de los mensajes pero en aplicaciones de tiem-

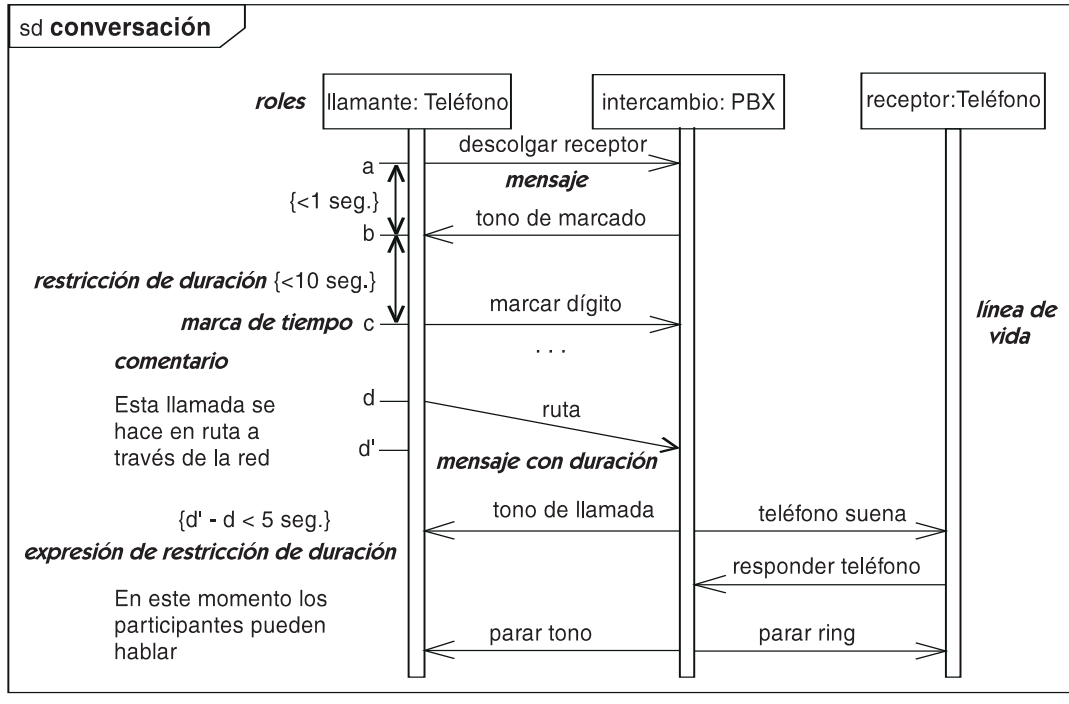


Figura 14.99 Diagrama de secuencia con flujos de control asíncronos

po real, el eje de tiempo puede ser un elemento métrico real. La posición horizontal de los objetos no es significativa.

Cada participante se muestra en una columna separada llamada línea de vida. Cada línea de vida juega un rol dentro de la interacción. Durante cada ejecución de la interacción, la línea de vida representa un objeto o conjunto de objetos. Se coloca un rectángulo en la cima del diagrama (si el objeto existe cuando la interacción empieza) o en el punto durante la interacción donde se crea el objeto, al final de una flecha del mensaje para la acción de creación. Se dibuja una línea hasta el punto donde el objeto se destruye (si eso pasa durante el tiempo mostrado por el diagrama). Esta línea se llama la línea de vida. Una *X* grande se pone en el punto en que el objeto cesa de existir, o a la cabeza de la flecha del mensaje que destruye el objeto o en el punto en que el objeto se destruye. Para cualquier período durante el que el objeto está activo, la línea de vida se ensancha a una línea sólida doble. Esto incluye la vida entera de un objeto activo o una activación de un período de objeto pasivo durante el cual una operación del objeto está en ejecución, incluyendo el tiempo durante el cual la operación espera el retorno de la operación que llamó. Si las llamadas del objeto son recursivas, directamente o indirectamente, entonces otra copia de la línea sólida doble es solapada en él para mostrar la activación doble (potencialmente podrían ser más de dos copias). La ordenación relativa de objetos no tiene significado, aunque es útil minimizar la distancia que las flechas del mensaje deben cubrir.

Se puede colocar un comentario sobre la activación en el margen cerca de él.

En el rectángulo en la cima de la línea de vida, se coloca una cadena con la sintaxis:

`nombre` ⌊ **`selector`** ⌋$_{\text{opc}}$: **`Tipo`**

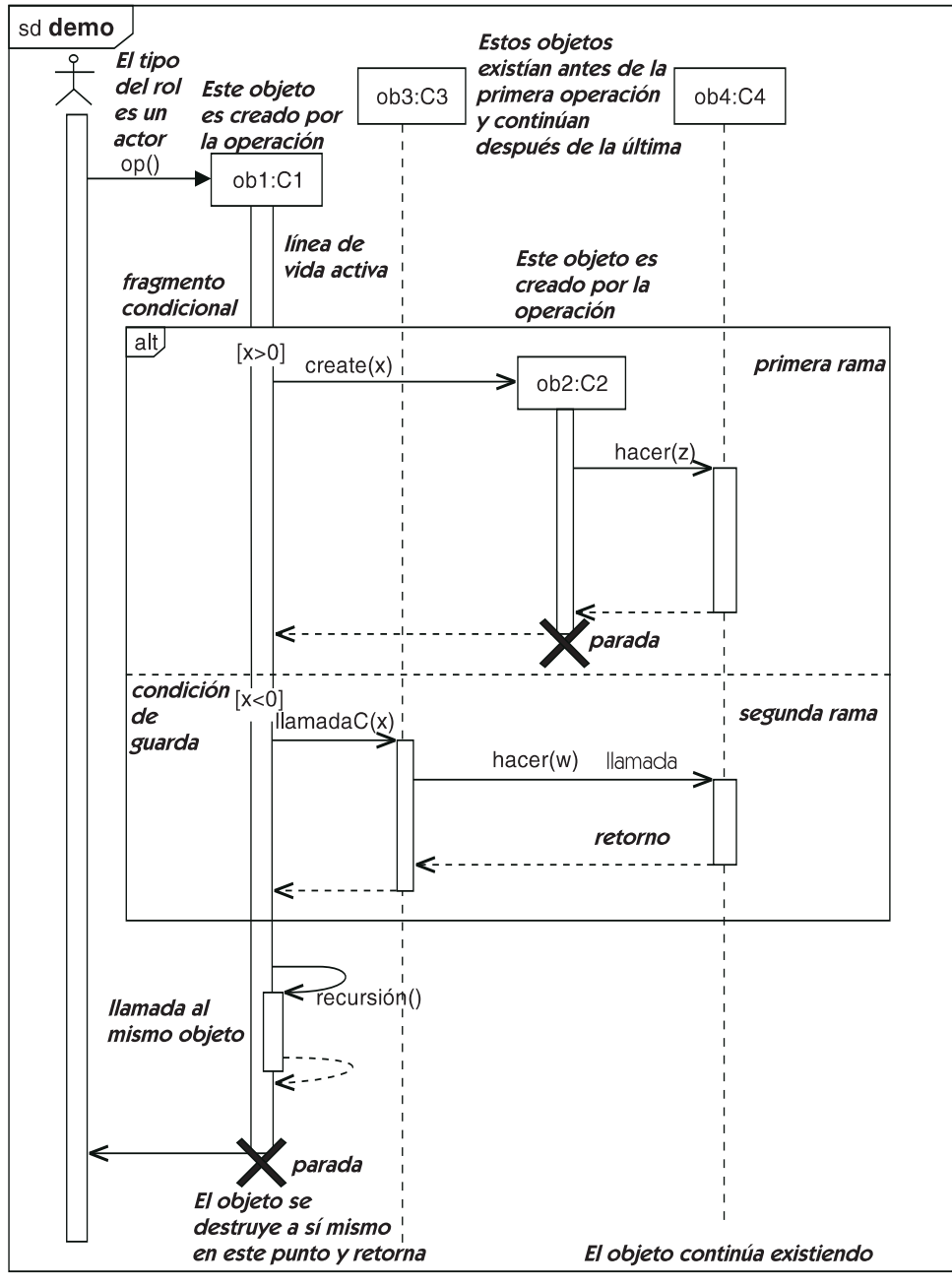


Figura 14.100 Diagrama de secuencia con flujo de control procedural

Tanto el nombre o el tipo (pero no ambos) pueden omitirse. El nombre es el nombre del papel o rol dentro de la interacción. El selector es una expresión para identificar un elemento de un conjunto si la multiplicidad es mayor que uno; se omite si la multiplicidad es uno. El tipo es el tipo del objeto.

Para indicar una línea de vida que se descompone en una interacción, añada la cadena:

ref **`nombre-de-la-interacción`**

a la cadena en el rectángulo de la línea de vida.

Cada mensaje se muestra como una flecha horizontal de la línea de vida del objeto que envía el mensaje a la línea de la vida del objeto que recibe el mensaje. Se puede poner una etiqueta en el margen opuesto con una flecha para denotar el tiempo en que el mensaje es enviado o recibido. La etiqueta de tiempo tiene una marca de picadillo horizontal para mostrar su situación exacta en el diagrama.

Se asume que los mensajes son instantáneos en muchos modelos, o por lo menos atómicos.

Si un mensaje requiere algún tiempo para alcanzar su destino, entonces la flecha del mensaje puede dibujarse descendente diagonalmente para que el extremo del receptor sea más bajo que el tiempo enviado. Ambos extremos pueden tener las etiquetas para denotar el tiempo en que el mensaje fue enviado o recibido.

Para el flujo asíncrono de control entre los objetos activos, los objetos se representan por líneas sólidas dobles y los mensajes se muestran como flechas. Dos mensajes pueden enviarse simultáneamente, pero no pueden recibirse dos mensajes simultáneamente.

No hay ninguna manera de garantizar la recepción simultánea. La Figura 14.99 muestra un diagrama de secuencia asíncrono.

La Figura 14.100 muestra el flujo procesal de mando en un diagrama de sucesión. Cuando el flujo procesal de un modelo de control, un objeto rinde el mando en una llamada hasta un subsecuente retorno. Una llamada síncrona se muestra con una punta de flecha llena sólida. La cabeza de una flecha de llamada puede empezar una activación o un nuevo objeto. Un retorno se muestra con una línea discontinua. La cola de una flecha de retorno puede terminar una activación o un objeto.

Varias estructuras de control de flujo, tales como condicionales, lazos y ejecución concurrente, se pueden mostrar empotrando un fragmento del diagrama dentro del diagrama total.

El fragmento del diagrama incluye un subconjunto de las líneas de vida. Tiene un operador etiqueta para indicar su propósito. Algunos tipos de fragmentos, como el condicional, son divididos en subregiones por líneas discontinuas horizontales. Cada región es un subdiagrama y la relación entre las subregiones es indicada por la etiqueta del fragmento. Por ejemplo en un condicional, cada rama se muestra en una región separada. *Véase* fragmento combinado para una lista de tipos del fragmento.

Un estado o condición de la línea de la vida pueden mostrarse como un símbolo de estado pequeño (rectángulo con esquinas redondeadas) puesto en la línea de vida. El estado o condición deben ser verdaderos en el momento del próximo evento en la línea de vida. La Figura 14.101 muestra los estados durante la vida de un boleto del teatro. Una línea de vida puede ser interrumpida por un símbolo de estado para mostrar un cambio de estado. Se puede asociar una flecha al símbolo de estado para indicar el mensaje que causó el cambio de estado.

Un diagrama de estados puede estar parametrizado para que pueda ser referenciado desde otros diagramas de estados. Una puerta es un punto de conexión en un diagrama de estados para los mensajes de fuera. *Véase* la Figura 14.243 para un ejemplo.

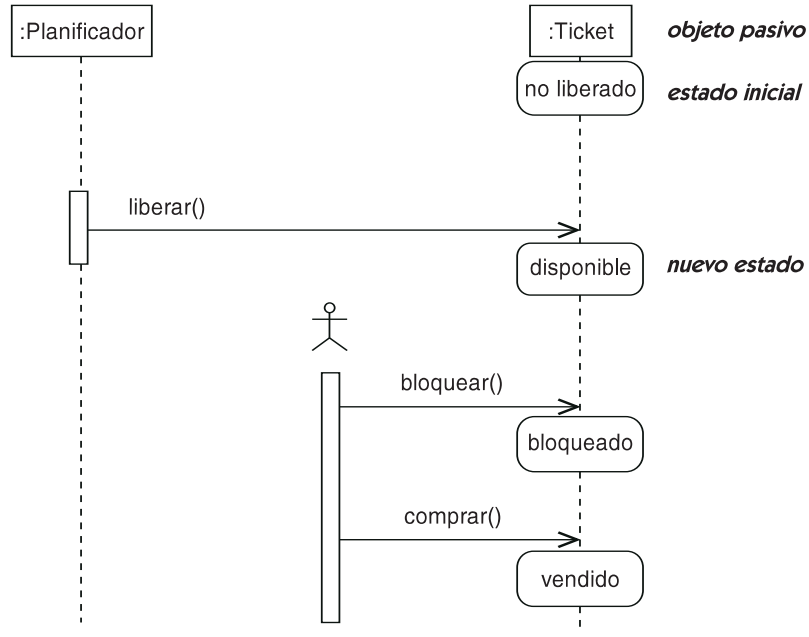


Figura 14.101 Estados de un objeto en un diagrama de secuencia

Un uso de interacción es realizar una referencia a un diagrama de estados dentro de otro diagrama de estados.

La referencia puede incluir un subconjunto de las líneas de vida en el diagrama de referencia.

Véase la Figura 14.286 para un ejemplo. Si el diagrama de estado referido tiene puertas, el diagrama de referencia puede conectar los mensajes al símbolo de interacción usado.

Historia

Mucha de la notación de diagrama de secuencia de UML2 se toma directamente de la notación ITU de Mapa (MSC) de secuencias de mensaje, aunque ha sido extendida.

diagrama de tiempos

Forma alternativa de mostrar un diagrama de secuencia que muestra explícitamente cambios en el estado de una línea de vida y tiempo métrico (unidades de tiempo). Pueden ser útiles en aplicaciones de tiempo real.

Notación

Un diagrama de tiempos (Figura 14.102) es una forma especial de un diagrama de secuencia con las siguientes diferencias:

- Los ejes normalmente están intercambiados, de forma que el tiempo se incrementa desde la izquierda a la derecha.

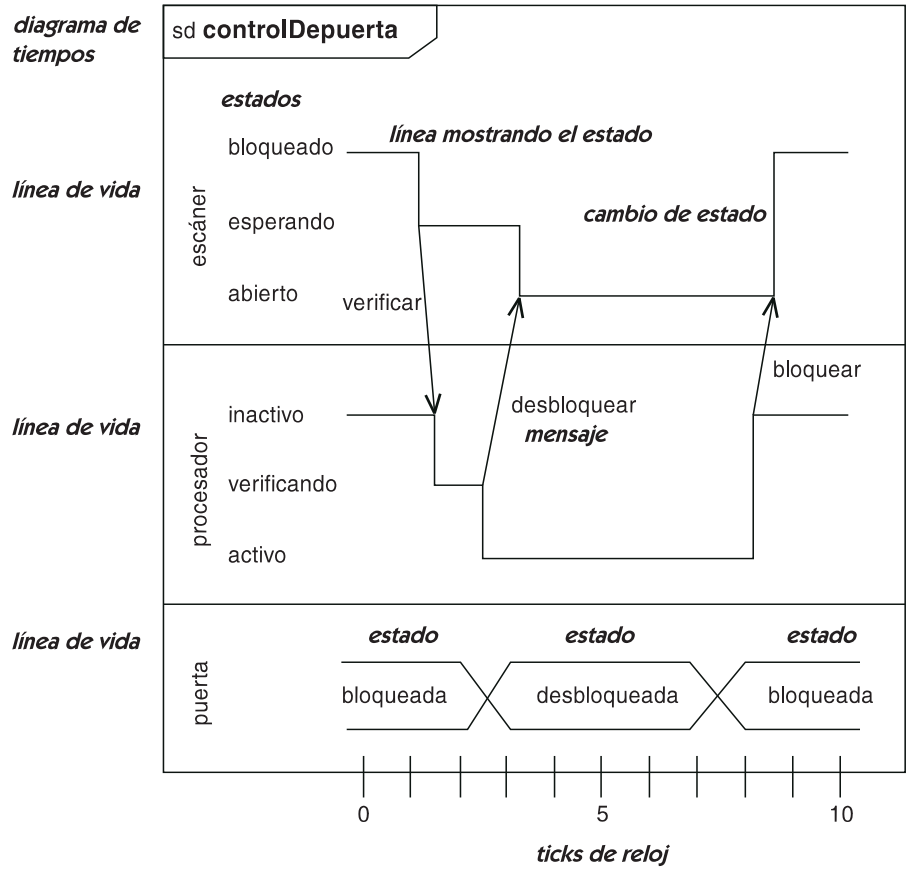


Figura 14.102 Diagrama de tiempos

- Las líneas de vida se muestran en compartimentos separados organizadas verticalmente.
- La línea de vida sube y baja para mostrar cambios en el estado. Cada posición vertical representa un estado diferente. La ordenación de los estados puede tener o no importancia.
- Alternativamente, una línea de vida puede seguir una sola línea con diferentes estados o valores mostrados en la línea.
- Se puede mostrar un eje de tiempo métrico. Las marcas de tiempo indican intervalos de tiempo, a veces tiempos discretos en los que se producen cambios.
- Los tiempos en las distintas líneas de vida están sincronizados.
- Se puede mostrar el valor que tiene un objeto.

difusión

Una señal asíncrona enviada a un conjunto implícito de objetos destino asociados con el sistema actual.

Véase acción.

Semántica

Una acción de difusión tiene un tipo de señal y una lista de argumentos como parámetros, pero no un objeto destino. La señal se envía a todos los objetos de un conjunto implícito dependiente del sistema. La intención es que se les envíe la señal a todos los objetos alcanzables por el sistema, pero en la práctica esto es imposible y una difusión normalmente irá destinada a un subconjunto de objetos del nodo local.

Discusión

El concepto de difusión es teóricamente problemático, porque el conjunto de nodos puede ser abierto, de forma que se debe considerar como una acción fundamentalmente específica de la implementación asociada con algunos conceptos de localidad y un universo cerrado de objetos.

diseño

Etapa del desarrollo de un sistema que describe cómo se implementará el sistema a nivel lógico por encima del código real. En el diseño, se toman decisiones estratégicas y tácticas para alcanzar la funcionalidad y calidad requeridas en un sistema. Los resultados de esta etapa se representan por medio de modelos de diseño, especialmente la vista estática, vista de máquina de estados y vista de interacción. Contrasta con: análisis, diseño, implementación y despliegue (fase de).

Véase etapas de modelado, proceso de desarrollo

disjoint

Palabra clave para un conjunto de generalizaciones cuyos subtipos son incompatibles.

Véase conjunto de generalizaciones.

disparador

Especificación de un evento cuya ocurrencia causa la ejecución de comportamiento, como hacer una transición elegible para su disparo. La palabra se puede utilizar como un nombre (para el propio evento) o como un verbo (para la ocurrencia del evento). Se utiliza para describir la asociación del evento con la transición.

Véase transición de finalización, transición.

Semántica

Cada transición (excepto una transición de finalización que se dispara en la finalización de la actividad interna) tiene una referencia a un evento como parte de su estructura. Si el evento se produce cuando un objeto está en un estado que contiene una transición de salida cuyo desencadenador es el evento de un antecesor del evento, entonces se prueba la condición de guarda de la transición. Si se satisface la condición, entonces la transición queda activada para su disparo. Si

está ausente la condición de guarda, entonces se estima que está satisfecha. Si hay más de una transición elegible para su disparo, realmente sólo se disparará una. La elección puede ser no determinista. (Si el objeto tiene más de un estado concurrente, se puede disparar una transición desde cada estado. Pero como mucho se puede disparar una transición de cada estado.)

Observe que la condición de guarda se prueba una vez, en el momento en que se produce el evento desencadenador (incluyendo un evento de finalización implícito). Si la ocurrencia de un evento no activa para su disparo ninguna transición, entonces simplemente se ignora el evento. Esto no es un error.

Los parámetros del evento desencadenador están disponibles para su utilización en una condición de guarda o en un efecto adjunto a la transición o a una actividad entrar en el estado destino.

Durante toda la ejecución de un paso de ejecutar hasta finalizar después de una transición, el evento desencadenador permanece disponible para las acciones de los subpasos de la transición como el evento actual. El tipo exacto de este evento puede no ser conocido en una acción entrar o en un segmento posterior en una transición de varios segmentos. Por tanto, se puede discriminar el tipo de evento en una acción utilizando una operación polimórfica o una sentencia case. Una vez que se conoce el tipo exacto del evento, se pueden utilizar sus parámetros.

El evento desencadenador puede ser la recepción de una señal, la recepción de una llamada, un evento de cambio, o un evento temporal. La finalización de actividad es un pseudodesencadenador.

Notación

El nombre y la signatura del evento desencadenador forman parte de la etiqueta de una transición.

Véase transición.

disparador cualquiera

Un disparador de una transición de una máquina de estados que se satisface con una ocurrencia de cualquier evento que no dispara otra transición en el mismo estado.

Semántica

Es la construcción “sino” para las transiciones. Por supuesto, es incorrecto tener múltiples disparadores cualquiera en el mismo estado.

Probablemente es permisible tener disparadores cualquiera en un estado y en el que lo contiene, teniendo mayor precedencia la transición más específica, pero la especificación no dice nada sobre este punto.

Notación

La palabra clave **all** se utiliza como nombre del disparador en lugar del nombre del evento.

disparador de cambio

Un disparador en una transición de una máquina de estados que se vuelve disponible si ocurre un evento de cambio específico.

Semántica

Un disparador de cambio especifica un evento de cambio mediante una expresión Booleana. No hay parámetros para el evento. El evento de cambio ocurre cuando las condición se convierte en verdadera (después de haber sido falsa) debido a un cambio en una o más variables de las cuales depende la condición. Cuando sucede el evento, se dispara el disparador de cambio y habilita la transición, haciendo que cualquier condición de guarda sea verdadera.

Un disparador de cambio implica una comprobación continua de la condición. Sin embargo, y en la práctica, analizando los instantes en los que las entradas de la condición pueden cambiar, una implementación a menudo puede realizar las pruebas en momentos discretos bien definidos, por lo que los sondeos continuos no suelen ser necesarios.

El evento ocurre cuando el valor de la expresión cambia de falso a verdadero (un cambio de estado de lógica positiva). El evento ocurre una vez cuando esto sucede y no se repite a menos que el valor primero vuelva a ser falso de nuevo.

Es un punto de variación semántica si el disparador de cambio permanece activo si las condiciones subyacentes se convierten en falsas antes de que el disparador habilite la transición.

Obsérvese la diferencia con respecto a una condición de guarda. La condición de guarda se evalúa *una vez* siempre que ocurre el evento de la transición. Si la condición es falsa, entonces la transición no se dispara y se pierde el evento (a menos que dispare alguna otra transición). Un evento de cambio se evalúa implícitamente de forma continua y ocurre una vez en el momento que el valor se convierte en verdadero. En ese momento puede disparar una transición o puede ser ignorado. Si es ignorado, el evento de cambio no dispara una transición en el subsiguiente estado sólo porque la condición todavía es verdadera. El evento de cambio ya ha sucedido y ha sido descartado. La condición se debe convertir en falsa y de nuevo verdadera para provocar otro evento de cambio.

Los valores en la expresión Booleana deben ser atributos del objeto al que pertenece 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 transiciones. Un disparador de cambios se muestra mediante la palabra clave **when** seguida de una expresión Booleana entre paréntesis: Por ejemplo:

```
when (self.esperandoClientes > 6)
```

Discusión

Un disparador de cambio es una prueba de la satisfacción de una condición. Puede ser costosa su implementación, aunque a menudo hay técnicas para compilarlas por lo que no deben ser com-

probadas continuamente. Sin embargo, es potencialmente costoso, al tiempo que oculta la relación directa causa y efecto entre el cambio de un valor y los efectos que son disparados por él. Algunas veces esto es deseable porque encapsula los efectos, pero los disparadores de cambio deben utilizarse con cuidado.

Un disparador de cambio está pensado para representar la prueba de valores visibles de un objeto. Si se pretende que un cambio de un atributo dentro de un objeto dispare un cambio en otro objeto que no es consciente del propio atributo, entonces la situación se modela como un disparador de cambio en el propietario de los atributos que dispara una transición interna para enviar una señal al segundo objeto.

Obsérvese que no se envía un evento de cambio explícitamente a cualquier sitio. Si está prevista una comunicación explícita con otro objeto, se debe utilizar una señal en su lugar.

La implementación de un disparador de cambio se puede hacer de varias formas, algunas de ellas realizan pruebas dentro de la propia aplicación en momentos apropiados y otras por medio de las facilidades del sistema operativo subyacente.

disparador de llamada

La especificación del efecto causado por la recepción de una llamada por un objeto activo. Si un objeto (a través de su clase) no tiene disparador para una operación, la recepción de una llamada causa un efecto inmediato. Es decir, un efecto ocurre concurrentemente con cualquier otro efecto en el sistema y, en particular, concurrentemente con cualquier actividad del propio objeto destino. El efecto inmediato más común es la ejecución de un método en el hilo de control del llamador. Obsérvese que muchos métodos se pueden ejecutar concurrentemente en el mismo objeto destino si son invocados desde hilos de control independientes. No hay sincronización de esas ejecuciones entre sí o con cualquier actividad principal del objeto destino.

En contraste, si un objeto activo tiene un disparador para una operación, una llamada no causa un efecto inmediato. En su lugar, se sitúa un evento de llamada en el pool de eventos del objeto destino. Un disparador implica la sincronización de múltiples llamadas en un único objeto. Cuando el objeto termina un paso de ejecución hasta la finalización y necesita otro disparador, se puede seleccionar el evento de llamada. (UML no especifica ninguna regla de ordenación genérica para seleccionar eventos del pool de eventos, pero se espera que los perfiles asociados con entornos de ejecución particulares puedan establecer tales reglas, como el primero que entra es el primero que sale.) Una vez que se selecciona el evento de llamada, éste experimenta el proceso de resolución para establecer un efecto. Este efecto puede ser la ejecución de un método, la habilitación de una transición de una máquina de estados u otra cosa. Esto es en general. En la vasta mayoría de modelos, la resolución será orientada a objetos. Si el disparador está asociado con un método, el método se ejecutará primero y puede devolver valores al llamador. Cuando se completa el método, el llamador es libre de seguir. Si no hay método, el llamador puede seguir inmediatamente (incluso si la llamada es síncrona) y no hay valores de retorno. Si el disparador está asociado con una transición de una máquina de estados, se permite la transición (sujeta a las condiciones de guarda) después de que se ejecute un posible método y continúa concurrentemente con la subsiguiente ejecución del llamador.

Ejemplo

La Figura 14.103 muestra llamadas a operaciones que se implementan como disparadores en transiciones internas de una máquina de estados. Una cuenta puede estar **Bloqueada** y **Desbloqueada**. La operación **depositar** siempre añade dinero a una cuenta, a pesar de su estado. La operación **retirar** saca todo el dinero si la cuenta está desbloqueada o no saca nada si está bloqueada. La operación **retirar** se implementa como un evento de llamada que dispara transiciones internas en cada estado. Cuando se da la llamada, se ejecuta una secuencia de acciones o la otra, dependiendo del estado activo. Si el sistema está bloqueado, se devuelve cero; si el sistema está desbloqueado, todo el dinero de la cuenta se devuelve y el contador se inicializa a cero.

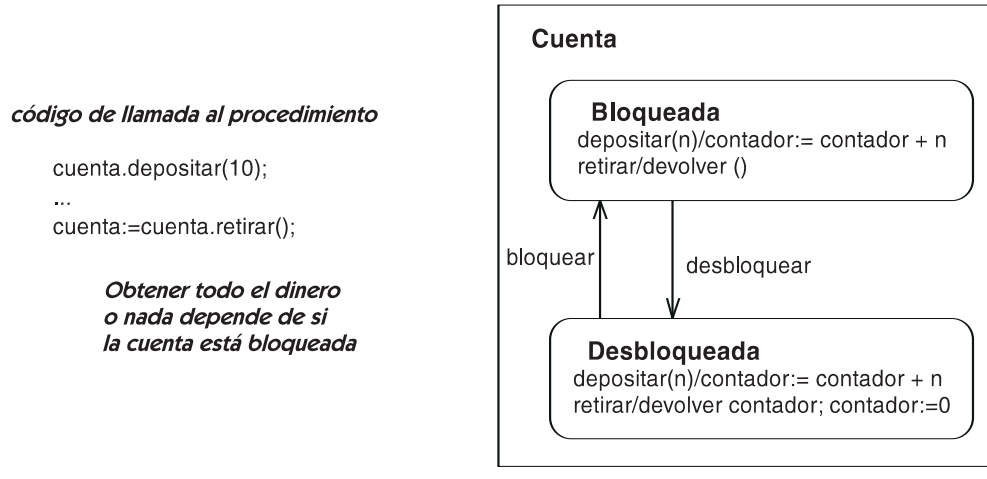


Figura 14.103 Eventos de llamada

Historia

El concepto de permitir a una llamada disparar transiciones se denominó, en UML1, *evento de llamada*, pero se ha ampliado e integrado mejor con las señales y otros tipos de comportamientos en UML2 en un cambio importante.

disparador de señal

Un disparador cuyo evento es un evento de señal.

disparar

Ejecutar una transición.

Véase también ejecutar hasta finalizar, disparador.

Semántica

Cuando se produce un evento requerido por una transición, y la condición de guarda de la transición se satisface, la transición realiza su acción y el estado activo cambia.

Cuando un objeto recibe un evento, el evento se guarda en un depósito si la máquina de estados está ejecutando un paso de ejecutar hasta finalizar. Cuando se completa el paso, la máquina de estados gestiona un evento del depósito, si hay alguno. Se desencadena una transición si su evento es manejado mientras su objeto propietario 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 desencadenador que es antepasado del tipo de evento que se produce. Si una transición compleja tiene varios estados origen, todos ellos deben estar activos para que la transición se active. Una transición de finalización está activa cuando su estado origen finaliza la actividad. Si es un estado compuesto, está activo cuando todos sus subestados directos han completado o alcanzado sus estados finales.

Cuando se gestiona el evento, se evalúa la condición de guarda (si existe). Si la expresión lógica de la condición de guarda se evalúa a verdadero, entonces se dice que la transición se *dispara*. Se ejecuta la acción de la transición, y el estado del objeto se convierte en el estado destino de la transición (sin embargo, no se produce un cambio de estado para transiciones internas). Durante el cambio de estado, se ejecutan todos los efectos de las actividades entrar y salir en el camino mínimo desde el estado original del objeto al estado destino de la transición. Observe que el estado original puede ser un subestado anidado del estado origen de la transición.

Si la condición de guarda no se satisface, no ocurre nada como resultado de esta transición, aunque alguna otra transición podría dispararse si se cumple su condición de guarda.

Conflictos. Si hay más de una transición apta para su disparo, sólo se disparará una de ellas. Una transición en un estado anidado tiene precedencia sobre una transición en un estado que lo englobe. Por lo demás, la elección de transiciones no está definida y puede ser no determinista. Esto es a menudo una situación del mundo real, y a veces puede ser deseable el no determinismo.

En la práctica, una implementación puede proporcionar ordenación de transiciones para su disparo. Esto no cambia la semántica, puesto que se puede conseguir el mismo efecto organizando las condiciones de guarda de forma que no se solapen. Pero normalmente es más sencillo poder decir, “Esta transición se dispara sólo si ninguna otra transición se dispara.”

Eventos diferidos. Si el evento o uno de sus antepasados están marcados para su aplazamiento en el estado o en un estado que lo englobe, y el evento no desencadena una transición, el evento es un evento diferido hasta que el objeto entre en un estado en el que el evento no esté diferido. Cuando el objeto entra en un nuevo estado, todos los eventos diferidos anteriormente que no sigan diferidos se transforman en *pendientes* y serán gestionados en un orden indeterminado. Si el primer evento pendiente no causa el disparo de una transición, se descarta y se produce otro evento pendiente. Si un evento diferido previamente está marcado para su aplazamiento en el nuevo estado, puede desencadenar una transición, pero permanece diferido si falla al desencadenar una transición. Si la ocurrencia de un evento causa una transición a un estado nuevo, todos los eventos pendientes y diferidos que queden serán reevaluados de acuerdo al estado de aplazamiento del nuevo estado y se establece un nuevo conjunto de eventos pendientes.

Una implementación podría imponer reglas más estrictas sobre el orden en que se procesan los eventos diferidos o proporcionar operaciones para manipular su orden.

dispositivo

Recurso físico computacional con capacidad de proceso en el que se pueden desplegar artefactos para su ejecución. Un dispositivo es un tipo de nodo. *Véase* nodo.

Semántica

No hay gran diferencia entre un nodo y un dispositivo. Los dispositivos representan dispositivos computacionales físicos, aunque las pautas son imprecisas. La distinción sería más significativa en un perfil que defina tipos particulares de dispositivos dentro de un determinado entorno de ejecución.

Notación

Un dispositivo se representa como un símbolo de nodo (un cubo) con la palabra clave **«device»** sobre el nombre del dispositivo.

división

En una máquina de estados, una transición compleja en la que se reemplaza un estado origen por dos o más estados destino, dando como resultado en un incremento del número de los estados activos. En una actividad, un nodo que copia un solo token de entrada en varias salidas concurrentes. Antónimo: unión.

Véase también transición compleja, estado compuesto, unión.

Semántica

En una máquina de estados, una división es una transición con un estado origen y dos o más estados destino. (Realmente es un pseudoestado, pero eso es un detalle interno que no es importante en la mayoría de los casos.) Si el estado origen está activo y se produce el evento de desencadenamiento, la acción de la transición se ejecuta y todos los estados destino se activan. Los estados destino deben estar en diferentes regiones de un estado compuesto.

En una actividad, un nodo de división es un nodo con una entrada y varias salidas. Se copia un token de entrada en cada una de las salidas, incrementando la cantidad de concurrencia. Las salidas pueden tener guardas, pero esta utilización introduce bastante peligro de generar modelos mal formados; recomendamos que las decisiones y las divisiones se separen.

Tanto en máquinas de estados como en actividades, las divisiones incrementan la cantidad de hilos de ejecución concurrentes. Una división normalmente se corresponde más tarde con una unión que decrementa la cantidad de concurrencia.

Notación

En un diagrama de máquinas de estado o en un diagrama de actividad, una división se representa como una barra gruesa con una flecha de transición entrante y dos o más flechas de transiciones salientes. Puede tener una etiqueta de transición (condición de guarda, evento desencadenador y acción). La Figura 14.104 muestra una división explícita en un estado compuesto concurrente.

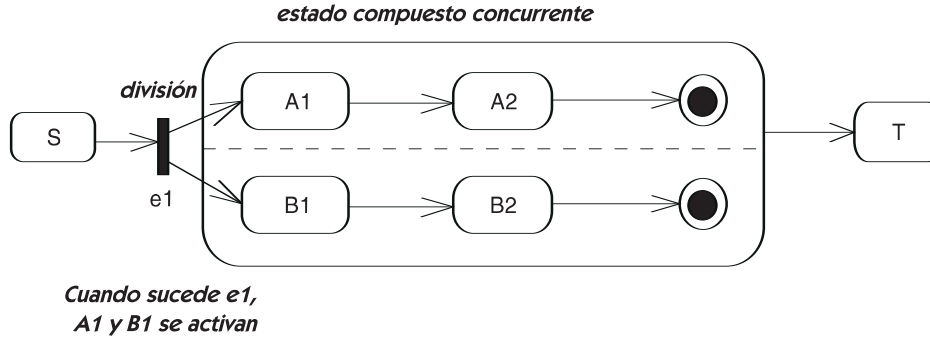


Figura 14.104 División

document (estereotipo de Componente)

Componente que representa un documento; archivo que no es un archivo fuente o un ejecutable.

Véase componente.

duración

Especificación del tiempo transcurrido entre dos eventos.

efecto

Acción o actividad que se ejecuta cuando se dispara una transición. La ejecución tiene semántica de ejecutar hasta finalizar; es decir, no se procesan eventos adicionales durante la ejecución.

Véase transición.

efecto lateral

Un cómputo que modifica el estado o valor de un sistema, como opuesto a uno que meramente devuelve un valor sin hacer ningún cambio permanente (una pregunta).

eficiencia de navegación

Indica si es posible cruzar eficazmente una asociación para obtener el objeto o conjunto de objetos asociados con un valor o tupla de valores del otro extremo o extremos. La eficiencia de la navegación normalmente se define a través de navegabilidad pero no es su propiedad primordial.

Véase también navegabilidad.

Semántica

La eficiencia de la navegación puede definirse de una manera general para que sea aplicable tanto al diseño abstracto como a los diferentes lenguajes de programación. Una asociación es navegable de manera eficaz, si el coste medio de obtener el conjunto de objetos asociados es proporcional al número de objetos en el conjunto (*no* al límite superior de la multiplicidad que puede ser ilimitada) más una constante fija. En términos de complejidad computacional, el coste es *O(n)*. Si la multiplicidad es uno o cero-uno, entonces el coste de acceso debe ser constante, lo que evita buscar en una lista de longitud variable. A la definición ligeramente más libre de eficiencia de la navegación permitiría un coste mínimo de *log(n)*.

Aunque una asociación navegable de multiplicidad uno se implementa normalmente mediante un puntero, también es posible una implementación externa usando tablas hash, que tienen un coste medio de acceso constante. Así, una asociación puede implementarse como una tabla externa de búsqueda de objetos y aún puede ser considerada navegable (En algunas situaciones de tiempo real, hay que limitar el costo del peor-caso en lugar del promedio. Esto no requiere cambios en la definición básica más que sustituir el tiempo por el del peor-caso, y el que algoritmos probabilísticos como las tablas hash deben eliminarse.)

Si una asociación no es navegable en una dirección dada, no significa que no pueda cruzarse en absoluto, pero sí que el coste de transversal puede ser significativo —por ejemplo una búsqueda a través de una lista grande. Si el acceso en una de las direcciónes es poco frecuente, la búsqueda puede ser una opción razonable. La eficiencia de la navegación es un concepto de diseño que permite a un diseñador definir los caminos de acceso de objeto con conocimiento de los costes de complejidad computacionales. Normalmente, la navegabilidad implica la eficiencia de la navegación.

Es posible (pero raro) tener una asociación que no es eficazmente navegable ninguna dirección. Tal asociación podría implementarse como una lista de enlaces donde deben realizarse búsquedas para atravesar la asociación en cualquier dirección. Sería posible pero ineficaz cruzarla. No obstante, el uso para una asociación de ese tipo es pequeño.

Observe que la navegación eficaz no requiere la implementación como punteros. Una tabla de base de datos puede implementar una asociación. La asociación es eficientemente navegable si la indexación evita la necesidad de realizar búsquedas por fuerza bruta.

Discusión

La eficiencia de la navegación indica la eficacia para obtener el conjunto de objetos relacionados a un objeto dado o tupla. Cuando la multiplicidad de una asociación binaria es 0..1 o 1 entonces la implementación obvia es un puntero en el objeto de origen. Cuando la multiplicidad es muchos, entonces la implementación usual es una clase contenedora que contiene un conjunto de punteros. La propia clase contenedora puede o no residir dentro de los datos que se graban para un objeto de la clase, dependiendo de si puede obtenerse con un coste constante (la situación usual para acceso con punteros). La clase contenedora debe ser eficaz para la navegación. Por ejemplo, una lista simple de todos los enlaces para una asociación no sería eficaz, porque los enlaces para un objeto se mezclarían con muchos otros enlaces poco interesantes y requeriría una búsqueda. Una lista de enlaces guardada con cada objeto es eficaz, porque no requiere ninguna búsqueda innecesaria.

En una asociación calificada, una marca de navegación en la dirección fuera del calificador normalmente indica que es eficaz para obtener el objeto o conjunto de objetos seleccio-

nados por un objeto del origen y valor del calificador. Esto es consistente con una implementación usando tablas hash o quizás una búsqueda binaria en árbol usando como índice el valor calificador (que es exactamente la razón para incluir los calificadores como un concepto de modelado).

Una asociación *n*-aria normalmente se implementa como un objeto independiente. Puede contener índices sobre diferentes extremos de la asociación para permitir la navegación eficaz.

ejecución

Procesamiento en tiempo de ejecución de la especificación de un comportamiento para modificar el estado de un sistema dentro de un entorno de ejecución particular.

Semántica

El concepto de ejecución de comportamiento es parte de la semántica dinámica de UML, pero no es parte de ningún modelo de UML, puesto que un modelo UML describe la especificación del propio comportamiento. Cuando se ejecuta el modelo, el uso del comportamiento es una ejecución de comportamiento.

Un modelo en tiempo de ejecución se puede construir utilizando interacciones. Una especificación de ejecución modela la ocurrencia de una ejecución de comportamiento durante una descripción de comportamiento.

ejecutar hasta finalizar

Una transición o serie de acciones que deben completarse en su integridad.

Véase también acción, máquina de estado, transición.

Semántica

Una máquina de estados sufre conceptualmente con el tiempo, una serie de pasos discretos en que se ocupa de un evento a la vez, perteneciente a un conjunto de eventos reconocidos por el objeto que posee la máquina de estados. Las ocurrencias de eventos reconocidas por el objeto se ponen en el contenedor de eventos. En general, las ocurrencias de eventos en el contenedor son no ordenadas, aunque los perfiles son libres para definir los ambientes de la ejecución con varias órdenes de prioridad o reglas. Durante cada paso, la máquina de estados selecciona un evento del contenedor (en el caso general, no determinísticamente) y, a continuación, lleva a cabo las consecuencias de gestionar ese evento. El paso de la ejecución puede contener subpasos, incluyendo concurrencia. Durante un paso de la ejecución, ningún otro evento se selecciona de los del contenedor hasta la ejecución completa del paso. Ya que se procesa un evento a la vez hasta que el comportamiento efectuado esté completo, este paso de la ejecución se llama *ejecutar-hasta-finalizar* el paso. Si ocurren eventos adicionales durante el paso ejecutar-hasta-finalizar, se ponen en el contenedor de eventos pero no interrumpen la ejecución.

Al inicio de un paso *ejecutar-hasta-finalizar,* se selecciona una ocurrencia de un evento del contenedor de eventos. En el caso general, la ocurrencia se selecciona de forma no determinista.

El evento se empareja con los estados en la configuración de estado activa para habilitar las transiciones a disparar. Si hay múltiples estados activos, para una transición activa, por región directa de una región ortogonal pueden habilitarse para que sean disparadas. Si se habilitan transiciones múltiples dentro de la misma región, una de ellas se selecciona (no determinísticamente) para ser disparada.

Las transiciones múltiples de regiones ortogonales pueden disparar concurrentemente. No se asume que se produzcan en serie y sus acciones pueden entrelazar. Un buen modelo asegura que esas transiciones ortogonales no actúan recíprocamente.

El encendido de cada transición comprende los pasos siguientes:

1. El *estado adjunto común* es el estado más específico que contienen ambos, el estado de la fuente y el estado designado de la transición. El estado actual se termina como una serie de subestados hasta que el estado actual simplemente esté dentro del estado adjunto común. Durante cada cambio de estado, la actividad de salida del estado terminada es ejecutada.
2. Entonces el efecto unido a la transición se ejecuta.
3. Entonces se escriben estados como una serie de subpasos hasta que el estado designado se toque.

Durante cada cambio de estado, la actividad de entrada del estado entrada se ejecuta.

La transición está ahora completa.

Cuando todas las transiciones han sido disparadas al completo, el paso *ejecutar-hasta-finalizar* está completo y de otra ocurrencia de evento puede ocuparse el contenedor de evento.

Segmentos de la transición y pseudoestados

Una transición puede componerse de segmentos múltiples colocados como una cadena y separados por pseudoestados. Varias cadenas pueden unirse juntas o pueden generar ramas separadamente, para que el modelo total pueda contener un gráfico de segmentos separados por pseudoestados. Sólo un segmento de una cadena puede tener un evento del disparador. La transición se activa cuando el evento del disparador es ocupado por la máquina de estado. Si la condición de guarda de todos los segmentos está satisfecha, la transición se habilita, y dispara, con tal de que no se dispare ninguna otra transición. Se ejecutan las acciones en los segmentos sucesivos. Una vez que la ejecución empieza, las acciones en todos los segmentos de la cadena deben ejecutarse antes de que el paso *ejecutar-hasta-finalizar* esté completo.

Durante la ejecución de una transición de *ejecutar-hasta-finalizar,* el evento del disparador que comenzó la transición está disponible a las acciones como el evento actual, por consiguiente, la entrada y actividades de la salida pueden obtener los argumentos del evento del disparador. Distintos eventos pueden causar la ejecución de una entrada o actividad de salida, pero una actividad puede diferenciar el tipo del evento actual.

Debido a la semántica de transiciones *ejecutar-hasta-finalizar,* deben ser usados para diseñar asignaciones, probando las banderas, aritmética simple y otros tipos de operaciones básicas. Deben diseñarse cómputos largos como actividades interrumpibles o hilos separados.

elaboración

La segunda fase de un proceso de desarrollo de software, durante el que se comienza el diseño del sistema y se desarrolla y prueba la arquitectura. Durante esta fase, se completa la mayoría de la vista de análisis, junto con las partes de la arquitectura de la vista de diseño. Si se construye un prototipo ejecutable, también estará realizada parte de la vista de implementación. *Véase* proceso de desarrollo.

elección

Un nodo en una máquina de estados en la que se realiza una evaluación dinámica de la condición de guarda subsiguiente.

Semántica

Una transición de una máquina de estados se puede estructurar en varios segmentos utilizando nodos de conjunción para separar segmentos. Sólo un segmento puede tener un disparador, pero cada segmento puede tener su propia condición de guarda y sus efectos. Sin embargo, y desde el punto de vista semántico, todas las condiciones de guarda a lo largo de una ruta completa se evalúan antes de que la ruta sea elegida o de que se ejecute algún efecto, por lo que la segmentación se hace por conveniencia. Los nodos de conjunción no se deben considerar estados en ningún sentido.

Un nodo de elección también divide una transición en dos segmentos, el primero de los cuales puede tener un disparador. Sin embargo, en el caso de un nodo de elección, el primer segmento está habilitado sin considerar cualquier segmento posterior, y cualquier efecto sobre él se ejecuta antes de que se evalúe alguna condición de guarda posterior. Cuando se alcanza el nodo de elección, se evalúan dinámicamente las condiciones de guarda de los segmentos salientes, incluyendo cualquier cambio causado por la ejecución del primer segmento. En ese momento, uno de los segmentos salientes cuya condición de guarda se evalúa como verdadera se dispara. Si ningún segmento saliente es verdadero, el modelo está mal formado. Para evitar esta posibilidad, se puede aplicar la condición de guarda **[else]** a uno de los segmentos, siendo elegido si todas las demás condiciones de guarda se evalúan como falsas.

Se puede considerar un nodo de elección como un estado real con la restricción de que se debe habilitar inmediatamente cualquier transición saliente utilizando únicamente condiciones de guarda.

Notación

Un nodo de elección se muestra como un rombo o diamante. Los segmentos salientes deben tener condiciones de guarda pero no disparadores (Figura 14.105).

Para el caso común en el que todas las condiciones de guarda son comparaciones con el mismo valor, el valor se puede situar dentro del rombo y se pueden escribir las condiciones de guarda omitiendo el primer valor compartido. Sin embargo, es poco probable que el ahorro de espacio supere a la pérdida de claridad.

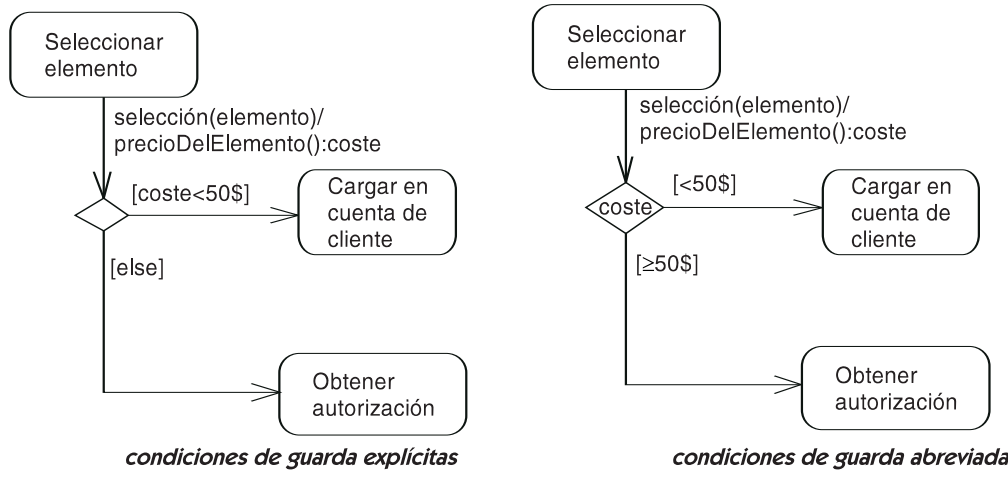


Figura 14.105 Nodo de elección

Historia

Las elecciones dinámicas dentro de las transiciones son nuevas en UML2.

elemento

Componente de un modelo. Este libro describe elementos que se pueden utilizar en los modelos de UML —elementos del modelo, que expresan información semántica, y elementos de presentación, que proporcionan presentaciones gráficas de información de los elementos del modelo.

Un elemento es la clase más abstracta en el metamodelo UML. Un elemento puede tener otros elementos, o pertenecer a otros elementos, y puede tener comentarios.

Véase también diagrama, elemento del modelo.

elemento conectable

Tipo de elemento que puede formar parte de la implementación de un clasificador estructurado y que se puede adjuntar a un conector.

Semántica

Esta metaclase abstracta incluye atributos, parámetros, partes, puertos y variables. Puesto que se define en términos generales, esta metaclase no limita las cosas demasiado.

Una definición más intuitiva es: Una parte puede representar una instancia o un valor en el contexto de una clase estructurada o de una colaboración. La parte puede ser una pieza inherente estructural del todo o puede representar una entidad transitoria, tal como un parámetro o un valor.

elemento de información

Un elemento de información representa un tipo de información que puede ser intercambiado sobre flujos de información.

elemento de representación

Una representación textual o gráfica de uno o más elementos del modelo.

Véase también diagrama.

Semántica

Elementos de la presentación (a veces llamados elementos de vista, aunque incluyen formas no gráficas de presentación) presentan la información en un modelo para la percepción humana. Son la notación. Muestran parte o toda la información semántica sobre un elemento del modelo. También pueden agregar la información estética útil a los humanos, por ejemplo, agrupando los elementos relacionados conceptualmente. Pero la información agregada no tiene ningún contenido semántico. La expectativa es que en una presentación del elemento es responsable de mantenerse correctamente, a pesar de los cambios a los elementos del modelo subyacentes, considerando que los elementos del modelo no necesitan ser conscientes de su presentación para operar correctamente.

Las descripciones de anotación de UML en este libro definen la cartografía de elementos del modelo a las presentaciones gráficas en una pantalla. La implementación de presentación de los elementos como objetos son responsabilidad de una implementación de la herramienta.

elemento del modelo

Elemento que es una abstracción hecha desde el sistema que se está modelando. Contrasta con elemento de presentación, que es una presentación (normalmente visual) de uno o más elementos de modelado para la interacción humana. Sin embargo, en la especificación UML2 sólo se describen elementos del modelo, de forma que simplemente se les llama elementos.

Semántica

Todos los elementos que tienen semántica son elementos del modelo, incluyendo conceptos del mundo real y conceptos de implementación de sistemas informáticos. Los elementos gráficos cuyo propósito es visualizar un modelo son elementos de presentación. No son elementos del modelo, puesto que no añaden semántica al modelo. La especificación UML2 no los describe.

Los elementos del modelo pueden tener nombres, pero la utilización y restricciones sobre dichos nombres varían según el tipo de elemento del modelo y se discuten en cada tipo. Cada elemento del modelo pertenece a un espacio de nombres apropiado al tipo de elemento. Todos los elementos del modelo pueden tener las siguientes propiedades adjuntas:

restricción	Se pueden adjuntar cero o más restricciones a un elemento del modelo. Las restricciones se expresan como cadenas lingüísticas en un lenguaje de restricción.

comentario	Cadena de texto que añade información significativa para el modelador. A menudo se utiliza para explicar las razones de las elecciones de diseño.

Además, los elementos del modelo pueden participar en relaciones de dependencia. Dependiendo de su tipo específico, los elementos del modelo pueden poseer otros elementos del modelo y pueden pertenecer a otros elementos del modelo. Todos los elementos del modelo pertenecen indirectamente a la raíz del modelo del sistema.

elemento derivado

Elemento que se puede calcular desde otros elementos y que se incluye por claridad o por propósitos de diseño aunque no añade información semántica.

Véase también restricción, dependencia.

Semántica

Un elemento derivado es lógicamente redundante dentro de un modelo puesto que se puede calcular desde otro u otros elementos. La fórmula de calcular un elemento derivado puede proporcionarse como una restricción.

Un elemento derivado se puede incluir en un modelo por varias razones. A nivel de análisis, un elemento derivado es semánticamente innecesario, pero se puede utilizar para proporcionar un nombre o una definición para un concepto significativo, como un tipo de macro. Es importante recordar que un elemento derivado no añade nada a la semántica del modelo.

A nivel de diseño, un elemento derivado representa una optimización —elemento que podría calcularse a partir de otros elementos pero que está presente físicamente en el modelo para evitar el coste o la problemática de recalcularlo. Como ejemplos tenemos un valor intermedio de un cálculo y un índice a un conjunto de valores. La presencia de un elemento derivado implica la responsabilidad de actualizarlo si el valor del que depende cambia.

Notación

Un elemento derivado se representa colocando una barra oblicua (/) delante del nombre del elemento derivado, como un atributo, un nombre de rol o un nombre de asociación (Figura 14.106).

Los detalles del cálculo de un elemento derivado se pueden especificar mediante una dependencia con el estereotipo **«derive»**. Normalmente es conveniente suprimir en la notación la flecha de dependencia desde la restricción hasta el elemento y poner simplemente una cadena de restricción cerca del elemento derivado, aunque la flecha se puede incluir cuando resulta de utilidad.

Discusión

Las asociaciones derivadas probablemente son el tipo más común de elemento derivado. Representan asociaciones virtuales que se pueden calcular desde dos o más asociaciones fundamentales. Por ejemplo, en la Figura 14.106, la asociación derivada **TrabajaParaLaCompañía** se puede calcular componiendo **TrabajaParaElDepartamento** con la composición **empleadora**. Una im-

plementación incluye explícitamente **TrabajaParaLaCompañía** para evitar recalcularla, aunque no representa ninguna información adicional.

Hay una diferencia con la generalización de la asociación (Figura 14.156), que representa dos niveles de detalle para una asociación. Normalmente no se podría implementar en ambos niveles. Normalmente sólo podrían ser implementadas las asociaciones hijas. A veces sólo podría implementarse la asociación padre, con las asociaciones hijas restringiendo los tipos de objetos que se pueden relacionar.

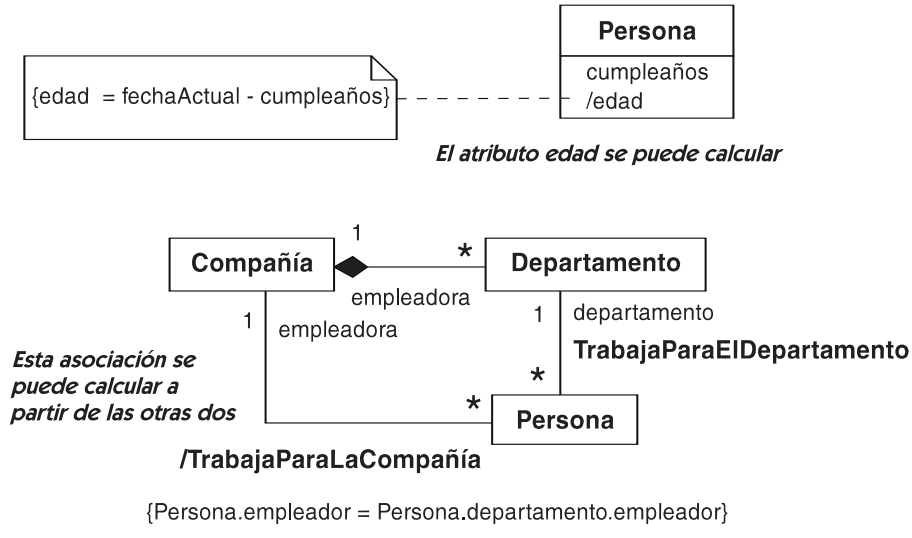


Figura 14.106 Atributo y asociación derivados

elemento empaquetable

Un elemento que puede ser poseído directamente por un paquete.

Semántica

Elementos empaquetables, incluyendo clasificadores, las dependencias, restricciones, especificaciones de instancias y otros paquetes, pueden contenerse como elementos directos de paquetes.

Otros elementos no pueden aparecer solos pero deben ser poseídos por tipos particulares de elementos. Por ejemplo, los atributos y operaciones son partes de clasificadores y los estados y transiciones son partes de máquinas de estado.

elemento generalizable

Este elemento de UML1 se ha retirado en UML2. La mayoría de su semántica se ha trasladado a clasificador. Los clasificadores ahora incluyen asociaciones, de forma que son generalizables. Los comportamientos, incluyendo interacciones y máquinas de estados, también son generalizables.

elemento ligado

Un elemento del modelo que se produce mediante la ligadura de los valores de los argumentos a los parámetros de una plantilla.

Véase también ligadura, plantilla.

Semántica

Una plantilla es una descripción parametrizada de un grupo de elementos potenciales. Para obtener un elemento real, los parámetros de la plantilla se deben *ligar* a los valores reales. El valor real para cada parámetro es una expresión proporcionada por el ámbito dentro del cual se da la ligadura. La mayoría de los argumentos son clases o enteros.

Si el propio ámbito es una plantilla, entonces se pueden utilizar los parámetros de la plantilla más externa en la ligadura de la plantilla original reparametrizándola. Pero los nombres de parámetros de una plantilla no tienen significado fuera de su cuerpo. No se puede asumir que se correspondan los parámetros de dos plantillas simplemente porque tengan el mismo nombre, de la misma forma que tampoco se podría asumir que los parámetros de una subrutina se corresponden basándose únicamente en sus nombres.

Un elemento ligado está completamente especificado por su plantilla. Se pueden declarar atributos adicionales en una clase ligada por comodidad. Esto es equivalente a declarar una clase anónima que realiza la ligadura, para, a continuación, declarar la clase ligada como una subclase suya y añadir nuevos atributos a la clase ligada. *Véase* plantilla para un ejemplo.

Ejemplo

La Figura 14.107 muestra cómo ligar más de una vez de una plantilla. **Polígono** es una plantilla con un único parámetro, el tamaño **n**. Queremos construir la plantilla utilizando la plantilla existente **FArray** que tiene dos parámetros, el tipo de elemento **T** y el tamaño **k**. Para construirla, se liga el parámetro **k** de la plantilla **FArray** al parámetro **n** de la plantilla **Polígono**. El parámetro **T** de la plantilla **FArray** se liga a la clase **Punto**. Esto tiene el efecto de eliminar un parámetro de la plantilla original. Para utilizar la plantilla **Polígono** para generar la clase **Triángulo**, se liga el parámetro de tamaño **n** con el valor 3. Para generar una clase **Cuadrilátero**, se liga con el valor 4.

La Figura 14.107 también muestra a la plantilla **Polígono** como hija de la clase **Figura**. Esto significa que cada clase ligada a partir de **Polígono** es una subclase de **Figura** (**Triángulo** y **Cuadrilátero** son subclases de **Figura**).

Notación

Un elemento ligado se muestra utilizando una flecha de línea discontinua desde la plantilla al elemento ligado, con la palabra clave **«bind»**. De forma alternativa, se puede mostrar utilizando la sintaxis de texto **`NombrePlantilla`** <**`argumento`**$_{\texttt{lista,}}$>, utilizando la correspondencia de nombres para identificar la plantilla. La forma de texto evita la necesidad de mostrar la plantilla o de dibujar una flecha hacia ella. Esta forma es especialmente útil cuando el elemento ligado se utiliza como clasificador para un atributo o un parámetro de una operación.

Véase ligadura para más detalles. La Figura 14.168 muestra un ejemplo.

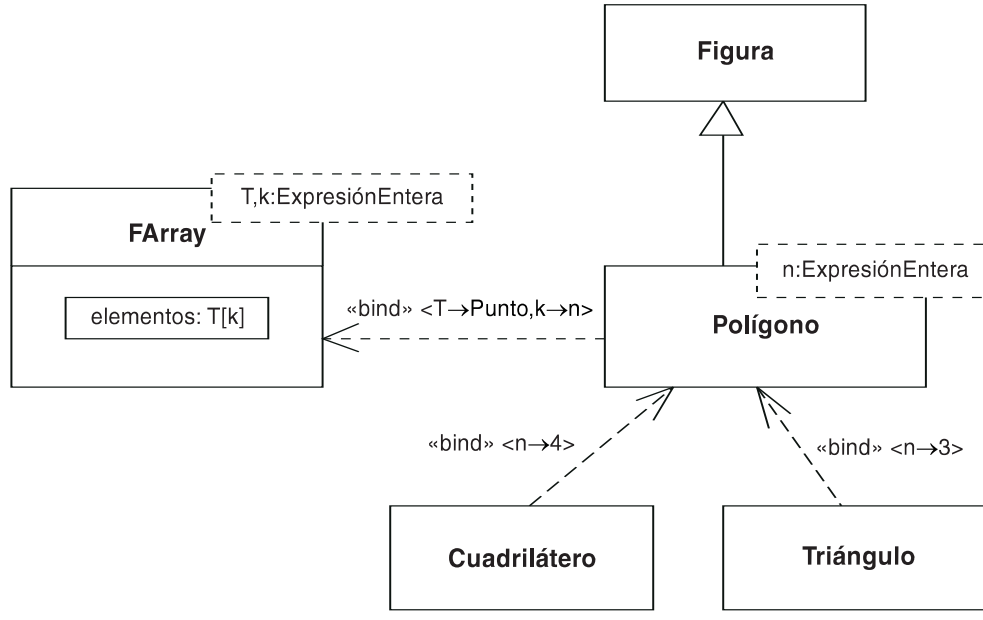


Figura 14.107 Ligado doble de una plantilla

Los compartimentos de los atributos y de las operaciones se suelen suprimir dentro de una clase ligada, ya que no se pueden modificar en un elemento ligado.

Se puede utilizar el nombre de un elemento ligado (tanto de la forma "anónima" de una línea utilizando los símbolos mayor y menor que, como de la forma explícita con la "flecha de ligadura") en cualquier sitio como un nombre de elemento. Por ejemplo, se podría utilizar el nombre de una clase ligada como un tipo de atributo o como parte de la signatura de una operación dentro de un símbolo de clase en un diagrama de clases. La Figura 14.108 muestra un ejemplo.

Discusión

Los clasificadores son candidatos obvios para la parametrización. Los tipos de sus atributos, operaciones o clasificadores asociados son parámetros comunes en las plantillas. Las colabora-

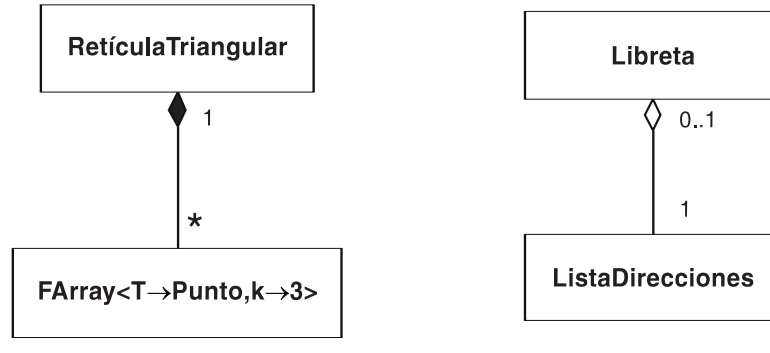


Figura 14.108 Uso de plantillas ligadas en asociaciones

ciones parametrizadas son patrones. Las operaciones, en cierto modo, están inherentemente parametrizadas. Los paquetes se pueden parametrizar. La utilidad de la parametrización de otros elementos no está clara, pero posiblemente se les encontrarán otros usos.

elemento parametrizado

Véase plantilla.

elemento redefinible

Un elemento dentro de un clasificador cuya especificación puede redefinirse dentro de una especialización del clasificador. *Véase* redefinición.

Semántica

Ordinariamente, un elemento definido dentro del contexto de una especificación del clasificador es simplemente heredado por descendientes del clasificador sin modificación. Esta invariabilidad apoya el polimorfismo asegurando que todos los descendientes de una porción del clasificador comparten la misma definición de sus elementos.

A veces, sin embargo, es deseable para poder modificar la definición de un elemento haciéndolo más específico o incluyendo las restricciones en el nuevo contexto. Invariablemente las restricciones en la definición original deben respetar las restricciones, sin embargo.

Pueden redefinirse los tipos siguientes de elementos: borde de actividad, nodo de actividad clasificador, punto de extensión, rasgo, región, estado, transición. La semántica de redefinición varía entre los tipos diferentes de elementos.

La bandera de la hoja indica que un elemento del redefinible no puede redefinirse más allá.

else

Palabra clave que indica una pseudocondición en una bifurcación que es verdadera si y sólo si ninguna otra condición es verdadera. Garantiza que una condicional tendrá al menos una elección válida.

Semántica

La pseudocondición else se puede especificar en cualquiera de las construcciones de bifucarción de UML, incluyendo: (para actividades) nodo condicional, nodo de decisión; (para interacciones) fragmento condicional; (para máquinas de estados) nodo de elección.

Notación

La pseudocondición else se expresa mediante la cadena **[else]**.

emisor

El objeto pasa una instancia de un mensaje a un objeto receptor.

Véase llamada, envío.

enlace

Tupla de referencias a objetos que es una instancia de una asociación o de un conector.

Semántica

Un enlace es una conexión individual entre dos o más objetos. Es una tupla (lista ordenada) de referencias a objetos. Es una instancia de una asociación. Los objetos deben ser instancias directas o indirectas de las clases de la posición correspondiente en la asociación. Una asociación puede contener enlaces duplicados para la misma asociación —es decir, dos tuplas idénticas de referencias a objetos— sólo si uno de los extremos está especificado como bolsa (incluyendo una lista).

Un enlace que es una instancia de una clase asociación puede tener una lista de valores de atributos además de la tupla de referencias a objetos. No están permitidos los enlaces duplicados con la misma tupla de referencias a objetos, incluso si sus valores de atributo son distintos, a no ser que la asociación tenga una bolsa en uno de sus extremos. La identidad de un enlace viene de su tupla de referencias a objetos, que debe ser única.

Un enlace puede ser utilizado para navegación. En otras palabras, un objeto que aparezca en una posición en un enlace puede obtener el conjunto de objetos que aparecen en otra posición. Entonces puede enviarles mensajes (conocido como “enviar un mensaje a través de una asociación”). Este proceso es eficiente si la asociación tiene la propiedad de navegabilidad en la dirección del destino. El acceso puede ser o no ser posible si la asociación no es navegable, pero probablemente sería ineficiente. La navegabilidad en direcciones opuestas se especifica de manera independiente.

Dentro de una colaboración, un conector es una relación contextual, a menudo transitoria, entre clasificadores. Una instancia de un conector también es un enlace, pero típicamente uno cuya vida está limitada a la duración de la colaboración. Dicha asociación transitoria normalmente no se llama asociación; la palabra se reserva normalmente para relaciones no contextuales.

Notación

Hablando con precisión, los objetos y los enlaces aparecen en el “mundo real” y no en los modelos o diagramas. Los modelos de objetos y los enlaces aparecen en algún contexto posiblemente implicado como especificaciones de instancia y conectores. Para comodidad en esta discusión, utilizamos las palabras objeto y enlace para significar los modelos de las entidades reales, pero aparecen implícitamente como partes dentro de un contexto.

Un enlace binario se muestra como una ruta entre dos objetos —es decir, uno o más arcos o segmentos de línea conectados. En el caso de una asociación reflexiva, la ruta es un bucle, con ambos extremos en un solo objeto.

Véase asociación para ver detalles sobre las rutas.

En cada extremo de un enlace se puede mostrar un nombre de rol. Un nombre de asociación puede mostrarse cerca de la ruta. No es necesario subrayar el nombre para indicar una instancia, puesto que una ruta entre dos objetos debe ser también una instancia. Los enlaces no tienen nombres de instancia. Cogen su identidad de los objetos que relacionan. La multiplicidad no se muestra en los enlaces puesto que las instancias no tienen multiplicidad; la multiplicidad es una propiedad de la asociación que limita cuantas instancias pueden existir. Para mostrar multiplicidad, simplemente se muestran varios enlaces desde la misma asociación. (Como siempre, un diagrama de instancia nunca puede mostrar el caso general.) Se pueden mostrar otros adornos de asociación (agregación, composición y navegación) en los roles del enlace.

Se puede mostrar un calificador en un enlace. El valor del calificador se puede mostrar en su caja. La Figura 14.109 muestra tanto enlaces ordinarios como calificados.

Otros adornos sobre los enlaces pueden mostrar propiedades de sus asociaciones, incluyendo la dirección de navegación, agregación o composición, estereotipos de implementación y visibilidad.

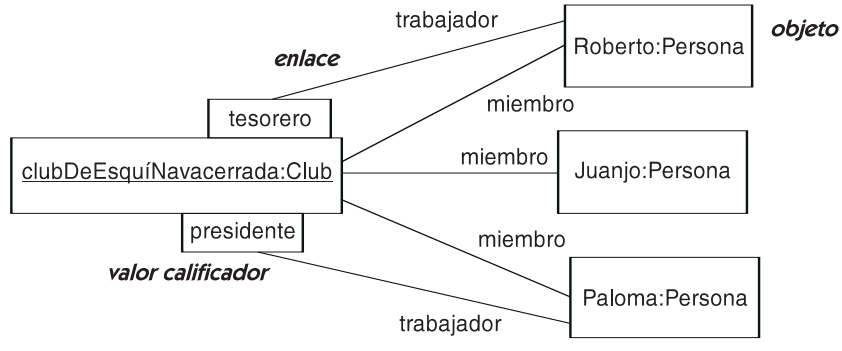


Figura 14.109 Enlaces

Enlace* n-*ario. Un enlace *n*-ario se representa como un diamante con una ruta a cada objeto participante. Los otros adornos de la asociación y los adornos de los roles tienen las mismas posibilidades que el 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 ocurre con los argumentos de los procedimientos, variables locales de los procedimientos, y el llamante de una operación? Deben existir como estructuras de datos reales, no simplemente como dependencias. Por tanto, se deben mostrar como enlaces. El llamante de un procedimiento requiere una referencia al objeto destino —esto es un enlace. Algunos enlaces pueden ser instancias de roles de asociación en colaboraciones, como la mayoría de los parámetros y variables locales. El resto de dependencias son relevantes para la propia clase y no para sus objetos individuales.

enlace transitorio

Enlace que existe por un tiempo limitado, como para la ejecución de una operación.

Véase también asociación, colaboración, uso.

Semántica

Durante la ejecución, algunos enlaces existen durante un tiempo limitado. Por supuesto, casi cualquier objeto o enlace tienen un período de vida limitado, si el período de tiempo es lo bastante grande. Sin embargo, algunos enlaces existen sólo en ciertos contextos limitados, como durante la ejecución de un método. Los argumentos de procedimientos y las variables locales se pueden representar mediante enlaces transitorios. Es posible modelar dichos enlaces como asociaciones, pero luego las condiciones de las asociaciones se deben establecer de forma muy amplia, y pierden mucha de su precisión restringiendo combinaciones de objetos. A cambio dichas situaciones se pueden modelar utilizando colaboraciones, que son configuraciones de objetos y enlaces que existen dentro de contextos especiales.

Un conector desde una colaboración puede ser considerado como un enlace transitorio que existe sólo dentro de la ejecución de una entidad de comportamiento, como un procedimiento. Aparece dentro de un modelo de clases como una dependencia de uso. Para ver los detalles completos, es necesario consultar el modelo de comportamiento.

Notación

Un enlace transitorio se puede representar como un conector dentro de una colaboración que representa la ejecución de un comportamiento.

entity (estereotipo de Componente)

Un componente de información persistente que representa un concepto de negocio.

Discusión

Este estereotipo podría haber sido inspirado por el término *Entity Bean* de la comunidad Enterprise Java Beans (EJB). Sin embargo, es un uso más restringido de una palabra común, y entra en conflicto con la utilización de la palabra *entity* en otras áreas, como en el modelo Entidad Relación familiar en las bases de datos.

entorno de ejecución

Tipo de nodo de despliegue que representa un tipo particular de plataforma de ejecución, como un sistema operativo, un motor de una estación de trabajo, un sistema gestor de bases de datos, etc.

También (y de forma más común) se utiliza de manera informal para describir el contexto en el que se produce la ejecución de un modelo.

Semántica

Un entorno de ejecución modela un entorno hardware y software que proporciona un conjunto de servicios estándar para su uso por componentes de aplicación. Los ejemplos incluyen sistemas operativos, sistemas gestores de bases de datos, motores de flujo de trabajo, etc. Los componentes se pueden desplegar en el entorno de ejecución. El entorno de ejecución normalmente forma parte de otro nodo que modela el hardware.

Notación

Un entorno de ejecución se representa como un símbolo de nodo (la imagen de un cubo) con la palabra clave **«executionEnvironment»**.

enumeración

Tipo de datos cuyas instancias forman una lista de valores literales con nombre. Normalmente se declaran tanto el nombre de la enumeración como sus valores literales.

Véase también clasificador, tipo de datos.

Semántica

Una enumeración es un tipo de datos definido por el usuario. Tiene un nombre y una lista ordenada de valores literales enumerados, cada uno de los cuales es un valor en el rango del tipo de datos —es decir, una instancia predefinida del tipo de datos.

Por ejemplo, **ColorRGB = { rojo, verde, azul}**, en el que el tipo de datos es **ColorRGB** y sus posibles valores son **rojo, verde** y **azul**. El tipo de datos **Boolean** es una enumeración predefinida con los literales **false** y **true**.

Los literales de enumeración se pueden comparar por igualdad o por posición relativa dentro de la lista de literales. Son atómicos y no tienen subestructura. El tipo enumerado puede definir operaciones que tengan argumentos literales y devuelvan literales como resultado. Por ejemplo, las distintas operaciones lógicas se definen sobre los valores **false** y **true**.

Notación

Una enumeración se representa con un rectángulo con la palabra clave «**enumeration**» encima del nombre del tipo de enumeración, en el compartimento superior (Figura 14.110). El segundo

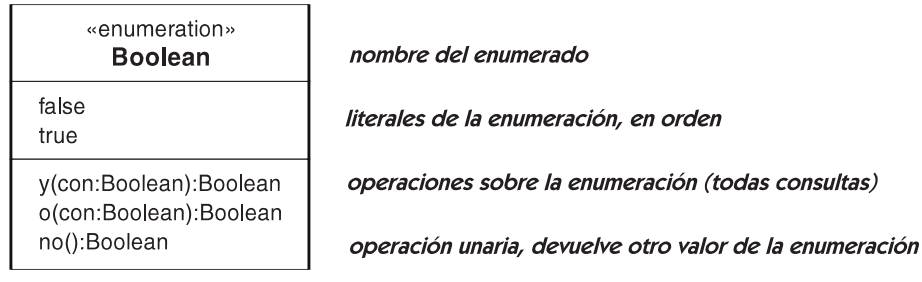


Figura 14.110 Declaración de una enumeración

compartimento contiene una lista de los nombres literales de la enumeración. El tercer compartimento (si existe) contiene un conjunto de operaciones sobre el tipo. Todas deben ser consultas (y por tanto no hace falta declararlas explícitamente como tales).

enviar

Para crear un mensaje que contiene un objeto (a menudo una señal) creado por un objeto del remitente y para transferirlo a un objeto del receptor para llevar información.

Véase también acción, mensaje, signo.

Semántica

Una acción de envío es una acción que un objeto puede realizar como parte de la ejecución de un comportamiento. Especifica una señal para enviar, una lista de argumentos o un objeto para el valor de la señal y un objeto designado para recibir la señal.

Un objeto envía un objeto a otro objeto. En muchos casos, el objeto es una señal pero puede enviarse cualquier clase de objeto. El remitente puede empotrar la acción de envío en un bucle o región de expansión para enviar un mensaje a un conjunto de objetos. Una acción de la transmisión envía un mensaje al conjunto de objetos, aunque, en la práctica, el conjunto de objetos destino pueda ser dependiente de la implementación y pueda variar de nodo a nodo.

Una acción de envío es asíncrona. Una vez que el mensaje se lanza hacia el objeto destino, el remitente es libre para continuar la ejecución concurrentemente con la transmisión y manejo subsecuente del mensaje por el objeto destino. No hay información de retorno como parte del envío de un mensaje, porque el remitente continúa la ejecución y su estado es desconocido. Una acción de llamada debe usarse en situaciones donde el objeto enviado necesita bloquearse hasta que se reciba un valor de retorno. El receptor de un mensaje puede enviar un mensaje después al objeto enviado original si tiene un enlace al objeto, pero no hay ninguna relación inherente entre los dos mensajes.

Los mensajes son enviados por valor. Un parámetro de envío puede ser sólo un parámetro “in”.

Una dependencia de uso de envío es un estereotipo de una dependencia de uso de la acción u operación que envía la señal al tipo de la señal.

Crear como envío

La creación de un nuevo objeto puede considerarse (conceptualmente) como enviar un mensaje a un objeto fábrica, tal como una clase, la cual crea la nueva instancia y entonces le pasa el mensaje como su “evento del nacimiento”. Esto provee un mecanismo a un creador para comunicarse con su creación. El evento de nacimiento puede considerarse como ir del creador al nuevo objeto, con el efecto lateral de inmediato del nuevo objeto a lo largo de la manera. La Figura 14.111 muestra la creación de un objeto que usa la sintaxis del texto y sintaxis gráfica. Éste es un uso informal de la acción de envío y no oficialmente aprobado.

Este modelo puede usarse aún cuando el lenguaje designado, como C++, no soporta clases como objetos en tiempo de ejecución. En ese caso, la acción de creación se compila (lo cual

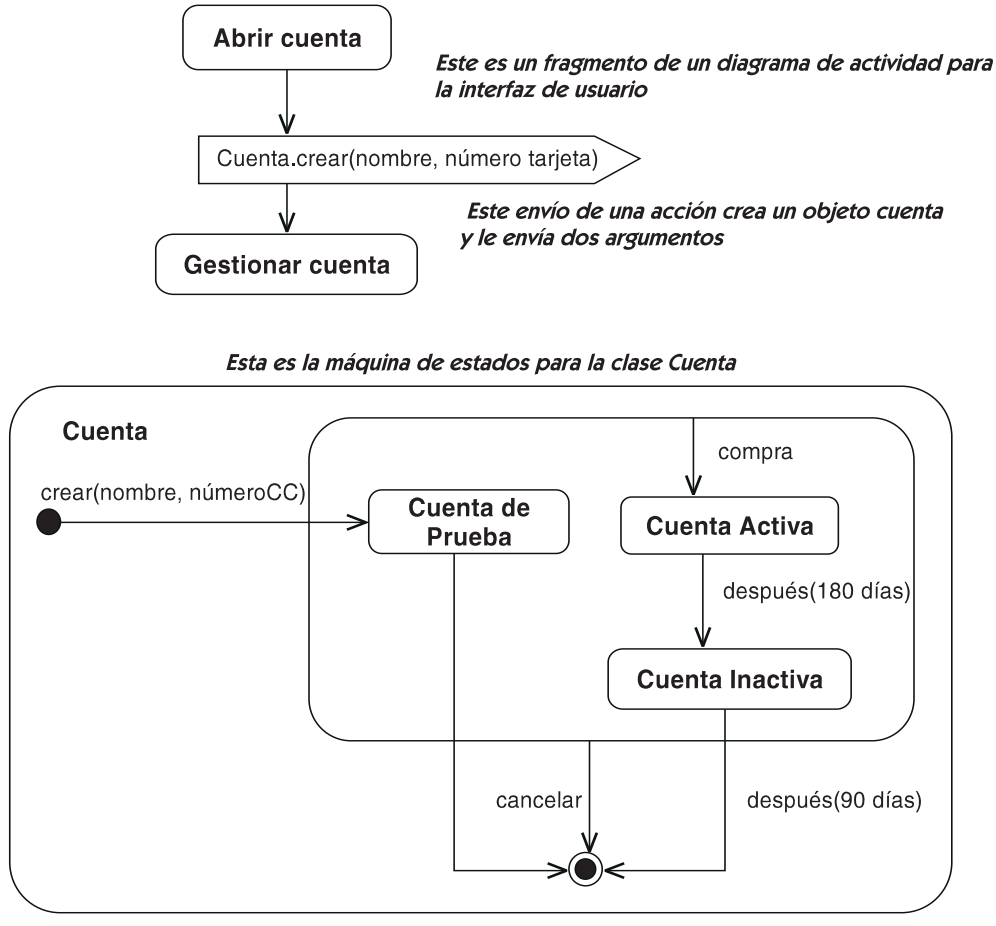


Figura 14.111 Creación de un objeto nuevo mediate el envío de un mensaje

impone algunas restricciones en su generalidad —por ejemplo, el nombre de la clase debe ser un valor literal) pero el intento subyacente es el mismo.

Notación del texto

Dentro de una transición, el envío de un signo es una acción en la que puede expresarse una expresión específica del lenguaje. Para los ejemplos en este libro, usamos la sintaxis:

Send **`objeto-destino.nombre-señal`**(**`argumento`**$_{\texttt{lista,}}$)

Ejemplo

Esta transición interna selecciona un objeto dentro de una ventana que usa la posición del cursor y, a continuación, le envía un signo del momento culminante. La sintaxis es informal:

```
presionar-boton-derecho-raton (posición)[posición in ventana]
/objecto:=seleccionar-objeto (posición);send objecto.resaltar ()
```

Notación del diagrama

Hay una sintaxis gráfica para mostrar una acción de envío en una máquina de estados o en un diagrama de actividad. Un pentágono convexo que contiene el nombre del signo puede ser incluido como parte de una cadena de acciones conectadas por flechas de la transición (Figura 14.112).

Figura 14.112 Notación de enviar acción

En la especificación de UML, no se da ninguna sintaxis para especificar el objeto designado, pero en este libro usamos la sintaxis:

`objeto-destino.nombre-señal``(`**`argumento`**$_{\texttt{lista,}}$`)`

No hay ninguna notación gráfica oficial para mostrar el objeto designado, pero sugerimos seguir el uso de la notación de la dependencia: El envío de un mensaje entre máquinas de esta-

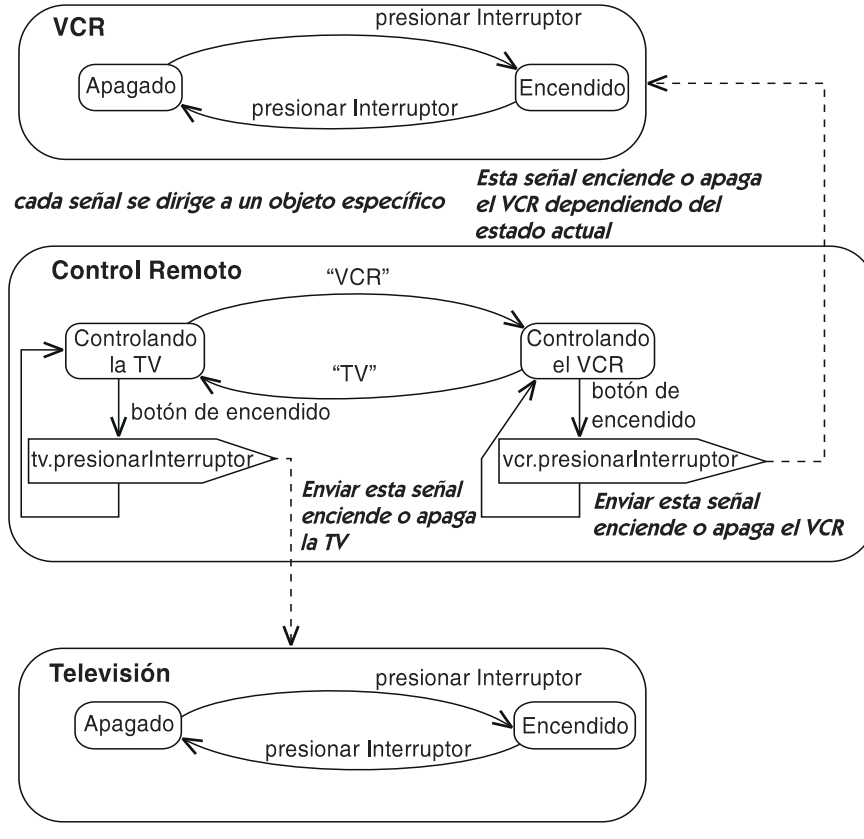


Figura 14.113 Envío de señales entre objetos

dos puede mostrarse dibujando una flecha de dependencia discontinua de la acción de envío a la máquina de estado receptora. La Figura 14.113 contiene diagramas de estado que muestran el envío de señales entre tres objetos.

Observe que esta notación también puede usarse en otros tipos de diagramas para mostrar el envío de eventos entre clases u objetos.

En un diagrama de secuencia, el envío de un mensaje se muestra dibujando una flecha de línea de vida del objeto enviado a la línea de la vida del objeto receptor. La punta de flecha está abierta. Se pone el nombre y argumentos del signo cerca de la flecha. *Véase* mensaje para ejemplos.

escenario

Una secuencia de acciones que ilustran el comportamiento. Un guión puede usarse para ilustrar una interacción o la ejecución de una ocurrencia de un caso de uso.

espacio de nombres

Parte del modelo en la que pueden definirse nombres y pueden usarse. Dentro de un espacio de nombres, cada nombre tiene un significado particular.

Semántica

Todos los elementos con nombre se declaran en un espacio de nombres y sus nombres tienen el alcance dentro de él. Los espacios de nombres de mayor nivel son los paquetes, contenedores cuyo propósito es agrupar elementos principalmente con el objetivo de facilitar el acceso humano y su comprensión, también tienen por objetivo organizar los modelos para el almacenamiento en la computadora y su manipulación durante el desarrollo. Los elementos primordiales del modelo, incluyendo clases, asociaciones, máquinas de estados, y colaboraciones, actúan como espacio de nombres para sus elementos contenidos, tales como atributos, extremos de asociación, estados y roles de la colaboración. El alcance de cada elemento del modelo se discute como parte de su descripción. Cada uno de estos elementos del modelo tiene su propio espacio de nombres distinto.

Los nombres definidos dentro de un espacio de nombres deben ser únicos para el tipo del elemento dado. Algunos elementos son anónimos y deben ser localizados mediante la relación con los elementos con nombre. Pueden anidarse espacios de nombres. Es posible realizar búsquedas dentro de un espacio de nombres anidados a partir de un nombre.

Dentro de un espacio de nombres, puede haber múltiples grupos de elementos del mismo tipo o de tipos similares. Los nombres de elementos dentro de un grupo deben ser únicos, pero el mismo nombre puede reutilizarse en un grupo diferente. La formación de los subgrupos es un poco desigual. En cualquier caso, el uso de nombres repetidos dentro de un solo espacio de nombres es peligroso, incluso si se tiene en cuenta que pueden ser distinguidos por tipo del elemento.

El concepto de unicidad puede variar por tipo. Por ejemplo, las operaciones en algunos lenguajes de programación son identificados por su prototipo, incluyendo el nombre de la opera-

ción y los tipos de sus parámetros. El prototipo es el nombre efectivo dentro del espacio de nombres de la operación.

Para acceder a nombres de otro espacio de nombres, un paquete puede acceder o puede importar otro paquete.

El propio sistema define el espacio de nombres extremo que provee la base a todos los nombres absolutos. Es un paquete, normalmente con paquetes anidados dentro de él a varios niveles hasta que se obtengan elementos primitivos finalmente. La lista de nombres desde la raíz hasta un elemento particular se conoce como el nombre calificado; identifica un elemento particular.

Notación

La notación para un nombre calificado, un camino a través de varios espacios de nombres anidados, se obtiene concatenando los nombres de los espacios de nombres (como paquetes o clases) separados mediante un par de dos puntos (::).

```
UserInterface::HelpFacility::HelpScreen
```

especialización

Producir una descripción más específica de un elemento del modelo agregando los rasgos, relaciones, restricciones, y otros detalles a un elemento del modelo original. La relación opuesta es generalización, la cual es usada también como nombre de la relación entre el elemento más específico y el elemento más general, ya que no hay ningún término bueno para la relación indirecta. Un elemento hijo es la especialización de un elemento padre. Recíprocamente, el padre es la generalización del hijo.

Véase generalización.

especificación de despliegue

Especificación detallada de los parámetros del despliegue de un artefacto en un nodo. *Véase* despliegue.

especificación de instancia

Descripción en un modelo de una instancia o un grupo de instancias. La descripción puede o no describir todos los detalles de la instancia.

Semántica

Una especificación de instancia es la descripción de una instancia individual en el contexto de su participación en un sistema y su relación con otras instancias. A diferencia de una instancia, que existe sólo como un individuo concreto en un sistema en ejecución, una especificación de instancia puede ser más o menos precisa. En un extremo, puede describir un solo objeto en una sola ejecución de un sistema con gran detalle. Más a menudo, describe varios aspectos de la ins-

tancia en ejecución y omite otros que no son de interés en una determinada vista. También puede representar varios objetos sobre varias ejecuciones de un escenario particular, de forma que el tipo de una especificación de instancia puede ser abstracto, aunque cada objeto en ejecución tiene necesariamente un tipo concreto. Recuerde que la especificación de instancia, como descripción, puede ser una abstracción de instancias reales.

Una especificación de instancia es tan parecida como poco parecida a un clasificador. Como un clasificador, restringe las instancias que describe. A diferencia de un clasificador, describe una instancia individual y puede tener una relación contextual con otras instancias y con el sistema en el que está embebida.

Estructura

clasificador	Clasificador o clasificadores que caracterizan a la instancia. El tipo de la instancia puede ser el mismo o un descendiente. Si se dan varios clasificadores, el tipo de la instancia debe ser un descendiente de todos ellos. Los clasificadores pueden ser abstractos. La instancia también puede tener clasificadores además de los listados, a menos que se prohíba de forma explícita.
especificación	Descripción opcional de cómo calcular, construir o derivar la instancia. Puede ser en un lenguaje formal o en uno informal. También puede ser completa o incompleta (y por tanto abstracta). Normalmente la especificación y los valores de las ranuras son mutuamente excluyentes.
ranuras	Puede haber una ranura para cada atributo del clasificador o clasificadores, pero no todo atributo debe ser incluido (en cuyo caso la descripción es abstracta). Cada ranura contiene una especificación de valor para el atributo dentro de la instancia. Una especificación de valor puede ser un valor literal, un árbol de expresión, una cadena de expresión u otra especificación de instancia.

Notación

Aunque los clasificadores y las especificaciones de instancia no son lo mismo, comparten muchas propiedades, incluyendo la misma forma (puesto que el descriptor debe describir la forma de la instancia). Por tanto, es conveniente elegir para cada par clasificador-instancia de forma que la correspondencia sea obvia visualmente. Hay un número limitado de formas de hacerlo, cada una con sus ventajas y sus inconvenientes. En UML, la distinción clasificador-instancia se representa utilizando el mismo símbolo gráfico para cada par de elementos y subrayando la cadena de nombre de un elemento de instancia. Esta distinción visual es obvia sin ser abrumadora incluso cuando un diagrama entero contiene elementos de instancia.

Para una especificación de instancia, el nombre del clasificador se reemplaza por una cadena de nombre de instancia con la siguiente forma:

nombre : tipo$_{\text{lista,}}$ $\lfloor$**=especificación**$\rfloor_{\text{opc}}$

El nombre es el nombre de la instancia. Se puede omitir, pero los dos puntos deben estar presentes. Puede aparecer o no hacerlo el nombre del clasificador o una lista de nombres de clasifi-

cadores. La especificación es opcional. Se proporciona como una expresión, cuya forma depende de la implementación. Normalmente no aparece si los valores se dan por ranuras. La especificación también puede ser listada en su propia línea debajo de la cadena de nombre. En dicho caso, se omite el signo igual.

Además del compartimento de nombre, una especificación de instancia puede tener un compartimento de ranuras de atributos que corresponde con el compartimento de atributos de un clasificador. Se puede incluir una ranura por cada atributo (incluidos los heredados) de cualesquiera de los tipos dados. Una especificación de ranura tiene la forma:

`nombre` ⌊**`:tipo`**⌋$_{\text{opc}}$ = **`valor`**

El nombre es el nombre del atributo. Por comodidad, se puede incluir el tipo del atributo, pero no es necesario ya que se conoce del clasificador. El valor se proporciona como una expresión de texto.

El compartimento de operaciones no forma parte de una especificación de instancia, puesto que todos los objetos de un tipo dado comparten las mismas operaciones especificadas en su clasificador.

Aunque la Figura 14.114 muestra especificaciones de objetos, se puede utilizar la convención de subrayado para otros tipos de instancias, como instancias de casos de uso, instancias de componentes e instancias de nodo.

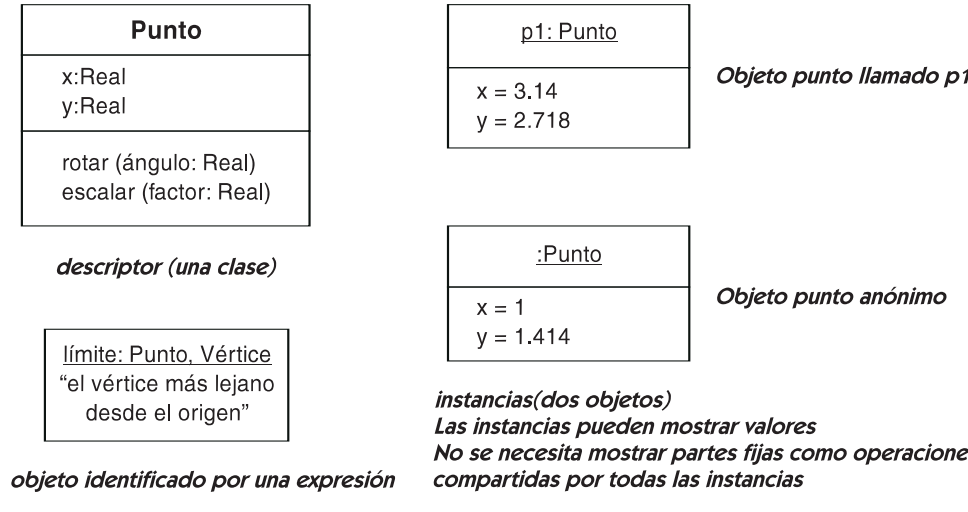


Figura 14.114 Descriptor e instancias

Las especificaciones de enlace (instancias de asociación) se muestran como caminos continuos que conectan símbolos de objetos. No es necesario subrayar los nombres puesto que una línea que conecta dos símbolos de objeto automáticamente es una especificación de enlace. Un símbolo de enlace puede tener los mismos adornos que una asociación (agregación, navegabilidad, nombres de extremo de asociación). La multiplicidad no es apropiada puesto que una especificación de enlace conecta especificaciones de instancia individuales. Puede haber varios

símbolos de enlace con el mismo nombre de asociación o extremo. Si los enlaces están ordenados, se puede dibujar a lo largo de los símbolos de enlace en orden una flecha discontinua con la palabra clave **ordered**; de lo contrario se pueden numerar los símbolos de enlace o los extremos de los enlaces. En principio se pueden ordenar ambos extremos de una asociación, pero es difícil de utilizar y difícil de mostrar; utilícelo lo menos posible.

Puesto que las instancias aparecen en modelos como especificaciones de instancia, normalmente sólo se incluyen detalles relevantes para un ejemplo particular. Por ejemplo, no se necesita incluir la lista entera de valores de atributos; o incluso se puede omitir la lista entera de valores si el enfoque está en otra cosa, como un flujo de mensajes entre objetos.

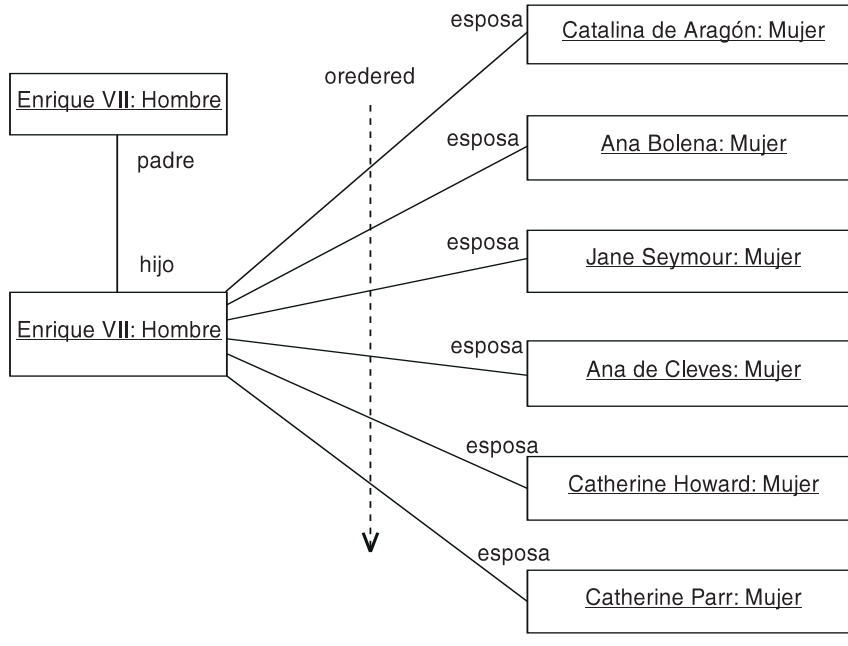


Figura 14.115 Especificación de enlace

Instancia de clase estructurada. Una instancia de una clase estructurada se representa utilizando un rectángulo con un compartimento gráfico. Para mostrar una instancia de una parte contenida, se dibuja un rectángulo. La cadena de nombre tiene la sintaxis:

nombre ⌊/ **nombre-de-rol**⌋$_{\texttt{lista}}$ **:** ⌊**nombre-de-tipo**⌋$_{\texttt{lista}}$,

Los diversos elementos son opcionales (incluyendo los dos puntos si se omite el tipo).

Un conector se dibuja como un camino continuo entre partes rectangulares. El nombre del conector (si existe) va subrayado.

Los valores de atributos para una parte pueden mostrarse en un compartimento de atributos con la sintaxis:

nombre = **valor**

La Figura 14.116 muestra una instancia de un clasificador estructurado.

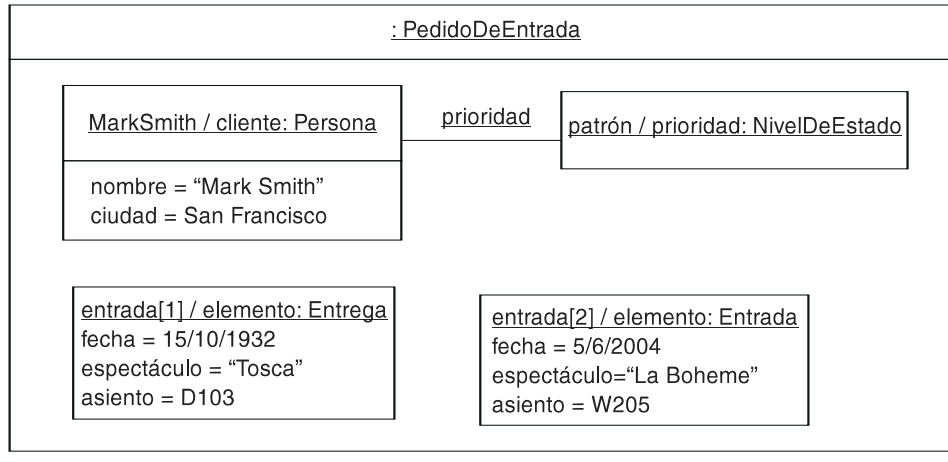


Figura 14.116 Instancia de una clase estructurada

Historia

En UML2, se ha hecho una clara distinción entre especificaciones de instancia e instancias, una diferencia sutil pero importante cuya ausencia origina una confusión semántica interminable en UML1.

especificación de objeto

El uso de un objeto de un tipo dado dentro de un contexto.

Semántica

Un objeto es un individuo único. Por consiguiente, raramente pueden usarse directamente objetos dentro de modelos o programas que representan modelos reusables aplicables a muchos objetos. Una clase describe muchos objetos con características similares, pero no tiene contexto. Una especificación de objeto es la descripción del uso de un objeto de un dado dentro de un contexto reusable particular. Por ejemplo, el valor de un atributo puede ser un objeto de una cierta clase. El atributo no tiene un objeto fijo como su valor; cada instancia que contiene el atributo tiene un objeto diferente como su valor. El valor del atributo es por consiguiente una especificación del objeto; describe a una familia de objetos dentro de un contexto específico, una instancia que contiene el atributo.

Una especificación del objeto es un tipo de especificación de valor.

especificación de una ejecución

La especificación de la ejecución de una actividad, operación u otra unidad de comportamiento dentro de una interacción. Una ejecución (a veces conocida como foco de control) representa el

periodo durante el que un objeto realiza un comportamiento, o bien a través de un comportamiento subordinado. Modela tanto la duración de la ejecución como la relación de control entre la ejecución y sus invocadores. En una computadora y lenguaje convencionales, la propia ejecución corresponde a un valor de la pila.

Véase llamada, diagrama de secuencia.

Semántica

La especificación de una ejecución modela la ejecución de un comportamiento u operación, incluyendo el periodo durante el que una operación llama a otras operaciones subordinadas (*véase* llamada). La especificación de una ejecución tiene dos eventos asociados, que representan su inicio y su finalización. Normalmente el evento de inicio es el destino de un mensaje de invocación, y el evento de finalización puede ser el origen de un mensaje de retorno.

Notación

La especificación de una ejecución se representa en un diagrama de secuencia como un fino y largo rectángulo (una barra hueca vertical), cuya parte superior está alineada con su evento de inicio y cuya parte inferior está alineada con su evento de finalización (Figura 14.117). El comportamiento que se lleva a cabo se representa mediante una etiqueta de texto cerca del símbolo de la especificación de la ejecución o en el margen izquierdo, dependiendo del estilo. Alternativamente, el símbolo de mensaje entrante puede indicar el comportamiento. En ese caso, la etiqueta se puede omitir en la propia especificación de la ejecución. Si el flujo de control es

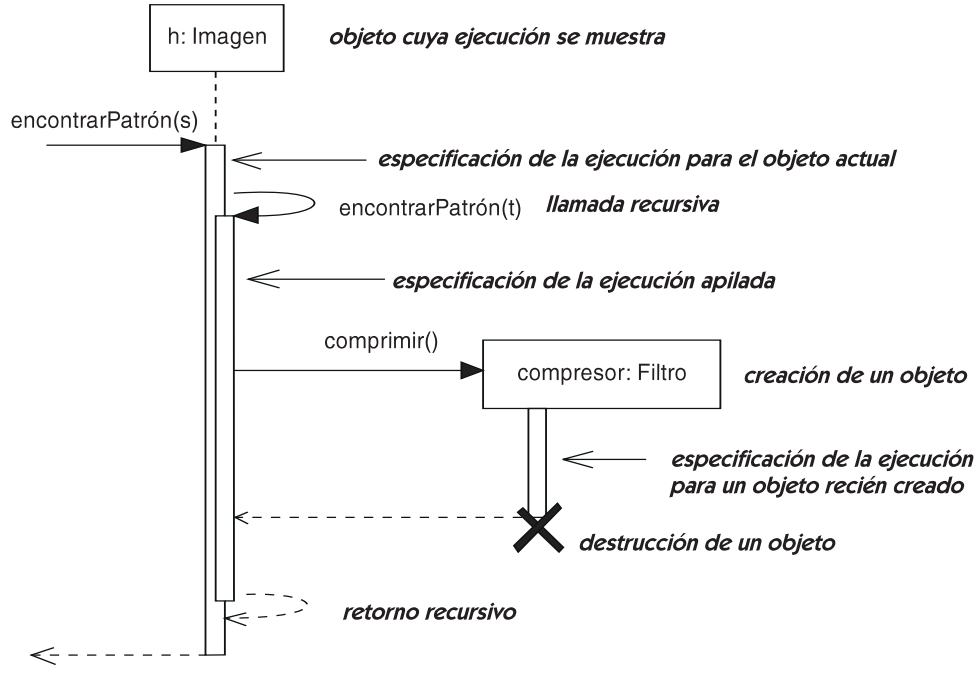


Figura 14.117 Especificaciones de ejecución

procedimental, entonces la parte superior del símbolo de la especificación de la ejecución está en el extremo de la flecha del mensaje entrante que inicia la ejecución de la operación y la parte inferior del símbolo se encuentra en la cola de la flecha del mensaje de retorno.

Si varios objetos tienen actividad concurrente, entonces cada especificación de la ejecución muestra la ejecución de un objeto concurrente. A no ser que los objetos se comuniquen, las ejecuciones concurrentes son independientes y sus tiempos relativos de ejecución son irrelevantes.

En el caso de código procedimental, una especificación de ejecución muestra la duración durante la que un procedimiento está activo en el objeto o un procedimiento subordinado llamado por el procedimiento original está activo, posiblemente en algún otro objeto. En otras palabras, se muestran simultáneamente todas las especificaciones de ejecución anidadas activas. Este conjunto de ejecuciones anidadas simultáneas corresponden a la pila de computación en una computadora convencional. En el caso de una segunda llamada a un objeto con una especificación de ejecución existente, el símbolo de la segunda especificación de ejecución se dibuja ligeramente a la derecha del primero, de forma que aparezcan visualmente "apilados". Las llamadas apiladas pueden anidarse a una profundidad arbitraria. Las llamadas pueden ser a la misma operación (llamada recursiva) o a otras operaciones diferentes sobre el mismo objeto.

Es posible (aunque a menudo innecesario) distinguir el periodo durante el que una especificación de ejecución está bloqueada puesto que ha llamado a una operación subordinada. El rectángulo puede colorearse de negro cuando la especificación de la ejecución representa ejecución directa y en blanco cuando representa una llamada a una operación subordinada. Esto no es notación oficial, pero la notación tolera dos colores para ciertas razones no fijadas, de forma que los modeladores pueden adoptar sus propias convenciones.

Discusión

Se llamaba *ocurrencia de ejecución*, pero dicho término entra en conflicto con el patrón general de nombrado de que *especificación* califica el nombre de una entidad de tiempo de ejecución cuando describe un grupo de entidades dentro de un contexto.

especificación de valor

La especificación de un valor en un modelo.

Semántica

Una especificación de valor no es un valor, sino el modelo de un valor. A diferencia de un valor real, que debe ser concreto y específico, una especificación de valor puede ser más o menos precisa. Puede especificar un valor específico, pero también puede especificar un rango de valores o incluso valores de tipos diferentes.

Las especificaciones de valor toman distintas formas en UML, incluyendo expresiones de prueba.

especificación del suceso

La especificación de la ocurrencia de un evento durante la ejecución de un sistema.

Semántica

Una especificación de una ocurrencia es un elemento ejemplar que representa un conjunto de ocurrencias potenciales dentro de un contexto específico. Una ocurrencia es un único individuo, considerando que una especificación de la ocurrencia representa muchas ocurrencias. Una especificación de la ocurrencia tiene una situación dentro de una sucesión, por consiguiente es más específico que un evento que describe todas las ocurrencias de un tipo dado sin localizarlos dentro de una sucesión. Las características técnicas de la ocurrencia se modelan dentro de las interacciones. Una línea de la vida representa una sucesión de especificaciones de la ocurrencia. El modelo del envío o la recepción de un mensaje por un objeto es una especificación de la ocurrencia, por consiguiente un mensaje conecta dos especificaciones de la ocurrencia en dos líneas de la vida. La ejecución de una actividad puede ser modelada por dos especificaciones de la ocurrencia (el principio y fin de la ejecución) en la misma línea de la vida.

Notación

Una especificación de la ocurrencia normalmente no se muestra explícitamente como un concepto separado. Normalmente se muestra por la intersección de una flecha del mensaje y una línea de vida. Una especificación de una ocurrencia aislada puede mostrarse en una línea de la vida utilizando un tic pequeño la línea de la vida y etiquetándolo con texto.

Véase diagrama de secuencia para los ejemplos.

Discusión

La especificación de la ocurrencia se llamó la ocurrencia de evento, pero ese término no distingue la entidad tiempo de ejecución (ocurrencia) del elemento ejemplar (especificación).

La nueva terminología tiene la correspondencia:

objeto: especificación de objeto: clase:: ocurrencia: especificación de la ocurrencia: evento.

especificador de interfaz

Esta construcción UML1 está obsoleta en UML2.

estado

Una condición o situación durante la vida de un objeto, durante la cual se satisface alguna condición, ejecuta alguna actividad a realizar o espera por algún evento.

Véase también acción, actividad, estado compuesto, hacer actividad, actividad de entrada, actividad de salida, transición de estado, transición interna, pseudoestado, máquina de estados, submáquina de estados, transición.

Semántica

Un objeto pasa por una serie de estados durante su vida. Cuando un objeto satisface la condición de un estado, se dice que el estado es activo. La configuración de estado activo es el conjunto de estados que están activos al mismo tiempo en cualquier punto. Si contiene más de uno entonces hay concurrencia dentro del objeto. El número de estados activos puede cambiar durante la vida de un objeto, debido a las bifurcaciones y uniones de control.

Un objeto permanece en un estado durante un tiempo finito (no instantáneo). Pueden introducirse estados tontos por conveniencia, los cuales realizan acciones triviales y de salida. Pero éste no es el propósito principal de los estados, y los estados tontos pueden, en principio, ser eliminados, aunque son útiles para evitar duplicidad.

Los estados están contenidos en una máquina de estados que describen cómo la historia de un objeto evoluciona con el tiempo, en respuesta a los eventos. Cada máquina de estados describe el comportamiento de los objetos de una clase. Cada clase puede tener una máquina de estados. Una transición describe la respuesta de un objeto en un estado, a la ocurrencia de un evento: El objeto ejecuta una actividad optativa enlazada a la transición y cambia a un nuevo estado. Cada estado tiene su propio conjunto de transiciones.

Una actividad puede enlazarse a una transición. Cuando el disparador de la transición se satisface, la transición se dispara y la actividad se ejecuta, entonces el estado de la fuente se desactiva y el estado designado se activa. Una actividad puede contener actividades anidadas, y estar finalmente compuesta de acciones atómicas.

Una actividad de continuación puede asociarse con un estado. La actividad *hacer* se ejecuta con tal de que el estado esté activo. Alternadamente, la actividad continuada puede planearse por un par de acciones, una actividad de entrada que inicia el *hacer* la actividad en entrada al estado y una actividad de salida que termina el *hacer* y ejecuta la actividad de salida del estado.

Un estado puede ser un estado simple, un estado compuesto o un estado de una submáquina de estados. Un estado simple no tiene ningún subestado anidado. Un estado compuesto contiene una o más regiones, cada una de las cuales tiene uno o más subestados anidados. Un estado de submáquina tiene una referencia a una definición de la máquina de estado que se extiende conceptualmente en lugar de la submáquina de estado.

Pueden agruparse estados en estados compuestos. Cualquier transición en un estado compuesto se aplica a todos los estados anidados dentro de él, para que eventos que afectan muchos subestados puedan ser planeados por una sola transición. Un estado compuesto tiene una o más regiones, cada una de las cuales contienen uno o más subestados. Si un estado compuesto está activo, cada una de sus regiones está activa. Un estado compuesto puede ser no ortogonal u ortogonal.

Un estado no ortogonal tiene una sola región. Sólo un subestado directo de un estado no ortogonal es a la vez activo. Un subestado directo de cada región de un estado ortogonal es concurrentemente activo. Un estado ortogonal se modela concurrentemente.

Para promover la encapsulación, un estado compuesto puede contener estados iniciales y estados finales. Éstos son pseudoestados, cuyo propósito es ayudar a la estructura de la máquina de estado. Una transición al estado compuesto es equivalente a una transición al inicio del estado de cada una de sus regiones, pero el estado puede usarse externamente sin conocimiento de su estructura interior.

Una transición a un estado final de una región de un estado compuesto representa la realización de actividad en la región. En un estado ortogonal, cada una de sus regiones debe alcanzar su estado final para la ejecución de todo el estado para estar completo. El hecho de completar una actividad en el disparador de un estado compuesto ha de completar una transición de finalización en el estado terminal para ser disparada. Una transición de finalización es una transición con un evento disparador no explícito (o, más precisamente, uno con el evento de finalización como su disparador implícito, aunque no sea explícitamente modelado). La realización del estado más externo de un objeto corresponde a su muerte.

Si un estado es un estado ortogonal, entonces todas sus regiones deben completarse antes de que el evento de la realización en el estado compuesto ocurra. En otros términos, una transición de finalización de un estado coexistente compuesto representa una unión de control de todos sus subhilos concurrentes. Espera por todos ellos para completar antes de proceder.

En una máquina de estado protocolar, un estado representa el intervalo invariable entre la ejecución de operaciones. La actividad de un estado indica las operaciones que pueden legalmente ser llamadas en ese punto del protocolo.

Estructura

Un estado tiene las partes siguientes.

Nombre. El nombre del estado, que debe ser único dentro del estado incluido.

El nombre puede omitirse, produciendo un estado anónimo. Cualquier número de estados anónimos distintos puede coexistir. Un estado anidado puede identificarse por su nombre calificado (si todos los estados adjuntos tienen nombres).

Subestados. Si una máquina de estado ha anidado la subestructura, se llama un estado compuesto.

Un estado compuesto contiene una o más regiones, cada una de las cuales contiene uno o más subestados directos. Un estado sin la subestructura (excepto las posibles acciones internas) es un estado simple. Un estado que hace referencia a la definición de otra máquina de estado es una submáquina de estado.

Actividades de entrada y de salida. Un estado puede tener una actividad de entrada y una actividad de salida.

El propósito de estas actividades es encapsular el estado para que pueda usarse externamente sin conocimiento de su estructura interna. Una actividad de entrada se ejecuta cuando el estado se escribe, después de que cualquier actividad enlazada a la transición entrante y antes de cualquier actividad interior. Una acción de salida se ejecuta cuando el estado se termina, después de la realización de cualquier actividad interna y antes de cualquier actividad enlazada a la transición saliente. En una transición que cruza algunos límites de estado, varias acciones de entrada y de salida pueden ejecutarse en modo anidado. Primero, se ejecutan actividades de salida, empezando con el estado más profundo y progresando hacia el estado extremo; entonces la actividad en la transición se ejecuta; y, a continuación, las actividades de entrada son ejecutadas, empezando por el extremo y acabando con el más profundo.

La Figura 14.118 muestra el resultado de disparar una transición por los límites de estado. Las actividades de entrada y de salida no pueden ser evadidas por cualquier medio, incluyendo la ocurrencia de excepciones. Proveen un mecanismo de encapsulación para la especificación de

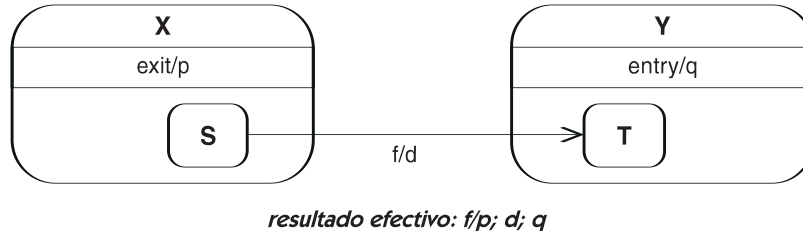


Figura 14.118 Transición entre dos estados límites, con acciones de salida y entrada

comportamiento de la máquina de estado, con una garantía de que se realizarán la actividades necesarias bajo todas las circunstancias.

Realización de actividad interna. Un estado puede contener un *haga la actividad interior* descrito por una expresión.

Cuando el estado se escribe, el *haga la actividad* empieza después de que la actividad de la entrada se completó. Si el *haga la actividad* termina, el estado está completo. Una transición de la realización es parte del estado que se activa. Por otra parte, el estado espera por una transición disparada para causar un cambio de estado. Si una transición dispara mientras el *haga la actividad* está realizándose, el *haga la actividad* se termina y la actividad de salida en el estado se ejecuta.

Transiciones interiores. Un estado puede tener una lista de transiciones interiores que parecen transiciones normales sólo que no tienen los estados designados y no causan un cambio de estado. Si su evento ocurre mientras un objeto está dentro del propio estado, la transición o un subestado anidado de él, entonces la acción en la transición interior se ejecuta pero ningún cambio de estado ocurre, ni se ejecutan acciones de entrada o acciones de salida. No hay ningún cambio de estado aún cuando el estado activo se anida dentro del estado que contiene la transición interior.

Esto lo diferencia de una transición-propia, planeada como una transición externa de un estado al mismo estado. En este caso, si el estado activo es un subestado del estado con la misma-transición, el estado final es el estado designado de la transición que fuerza la ejecución de cada actividad de la salida de los estados anidados dentro del estado con la misma-transición, la ejecución de su actividad de salida, y la ejecución de su actividad de entrada. Las acciones incluso se ejecutan en una misma-transición al estado actual que se termina y, a continuación, vuelve a entrar. Si una misma-transición es un estado adjunto activo actual, entonces el estado final es el propio estado adjunto, no el estado corriente. En otros términos, una misma-transición puede forzar la salida de un estado anidado, pero una transición interna no lo hace.

Submáquina. El cuerpo de un estado puede representar una copia de una máquina de estado separada referenciada por nombre. La máquina de estado referenciada se llama submáquina porque se anida dentro de la máquina de estado más grande y la fabricación de estado que la referencia es llamada estado de la submáquina. Una submáquina puede enlazarse a una clase que proporciona el contexto para las acciones dentro de él, como atributos que pueden leerse y pueden escribirse. Una submáquina es pensada para ser reusada en muchas máquinas de estado para evitar la repetición del mismo fragmento de la máquina de estado. Una submáquina es un tipo de subprograma de máquina de estado.

Dentro del estado de la submáquina, la submáquina es referenciada por nombre con una posible lista del argumento. El nombre debe ser el nombre de una máquina de estado que tiene un estado inicial y un estado final o punto de entrada explícito y estados de punto de salida. Si la submáquina tiene los parámetros en su transición inicial, entonces la lista del argumento debe tener argumentos concordantes. Cuando el estado de la submáquina se escribe, su acción de entrada se realiza primero, entonces la ejecución de la submáquina empieza con su estado inicial. Cuando la submáquina alcanza su estado final, cualquier acción de salida en el estado de la submáquina se realiza.

Entonces el estado de la submáquina es considerado completado y se toma una transición basada en la realización implícita de actividad.

Una transición a un estado de la submáquina activa el estado inicial de la submáquina designada.

Pero a veces una transición a un estado diferente en la submáquina es deseada.

Un punto de entrada es un pseudoestado puesto dentro de una submáquina que identifica *un declare* dentro de la submáquina. Pueden conectarse transiciones a un punto de conexión en la submáquina *declare* que empareja el punto de la entrada en la definición de la máquina de estado.

La transición realmente va al estado interior identificado por el punto de entrada, pero la transición externa necesitada no sabe nada sobre la estructura interior de la máquina de estado. Pueden hacerse conexiones similares a los puntos de entrada.

Una submáquina anidada representa una actividad 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 estado, una expresión de procedimiento puede quedar anexada a la submáquina (esto es un *haga la actividad*). Un *haga la actividad* puede considerarse como definir la serie de estados, uno para expresión primitiva, que es el interrumpible y puede aceptar eventos entre cualesquiera dos pasos. No es igual que un efecto anexado a una transición, que tiene la subestructura pero no acepta eventos durante la ejecución porque obedece semánticas *ejecutar-a-realización*.

Eventos diferibles. Un evento diferible es un evento cuyo reconocimiento en el estado es pospuesto si no activa una transición, para que no esté perdido. Esto permite una limitada capacidad de la región crítica en la que pueden procesarse ciertos eventos sin perder otros eventos. El reconocimiento de un evento diferido se pospone con tal de que el estado activo lo declare como diferible. Cuando la configuración de estado activo incluye un estado en el cual el evento no es diferido, se procesa. La implementación de tales eventos diferidos envolvería una cola interna de eventos.

Estados redefinidos. Los estados pueden definirse cuando se define una máquina de estado para una subclase.

Véase redefinición (máquina de estado).

Notación

Un estado se muestra como un rectángulo con esquinas redondeadas. Puede tener uno o más compartimientos. Los compartimientos son optativos. Los compartimientos siguientes pueden ser incluidos:

Nombre del compartimiento. Sostiene el nombre (optativo) del estado como una cadena. Los estados a menos que los nombres sean anónimos y sean todos distintos. Es indeseable repetir el mismo símbolo de estado nombrado dos veces en el mismo diagrama, sin embargo, pues es confuso.

El nombre puede ponerse opcionalmente dentro de una etiqueta rectangular enlazada al lado superior del símbolo de estado. Esta anotación es particularmente útil con estados compuestos, porque el nombre es claramente distinguido de los nombres y volúmenes de las regiones anidadas.

Región anidada. Muestra un fragmento del diagrama de estado de una región como compuesto de estados secundarios anidados. El fragmento del diagrama de estado es arrastrado dentro del límite del estado exterior. Las transiciones pueden conectar directamente a los estados anidados, así como al límite del estado exterior. Dentro de un estado no ortogonal, los subestados son arrastrados directamente dentro del estado compuesto. En un estado ortogonal, el símbolo de estado compuesto es dividido en regiones por líneas discontinuas (es decir, se embaldosa), y un fragmento de la máquina de estado se muestra dentro de cada región.

Véase estado compuesto para los detalles y ejemplos.

Compartimiento de la transición interior. Sostiene una lista de acciones interiores o actividades realizadas en respuesta a los eventos recibidos mientras el objeto está en el estado, sin que el estado cambie. Una transición interior tiene el formato:

nombre-evento$_{opc}$⌊(**argumento**$_{lista,}$)⌋$_{opc}$⌊[**condición-de-guarda**]⌋$_{opc}$

⌊/ **expresión-de-actividad**⌋$_{opc}$

Las expresiones de actividad pueden usar atributos y eslabones del objeto propio y parámetros de transiciones entrantes (si aparecen en transiciones todas entrantes).

La lista del argumento (incluyendo paréntesis) puede omitirse si no hay ningún parámetro.

La condición del guardia (incluyendo los paréntesis) y la expresión de acción (incluyendo la raya vertical) es optativo.

Las acciones de entrada y de salida tienen la misma forma, pero las palabras de uso reservado **entry** y **exit** no pueden usarse para nombres de evento.

Entry / **expresión-de-actividad**

Exit / **expresión-de-actividad**

Las actividades de entrada y de salida no pueden tener argumentos o condiciones de guardia (porque se invocan implícitamente, no explícitamente). Para obtener los parámetros en una acción de entrada al evento actual pueden ser accedidos por una acción. Esto es particularmente útil para obtener los parámetros de creación para un nuevo objeto.

El nombre de actividad reservado **defer** indica un evento que es diferible en un estado y sus subestados. La transición interior no debe tener una condición o acciones de guardia.

nombre-evento / **defer**

La palabra reservada **do** representa una expresión para una actividad **do** no atómica que haga la actividad.

do / **expresión-de-actividad**

Estado de submáquina. La invocación de una submáquina anidada es mostrada por una cadena nombre de la forma siguiente:

estado-nombre: **Máquina-nombre**

Una conexión de punto de entrada es mostrada por un círculo pequeño en el límite del símbolo de estado de submáquina (Figura 14.119). Una transición puede conectarse de un estado a la conexión del punto de entrada. Una conexión de punto de salida se muestra por un círculo pequeño que contiene una X en el límite del símbolo de estado. Una transición puede conectarse de la conexión de punto de salida a otro estado. *Véase* punto de conexión.

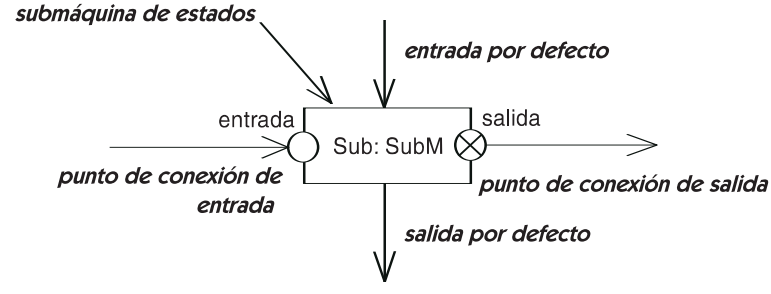


Figura 14.119 Estado de la submáquina

Ejemplo

La Figura 14.120 muestra un estado con transiciones interiores. La Figura 14.121 muestra la declaración y uso de una submáquina.

estado compuesto

Estado que contiene una o más regiones, cada una de las cuales contiene a su vez uno o más subestados directos.

Véase también transición compleja, región ortogonal, estado ortogonal, región, estado simple, estado.

Semántica

Un estado compuesto es un estado con subestados. Un estado compuesto se puede descomponer, utilizando relaciones y, en una o más regiones, cada una de las cuales se descompone, utilizando relaciones o, en uno o más subestados directos mutuamente excluyentes. Si un estado com-

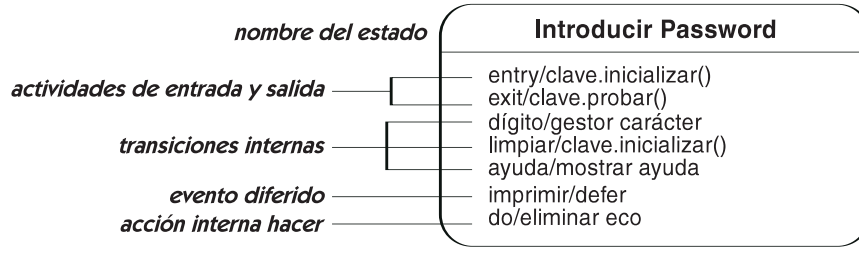


Figura 14.120 Transiciones internas, con acciones de entrada y salida y eventos diferidos

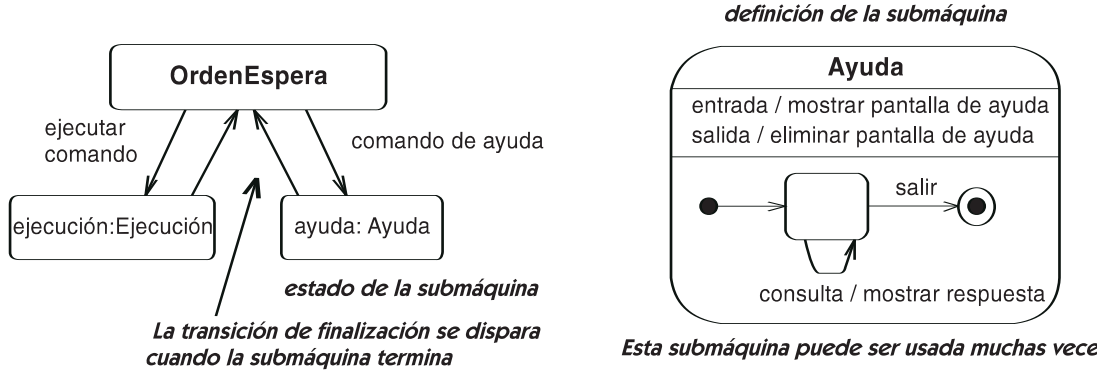


Figura 14.121 Submáquina

puesto está activo, en cada una de sus regiones está activo exactamente un subestado directo. El resultado es un árbol y/o de estados activos. Cada máquina de estados tiene un estado de nivel superior, que es un estado compuesto.

Un estado compuesto con exactamente una región es un estado no ortogonal. Si está activo, exactamente un subestado directo está activo. Añade una capa de subestructura pero no añade concurrencia adicional.

Un estado compuesto con más de una región es un estado ortogonal. Sus regiones se conocen como regiones ortogonales. Si está activo, está activo exactamente un subestado directo en cada región ortogonal. Introduce una concurrencia igual al número de regiones ortogonales, así como una capa de subestructura dentro de cada región ortogonal.

Un sistema puede manejar múltiples estados al mismo tiempo. El conjunto de estados activos se conoce como configuración de estados activos. 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 estado ortogonal.

Véase transición compleja para una discusión de ejecución concurrente; la Figura 14.277 muestra un árbol y/o.

Una transición de grupo es una transición que directamente abandona un estado compuesto. Si es una transición de finalización, se activa cuando se alcanza el estado final de cada región.

Cada región de un estado compuesto puede tener, como mucho, un estado inicial y un estado final. Cada región también puede tener, como mucho, un estado de historia superficial y un estado de historia profundo.

Un objeto recién creado comienza en su estado inicial, que debe tener el estado compuesto más externo. El evento que crea el objeto puede utilizarse para disparar una transición desde el estado inicial. Los argumentos del evento de creación están disponibles para esta transición inicial. Un objeto que transiciona a su estado final más externo es destruido y deja de existir.

Notación

Un estado compuesto es un estado con detalle subordinado. Tiene un nombre de compartimento, 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 apariencia, los compartimentos de texto (nombre y transiciones internas) pueden ser reducidos a lengüetas dentro de la región gráfica, en vez de atravesarlo horizontalmente.

Una expansión de un estado compuesto ortogonal en regiones ortogonales se representa dividiendo el compartimento gráfico del estado utilizando líneas discontinuas para dividirlo en subregiones. Un estado compuesto no ortogonal tiene una sola subregión. Cada subregión puede tener un nombre opcional, y debe contener un diagrama de estados anidado con subestados directos. El compartimento de nombre y otros compartimentos de texto del estado global están separados de las regiones por una línea continua. Alternativamente, el nombre del estado compuesto puede colocarse en una pequeña lengüeta adjunta al símbolo del estado global, de forma que el compartimento de nombre no parezca una región.

Un estado inicial se representa como un pequeño círculo negro. En una máquina de estados de nivel superior, la transición de un estado inicial se puede etiquetar con el evento que crea el objeto. De lo contrario, debe estar sin etiquetar. Si no está etiquetada, 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 dicho estado, sino que debe pasar a un estado real.

Un estado final se representa como un círculo rodeando un pequeño círculo negro (un ojo de buey). Representa la finalización de actividad en el estado que lo engloba, y dispara una transición en el estado que lo engloba, etiquetada por el evento implícito de finalización de la actividad (habitualmente se representa como una transición sin etiquetar).

Ejemplo

La Figura 14.122 muestra un estado compuesto no ortogonal que contiene dos subestados directos, un estado inicial, y un estado final. Cuando el estado compuesto se activa, en primer lugar se activa el subestado **Inicio** (el destino del estado inicial).

Se puede ocultar el contenido de un estado compuesto en una vista particular. Una región oculta se muestra por medio de un pequeño icono que representa dos símbolos de estado unidos. La Figura 14.123 muestra la descomposición de una región compuesta más externa. La descomposición de sus dos subestados está oculta.

La Figura 14.124 muestra un estado compuesto ortogonal que contiene tres subregiones ortogonales. Cada subregión ortogonal se descompone a su vez en subestados directos. Cuando el estado compuesto **Incompleto** se activa, se activan los destinos de los estados iniciales. Cuando

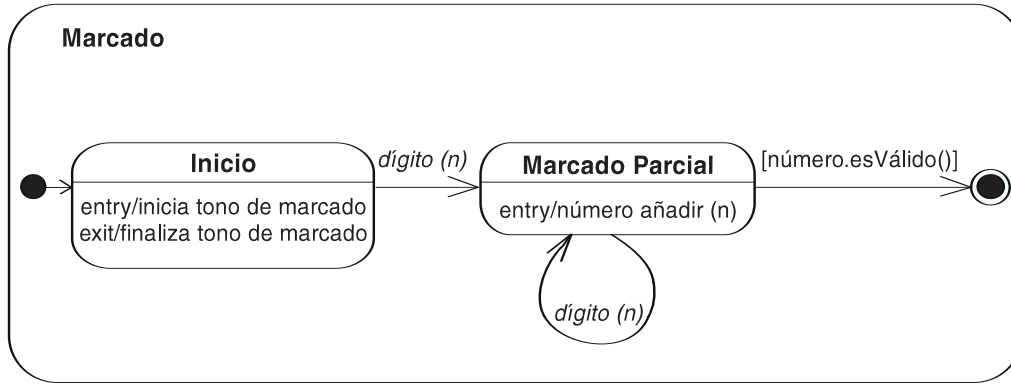


Figura 14.122 Estado compuesto no ortogonal

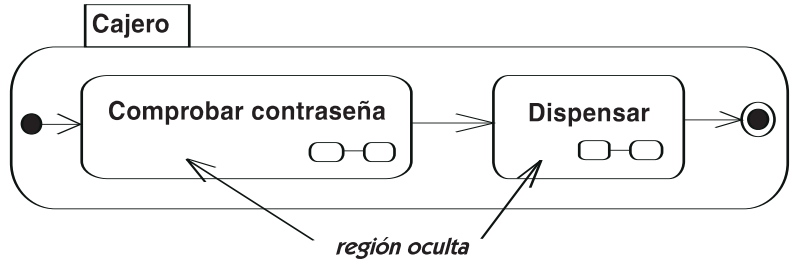


Figura 14.123 Región oculta en un estado compuesto

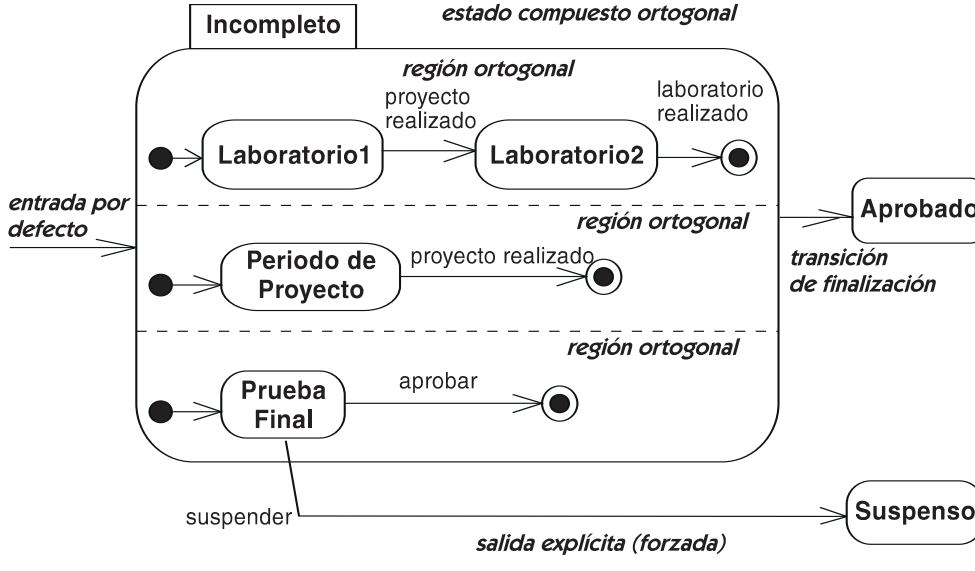


Figura 14.124 Estado compuesto ortogonal

las tres subregiones alcanzan su estado final, se dispara la transición de finalización del estado compuesto externo **Incompleto** y se activa el estado **Aprobado**. Si se produce el evento **suspender** mientras está activo el estado **Incompleto**, se terminan las tres subregiones ortogonales y se activa el estado **Suspenso**.

estado de flujo de objetos

Este concepto de UML1 es obsoleto en UML2. Ha sido reemplazado por el nodo de objeto.

estado de historia

Pseudoestado que indica que el estado compuesto que lo engloba recuerda su subestado activo previamente después de su salida.

Véase también estado compuesto, pseudoestado, máquina de estados, transición.

Semántica

Un estado de historia permite a un estado compuesto recordar el último subestado que estaba activo en él antes de la salida más reciente desde el estado compuesto. Una transición al estado de historia causa al primer subestado activo hacerse activo de nuevo después de ejecutar cualquier actividad o actividades entrar especificadas en el camino del subestado. Las transiciones entrantes se pueden conectar al estado de historia desde fuera del estado compuesto o desde el estado inicial.

Un estado de historia puede tener una transición saliente sin etiquetar. Esta transición indica el estado de historia inicial por defecto. Se utiliza si se produce una transición al estado de historia cuando no está presente ningún estado almacenado. El estado de historia no puede tener transiciones entrantes desde otros estados dentro del estado compuesto puesto que ya está activo.

Un estado de historia puede recordar *historia superficial* o *historia profunda*. Un estado de historia superficial recuerda y reactiva un estado en el mismo nivel de anidamiento que el propio estado de historia. Si una transición desde un subestado anidado sale directamente del estado compuesto, se activa el subestado que lo encierra en el nivel superior dentro del estado compuesto, pero ningún subestado anidado. Un estado de historia profunda recuerda a un estado que puede haber estado anidado a cualquier profundidad dentro del estado compuesto. Para recordar un estado profundo, una transición debe haber tomado el estado profundo directamente fuera del estado compuesto. Si una transición desde un estado profundo va a un estado más superficial, que luego transiciona fuera del estado compuesto, entonces el estado más superficial es el único que es recordado, puesto que es el origen de la salida más reciente. Una transición a un estado de historia profunda restaura el estado activo previo a cualquier profundidad. En el proceso, las actividades entrar se ejecutan si están presentes en estados interiores que contienen al estado recordado. Un estado compuesto puede tener tanto un estado de historia superficial como un estado de historia profunda. Una transición entrante debe conectarse a uno o al otro.

Si un estado compuesto alcanza su estado final, entonces pierde su historia almacenada y se comporta como si no se hubiera entrado en él la primera vez.

Notación

Un estado de historia superficial se representa como un pequeño círculo que contiene la letra **H**, como en la Figura 14.125. Un estado de historia profunda se representa como un círculo que contiene las letras **H***.

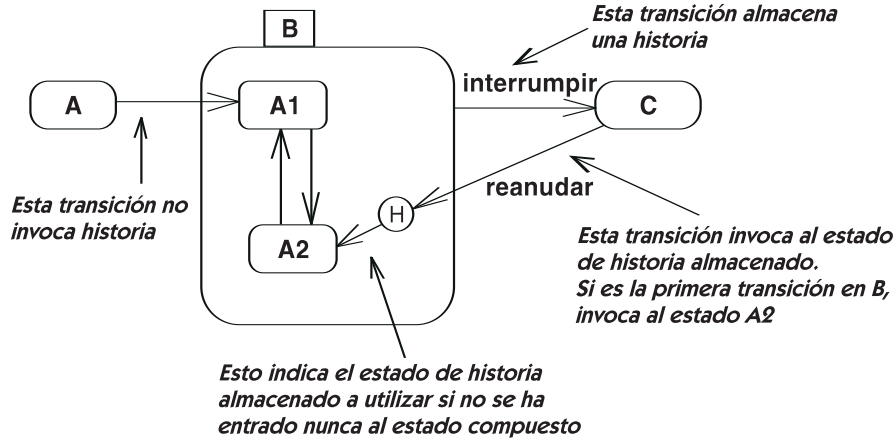


Figura 14.125 Estado de historia

estado de origen

El estado dentro de una máquina de estados de la que parte una transición. La transición se aplica al estado de origen. Si la configuración del estado que se activa incluye el estado origen o un estado anidado dentro de él, entonces la transición es una candidata para disparse.

estado de protocolo

El estado de una máquina de estado protocolar.

Semántica

Un estado protocolar no puede tener una actividad de entrada, actividad de salida, o realizar la actividad.

El rango completo de estados compuestos y estados de la submáquina están permitidos.

Notación

La notación es igual que un estado conductual, es decir, un rectángulo redondeado que contiene el nombre del estado.

estado de referencia a submáquina

Un estado que referencia a una submaquina (otra máquina de estados), una copia de la que es implícitamente parte de la máquina de estados contenedora en el lugar del estado de la submáquina referenciado. Es conceptualmente como una "llamada" en una "subrutina" de una máquina de estados (pero puede implementarse de forma diferente). Puede contener referencias a puntos de entrada y salida, que identifiquen los estados en la submáquina.

Véase también estado, máquina de estados, punto de entrada, punto de salida.

Semántica

Un estado de submáquina es semánticamente equivalente a insertar una copia de la submáquina en el lugar que referencia al estado. El estado de submáquina no tiene estructura en sí mismo; toda la estructura semántica proviene de la máquina de estados referenciada.

Notación

Un estado de submáquina se dibuja como un símbolo de estado con una etiqueta de la forma:

`Nombre-estado`: **`nombre-submáquina`**

Las flechas de transición se pueden dibujar al límite de la submáquina. Una transición a la submáquina de estados establece el estado inicial de la submáquina; si no hay estado inicial, no se permiten transiciones hacia el límite de la submáquina. Una transición, también se puede dibujar hacia un punto de conexión con nombre en el límite de la submáquina de estados. Una conexión de punto de entrada se muestra como un pequeño círculo. Un punto de conexión de salida, se muestra como un pequeño círculo que contiene una X. Una transición de o hacia un punto de conexión, es equivalente a una transición al correspondiente punto de entrada o salida de la submáquina de estados.

Ejemplo

La Figura 14.126 muestra parte de una máquina de estados que contiene una máquina de subastados. La máquina de estados contenida vende tickets a clientes con cuentas. Debe identificar a los clientes como parte de su trabajo. El identificar a los clientes es requisito de otra máquina de estados, así que ha sido hecho en una máquina separada. La Figura 14.127 muestra la definición de una máquina de estados **Identificar**, que se usa como una submáquina por las otras máquinas de estados. La entrada normal a la submáquina permite leer la tarjeta del cliente, pero también hay un punto de entrada explícito que permite la entrada manual del nombre del cliente en la oficina del dependiente. Si el proceso de identificación tiene éxito, la submáquina termina en su estado final. De lo contrario pasa a su estado de **Fallo**.

En la Figura 14.126, la submáquina de estados está representada por un icono de estado con un nombre local y el nombre de la submáquina. El punto normal de entrada a la submáquina se muestra mediante una flecha a su límite. Esta transición activa el estado inicial de la submáquina. La salida normal se representa por una transición de terminación desde el límite. Esta transición se dispara si la submáquina termina normalmente.

La entrada en el punto de entrada **EntradaManual** se representa por una transición a un punto de conexión en un símbolo de la submáquina de estados. El punto de conexión está etiqueta-

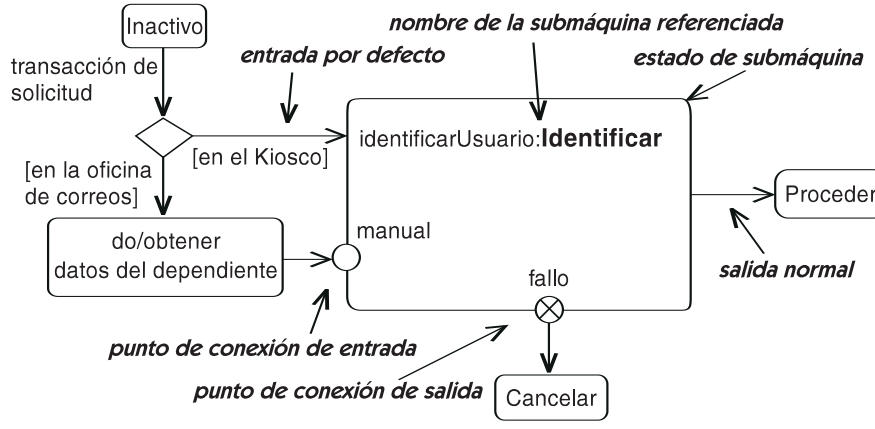


Figura 14.126 Estado de la submáquina

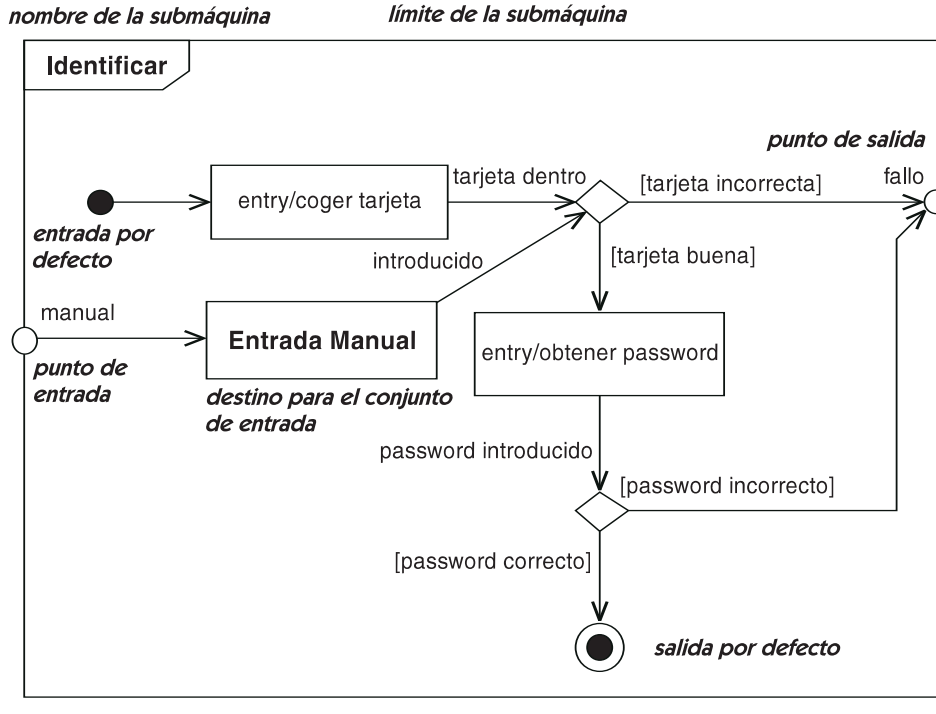


Figura 14.127 Definición de la submáquina

do con el nombre del punto de entrada en la submáquina. De forma similar, la salida de un estado explícito **Fallo** se representa por una transición a un punto de salida.

estado de sincronización

Este concepto de UML1, ha sido eliminado de UML2. No se necesita debido a que se han eliminado restricciones en las actividades.

estado destino

El estado de máquina de estados que resulta del disparo de una transición. Después de que un objeto maneja un evento que causa que se dispare una transición, el objeto pasa a estar en el estado destino de la transición (o estados destinos, si es una transición compleja con múltiples estados destino). No es aplicable a una transición interna que no causa cambio de estado.

Véase transición.

estado final

Estado especial que cuando se entra en él indica que se ha finalizado la ejecución de la región que lo contiene. El estado que contiene la región se finaliza si todas las demás regiones contenidas (si las hay) también han finalizado. Cuando finaliza un estado compuesto se desencadena una transición de finalización abandonando el estado compuesto y se puede disparar si se satisface su condición de guarda.

Véase también actividad hacer, transición de finalización, destrucción.

Semántica

Para promover la encapsulación es deseable separar todo lo posible la vista externa de un estado compuesto de sus detalles internos. Desde fuera, el estado se ve como una entidad opaca con una estructura interna oculta. Desde el punto de vista externo, conectan a subestados dentro del estado. Un estado inicial o un estado final es un mecanismo para dar soporte a la encapsulación de estados.

Un estado final es un estado especial que indica que la actividad de una región dentro del estado compuesto ha finalizado y que se activa una transición de finalización abandonando el estado compuesto. Si hay varias regiones en el estado compuesto, todas ellas deben alcanzar su finalización. Un estado final no es un pseudoestado. Un estado final puede estar activo durante un período de tiempo, a diferencia de un estado inicial que transiciona inmediatamente a su sucesor. El control puede permanecer dentro de un estado final que espera a la finalización de otras regiones ortogonales del estado compuesto —es decir, mientras espera a la sincronización de varios hilos de control. Sin embargo, no se permiten las transiciones salientes desencadenadas por eventos desde un estado final (por lo demás, es como un estado normal). Un estado final puede tener cualquier número de transiciones entrantes desde dentro del estado compuesto que lo engloba, pero ninguna desde fuera de dicho estado. Las transiciones entrantes son transiciones normales y pueden tener desencadenadores, 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 con un icono de ojo de buey —es decir, un pequeño círculo negro rodeado por otro círculo. El símbolo se coloca dentro del estado compuesto que lo engloba y

cuya finalización representa (Figura 14.128). Sólo se puede producir un estado final (directamente) dentro de cada región ortogonal de un estado compuesto. Sin embargo, se pueden producir estados finales adicionales dentro de estados compuestos anidados. Por comodidad, el símbolo de estado final se puede repetir dentro de un estado, pero cada copia representa el mismo estado final.

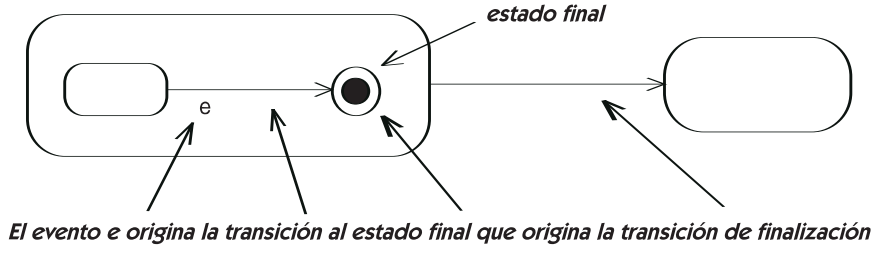


Figura 14.128 Estado final

estado inicial

Pseudoestado que indica el estado de partida por defecto en la región que lo engloba. Se utiliza cuando una transición alcanza la frontera de un estado compuesto.

Véase también estado compuesto, creación, actividad entrar, punto de entrada, punto de salida, inicialización, conjunción.

Semántica

Para promover la encapsulación es deseable separar la vista externa de un estado compuesto de los detalles internos tanto como sea posible. Desde fuera, el estado se ve como una entidad opaca con una estructura interna oculta. Desde el punto de vista exterior, las transiciones van hacia y desde el propio estado. Desde el punto de vista interno, las transiciones conectan con subestados dentro del estado.

Un estado inicial es un estado (pseudoestado) que designa el estado de partida por defecto en la región que lo contiene. Mejor dicho, es una manera sintáctica de indicar dónde debería ir el control. Un estado inicial debe tener una transición saliente sin desencadenadores (una transición sin desencadenador de eventos, y por tanto automáticamente habilitada tan pronto como se entra en el estado inicial). La transición conecta con un estado real en el estado compuesto. La transición puede tener una acción.

Cuando se dispara una transición externa hacia la frontera del estado compuesto que la engloba, se produce el siguiente comportamiento, en orden: Se ejecuta cualquier acción en la transición saliente; se activa cada subregión ortogonal del estado compuesto; se ejecuta cualquier actividad entrar para una subregión dada; se activa el estado inicial de la subregión y origina tomar su transición de salida; se ejecuta cualquier acción sobre la transición saliente; se activa el estado de partida por defecto de la subregión; la ejecución de la transición ahora se ha finaliza-

do. A la finalización de la transición, un estado estará activo en cada subregión del estado compuesto.

La acción entrar del estado compuesto y la transición de finalización del estado inicial pueden acceder al evento actual implícito —es decir, el evento que desencadenó el primer segmento en la transición que finalmente causó la transición al estado inicial.

La transición sobre un estado inicial no puede tener un desencadenador de evento. Un estado inicial es un pseudoestado y no puede permanecer activo.

La mayoría de las veces la transición desde el estado inicial no tiene guardas. En ese caso, debe ser la única transición desde el estado inicial. Un conjunto de transiciones salientes pueden tener condiciones de guarda, pero las condiciones de guarda deben cubrir completamente todos los posibles casos (o, más fácil, una de ellas debe tener la condición de guarda **else**). La idea es que el control debe abandonar el estado inicial inmediatamente. No es un estado real, y se debe disparar alguna transición.

El estado inicial en el estado de más alto nivel de una clase representa la creación de una nueva instancia de la clase. Cuando la transición que abandona el estado inicial se dispara, el evento actual implícito es el evento de creación del objeto y tiene los valores de los argumentos pasados por la operación de construcción. Esos valores están disponibles dentro de acciones en la transición saliente.

Creación de objetos

El estado inicial del estado compuesto más alto de una clase es ligeramente diferente. Puede tener un desencadenador con el estereotipo **«create»**, junto con un desencadenador de evento con nombre y parámetros. Puede haber varias transiciones de este tipo con diferentes desencadenadores. La signatura de cada desencadenador debe concordar con una operación de creación en la clase. Cuando se instancia un nuevo objeto de la clase, la transición correspondiente con su operación de creación se dispara y recibe los argumentos de la llamada a la operación de creación.

Notación

Un estado inicial se muestra como un pequeño círculo negro dentro del símbolo de su estado compuesto. Las flechas de transición salientes deben ir conectadas a él. Sólo se puede producir un estado inicial (directamente) dentro de un estado compuesto. Sin embargo, se pueden producir estados iniciales adicionales dentro de estados compuestos anidados.

Ejemplo

En la Figura 14.129, empezamos en el estado **X**. Cuando sucede el evento **e**, la transición se dispara y se realiza la acción **a**. La transición va al estado **Y**. Se realiza la acción entrar **b**, y el estado inicial se activa. La transición saliente se dispara inmediatamente, realizando la acción **c** y cambiando al estado **Z**.

Por otro lado, si se produce el evento **f** cuando el sistema está en el estado **X**, entonces se dispara la otra transición y se realiza la acción **d**. Esta transición va directamente al estado **Z**. El estado inicial no está implicado. Puesto que el control pasa al estado **Y**, se realiza la acción **b**, pero la acción **c** no se realiza en este caso.

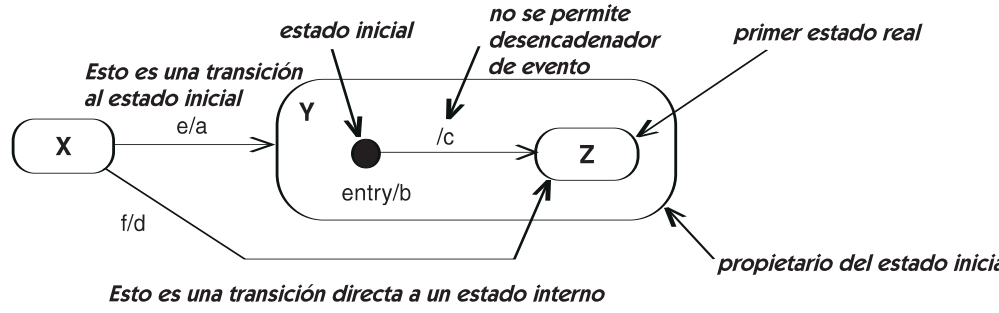


Figura 14.129 Estado inicial

En la Figura 14.130, el estado inicial tiene una bifurcación. De nuevo, suponga que el sistema comienza en el estado **X**. Cuando se produce el evento **e**, se realiza la acción **a**, el sistema cambia al estado **Y**, y se realiza la acción entrar **b**. El control va al estado inicial. Se prueba el atributo tamaño del objeto propietario. Si es 0, el control va al estado **Z**; si no lo es, va al estado **W**.

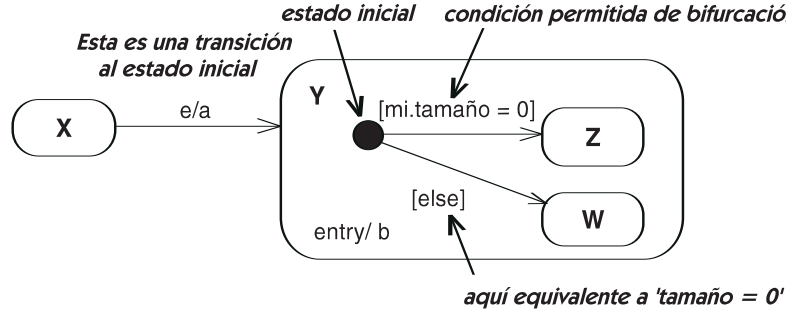


Figura 14.130 Estado inicial con bifurcación

estado no ortogonal

Un estado compuesto por una sola región. *Véase* estado compuesto.

Semántica

Un estado compuesto por una sola región es un “y” descomposición en subestados directos.

Cuando el estado compuesto está activo, exactamente uno de sus subestados directos está activo.

Si se alcanza un estado final directo, el estado no ortogonal que lo contiene se completa y una transición de completitud que sale de él se dispara.

estado ortogonal

Un estado compuesto que contiene más de una región. Si el estado compuesto está activo un subestado de cada región está activo. Concurrentemente los subestados activos de las regiones diferentes son independientes de cada uno de los otros.

Véase transición compleja, estado compuesto, región ortogonal, región.

Semántica

Un estado compuesto tiene una o más regiones. Si un estado compuesto es un subestado activo, debe ser activo en cada región. Si hay más de una región, los subestados de las regiones diferentes son concurrentemente activas. Por conveniencia, un estado compuesto con una región sencilla se llama estado no ortogonal; un estado compuesto con más de una región se llama estado ortogonal. Cualquier transición compleja dentro o fuera de un estado ortogonal envuelve un bifurcador o un ensamblador que cambia el número de estados activos en la configuración de estado activa. Una transición debe incluir (explícitamente o implícitamente), por lo menos un estado de cada región de un estado ortogonal.

Historia

El concepto de regiones fue incluido en UML2 para hacer la descomposición de estados más uniforme que en UML1, pero con poder equivalente.

estado simple

Un estado que no tiene estados jerarquizados dentro de él. Un conjunto de estados jerarquizados forman un árbol y los estados simples son las hojas. Un estado simple no tiene subestructuras. Puede tener transiciones internas, una actividad de entrada, una actividad de salida y una actividad de hacer. Contraste: estado compuesto, estado de submáquina.

estereotipo

Un nuevo tipo de elemento del modelo definido dentro de un perfil basado en un tipo existente de elemento del modelo. Es esencialmente una nueva metaclase. Los estereotipos pueden extender la semántica pero no la estructura de las clases existentes del metamodelo. *Véase también* restricción, valor etiquetado.

Semántica

Un estereotipo representa una variación de un elemento del modelo existente con la misma forma (como atributos y relaciones) pero con la intención modificada. Generalmente, un estereotipo representa una distinción de uso. Un elemento estereotipado puede tener restricciones más allá de aquéllos del elemento base, así como una imagen visual distinta y propiedades adicionales (metaatributos) definidos a través de definiciones etiquetadas. Se espera que los generadores de código y otras herramientas puedan tratar elementos estereotipados, generando el código

diferente, por ejemplo. El objetivo es que una herramienta genérica, como un editor de modelos o un repositorio, debe tratar a un elemento estereotipado de forma especial para más propósitos que un elemento ordinario con un poco de información textual adicional, mientras diferencia a el elemento para ciertas operaciones semánticas, detección de elementos bien formados, generación del código, y generación de informes. Los estereotipos representan uno de los mecanismos de extensibilidad incorporados de UML.

Los estereotipos están definidos dentro de los perfiles. Cada estereotipo se deriva de una clase base del modelo. Todos los elementos que llevan el estereotipo tienen las propiedades de la clase del elemento base del modelo.

Un estereotipo también puede especializarse en otro estereotipo. El estereotipo hijo tiene las propiedades del estereotipo padre. Finalmente, cada estereotipo está basado en alguna clase de elemento del metamodelo.

Un estereotipo puede definir las propiedades modelo-tiempo de elemento a través de una lista de definiciones etiquetadas. Una definición etiquetada puede tener un valor predefinido que se usa si no se proporciona ningún valor etiquetado explícito. También puede especificarse el rango permitido de valores para cada etiqueta. Cada elemento que lleva el estereotipo debe de haber etiquetado los valores con las etiquetas disponibles. Las etiquetas con valores predefinidos son implícitas de forma automática si no están explícitas en un elemento estereotipado.

Una definición de una etiqueta es, esencialmente, un metaatributo para la metaclase a la que se aplica el estereotipo. Un elemento del modelo que lleva el estereotipo puede tener un valor para la definición de la etiqueta. Observe que éste es un valor del propio elemento del modelo, no de la instancia del elemento del modelo.

Un estereotipo puede tener una lista de restricciones que se agregan a las condiciones por encima de las del elemento base. Cada restricción se aplica a cada elemento del modelo. Cada elemento del modelo también está sujeto a las restricciones aplicables al elemento base.

Un estereotipo es un tipo de metaclase virtual (es decir, no se manifiesta en el metamodelo), eso se agrega dentro de un modelo en lugar de modificar los predefinidos por el metamodelo de UML. Por esa razón, los nombres de nuevos estereotipos deben diferir de los nombres de las metaclases de UML existentes u otros estereotipos o palabras claves.

Los elementos del modelo pueden tener cero, uno, o más de un estereotipo.

UML cuenta con ciertos estereotipos predefinidos; otros pueden ser definidos por el usuario. Los estereotipos son uno de los tres mecanismos de extensibilidad de UML.

Véase restricción, valor etiquetado.

Notación

Declaración de estereotipo. Los estereotipos se definen dentro de un perfil. En un diagrama cada estereotipo se muestra como un símbolo de la clase (rectángulo) con la palabra clave del «**estereotipo**.» Un estereotipo aplicado a una metaclase del metamodelo base, representado con la palabra clave «**metaclase**». La relación es una relación de extensión, mostrada por una flecha del estereotipo a la metaclase con una punta de flecha llena triangular (Figura 14.131). Si el estereotipo es obligatorio en todos los elementos de la metaclase la restricción {**requerido**} puede ponerse en la flecha de la extensión. Es posible aplicar más de un estereotipo a la misma metaclase y el mismo estereotipo puede que se aplique a más de una metaclase.

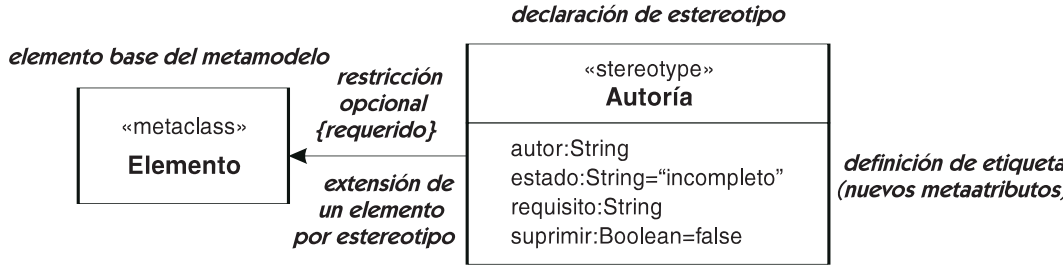


Figura 14.131 Declaración de estereotipos

Un estereotipo se declara con un rectángulo de clasificador con la palabra clave "estereotype" sobre el nombre del estereotipo. El rectángulo contiene un compartimiento de atributos en el que se definen etiquetas usando la sintaxis de los atributos. Se puede mostrar un valor predefinido para una etiqueta con el signo igual. Los estereotipos pueden mostrar generalización (Figura 14.132).

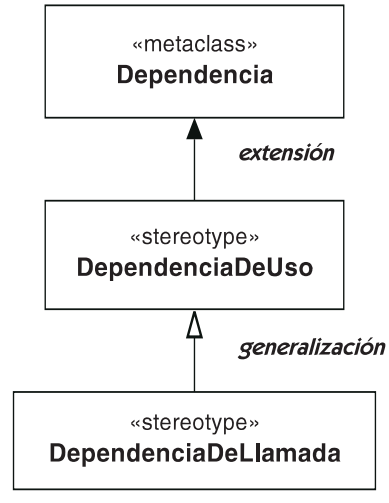


Figura 14.132 Estereotipo de generalización

Uso de estereotipos. La notación general para el uso de un estereotipo es usar el símbolo para el elemento base pero poniendo una palabra clave de tipo cadena sobre el nombre del elemento (si lo hay). La cadena de la palabra clave es el nombre del estereotipo dentro de llaves que son los símbolos de la comilla francesa y algún otro idioma —por ejemplo: «**foo**». (Nótese que una llave se parece a unos ángulos dobles pero es un solo carácter en la mayoría de las fuentes extendidas. La mayoría de las computadoras tiene una utilidad de mapa de carácter en la que pueden encontrarse símbolos especiales. Los ángulos dobles son usados por el cambio tipográfico.) La palabra clave normalmente se coloca delante del nombre del elemento que describe

La cadena de la palabra clave también puede usarse como un elemento en una lista. En ese caso, se aplican los elementos de la lista subsecuentes hasta que otra cadena de estereotipo la reemplace o hasta que la cadena del estereotipo esté vacía (« »). Observe que un nombre de este-

reotipo no debe ser idéntico a una palabra clave predefinida aplicable al mismo tipo de elemento. (Evitar confusión, se debe evitar usar un nombre de palabra clave predefinida para cualquier estereotipo incluso si es discernible en principio.) Si un elemento tiene estereotipos múltiples, sus nombres son incluidos en una lista separada por comas. *Véan* la Figura 14.133.

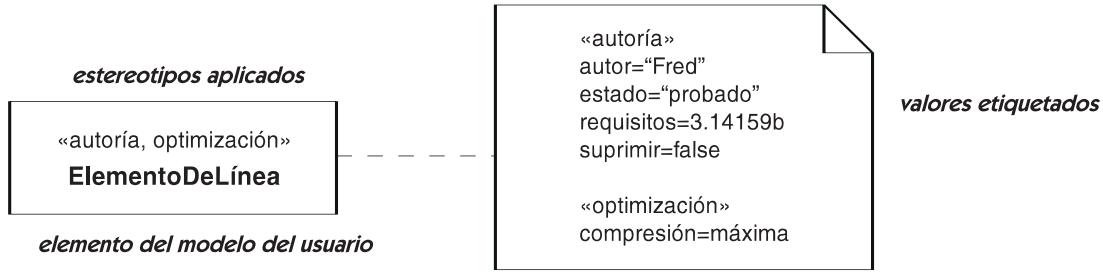


Figura 14.133 Aplicando estereotipos a elementos

Se pueden mostrar los valores de las etiquetas en un símbolo de comentario asociado al elemento del modelo. El nombre del estereotipo se pone sobre una lista de los valores de la etiqueta. Si un elemento tiene estereotipos múltiples, se puede mostrar más de una lista de valores.

Para permitir una extensión gráfica limitada de la notación UML, se puede asociar un icono gráfico o un marcador gráfico (como texto o color) con un estereotipo.

UML no define la forma de la especificación gráfica, pero existen muchos formatos de mapas de bits y podrían ser usados por un editor gráfico (aunque su portabilidad es un problema difícil). Un icono puede usarse de dos maneras. En un caso, puede usarse en lugar de, o además de,

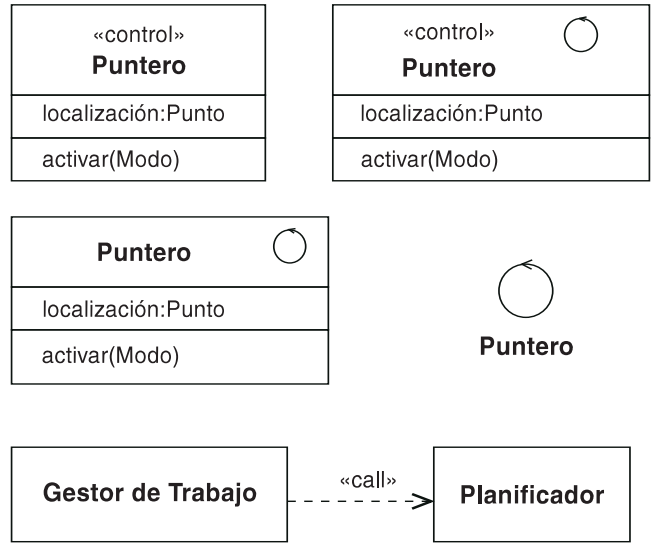


Figura 14.134 Variedad de la notación de estereotipos

la cadena de la palabra clave del estereotipo dentro del símbolo para el elemento base del modelo en el que el estereotipo está basado. Por ejemplo, en un rectángulo de un clasificador, se pone en la esquina superior-derecha del compartimiento del nombre. De esta forma, pueden verse las características normales del elemento en su símbolo. Alternadamente el símbolo del elemento puede convertirse en un icono que contiene el nombre del elemento o tiene el nombre sobre o debajo del icono. Otra información contenida por el símbolo del elemento del modelo base se suprime. La Figura 14.134 muestra varias formas de dibujar una clase estereotipada.

UML evita el uso de marcadores gráficos, como color, ya que son una limitación para ciertas personas (los ciegos al color) y para ciertos tipos de equipo (tal como copiadoras, impresoras y el fax). Ninguno de los símbolos de UML requiere el uso de tales marcadores gráficos. Los usuarios pueden usar los marcadores gráficos libremente para sus propósitos (como para resaltar algo dentro de una herramienta) pero debe ser consciente de sus limitaciones para el intercambio de información y estar preparados para usar las formas canónicas cuando sea necesario.

estructura compuesta

Describe la interconexión de objetos dentro de un contexto para formar una entidad con un propósito global. Las estructuras compuestas incluyen clasificadores estructurados y colaboraciones.

Véase clasificador estructurado, colaboración, conector, parte estructurada, puerto.

estructura interna

Partes, puertos y conectores interconectados que componen el contenido de un clasificador estructurado.

Véase clasificador estructurado.

etapas de modelado

El estado de desarrollo de un elemento o un modelo que pasa a través del proceso de diseño y construcción de un sistema. *Véase también* proceso de desarrollo.

Discusión

El esfuerzo de desarrollo total puede ser dividido en actividades enfocadas a diferentes fines. Estas actividades no se realizan en secuencia; más bien, se realizan iterativamente durante las fases del proceso de desarrollo. El análisis trata de captar requerimientos y de entender las necesidades del sistema. El diseño crea un acercamiento práctico al problema dentro de las restricciones de las estructuras de datos, los algoritmos y los pedazos del sistema existente. La implementación trata de construir la solución en un lenguaje ejecutable o medio (como una base de datos o hardware digital). El desarrollo trata de ubicar la solución práctica en un ambiente físico específico. Estas divisiones son un tanto arbitrarias y no siempre claras, pero siguen siendo pautas útiles.

Sin embargo, no deben igualarse estas vistas de desarrollo con fases secuenciales del proceso de desarrollo. En el proceso tradicional de cascada de hecho, se trata el desarrollo como fases

distintas. En un proceso de desarrollo iterativo más moderno, sin embargo, no hay fases distintas. En un punto de tiempo determinado, actividades de desarrollo pueden existir a varios niveles y pueden entenderse mejor como tareas diferentes necesarias para realizar cada elemento del sistema, no todo al mismo tiempo.

Piense en un grupo de edificios, cada uno con una función, paredes y tejado; todos los elementos deben completarse para todos los edificios, pero no todo al mismo tiempo. Normalmente se completan las partes de cada edificio más o menos en orden. A veces sin embargo, el tejado puede empezarse antes de que todas las paredes estén completas. De vez en cuando, la distinción entre las paredes y el tejado se pierde.

UML contiene un rango de estructuras apropiado para las distintas fases del desarrollo.

Algunas estructuras (como asociación y estado) no tienen significado en todas las fases. Algunas estructuras (como navegabilidad y visibilidad) son significantes durante el diseño, pero representan un detalle de implementación innecesario durante el análisis. Esto no evita su definición en una fase temprana de trabajo. Algunas estructuras (como sintaxis específica del lenguaje de programación) sólo son significativas durante la implementación y dañan el proceso de desarrollo si se introducen prematuramente.

Los modelos cambian durante el desarrollo. Un modelo UML toma una forma diferente en cada fase de desarrollo, con énfasis variable en diferentes estructuras de UML.

El diseño debe realizarse entendiendo que no todas las estructuras son útiles en todas las fases.

Véase proceso de desarrollo para una discusión de la relación de fases de modelado y fases de desarrollo.

etiqueta

Término para la utilización de una cadena en un diagrama. Es simplemente un término rotacional.

Véase también diagrama.

Notación

Una etiqueta es una cadena gráfica que está adjunta lógicamente a otro símbolo en un diagrama. Visualmente, el mecanismo de adjuntar es una forma de contener la cadena en una región cerrada o colocarla cerca del símbolo. Para algunos símbolos la cadena se coloca en una posición definida (como debajo de una línea), pero para la mayoría de símbolos la cadena debe estar “cerca” de una línea o icono. Una herramienta de edición puede mantener un vínculo gráfico interno explícito entre una etiqueta y un símbolo gráfico, de forma que la etiqueta permanezca conectada lógicamente al símbolo incluso si se ve separada visualmente. Pero la apariencia final del diagrama es un asunto de juicio estético y debería hacerse de forma que no haya confusión acerca de a qué símbolo pertenece una etiqueta. Aunque el adjunto puede no ser obvio a partir de una inspección visual del diagrama, el adjunto es claro y no es ambiguo en el nivel de estructura gráfica (y por tanto no plantea ambigüedad en el mapeo semántico). Una herramienta puede mostrar de forma visual el adjuntado de una etiqueta a otro símbolo utilizando diversas ayudas (como una línea coloreada o el resaltado de los elementos concordantes) como crea conveniente.

evento

Tipo de ocurrencia digna de mención que tiene una ubicación en el tiempo y en el espacio.

Véase también máquina de estados, transición, disparador.

Semántica

Un evento es algo que ocurre durante la ejecución de un sistema que es digno de modelar. Dentro de una máquina de estados, la ocurrencia de un evento puede disparar una transición de estado. Un evento tiene una (posiblemente vacía) lista de parámetros que transmiten información desde el creador del evento a su receptor. El momento de tiempo en que se produce el evento es implícitamente un parámetro de cada evento. Otros parámetros forman parte de la definición de un evento.

Una ocurrencia (instancia) de un evento tiene un argumento (valor real) en correspondencia a cada parámetro del evento. El valor de cada argumento está disponible para una acción adjunta a una transición disparada por el evento.

Observe que los eventos no aparecen explícitamente en los modelos. Un disparador en una transición de máquina de estados especifica un evento cuya ocurrencia habilita la transición.

Hay cuatro tipos de eventos que se pueden utilizar en disparadores:

evento de llamada	Recepción de una petición para invocar una operación, es decir, la recepción de una llamada. El resultado esperado es la ejecución de la operación disparando una transición en el receptor. El disparador especifica la operación. Los parámetros del evento son los parámetros de la operación e, implícitamente, un puntero de retorno. El emisor recupera el control cuando se completa la transición (o inmediatamente si no se dispara ninguna transición).
evento de cambio	Satisfacción de una condición lógica especificada por una expresión en el evento. El disparador especifica una condición lógica como una expresión. No hay parámetros del evento. Este tipo de evento implica una prueba continua de la condición. El evento se produce cuando la condición cambia de falsa a verdadera. Sin embargo, en la práctica, los momentos en los que la condición puede ser satisfecha pueden ser restringidos a la ocurrencia de otros eventos, de forma que no se suele necesitar el sondeo.
evento de señal	Recepción de una señal, que es una entidad explícitamente nombrada prevista para comunicación explícita entre objetos. El disparador especifica el tipo de señal. Los parámetros del evento son los parámetros de la señal. Una señal es enviada explícitamente por un objeto a otro objeto o conjunto de objetos. Una difusión general de un evento puede ser considerada como el envío de una señal al conjunto de todos los objetos, aunque en la práctica, podría ser implementado de forma diferente por cuestiones de eficiencia. El emisor especifica explícitamente los argumentos de la señal en el momento

	en que la envía. Una señal enviada a un conjunto de objetos puede disparar cero o una transición en cada uno de ellos. Las señales son formas explícitas por las que los objetos se pueden comunicar con los demás de manera asíncrona. Para realizar comunicación síncrona, se pueden utilizar dos señales asíncronas, una en cada dirección de la comunicación, o si no se puede utilizar una llamada síncrona. Las señales son generalizables. Una señal hija se deriva de una señal padre; hereda los parámetros del padre y puede tener sus propios parámetros adicionales. Una señal hija satisface un disparador que especifica a cualquiera de sus antecesores.
evento temporal	Satisfacción de una expresión temporal, como la ocurrencia de un tiempo absoluto o el paso de una cantidad de tiempo después de que un objeto entre en un estado. El disparador especifica la expresión temporal. Observe que tanto el tiempo absoluto como el tiempo transcurrido pueden ser definidos con respecto a un reloj del mundo real o a un reloj virtual interno (en cuyo caso, puede diferir para objetos diferentes).

También hay varios tipos de ocurrencias formales que podrían considerarse eventos desde algunos puntos de vista, incluyendo el inicio de la ejecución, finalización de la ejecución, creación y destrucción.

Notación

Véanse los tipos específicos de disparadores asociados con un evento para ver detalles de notación.

evento actual

Evento que dispara un paso de ejecución hasta finalizar en la ejecución de una máquina de estados.

Véase también ejecutar hasta finalizar, máquina de estados, transición.

Semántica

Una máquina de estados puede atravesar varios segmentos de transición conectados en respuesta a un evento. Todos los segmentos de transición menos el último van a pseudoestados —es decir, estados cuyo propósito es ayudar a estructurar la máquina de estados, pero que no esperan eventos de fuera. En principio, todos los segmentos podían estar juntos en una transición, pero la separación en varios segmentos utilizando pseudoestados permite compartir subsecuencias comunes entre varias transiciones.

La ejecución de una cadena de segmentos de transición es parte de un único paso de ejecución hasta finalizar y que no puede ser interrumpido por un evento externo. Durante la ejecución de dicha cadena de transiciones, las acciones y condiciones de guarda adjuntas a los segmentos

tienen acceso implícito al evento que disparó el primer segmento y a los parámetros de dicho evento. Este evento es conocido como el evento actual durante una transición.

El evento actual es particularmente útil para la transición inicial de un objeto nuevo, para obtener los parámetros de creación. Cuando se crea un objeto nuevo, el evento que lo crea se convierte en el evento actual y sus parámetros quedan disponibles durante la transición inicial de la máquina de estados del nuevo objeto.

Notación

Los parámetros del evento actual pueden usar el formato:

`nombre-del-evento` . **`nombre-del-atributo`**

Ejemplo

La Figura 14.135 muestra una transición desde el estado **Inactivo** al estado **Compra** disparada por el evento **petición**. La acción de entrada de **Compra** llama a la operación **inicializar**, que usa el parámetro producto del evento actual.

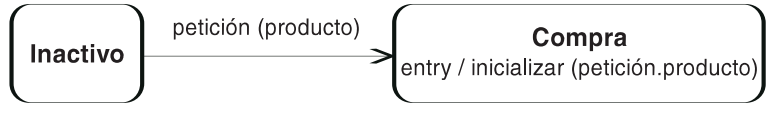


Figura 14.135 Utilización del evento actual

evento de cambio

El evento de una expresión Booleana que se satisface debido al cambio de uno o más valores de aquellos a los que hace referencia.

Véase también condición de guarda.

evento de envío

El envío de un evento (normalmente una señal) por un objeto.

Véase evento de señal.

Semántica

Los eventos pueden representar tanto el envío como la recepción de mensajes. Ambos tipos de eventos aparecen en interacciones. Como disparadores, sin embargo, el envío de eventos es menos interesante que los eventos de recepción, porque el remitente simplemente puede realizar las acciones necesarias como parte del mensaje a enviar. La implementación explícita de los eventos de envío podría ser útiles sin embargo, si el comportamiento es activado automáticamente sin modificar el código enviado.

evento de llamada

El evento de recibir una llamada por un objeto destino para una operación. Se puede implementar mediante métodos en clase del objeto, mediante acciones en transiciones de una máquina de estados o mediante otras formas que se determinan en las reglas de resolución para el destino.

Véase llamada, disparador de llamada.

evento de recepción

El recibo de un mensaje (normalmente una señal) por un objeto.

Véase evento de señal.

evento de señal

Un evento que es la recepción por un objeto de una señal de envío a él, el cual puede activar una transición en su máquina de estados.

Semántica

Por ser más concretos, hay eventos de recibo y envío. El término *evento de señal* se usa a menudo para significar *evento de recibo,* porque las máquinas de estados usualmente actúan sobre el recibo de mensajes de otros objetos. Ambos tipos de evento son posibles sin embargo, bajo ciertos tipos de sistemas.

evento diferible

Evento cuyo reconocimiento puede ser diferido mientras que un estado está activo.

Véase evento diferido

evento diferido

Evento cuya aparición se difiere porque se ha declarado para ser un evento diferible en el estado activo. *Véase también* máquina de estados, transición.

Semántica

Un estado puede designar un conjunto de eventos como *diferibles*. Si se produce un evento mientras que un objeto está en un estado para el que el evento es diferible y el evento no dispara una transición, el evento es diferido: El evento no tiene efecto inmediato; se guarda hasta que el objeto entra en un estado en el que el evento dado no es diferido. Si se producen otros eventos mientras el estado está activo, se manejan de la forma habitual. Cuando el objeto entra en un nuevo estado, los eventos diferidos que ya no son diferibles se van produciendo uno a uno y pueden dis-

parar transiciones en el nuevo estado (el orden en que se producen los eventos anteriormente diferidos es indeterminado, y es arriesgado depender de un orden particular). Si un evento no dispara ninguna transición en el estado no diferible, es ignorado y perdido.

Los eventos diferibles deberían utilizarse con cuidado en máquinas de estados ordinarias. A menudo se pueden modelar de manera más directa por medio de un estado concurrente que los responda mientras la computación principal está haciendo algo más. Pueden ser útiles para permitir que las computaciones sean secuenciadas sin perder mensajes asíncronos.

Si un estado tiene una transición disparada por un evento diferible, la transición anula el aplazamiento y el evento dispara la transición, sin resistirse a la especificación diferible.

Notación

Un evento diferible se indica por una transición interna en el evento con la palabra reservada **defer** en lugar de la acción. El aplazamiento se aplica al estado y sus subestados anidados (Figura 14.136).

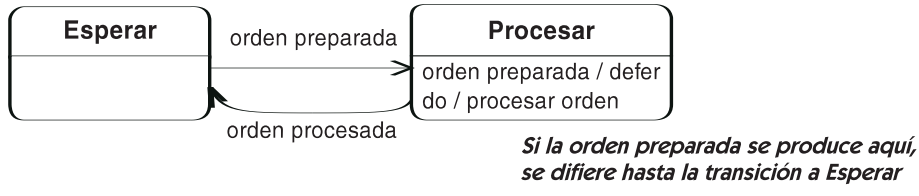


Figura 14.136 Evento diferible

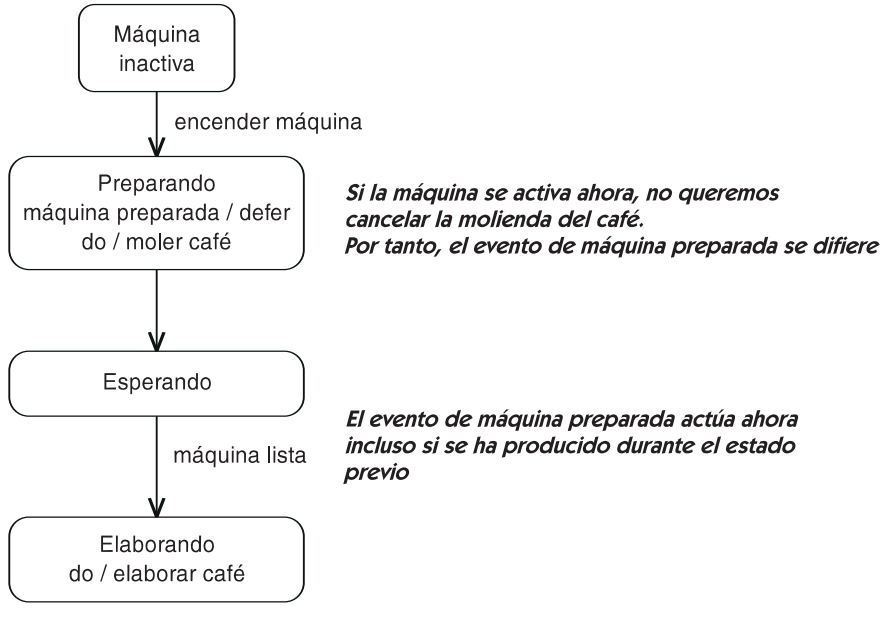


Figura 14.137 Ejemplo de evento diferible

La Figura 14.137 muestra los pasos para hacer una taza de café. La secuencia comienza cuando la máquina se enciende. La persona que está haciendo el café tiene que moler el café y esperar a que la máquina se caliente antes de elaborar el café. Eso podría hacerse en cualquier orden. En este modelo, tratamos a moler el café como una actividad hacer. En caso de que la máquina esté lista antes de que el café haya sido molido, se difiere el evento **máquina preparada** en el estado **Preparando**. De cualquier otra forma, la molienda del café podría ser interrumpida por el evento **máquina preparada**. Cuando se ha completado la molienda, la actividad hacer termina y la transición de finalización lleva al sistema al estado **Esperando**. Después de moler el café, es necesario esperar hasta que la máquina esté lita. Si el evento **máquina preparada** ya se ha producido durante el estado anterior, disparará la transición **máquina preparada** inmediatamente cuando se entra en el estado **Esperando**, de otra forma el sistema permanecerá en el estado **Esperando** hasta que se produzca el evento.

Un evento diferible es una forma de modelar un evento necesario que podría llegar antes o después de otro evento necesario.

evento temporal

Evento que denota la satisfacción de una expresión de tiempo, como la ocurrencia de un tiempo absoluto o el paso de una cantidad de tiempo determinada después de que un objeto entre en un estado.

Semántica

Un evento temporal es un evento que depende del paso del tiempo y por tanto depende de la existencia de un reloj. En el mundo real, el reloj es implícito. En una computadora, es una entidad física, y puede haber diferentes relojes en diferentes computadoras. El evento temporal es (esencialmente) un mensaje desde el reloj hasta el sistema. Observe que tanto el tiempo absoluto como el tiempo transcurrido pueden ser definidos respecto a un reloj del mundo real o a un reloj interno virtual. En el último caso, puede diferir para los distintos objetos.

Los eventos temporales pueden basarse en tiempo absoluto (la hora del día o el ajuste de un reloj dentro de un sistema) o en tiempo relativo (el tiempo transcurrido desde la entrada en un estado determinado o el suceso de un evento).

Notación

Los eventos temporales no se declaran como eventos con nombre como en el caso de las señales. En su lugar, se utiliza simplemente una expresión de tiempo como la desencadenadora de la transición.

Discusión

En cualquier implementación real, los eventos temporales no vienen del universo —vienen de algún objeto reloj dentro o fuera del sistema. Por tanto, se vuelven casi indistinguibles de las señales, especialmente en sistemas de tiempo real y distribuidos. En dichos sistemas se debe determinar el asunto de qué reloj se utiliza —no hay ninguna cosa como el “tiempo real”. (Tampoco existe en el universo real —pregunte a Einstein.)

excepción

Indicación de una situación inusual elevada en respuesta a errores de comportamiento por el mecanismo de ejecución subyacente o explícitamente elevada por una acción. La ocurrencia de una excepción aborta el flujo normal de control y origina la búsqueda de un manejador de excepción en un nodo de actividad que lo engloba.

Semántica

Una excepción es una indicación de que se ha producido una situación anormal que previene la ejecución normal. Una excepción se genera normalmente de forma implícita por el mecanismo de implementación subyacente en respuesta a un fallo durante la ejecución. Por ejemplo, un intento de utilizar un índice de array inválido para acceder a un elemento del array podría ser tratado como una excepción. Una excepción también puede ser elevada de forma explícita por una acción. Tales "excepciones software" se pueden utilizar para indicar situaciones molestas que no son bien manejadas por la secuencia de procesamiento normal.

Con el fin de que un programa pueda responder correctamente a una excepción, es necesario conocer qué excepciones se producen y, en muchos casos, parámetros adicionales de la excepción. Ambas necesidades se satisfacen modelando las excepciones como objetos. El tipo del objeto excepción indica la naturaleza de la excepción. Los atributos del objeto excepción representan los parámetros de la excepción. Puesto que diferentes tipos de objetos tienen diferentes atributos, las excepciones pueden representar exactamente tanta información como sea necesario (a diferencia de los mecanismos en algunos lenguajes de programación más antiguos en los que las excepciones estaban limitadas a valores únicos). Puesto que los tipos de excepción son clasificadores ordinarios, pueden formar jerarquías de generalización.

Algunas acciones causan excepciones si se producen condiciones específicas durante la ejecución de la acción. Por ejemplo, un intento de acceder a los atributos de un objeto nulo podría estar especificado para elevar una excepción de "referencia nula". Para cada tipo de excepción predefinida, se debe especificar las condiciones que la causan, junto con un tipo de excepción y una lista de valores de atributo proporcionados por el mecanismo de ejecución si se produce la excepción.

También hay una acción explícita **RaiseException** que permite al modelador elevar una excepción arbitraria. El argumento para esta acción es un único objeto. El tipo de este objeto representa el tipo de excepción, y sus atributos representan los parámetros de la excepción. Potencialmente, cualquier clase puede se utilizada como un tipo de excepción, aunque algunas implementaciones pueden requerir que los tipos de excepción sean descendientes de una clase de excepción raíz designada.

Cuando se produce una excepción, la ejecución de la acción actual se abandona sin generar ningún valor de salida. Se crea un token de excepción para representar los parámetros de la excepción.

El mecanismo de ejecución realiza una búsqueda para encontrar un manejador de excepción que sea capaz de manejar la excepción. Los manejadores de excepción están adjuntos a nodos de actividad. Cada manejador de excepción designa un tipo de excepción que maneja. Primero es examinada la acción que elevó la excepción para ver si tiene un manejador de excepción. Si lo tiene, se ejecuta el cuerpo del manejador de excepción con el token de excepción como valor de

entrada. Cuando se completa la ejecución del cuerpo del manejador de excepción, los valores de salida del manejador reemplazan a los (todavía no generado) valores de salida de la acción original, y se considera completa la ejecución de la acción. El manejador de excepción debe tener el mismo número y tipos de salida que la acción a la que protege. Después de la finalización del manejo de la excepción, se continúa la ejecución de las acciones subsecuentes como si la acción original se hubiera completado normalmente.

Un manejador de excepción captura una excepción cuyo tipo es el mismo o descendiente del tipo designado por el manejador. En otras palabras, un manejador de excepción puede ignorar distinciones entre subtipos cuando captura excepciones.

Si en la acción original no se encuentra un manejador para la excepción, la excepción se propaga al nodo de actividad que contiene la acción original, y así en los nodos anidados sucesivos hasta que se encuentra un nodo de actividad con un manejador de excepción que capture la excepción dada. Cada vez que la excepción se propaga hacia fuera, se termina la ejecución del nodo de actividad actual y no se generan los valores de salida normales. Si hay tokens concurrentes activos en un nodo de actividad, no se puede producir la propagación hasta que sean eliminados. La especificación UML no especifica un mecanismo para la eliminación de tokens concurrentes, y son compatibles con el mecanismo de manejo de excepción varios mecanismos semánticamente diferentes, incluyendo esperar a que termine la actividad concurrente, abortar automáticamente toda la actividad concurrente o proporcionar algún tipo de mecanismo de interrupción que permita al modelo terminar la actividad concurrente de una forma controlada. Esas elecciones son puntos de variación semántica en la especificación UML. La asunción por defecto es que la actividad concurrente es abortada por la excepción.

Una vez que se encuentra un manejador para el tipo de la excepción, se ejecuta su cuerpo del modo descrito previamente. Las salidas del manejador de excepción reemplazan las salidas originales del nodo de actividad protegido, y la ejecución continúa con los nodos sucesivos (pero la ejecución de los niveles embebidos han sido abortados irrevocablemente).

Si una excepción se propaga al nivel superior de una actividad sin haber sido manejada, se abandona la ejecución de toda la actividad. Si la actividad fue invocada de forma asíncrona, no se producen efectos adicionales y el manejo de excepción se completa. Si la actividad fue invocada de forma síncrona, la excepción se devuelve al marco de ejecución del emisor, donde se vuelve a elevar la excepción en el ámbito de la acción de llamada, donde puede ser manejada o propagada más allá.

Si una excepción se propaga a la raíz del sistema, el arquitecto del sistema probablemente ha hecho un trabajo pobre de diseño y el sistema probablemente se caerá.

Una operación puede declarar la excepción o las excepciones que podría elevar (incluyendo excepciones propagadas por otras operaciones a las que llame). Las excepciones representan parte de su signatura completa.

Notación

La ocurrencia de una excepción dentro de una actividad se puede representar como un pin de salida en el límite de la caja de actividad con un pequeño triángulo cerca de él. El símbolo del pin de salida se etiqueta con el tipo de la excepción (Figura 14.138).

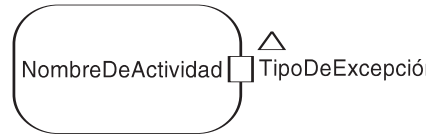


Figura 14.138 Excepción elevada por una actividad

Un manejador de excepción adjunto a un nodo de actividad se representa dibujando una flecha dentada ("rayo") desde el nodo protegido hasta un símbolo de pin de entrada en el límite del símbolo de actividad del manejador. El pin de entrada se etiqueta con el tipo de la excepción (Figura 14.139).

Figura 14.139 Manejador de excepción en una actividad

No parece haber una notación definida para la declaración de una excepción potencialmente elevada por una operación.

Ejemplo

La Figura 14.140 muestra un cómputo de matriz protegido por dos manejadores de excepción. Si se produce la excepción **MatrizSingular**, se sustituye una solución. Si se produce una excepción **Desbordamiento**, se sustituye otra solución. Si no se produce ninguna excepción, se utiliza la solución calculada. En cualquier caso, se imprime el resultado final.

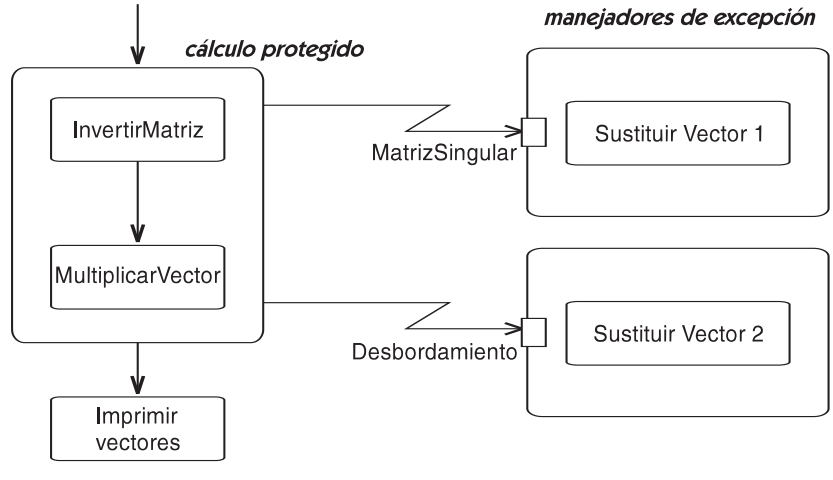


Figura 14.140 Manejadores de excepción en cálculo

Historia

En UML1, una excepción era considerada como un tipo de señal enviada entre objetos independientes. Aunque esto era una forma razonable de modelar mensajes entre objetos concurrentes, no concordaba con el concepto de excepción existente en muchos lenguajes de programación durante muchos años, en los que el manejo de excepciones era un mecanismo para escaparse del flujo normal de control para manejar situaciones inusuales sin confundir la lógica principal de un programa. UML2 ha cambiado al concepto estándar de excepción encontrado en lenguajes como C++.

Todavía hay unos pocos lenguajes residuales que describen una excepción como una señal. Esto es probablemente un descuido de los autores que debería ser ignorado.

executable (estereotipo de Artefacto)

Archivo de programa que se puede ejecutar en un sistema informático.

Véase artefacto, file.

exportar

En el contexto de los paquetes, hacer un elemento accesible fuera del espacio de nombres que lo engloba ajustando su visibilidad. Contrastar con acceso e importar, que hacen los elementos externos accesibles dentro de un paquete.

Véase también acceso, importar, visibilidad.

Semántica

Un paquete exporta un elemento estableciendo su visibilidad a un nivel que lo permita ser visto por otros paquetes (público para paquetes que lo importan, protegido para sus propios hijos).

expresión

Árbol estructurado que denota un valor (posiblemente un conjunto) cuando se evalúa en el contexto apropiado.

Semántica

Una expresión es un árbol de símbolos que se evalúa a un (posiblemente vacío) conjunto de instancias o valores cuando se ejecuta en un contexto. Una expresión no debería modificar el entorno en el que se evalúa. Una expresión tiene un tipo. El o los valores resultado son del tipo dado.

Una expresión consta de un símbolo y una lista de operandos, que pueden ser valores literales (como valores lógicos o números), instancias o subexpresiones. Una expresión es por tanto un árbol cuyas hojas son valores literales, instancias o cadenas de expresión en un lenguaje específico.

Un operando también puede ser una expresión opaca, es decir, una cadena que se evalúe en la sintaxis de un lenguaje especificado. 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 lenguaje humano. Por supuesto, si una expresión se escribe en lenguaje humano, no podrá ser evaluada automáticamente por una herramienta y debe ser simplemente para consumo humano.

Distintas subclases de expresiones producen distintos tipos de valores. Eso incluye expresiones lógicas, expresiones enteras y expresiones temporales.

Las expresiones aparecen en acciones, restricciones, condiciones de guarda y en otros lugares.

Notación

Una expresión se muestra como una cadena definida en un lenguaje. La sintaxis de la cadena es responsabilidad de una herramienta y un analizador lingüístico del lenguaje. La asunción es que el analizador puede evaluar cadenas en tiempo de ejecución para producir valores del tipo apropiado o que puede producir estructuras semánticas para capturar el significado de la expresión. Por ejemplo, una expresión lógica se evalúa a un valor verdadero o falso. El propio lenguaje es una herramienta de modelado, pero normalmente está implícito en el diagrama bajo la asunción de que la forma de la expresión hace su propósito claro. El nombre del lenguaje se puede colocar entre paréntesis antes de la cadena.

Ejemplo

```
mi.coste < autorización.costeMaximo
```

```
{OCL} i > j and mi.tamaño > i
```

Discusión

El conjunto de símbolos de operadores no está especificado en el documento UML, y por tanto el concepto de expresión no está totalmente especificado.

expresión Booleana

Una expresión que se evalúa mediante un valor Booleano. Es útil en las condiciones de guarda.

expresión de acción

Término obsoleto de UML1 substituido en su mayor parte por el comportamiento y la actividad hacer.

expresión de actividad

Este término de UML está obsoleto. Las actividades se modelan ahora directamente como construcciones de primera clase.

expresión de iteración

Especificación dentro de una interacción del rango del número de iteraciones de un bucle.

Semántica

En un bucle dentro de una interacción, el rango de iteraciones se puede especificar mediante unos valores mínimo y máximo.

Notación

Diagrama de secuencia. En un diagrama de secuencia, los límites del número de iteraciones de un bucle se incluyen entre paréntesis como parte de la etiqueta después de la palabra clave loop:

loop	Mínimo = 0, máximo ilimitado
loop (**`repetir`**)	Mínimo = máximo = repetir
loop (**`mínimo`**, **`máximo`**)	Límites mínimo y máximo explícitos

Además de los límites, se puede incluir una expresión lógica como guarda en una línea de vida. El bucle continuará iterando si la expresión es verdadera, pero iterará al menos el valor mínimo y no más del valor máximo, independientemente de la expresión de guarda.

La Figura 14.141 muestra un bucle con límites y una condición de guarda.

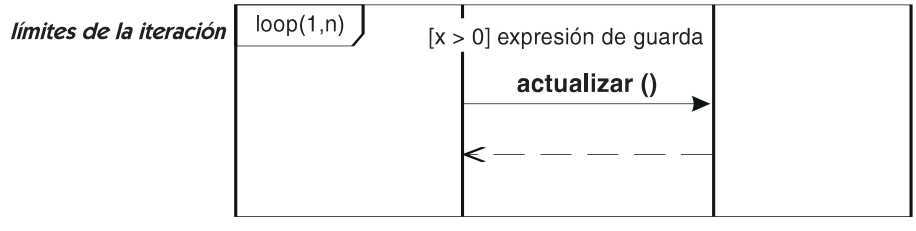


Figura 14.141 Límites de iteración en un bucle

Diagrama de comunicación. Detrás del número o del nombre que representa un nivel de anidamiento en el número de secuencia puede aparecer una expresión de iteración. La expresión de iteración representa ejecución condicional o iterativa. Representa la ejecución de cero o más mensajes dependiendo de las condiciones implicadas. Las posibilidades son

* [**`clausula-de-iteración`**]	Una iteración
[**`clausula-de-condición`**]	Una bifurcación

Una iteración representa una secuencia de mensajes. La **`clausula-de-iteración`** muestra los detalles de las variables de iteración y prueba, pero se puede omitir (en cuyo caso las condiciones de iteración no están especificadas). La **`clausula-de-iteración`** se expre-

sa en pseudocódigo o en un lenguaje de programación real. UML no prescribe su formato. Un ejemplo podría ser:

```
* [ i := 1..n ]
```

Una condición protege un mensaje cuya ejecución está sujeta a la verdad de la **`clausula-de-condición`**. La **`clausula-de-condición`** se expresa en pseudocódigo o en un lenguaje de programación. UML no prescribe su formato. Un ejemplo podría ser:

```
[ x > y ]
```

Observe que la notación de una bifurcación es la misma que la de una iteración quitando el asterisco. Puede pensar que es como una iteración restringida a una sola ocurrencia.

La notación de la iteración asume que los mensajes de la iteración se ejecutarán secuencialmente. También existe la posibilidad de ejecutarlos de forma concurrente. En ese caso la notación es un asterisco seguido por una línea vertical doble, que indica paralelismo (*||). Por ejemplo,

```
* [ i := 1..n ] || q[i].calculaPuntuación( )
```

Observe 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 engloba.

Ejemplo

Por ejemplo, un número de secuencia completo podría ser:

```
3b.1*[i:=1..n].2
```

que significa:

el tercer paso del procedimiento externo

el segundo hilo concurrente dentro del último paso

el primer paso dentro de ese hilo, que es un bucle desde 1 a n

el segundo paso dentro del bucle

La Figura 14.142 muestra un diagrama de comunicación sencillo en el que las iteraciones del bucle se ejecutarán concurrentemente.

expresión de tiempo

Expresión que se evalúa a un valor de tiempo absoluto o relativo. Utilizado en la definición de un evento temporal.

Semántica

La mayoría de las expresiones de tiempo son o bien un tiempo transcurrido después de la entrada a un estado o la ocurrencia de un tiempo absoluto particular. Se deben definir otras expresiones de tiempo de una forma ad hoc.

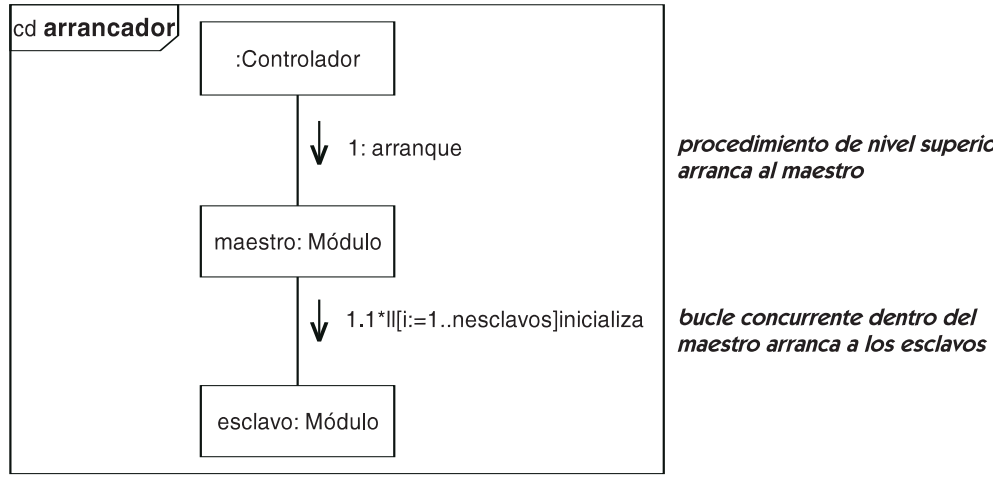


Figura 14.142 Expresión de interacción en un diagrama de comunicación

Una expresión de tiempo se define con un tiempo mínimo y máximo. Estos pueden ser los mismos para definir un solo punto en el tiempo.

Notación

Tiempo transcurrido. Un evento que denota el paso de cierta cantidad de tiempo después de la entrada en el estado que contiene la transición se anota con la palabra clave **after** seguida por una expresión que se evalúa (en tiempo de modelado) a una cantidad de tiempo.

after (10 segundos)

after (10 segundos sede la salida del estado A)

Si no se especifica ningún punto de inicio, entonces es el tiempo transcurrido desde la entrada en el estado que contiene la transición.

Tiempo absoluto. Un evento que denota la ocurrencia de un tiempo absoluto se anota con la palabra clave **when**, seguida por una expresión lógica, entre paréntesis, que implique al tiempo.

when (fecha= 1 de enero de 2000)

expresión de tipo

Expresión que se evalúa a una referencia a uno o más tipos de datos. Por ejemplo, una declaración de tipo de atributo en un lenguaje de programación típico es una expresión de tipo. La sintaxis de las expresiones de tipo no está especificada en UML.

Ejemplo

En C++ las siguientes son expresiones de tipo:

```
Persona*
Orden[24]
Boolean (*) (String,int)
```

expresión opaca

Una cadena de texto expresada en un lenguaje específico que, cuando es evaluado, rinde un valor.

Semántica

Una expresión opaca es una convención para permitir que las expresiones en otros lenguajes sean usadas en UML cuando se necesitan valores. Ya que los otros lenguajes están, por definición, fuera de UML, son opacos con respecto a UML. Todo lo que se sabe es que, cuando se evalúa, producen los valores; opcionalmente se puede especificar un tipo para los valores.

El modelo de UML incluye una cadena de texto y el nombre del lenguaje designado. El lenguaje OCL permite explícitamente otros lenguajes, tales como lenguajes de programación, y pueden usarse a la discreción de la herramienta modelada.

Un lenguaje importante (familia de idiomas, realmente) es el lenguaje natural. Esto es una buena forma de expresar los modelos de nivel superior para ser comunicado a los humanos.

Notación

Una expresión opaca se expresa en la sintaxis del lenguaje designado. A menudo el lenguaje destino es implícito dentro del modelo de un usuario particular, pero puede declararse poniendo el nombre del lenguaje (el nombre oficial como se expresa en su definición normal) entre llaves delante de la expresión del texto. Por ejemplo, una restricción de UML se pone a menudo entre llaves, así que un ejemplo de una restricción de OCL podría ser:

```
{ {OCL} self.size > 0 y self.size < n}
```

extender

Relación entre un caso de uso de *extensión* y un caso de uso *base* (extendido), que especifica cómo el comportamiento definido para el caso de uso de extensión se puede insertar en el comportamiento definido por el caso de uso base. Los casos de uso de extensión modifican incrementalmente el caso de uso base de una 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 de extensión con un caso de uso base (Figura 14.143). El caso de uso de extensión en esta relación no es necesariamente un clasificador instanciable por separado. En cambio, consiste un uno o más segmentos que describen secuencias de comportamiento adicional que modifican incrementalmente el comportamiento del caso de uso base. Cada segmento en un caso de uso de extensión se puede insertar en una ubicación separada del caso de uso base. La relación de extensión tiene una lista de nombres de puntos de extensión, iguales en número al número de segmentos en el caso de uso de extensión. Cada punto de extensión debe estar definido en el caso de uso base. Cuando la ejecución de una instancia de un caso de uso alcanza una ubicación en el caso de uso base referenciada por el punto de extensión y se satisfacen todas las condiciones de la extensión, entonces la ejecución de la instancia puede transferirse a la secuencia de comportamiento del correspondiente segmento en el caso de uso de extensión; cuando la ejecución del segmento de extensión ha finalizado, el control vuelve al caso de uso original en el punto referenciado. *Véase* la Figura 14.144 para ver un ejemplo de secuencias de comportamiento.

Se pueden aplicar varias relaciones de extensión al mismo caso de uso base. Una instancia de un caso de uso puede ejecutar más de una extensión durante su vida. Si varios casos de uso extienden a un caso de uso base en el mismo punto de extensión, entonces su orden relativo de ejecución no es determinista. Incluso puede haber varias relaciones de extensión entre los mismos casos de uso de extensión y base, a condición de que el caso de uso de extensión se inserte en diferentes ubicaciones del caso de uso base. Los casos de uso de extensión incluso pueden tener otros casos de uso de extensión anidados.

Un caso de uso de extensión en una relación de extensión puede acceder y modificar atributos definidos por el caso de uso base. Sin embargo, el caso de uso base no puede ver los casos de

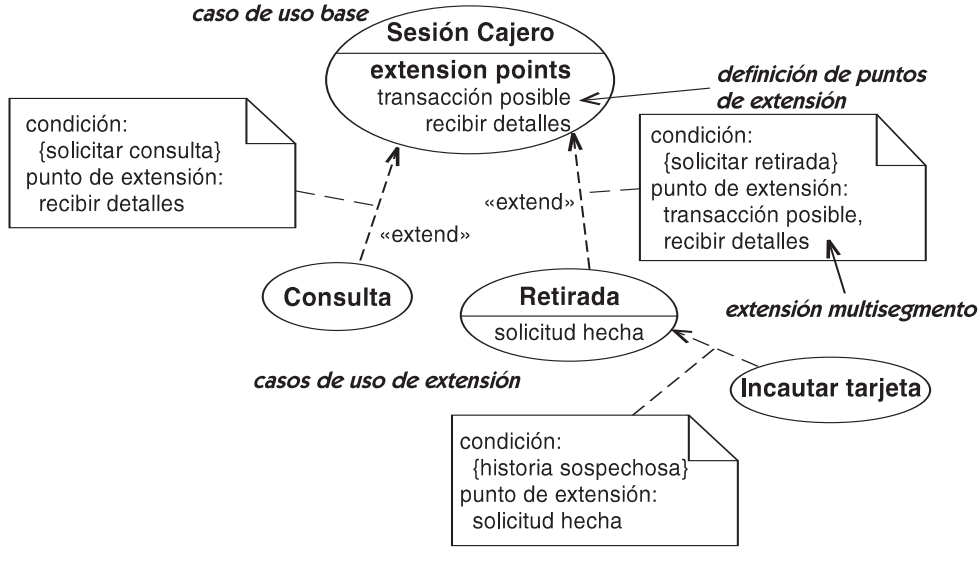


Figura 14.143 Relación de extensión

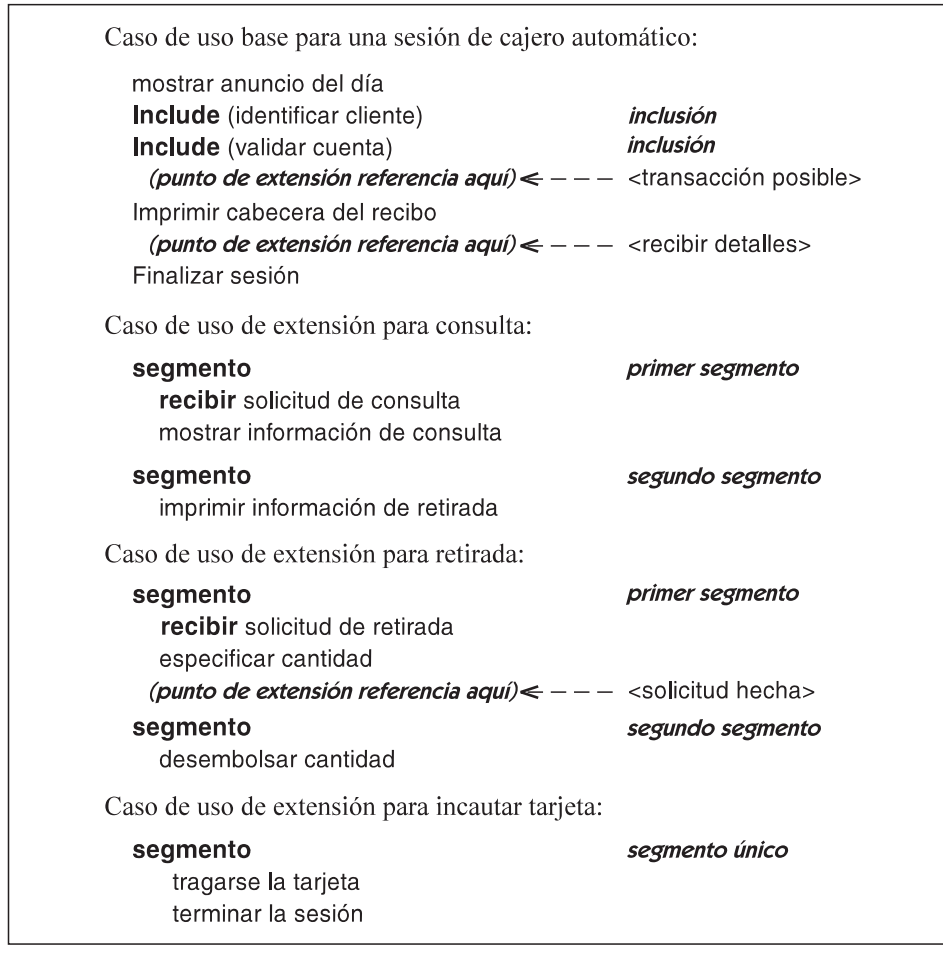


Figura 14.144 Secuencias de comportamiento para casos de uso

uso de extensión y no puede acceder a sus atributos u operaciones. El caso de uso base define un marco de trabajo modular en el que se pueden añadir casos de uso de extensión, pero la base no tiene visibilidad sobre dichos casos de uso de extensión. Los casos de uso de extensión modifican implícitamente el comportamiento del caso de uso base. Observe la diferencia con la generalización de casos de uso. Con la relación de extensión, los efectos del caso de uso de extensión se añaden a los efectos de caso de uso base en una instanciación del caso de uso. Con la generalización, los efectos del caso de uso hijo se añaden a los efectos del caso de uso padre en una instanciación del caso de uso hijo, mientras que una instanciación del padre no obtiene los efectos del caso de uso hijo.

Un caso de uso de extensión puede extender más de un caso de uso base, y un caso de uso base puede ser extendido por más de un caso de uso de extensión. Esto no indica ninguna relación entre los casos de uso base.

Un caso de uso de extensión por sí mismo puede ser la base en una relación de extensión, inclusión o generalización.

Estructura (de caso de uso de extensión)

Un caso de uso de extensión contiene una lista de uno o más *segmentos de inserción*, cada uno de los cuales es una secuencia de comportamiento.

Estructura (de caso de uso base)

Un caso de uso base define un conjunto de puntos de extensión, cada uno de los cuales referencia una ubicación o conjunto de ubicaciones en el caso de uso base donde se puede insertar comportamiento adicional.

Estructura (de relación extender)

La relación de extensión tiene una lista de nombres de puntos de extensión, que deben estar presentes en el caso de uso base. El número de nombres debe ser igual al número de segmentos en el caso de uso de extensión.

La relación de extensión puede tener una condición, una expresión en términos de atributos del caso de uso base o la ocurrencia de eventos como la recepción de una señal. La condición determina si el caso de uso de extensión es realizado cuando la ejecución de una instancia del caso de uso alcanza una ubicación referenciada por el primer punto de extensión. Si la condición no existe, se considera que siempre es cierta. Si se satisface la condición para un caso de uso de extensión, entonces se procede a la ejecución del caso de uso de extensión. Si el punto de extensión referencia a varias ubicaciones en el caso de uso base, el caso de uso de extensión se puede ejecutar en cualquiera de ellos.

La extensión se puede realizar más de una vez si la condición permanece siendo verdadera. Se ejecutan todos los segmentos del caso de uso de extensión el mismo número de veces. Si se debe restringir el número de ejecuciones, debería definirse la condición de forma adecuada.

Semántica de ejecución

Cuando una instancia de un caso de uso que está realizando el caso de uso base alcanza una ubicación en el caso de uso base que es referenciada por una relación de extensión, entonces se evalúa la condición en la relación de extensión. Si no hay condición o ésta es verdadera, entonces se realiza el caso de uso de extensión. En muchos casos, la condición incluye la ocurrencia de un evento o la disponibilidad de valores necesitados por el propio segmento del caso de uso de extensión —por ejemplo, una señal de un actor que comienza el segmento de extensión. La condición puede depender del estado de la instancia del caso de uso, incluyendo valores de atributos del caso de uso base. Si el evento no se produce o la condición es falsa, no comienza la ejecución del caso de uso de extensión. Cuando se finaliza la realización de un segmento de extensión, la instancia del caso de uso reanuda la realización del caso de uso base en la ubicación en que quedó.

Si se satisface la condición, se pueden realizar inmediatamente inserciones adicionales del caso de uso de extensión. Si el punto de extensión referencia varias ubicaciones en el caso de uso base, la condición puede ser satisfecha en cualquiera de ellos. La condición puede volverse verdadera en cualquier ubicación dentro del conjunto.

Si hay más de una secuencia de inserción en un caso de uso de extensión, la extensión se ejecuta si la condición es verdadera en el primer punto de extensión. La condición no se vuelve a

evaluar para los siguientes segmentos, que se insertan cuando el caso de uso alcanza las correspondientes ubicaciones dentro del caso de uso base. La instancia del caso de uso reanuda la ejecución del caso de uso base entre las inserciones en diferentes puntos de extensión. Una vez que empiezan, se deben realizar todos los segmentos.

Observe que en general, un caso de uso es una máquina de estados no determinista (como en una gramática) más que un procedimiento ejecutable. Esto es porque las condiciones pueden incluir la ocurrencia de eventos externos. Para realizar un caso de uso como una colaboración de clases se puede requerir una transformación en mecanismos explícitos de control, así como la implementación de una gramática requiere una transformación a una forma ejecutable que sea eficiente pero más complicada de entender.

Observe que los términos *base* y *extensión* son relativos. Un caso de uso de extensión puede servir a su vez como base para otra relación de extensión. Esto no presenta ninguna dificultad, y todavía son de aplicación las reglas anteriores —las inserciones se anidan. Por ejemplo, suponga que el caso de uso B extiende al caso de uso A en un punto de extensión x, y suponga que el caso de uso C extiende al caso de uso B en un punto de extensión y (Figura 14.145). Cuando una instancia de A llega al punto de extensión x, comienza a realizar el caso de uso B. Cuando la instancia llega al punto de extensión y dentro de B, comienza a realizar el caso de uso C. Cuando se finaliza la ejecución del caso de uso C, se continúa la realización del caso de uso B. Cuando se finaliza el caso de uso B, se continúa la realización del caso de uso A. Es similar a llamadas a procedimientos anidados o cualquier otra construcción anidada.

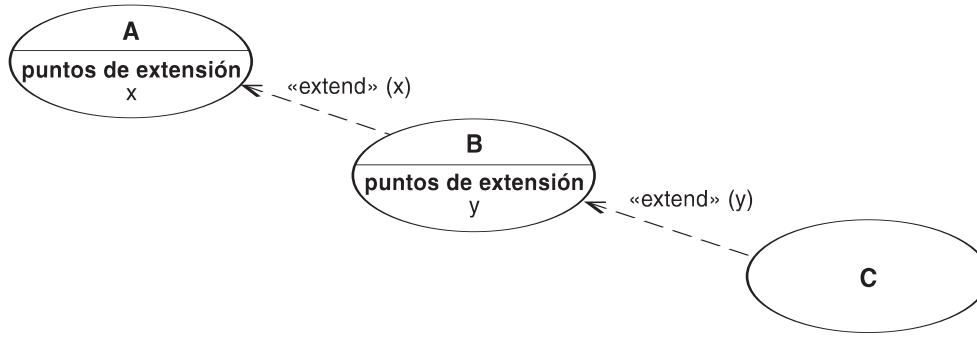


Figura 14.145 Relaciones extender anidadas

Notación

Se dibuja una flecha discontinua desde el símbolo del caso de uso de extensión hasta el símbolo del caso de uso base con una cabeza de flecha en la base. Se coloca la palabra clave **«extend»** en la flecha. Puede aparecer una lista de nombres de puntos de extensión entre paréntesis después de la palabra clave.

La Figura 14.143 muestra casos de uso con relaciones de extensión. No hay notación oficial para especificar secuencias de comportamiento. La Figura 14.144 es simplemente una sugerencia.

Discusión

Las relaciones de extensión, inclusión y generalización añaden comportamiento a un caso de uso inicial. Tienen muchas similitudes, pero en la práctica es conveniente separarlas. La Tabla 14.2 compara los tres puntos de vista.

Tabla 14.2 Comparación de las relaciones entre casos de uso

Propiedad	*Extender*	*Incluir*	*Generalización*
Comportamiento base	Caso de uso base	Caso de uso base	Caso de uso padre
Comportamiento añadido	Caso de uso de extensión	Caso de uso de inclusión	Caso de uso hijo
Dirección de referencia	Caso de uso de extensión referencia al caso de uso de base	Caso de uso base referencia al caso de uso de inclusión	Caso de uso hijo referencia al caso de uso padre
¿Base modificada por la adición?	La extensión modifica implícitamente el comportamiento de la base. La base debe estar bien formada sin la extensión, pero si la extensión está presente, una instanciación de la base puede ejecutar la extensión.	La inclusión modifica explícitamente el efecto de la base. La base puede estar o no bien formada sin la inclusión, pero una instanciación de la base ejecuta la inclusión.	El efecto de ejecutar al padre no se ve afectado por el hijo. Para obtener los efectos de la adición, se debe instanciar al hijo, no al padre.
¿La adición es instanciable?	El caso de uso de 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.
¿La adición puede acceder a los atributos de la base?	La extensión puede acceder y modificar el estado de la base	La inclusión puede acceder al estado de la base. La base puede proporcionar los atributos apropiados esperados por la inclusión.	El hijo puede acceder y modificar el estado de la base (por el mecanismo habitual de herencia).
¿La base puede ver a la adición?	La base no puede ver la extensión 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

Observe que los términos segmento y ubicación no son precisos, así que deben considerarse puntos de variación semántica sujetos a implementación.

extensión (extent)

Conjunto de instancias descritas por un clasificador.

Contrastar con intención.

Semántica

Un clasificador, como una clase o una asociación, tiene tanto una descripción (su intención) como un conjunto de instancias a las que describe (su extensión). El propósito de la intención es especificar las propiedades de las instancias que componen la extensión. No hay ninguna asunción de que la extensión sea una manifestación física o de que se pueda obtener en tiempo de ejecución. Por ejemplo, incluso en principio un modelador no debería asumir que el conjunto de objetos que son instancias de una clase se puedan obtener.

extensión (extension)

Adjuntar un estereotipo a una metaclase, extendiendo por tanto la definición de la metaclase para incluir el estereotipo.

Semántica

Una extensión es una relación entre un estereotipo y una metaclase UML (una clase en la definición del propio modelo UML, como una **Clase** u **Operación**). Indica que las propiedades definidas en el estereotipo pueden ser aplicadas a las instancias de la metaclase que lleve el estereotipo. Observe que una instancia de una metaclase es una clase ordinaria como se define en un modelo de usuario, de forma que definir una extensión a una metaclase significa que un elemento del modelo de usuario de la metaclase dada tiene propiedades adicionales en tiempo de modelado cuyos valores puede especificar el modelador. Una extensión puede ser opcional o requerida. Si una extensión es requerida, todas las instancias de la metaclase deben llevar la extensión. Si una extensión es opcional, las instancias de la metaclase pueden o no llevar la extensión. Más aún, una instancia de la metaclase puede añadir o eliminar la extensión durante su vida.

Notación

Una extensión se representa mediante una flecha continua desde el rectángulo del estereotipo al rectángulo de la metaclase con una cabeza de flecha sólida de color negro en el extremo asociado a la metaclase. Se utiliza la palabra clave **{required}** para indicar una extensión requerida; en caso contrario la extensión es opcional. Se puede colocar la palabra clave **«metaclass»** en el rectángulo de la metaclase.

Ejemplo

La Figura 14.146 muestra la extensión de la metaclase **Clase** por el estereotipo **Autoría**, que define las propiedades **autor** y **fecha de creación**. La extensión de **Clase** indica que una clase defi-

nida en un modelo de usuario puede llevar el estereotipo **Autoría**. Puesto que la palabra clave **{required}** no está presente, el uso del estereotipo es opcional. El modelo que define dicha clase puede definir valores para dichas propiedades. Por ejemplo, pueden identificar a la persona que creó la clase y la fecha en que fue creada.

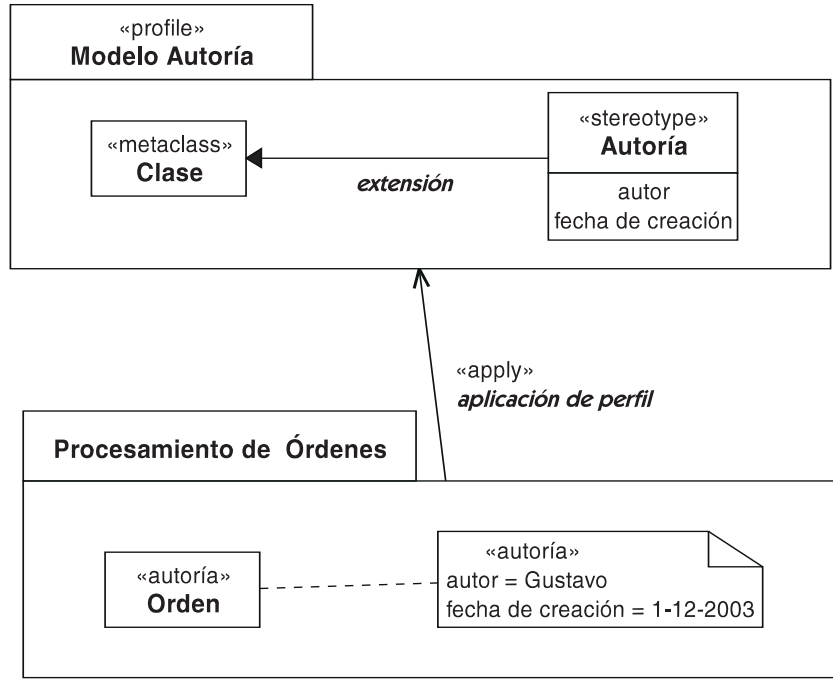


Figura 14.146 Extensión de una metaclase por un estereotipo

Discusión

La palabra *extensión* se usa de forma diversa para la relación extender de casos de uso, la extensión de clases por estereotipos, y las instancias de una clase en tiempo de ejecución. Este solapamiento es algo confuso puesto que no son los mismos conceptos.

extremo de la asociación

Una parte de una asociación que define la participación de una clase en la asociación. Se puede conectar una clase a más de un extremo en la misma asociación. Los extremos de la asociación dentro de una asociación tienen distintas posiciones, tienen nombres y, en general, no son intercambiables. Un extremo de la asociación no tiene una existencia independiente ni un significado fuera de su asociación.

Véase también asociación.

Semántica

Estructura

Un extremo de la asociación tiene una referencia a un clasificador destino. Define la participación del clasificador en la asociación. Una instancia de la asociación (un enlace) debe contener una instancia del clasificador dado, o de uno de sus descendientes, en una posición dada. Los hijos de un clasificador heredan la participación en una asociación.

Un atributo es una asociación degenerada en la cual un extremo de la asociación es propiedad de un clasificador que se encuentra conceptualmente en el otro extremo no modelado de la asociación. Una asociación es más apropiada en una situación de modelado en la cual todos los clasificadores participantes tienen una importancia bastante similar o la relación se puede ver en más de una dirección. Un atributo es más apropiado en una situación en la cual la relación siempre se ve en una única dirección.

Debido a la similitud subyacente de los extremos de la asociación y los atributos, la mayoría de sus propiedades de modelado se combinan en un único elemento de modelado, denominado propiedad estructural. *Véase* propiedad para una lista completa de configuraciones de modelado tanto para el extremo de la asociación, como para el atributo.

Notación

Véase propiedad para la mayor parte de la notación de los extremos de la asociación. Las propiedades de los extremos de la asociación se muestran situando adornos gráficos o de texto sobre o junto al final de las líneas (Figura 14.147).

Si hay múltiples adornos en un único extremo de la asociación, estos se muestran desde el final de la línea hacia el símbolo de clase, en el siguiente orden (Figura 14.23):

calificador

símbolo de agregación o de composición

flecha de navegación

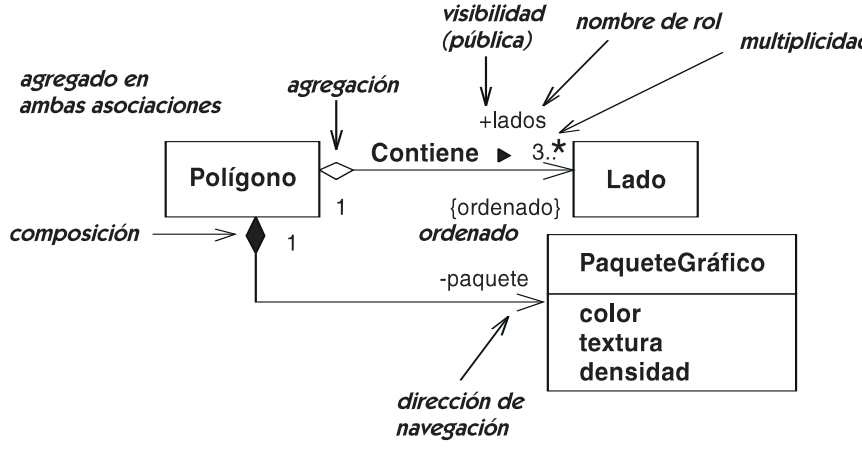


Figura 14.147 Diversos adornos para los extremos de la asociación

Los nombres de los extremos y las cadenas de multiplicidad se deben situar cerca del final de la ruta de forma que no creen confusión con otra asociación distinta. Se deben colocar en cualquier lado de la línea. Se intenta pedir que siempre se sitúen en un lado de la línea (en el sentido de las agujas del reloj o en el contrario), pero esto a veces no se puede tener en cuenta por la necesidad de claridad en una presentación muy sobrecargada. Un nombre de extremo y una multiplicidad se puede colocar en lados opuestos del mismo rol, o se pueden colocar juntos (por ejemplo, *** empleado**).

Otras configuraciones se muestran como cadenas de propiedad entre llaves ({}). Varias cadenas de propiedad se pueden colocar dentro del mismo par de llaves separadas por comas, o se pueden mostrar por separado. Las etiquetas de las cadenas de propiedad se deben situar cenca de los extremos de la ruta, de forma que no interfieran con los nombres de los extremos o las cadenas de multiplicidad.

extremo del enlace

Instancia de un extremo de asociación.

facade

El estereotipo facade de UML1 ha sido retirado.

fase de transición

Cuarta fase de un proceso de desarrollo de software, durante la que se configura la implementación del sistema 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 no fueron completadas en las fases previas.

Véase proceso de desarrollo.

file (estereotipo de Artefacto)

Archivo físico en el contexto del sistema de destino.

Véase artefacto.

finalización

Una especificación de la terminación de la instancia representada por una línea de vida en una interacción.

Semántica

Una parada indica la destrucción de la instancia modelada. No puede existir ninguna especificación de la instancia después de la parada.

Notación

Una parada se muestra por una X en una línea de vida. Un mensaje puede partir del símbolo (si el objeto se destruyó) o termina en el símbolo (si el objeto es destruido por la acción de otro objeto).

La Figura 14.100 muestra ejemplos de parada.

flag de aislamiento

Indicador que garantiza que la ejecución de un nodo de actividad no entrará en conflicto con la ejecución de otros nodos que comparten acceso a los mismos objetos.

Semántica

Varias ejecuciones concurrentes no tienen dependencias de secuenciación si los objetos a los que acceden son disjuntos, puesto que un cambio en el orden relativo de su ejecución no cambiará los resultados. Si nodos de actividad diferentes comparten acceso a los mismos objetos, el orden de ejecución relativo puede afectar a los resultados. Se puede evitar la indeterminación ordenando totalmente la secuencia de ejecución. Sin embargo, a veces, la ejecución es determinada si cada nodo de actividad puede ejecutarse completamente sin intercalar su ejecución con los otros nodos de actividad, independientemente de qué nodo de actividad se ejecute primero. Por ejemplo, considere dos fragmentos que cada uno incrementa el valor de un contador. El resultado es determinado si cualquier incremento se ejecuta por completo, pero puede ser defectuoso si dos actividades acceden simultáneamente a la variable y después escribe de vuelta el valor incrementado.

El establecimiento del indicador de aislamiento es una sentencia ejecutable de que un nodo de actividad se va a ejecutar de forma que no se produzca solapamiento con otras ejecuciones. No es una aserción; tiene un efecto ejecutable. No requiere que las ejecuciones sean secuenciales, siempre que el resultado no se vea afectado por ninguna ejecución concurrente. En la práctica, la implementación eficiente del indicador de aislamiento podría implicar cierto análisis previo de secuencias de ejecución potenciales para identificar fuentes de conflicto y sólo secuenciar las actividades potencialmente conflictivas.

flujo

Término genérico utilizado para describir varios tipos de información entre orígenes y destinos, incluyendo datos y foco de ejecución (control). El modelo de actividad proporciona conceptos y notación para modelar flujo de datos y flujo de control explícitamente. Otros modelos proporcionan otras formas de modelar flujo de información de forma implícita.

Discusión

UML2 contiene un modelo de flujo de información autónomo que está destinado a añadir una capacidad rudimentaria de modelar flujo de datos, pero que no recomendamos. El modelo de actividad proporciona una capacidad de modelar flujos mucho más completa y que está adecuadamente integrado con el resto de UML.

flujo de control

Restricción de secuenciación en la ejecución de nodos de actividad.

Véase también flujo de datos.

Semántica

Una actividad es un gráfico de nodos de actividad y transiciones de actividad. Un nodo de actividad es una acción básica, un nodo objeto, un nodo de control o un agrupamiento de nodos a alto nivel. Una transición de actividad es una relación que gobierna el flujo de control y datos entre dos nodos. Un nodo puede tener transiciones de entrada y de salida. Un nodo sólo puede empezar la ejecución cuando todos los prerrequisitos especificados por sus transiciones de entrada están satisfechos.

Un flujo de control es una transición que especifica flujo de control en vez de datos. Si un nodo tiene un flujo de control de entrada, puede empezar su ejecución sólo después de que el nodo al otro lado de la transición haya completado su ejecución. (Ciertos tipos de nodos de control pueden empezar la ejecución cuando un subconjunto designado de nodos tiene tokens.) El flujo de control es una transición que sale del otro nodo. En otras palabras, cuando un nodo completa su ejecución, se coloca un token de control en cada uno de sus flujos de control de salida. Si el flujo de control sirve como una transición de entrada para otro nodo, el nodo se activará después de que todas sus transiciones de entrada contengan tokens. Si un nodo tiene varias transiciones de entrada, el nodo sincroniza la ejecución de todos sus hilos de entrada. Algunos tipos de nodos de control se ejecutan cuando un subconjunto de sus transiciones de entrada contiene tokens; las reglas de activación se describen para cada tipo de nodo de control.

Un flujo de control puede ser considerado como una transición de flujo de datos degenerada, sin ningún dato, de forma que las reglas de secuenciación se pueden expresar en términos de tokens en las transiciones.

Notación

El flujo de control en un diagrama de actividad se representa como una flecha continua desde un nodo a otro. El flujo de control es una salida del nodo en la cola de la flecha y una entrada del nodo en la punta de la flecha.

Ejemplo

En la Figura 14.148, **recibir orden** tiene dos flujos de control de salida. Cuando se completa, **enviar orden** y **facturar cliente** se activan puesto que cada uno tiene un solo flujo de control de entrada. Esas actividades pueden ejecutarse concurrentemente. Cuando ambas se completan, **enviar confirmación** se activa.

Historia

Los flujos de control y de datos se han añadido como parte de la gran expansión del modelo de actividad de UML2.

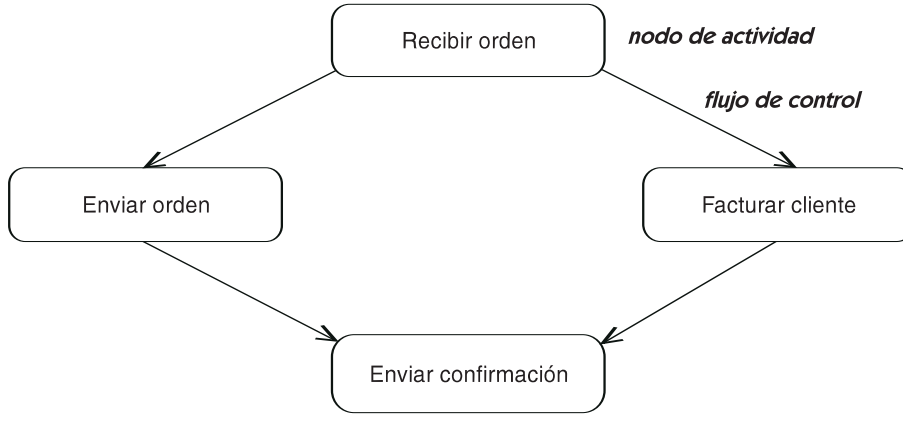


Figura 14.148 Flujos de control

flujo de datos

Relación en una actividad entre nodos de actividad que proporcionan y consumen valores de datos e, implícitamente, la restricción de secuenciación de control implicada por la relación.

Semántica

Una actividad es un gráfico de nodos de actividad y transiciones de actividad. Un nodo de actividad es una acción básica o un grupo de acciones de mayor nivel. Una transición de actividad es una relación que gobierna el flujo de datos y control entre dos nodos. Un nodo puede tener transiciones de entrada y de salida. Un nodo puede empezar la ejecución sólo cuando todos los prerrequisitos especificados por sus transiciones de entrada han sido satisfechos, incluyendo la presencia de todos los valores de datos requeridos y todos los tokens de control requeridos. Las acciones requieren tokens en todas las transiciones de entrada. Algunos tipos de nodos de control requieren entradas en un subconjunto designado de transiciones de entrada.

Una transición de flujo de objeto especifica el flujo de datos más que sólo flujo de control. Si un nodo tiene una transición de flujo de objeto de entrada, el valor de los datos es un parámetro de entrada del nodo. El nodo puede comenzar su ejecución sólo después de que se haya producido un valor de datos por el nodo que se encuentra en el extremo proveedor de la transición. La transición de flujo de objeto es una transición saliente del otro nodo. En otras palabras, cuando un nodo finaliza su ejecución, se coloca un token de datos en cada una de sus transiciones de flujo de objeto de salida. Si un nodo sucesor utiliza la transición de flujo de objeto como un parámetro de entrada, el nodo se activa después de que los valores de datos estén disponibles en todas sus transiciones de flujo de objeto y de que los tokens de control estén disponibles en todas sus transiciones de flujo de control de entrada (con la excepción de ciertos tipos de nodos de control que unen de forma explícita flujo de control desde una o varias entradas).

Los valores de datos en las transiciones son copias individuales que no son ni compartidas ni almacenadas en un repositorio central. Los valores pueden incluir referencias a objetos, pero en ese caso el objeto está separado del valor y no forma parte del token de datos. El destino de una transición de flujo de objeto utiliza el valor consumiéndolo.

El flujo de datos implica flujo de control, puesto que un nodo no se puede ejecutar hasta que todos sus valores de datos de entrada estén disponibles. Puede considerarse un flujo de control como una transición de flujo de objeto degenerada sin datos, de forma que las reglas de secuenciación se pueden expresar en términos de tokens en las transiciones.

Notación

El flujo de datos se representa por medio de una flecha continua entre los pines de los nodos de actividad. Alternativamente, también se puede representar por un símbolo de nodo objeto (rectángulo) entre dos nodos de actividad, con una flecha continua desde el nodo origen al nodo objeto y otra flecha continua desde el nodo objeto al nodo destino. *Véase* la Figura 14.149.

Véase flujo de objetos, nodo objeto y pin para ver la notación y los ejemplos.

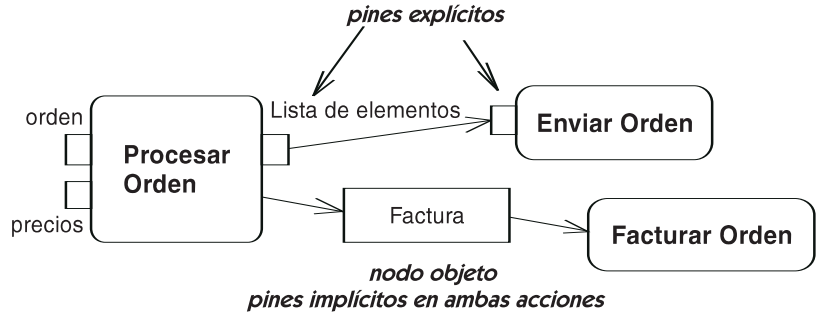


Figura 14.149 Flujos de datos

Historia

Los flujos de control y de datos se han añadido como parte del modelo de actividad enormemente expandido en UML2.

flujo de información

Declaración de intercambio de información entre dos objetos de tipos dados.

Semántica

Un flujo de información especifica un clasificador origen y otro destino y uno o más clasificadores que representan tipos de información que se pueden enviar desde un objeto del tipo origen a un objeto del tipo destino.

Notación

Se representa como una línea discontinua (una dependencia) con la palabra clave “flow”.

Discusión

El concepto de flujo de información y elemento de información son tan imprecisos como para cuestionar su utilidad. En el modelo de actividad se encuentra un modelo de control y flujo de datos más robusto.

flujo de objetos

Un tipo de transición de actividad es una actividad que representa el flujo de valores entre dos nodos de actividad.

Véase también control de flujo, nodo de objeto.

Semántica

Un flujo del objeto es un tipo de transición de actividad que conecta dos nodos de actividad, normalmente un nodo ejecutable y un nodo de objeto. (Para ser preciso, se conectan a los pines de los nodos ejecutables, pero en efecto conectan los nodos.) Representa la producción de un valor por una acción origen o el consumo de un valor por una acción destino. También puede conectar dos nodos de objeto. *Véase* nodo del objeto para más detalles.

Un multicast conecta un nodo de objeto a múltiples nodos de actividad; un multireceptor conecta múltiples nodos de actividad a un nodo de objeto. Se tiene la intención de que permitan implementar arquitecturas de publicación-suscripción, pero pueden usarse con carácter general.

Notación

Un flujo del objeto se muestra como una flecha sólida que conecta dos nodos de actividad. Puede tener una condición de guarda que indica que el flujo sólo puede ocurrir si la condición es satisfecha.

Se pueden aplicar las palabras clave “multicast” y “multireceive” a la flecha (pero no ambas a la vez).

foco de control

Símbolo en un diagrama de secuencia que muestra el periodo de tiempo durante el que un objeto está realizando una acción, o bien directamente o bien mediante un procedimiento subordinado. En UML2 ahora se conoce como *especificación de una ejecución*.

Véase especificación de una ejecución.

focus (estereotipo de Clase)

Clase que define la lógica o control fundamentales en conjunción con una o más clases auxiliares que la dan soporte proporcionando mecanismos subordinados.

fragmento combinado

Construcción dentro de una interacción que consta de un operador palabra clave y uno o más operandos interacción, cada uno de los cuales es un fragmento de una interacción. Se representa como una región anidada dentro de un diagrama de secuencia.

Semántica

Las interacciones incluyen varias construcciones para representar comportamiento circunstancial, como condicionales, bucles y demás. Puesto que muchas de esas construcciones comparten información estructural, son agrupadas juntas como tipos de fragmentos combinados. Un fragmento combinado tiene una palabra clave que especifica qué construcción es.

Dependiendo de la palabra clave hay uno o más operandos embebidos, cada uno de los cuales es un subfragmento estructurado de la interacción global.

Un fragmento combinado también puede tener cero o más entradas, que especifican la interfaz entre el fragmento combinado y otras partes de la interacción.

La palabra clave debe ser una de las siguientes:

alt	Una condición tiene múltiples operandos. Cada operando tiene una condición de guarda. La ausencia de la guarda implica una condición verdadera. La condición **else** es verdadera si ninguna otra guarda se evalúa a verdadera. Se ejecuta sólo uno de los operandos cuya guarda se evalúa a verdadero, a no ser que ninguna guarda sea verdadera. Si se evalúa a verdadero más de un operando, la elección puede ser no determinista.
assert	Una afirmación tiene un subfragmento. Si la ejecución alcanza el comienzo de la construcción, debe producirse el comportamiento del subfragmento. A menudo se combina con ignorar o considerar para declarar el comportamiento de un tipo de mensaje particular.
break	Una construcción de ruptura tiene un subfragmento con una condición de guarda. El subfragmento se ejecuta si la guarda es verdadera, y el resto del fragmento de interacción rodeado no se ejecuta. De lo contrario la ejecución continúa de forma normal.
consider	Una construcción considerar tiene un subfragmento y una lista de tipos de mensaje. Sólo se representan dentro del subfragmento los tipos de mensaje listados. Esto significa que pueden producirse otros mensajes en el sistema real, pero la interacción es una abstracción que los ignora. Es lo contrario a la construcción ignorar.
critical	Una región crítica tiene un subfragmento. Una secuencia de eventos en una sola línea de vida en la región crítica no debe ser intercalada con cualesquiera otros eventos en otras regiones. No hay ninguna restricción sobre eventos en otras líneas de vida, así que esto no excluye la actividad concurrente que no afecta a la región crítica. Esta construcción anula una construcción paralela que de otra forma podría permitir el intercalado.

ignore — Una construcción ignorar tiene un subfragmento y una lista de tipos de mensajes. Sólo se representan dentro del subfragmento los tipos de mensajes listados. Esto significa que los tipos declarados pueden producirse en el sistema real, pero la interacción es una abstracción que los ignora. Por ejemplo, cualesquiera restricciones en los tipos dentro de la actividad mayor no se aplican a las apariciones reales de los tipos en el fragmento ignorar, puesto que los tipos son invisibles para los propósitos de modelado en el fragmento. Es lo contrario a la construcción considerar.

loop — Una construcción de bucle tiene un subfragmento con una guarda. La guarda debe tener un valor mínimo y máximo, así como una condición lógica. El subfragmento se ejecuta siempre que la condición de guarda se evalúe a verdadera, pero se ejecuta al menos el valor mínimo, y no más del valor máximo. Si la guarda no está presente, se trata como verdadera, y el bucle depende de los valores de repetición.

neg — Una construcción negativa tiene un subfragmento. El subfragmento define secuencias de ejecución que no pueden suceder. En muchos casos, el modelo puede no listar todas las secuencias que están prohibidas, por lo que es arriesgado sacar conclusiones acerca de la ausencia de construcciones negativas.

opt — Una construcción opcional tiene un subfragmento con una condición de guarda. El subfragmento se ejecuta si la guarda es verdadera y no se ejecuta en cualquier otro caso. Esta construcción es equivalente a una condicional con un solo subfragmento con guarda y una cláusula else vacía.

par — Una construcción paralela tiene dos o más subfragmentos que se ejecutan concurrentemente. El orden de ejecución de los elementos individuales en los subfragmentos paralelos puede ser intercalado en cualquier orden posible (a no ser que esté prohibido por una construcción crítica). La concurrencia es lógica y necesita no ser física; las ejecuciones concurrentes pueden ser intercaladas en una sola ruta de ejecución.

seq — Una construcción de secuenciación débil tiene dos o más subfragmentos. Es lo mismo que la ejecución paralela, excepto en que los eventos en la misma línea de vida de diferentes subfragmentos son ordenados en el mismo orden que los subfragmentos dentro del fragmento de secuenciación débil. Eventos en distintas líneas de vida que no aparecen en subfragmentos múltiples pueden intercalarse en cualquier orden.

strict — Una construcción de secuenciación estricta tiene dos o más subfragmentos. Los subfragmentos se ejecutan en el orden en que aparecen dentro de la construcción. Los eventos en diferentes líneas de vida son ordenados de acuerdo a los subfragmentos, a diferencia del caso de una construcción paralela.

Véase cada tipo de construcción para más información y notación.

Notación

La notación general para un fragmento combinado en un diagrama de secuencia es un rectángulo con un pequeño pentágono en la esquina superior izquierda que contiene la palabra clave para la construcción. Si el fragmento tiene más de un subfragmento, se separarán con líneas horizontales discontinuas. El rectángulo se anida dentro del fragmento que lo contiene o dentro del diagrama de secuencia como un todo.

Como forma de abreviar, el pentágono puede contener múltiples palabras clave para indicar construcciones anidadas. La palabra clave a la izquierda indica la construcción más externa. Por ejemplo, **sd strict** indica un diagrama de secuencia que contiene una construcción de secuenciación estricta.

Parámetros. Algunas de las construcciones admiten parámetros después de la palabra clave, como sigue:

Las construcciones ignorar y considerar admiten una lista entre llaves de nombres de mensajes separados por comas, por ejemplo:

```
consider {inicio, fin}
```

La construcción de bucles puede contener un límite inferior y un límite superior entre paréntesis, por ejemplo:

```
loop(2,5)
```

Si se omite el límite superior, será el mismo que el límite inferior:

```
loop(2)
```

Se puede usar un asterisco para representar un límite superior ilimitado:

```
loop(1,*)
```

Si no se muestra ningún límite, el límite inferior será cero y el límite superior será ilimitado.

Historia

La utilización de construcciones estructuradas tales como fragmentos combinados fue una razón importante para el reemplazo de la notación del diagrama de secuencia de UML1 por la notación actual, basada en gran parte en la notación DSM (Diagrama de secuencia de mensajes)

fragmento condicional

Un fragmento combinado en una interacción que representa una elección dinámica entre varios subfragmentos operandos. Se denomina *alternativa* dentro de la especificación de interacción.

Semántica

Una condicional en una interacción se representa como un fragmento combinado con la palabra clave **alt.** Una condicional tiene varios operandos, siendo cada uno de ellos un subfragmento. Cada operando tiene una condición de guarda. La ausencia de una guarda implica una condición verdadera. La condición **else** es verdadera si ninguna otra guarda se evalúa a verdadera. Se ejecuta exactamente un operando cuya guarda se evalúe a verdadero, a no ser que ninguna guarda sea verdadera. Si se evalúa a verdadero más de un operando, la elección puede ser no determinista.

Notación

Una condicional se representa como una región rectangular con la etiqueta **alt** dentro de un pequeño pentágono en la esquina superior izquierda (Figura 14.150). La región se divide en subregiones por medio de líneas horizontales discontinuas. Cada subregión representa un operando de la condicional. Se puede colocar una restricción que represente una condición exitosa en la línea de vida del objeto cuyos valores se están probando, en la parte de arriba de la subregión. Normalmente las condiciones para todas las regiones estarán en la misma línea de vida. En vez de una restricción, se puede colocar la pseudorrestricción **[else]** en la línea de vida para indicar una rama que se toma si ninguna restricción es verdadera.

Historia

La notación de fragmento condicional basada en DSM es una gran mejora en potencia y claridad sobre la notación para alternativas de UML1 en los diagramas de secuencia.

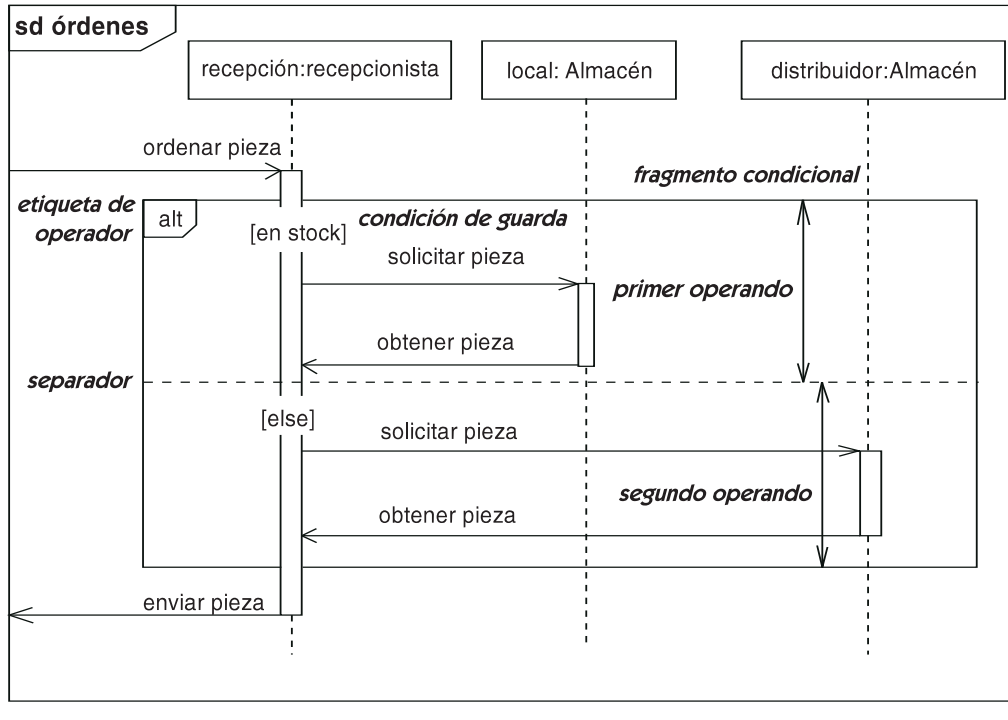


Figura 14.150 Fragmento condicional *(alt)*

fragmento de interacción

Pieza estructural de una interacción

Semántica

Un fragmento de interacción es una pieza de una interacción, una de:

fragmento combinado	Parte estructurada de una interacción que puede incluir uno o más operandos (subfragmentos). Hay varios tipos de fragmentos combinados con distinta semántica y sintaxis, distinguidos por su parámetro de operación de interacción. Incluye entre otras, construcciones repetitivas, condicionales y paralelas.
continuación	Etiqueta que permite a las condicionales ser divididas en dos piezas semánticamente combinadas.
especificación del suceso	Punto notable en la secuencia de actividad de una interacción.
especificación de una ejecución	Región temporal durante la que se ejecuta una acción u operación.
interacción	El cuerpo de una interacción es un fragmento.
uso de la interacción	Referencia parametrizada a una interacción dentro de otra interacción.
operando de la interacción	Pieza constituyente de un fragmento combinado.
invariante del estado	Restricción en el estado o valores de una línea de vida.

Notación

Véase diagrama de secuencia y diagrama de comunicación para ver la notación.

framework (estereotipo de Paquete)

Arquitectura genérica que proporciona una plantilla extensible para las aplicaciones dentro de un dominio. Un marco de trabajo es el punto de partida para construir una arquitectura. Típicamente, los elementos se modifican, especializan y extienden para confeccionar la arquitectura genérica de un problema específico.

Véase paquete.

función primitiva

La declaración de una función matemática que acepta una lista de valores de entrada (posiblemente vacía), y produce una lista de valores de salida (no vacía), sin efectos laterales o acceso a cualquier objeto.

Semántica

UML2 no tiene un conjunto predefinido de funciones primitivas, como funciones aritméticas. Esto sorprende a muchas personas, pero ningún lenguaje de programación tiene exactamente el mismo juego de funciones primitivas y ha habido muchas luchas amargas sobre cuál es el juego "correcto" de funciones primitivas. Por ejemplo, pocos lenguajes llevan a cabo (incluso dentro de los límites de memoria) la aritmética de enteros; casi todos implementan el módulo-aritmético con varias bases. UML2 asume que el ambiente particular tendrá un conjunto predefinido de funciones primitivas. Porque las funciones primitivas son totalmente autónomas y no pueden actuar recíprocamente con el ambiente, no agregan ninguna semántica interesante y el no enumerar un juego predefinido preferido no es ningún impedimento. Un perfil para un ambiente particular de la ejecución o lenguaje de programación debe declarar un juego de funciones primitivas como parte de sus contenidos.

La acción implementada, implementa una función primitiva a un juego de valores de entrada para generar un conjunto de valores de salida. Porque las funciones primitivas son totalmente autónomas, esta acción no impone restricciones en absoluto en la implementación y puede ser totalmente concurrente con todas las otras ejecuciones en el ambiente.

La intención es que no puedan descomponerse funciones primitivas o puedan analizarse dentro del propio UML. Se asume que son llevadas a cabo directamente por el ambiente de ejecución como acciones atómicas.

fusión

Lugar en una máquina de estados o diagrama de actividad donde dos o más rutas de control alternativo se juntan; unir una bifurcación. Antónimo: bifurcación. *Véase también* conjunción.

Semántica

Una fusión simplemente es una situación en la que dos o más rutas de control se unen. No cambia la cantidad de concurrencia.

En una máquina de estados, un estado puede tener más de una transición de entrada. Ni se requiere ni se proporciona ninguna construcción del modelo especial para indicar una fusión. Se puede indicar mediante una conjunción si es parte de una ruta de ejecutar hasta finalizar. Si el control llega a un estado a través de cualquier transición de entrada, el estado se activa.

En una actividad, una fusión se indica mediante un nodo de fusión (Figura 14.151). Un nodo de fusión tiene dos o más flujos de entrada y un flujo de salida. Si un token llega a cualquier entrada, se copia a la salida inmediatamente.

En una interacción, los flujos de control no estructurados no están soportados. Una fusión sucede implícitamente al final de un fragmento condicional.

Notación

Una fusión se puede indicar en un diagrama de actividad con un diamante con dos o más transiciones de entrada y una sola transición de salida. No son necesarias condiciones; una fusión sucede automáticamente cuando un token aparece en cualquier entrada. La Figura 14.151 muestra un ejemplo.

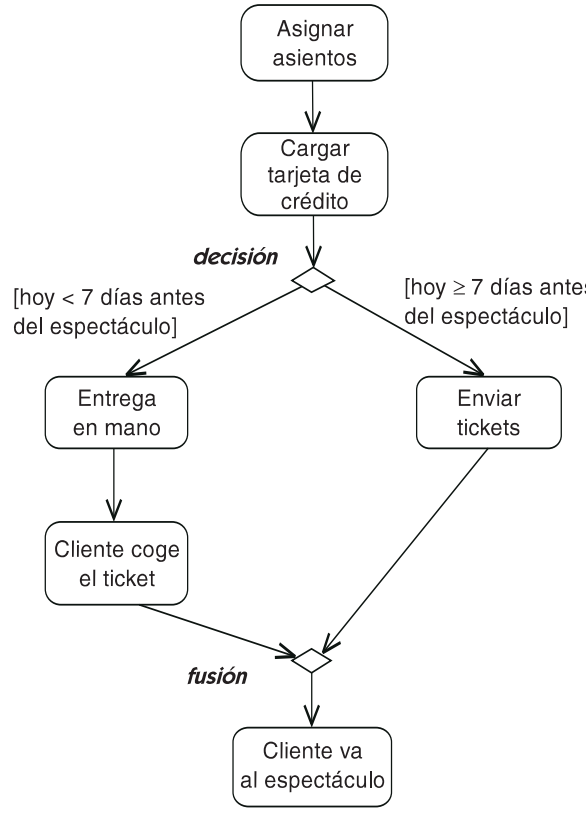


Figura 14.151 Fusión en un diagrama de actividad

Un diamante también se usa para una bifurcación (la inversa de una fusión), pero una bifurcación se distingue claramente puesto que tiene una transición de entrada y varias transiciones de salida, cada una de ellas con su propia condición de guarda. La utilización de los mismos símbolos muestra la simetría entre una bifurcación y una fusión.

Combinar una bifurcación y una fusión es legal pero de utilidad limitada. Debería tener varias transiciones de entrada y varias transiciones de salida etiquetadas.

Una bifurcación y una fusión normalmente van emparejadas de forma anidada.

En un diagrama de máquina de estados, una fusión se representa simplemente dibujando varias flechas de transición que entran al mismo estado. No es necesario ningún icono de fusión.

Discusión

Asegúrese de distinguir fusión y unión. La fusión combina dos o más rutas de control alternativas. En una ejecución, sólo se elegirá una ruta en un momento dado. No es necesaria ninguna sincronización.

La unión combina dos o más rutas de control concurrentes. En cualquier ejecución, se tomarán todas las rutas, y la unión se disparará sólo cuando todas ellas hayan alcanzado los estados origen de la unión.

fusión de paquetes

Un mecanismo para fusionar los contenidos de paquetes, incluso las reglas para importar elementos, resolviendo conflictos de nombres usando especialización y redefinición. Esto es una característica de un modelado avanzado orientado sobre todo a constructores de metamodelos forzados a rehusar al mismo modelo para varios propósitos diferentes, divergentes. Su uso puede llevar a la confusión y debe evitarse si es posible. No se considera necesario para el uso de un diseñador ordinario. *Véase* el documento de especificación de UML2 para la semántica, la cual es compleja y difícil.

generalización

Relación taxonómica entre un elemento más general y un elemento más específico. El elemento más específico es totalmente consistente con el elemento más general y contiene información adicional. Una instancia del elemento más específico es una instancia indirecta del elemento más general y hereda sus características.

Véase también generalización de la asociación, conjunto de generalizaciones, herencia, herencia múltiple, principio de capacidad de sustitución, generalización de casos de uso.

Semántica

Una relación de generalización es una relación dirigida entre dos clasificadores del mismo tipo, como clases, casos de uso o asociaciones. Un elemento se llama padre y el otro hijo. Para clases, el padre se conoce como la superclase y el hijo como la subclase. El padre es la descripción de un conjunto de instancias (indirectas) con propiedades comunes sobre todos los hijos; el hijo es una descripción de un subconjunto de dichas instancias que tiene las propiedades del padre pero también tiene propiedades adicionales propias del hijo.

La generalización es una relación transitiva y antisimétrica. Una dirección de la relación lleva al padre; la otra dirección lleva al hijo. Un elemento relacionado en la dirección del padre a través de una o más generalizaciones se conoce como antecesor; un elemento relacionado en la dirección del hijo a través de una o más generalizaciones se conoce como descendiente. No se permiten ciclos de generalización dirigidos. Una clase no puede ser antecesora o descendiente de ella misma. Una instancia de un clasificador es una instancia indirecta de todos los antecesores del clasificador.

En el caso más sencillo, una clase (u otro elemento generalizable) tiene un único padre. En una situación más complicada, un hijo puede tener más de un padre. El hijo hereda la estructura, comportamiento y restricciones de todos sus padres. Esto se conoce como herencia múltiple (mejor podría llamarse generalización múltiple). Un elemento hijo referencia a sus padres y debe tener visibilidad hacia él.

Los elementos de comportamiento son clases, y por tanto pueden participar en la generalización, de forma similar a las máquinas de estado e interacciones.

Cada tipo de clasificador tiene su propia semántica de generalización. Para la aplicación de generalización a asociaciones, *véase* generalización de la asociación. Para la aplicación de generalización a casos de uso, *véase* generalización de casos de uso. Los nodos y componentes son

prácticamente como clases, y la generalización que se les aplica se comporta igual que para las clases.

Un hijo hereda los atributos, asociaciones, operaciones y restricciones de su padre. Por defecto, permanecen sin cambios, aunque en ciertos casos se pueden redefinir en el hijo. *Véase* redefinición.

Tipos y capacidad de sustitución

Normalmente la generalización implica que un elemento hijo se puede utilizar en cualquier lugar donde se defina un elemento padre, por ejemplo, para sustituir un parámetro de un tipo dado. Tal generalización se dice que tiene capacidad de sustitución. Las generalizaciones con capacidad de sustitución soportan el polimorfismo, puesto que cualquier semántica del elemento padre debe poder ser garantizada por cualquier elemento hijo.

El modelado cercano a las construcciones de un lenguaje de programación a veces requiere generalizaciones sin capacidad de sustitución. En otras palabras, no se puede asumir que un elemento hijo se pueda utilizar en cualquier lugar en que esté definido un elemento padre. Esto puede suceder, por ejemplo, debido a que los componentes internos son anulados o redefinidos. UML permite la definición de generalizaciones con capacidad de sustitución o sin ella. Aunque el uso ocasional de generalizaciones sin capacidad de sustitución puede ser conveniente para compartir estructura, frustran el amplio propósito de la generalización y debería evitarse tanto como sea posible.

Conjuntos de generalizaciones

Un conjunto de clasificadores que comparten un padre común se pueden agrupar en un conjunto de generalizaciones. Cada conjunto de generalizaciones representa una dimensión o aspecto diferentes en los que se puede especializar el supertipo. *Véase* conjunto de generalizaciones.

Notación

La generalización entre clasificadores se representa mediante una línea continua desde el elemento hijo (como una subclase) hasta el elemento padre (como una superclase), con un gran triángulo hueco al final del camino en la unión con el elemento más general (Figura 14.152). Las líneas hacia el padre se pueden combinar para producir un árbol (Figura 14.153).

La generalización se puede aplicar a asociaciones, así como a clasificadores, aunque la notación puede ser confusa debido a las diversas líneas. Una asociación puede representarse como una clase de asociación con el fin de adjuntar flechas de generalización.

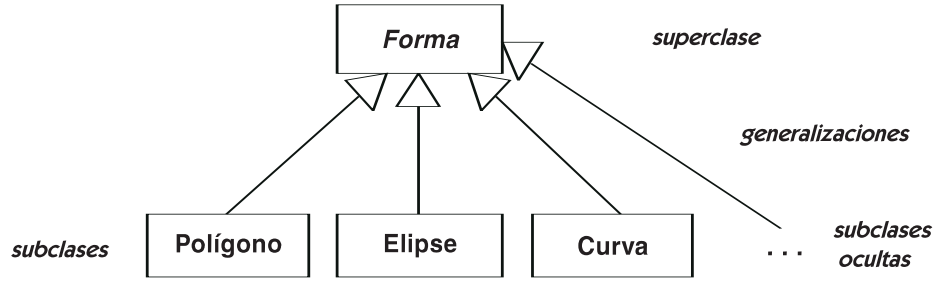


Figura 14.152 Generalización

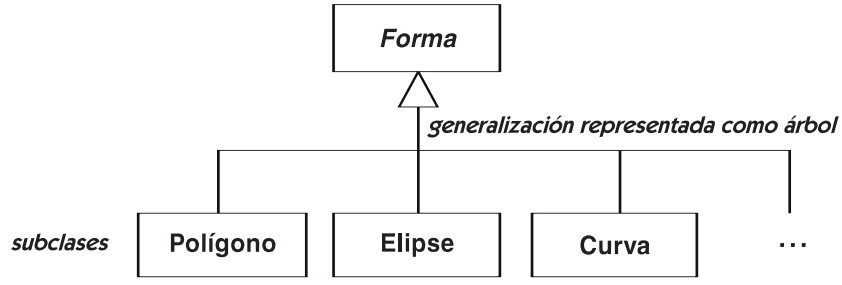


Figura 14.153 Generalización utilizando estilo de árbol

Una etiqueta de texto siguiendo a dos puntos colocados en una línea de generalización indica que la generalización pertenece al conjunto de generalizaciones del nombre dado. Si dos generalizaciones con el mismo padre tienen el mismo nombre, están en el mismo conjunto de generalizaciones.

Opciones de presentación

Un grupo de caminos de generalización para una superclase dada se puede representar como un árbol con un segmento compartido (incluyendo el triángulo) para la superclase, dividiéndose en varios caminos hacia cada subclase. Esto es simplemente un recurso de notación que no indica ninguna relación *n*-aria. En el modelo subyacente, hay una generalización para cada par subclase-superclase. No hay diferencia semántica si los arcos se dibujan de forma separada.

Si se coloca una etiqueta de texto en un triángulo de generalización compartido por varios caminos de generalización hacia subclases, la etiqueta se aplica a todos los caminos. En otras palabras, todas las subclases comparten las propiedades dadas.

Discusión

El elemento padre en una relación de generalización se puede definir sin conocer a los hijos, pero los hijos en general deben conocer la estructura de sus padres para funcionar correctamente. Sin embargo, en muchos casos el padre se diseña para ser extendido por sus hijos e incluye más o menos conocimiento de los hijos esperados. Sin embargo, una de las glorias de la generalización es que se descubre que los nuevos hijos a menudo no habían sido anticipados por el diseñador del elemento padre, llevando a una expansión en potencia que es compatible en espíritu con la intención original.

La relación de realización es como una generalización en la que sólo se hereda una especificación de comportamiento en vez de una estructura o implementación. Si el elemento de especificación es una clase abstracta sin atributos y asociaciones, sino con sólo operaciones abstractas, entonces la generalización y la realización son más o menos equivalentes puesto que no habría nada que heredar más que una especificación de comportamiento. Sin embargo, observe que la realización no genera realmente al cliente; por tanto, las operaciones deben estar en el cliente o ser heredadas de algún otro elemento.

Historia

En UML2, la generalización se restringe a los clasificadores, pero teniendo en cuenta que casi todo es un clasificador, no se pierde gran cosa.

generalización de casos de uso

Relación taxonómica entre un caso de uso (el hijo) y el caso de uso (el padre) que describe las características que el hijo comparte con otros casos de uso que tienen el mismo padre. Esta es la generalización aplicable a los casos de uso.

Semántica

Un caso de uso padre se puede especializar en uno o más casos de uso hijos que representan formas más específicas del padre (Figura 14.154). Un hijo hereda todos los atributos, operaciones y relaciones de su padre, puesto que un caso de uso es un clasificador. La implementación de una operación heredada se puede anular mediante una colaboración que realice un caso de uso hijo.

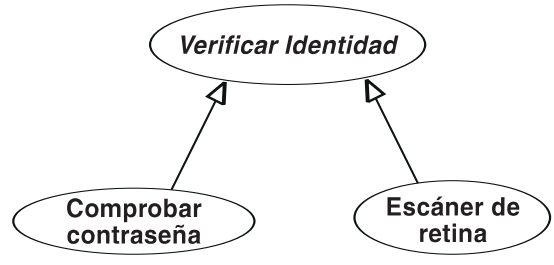


Figura 14.154 Generalización de casos de uso

El hijo hereda la secuencia de comportamiento de su padre y puede insertarle comportamiento adicional (Figura 14.155). El padre y el hijo son potencialmente instanciables (si no son abstractos), y las diferentes especializaciones del mismo padre son independientes, a diferencia de la relación de extensión, en la que varias extensiones modifican implícitamente el mismo caso de uso base. Se puede añadir comportamiento al caso de uso hijo añadiendo pasos en la secuencia de comportamiento heredada del padre, así como declarando relaciones de inclusión y extensión al hijo. Si el padre es abstracto, su secuencia de comportamiento puede tener secciones explícitamente incompletas en el padre y que deben ser proporcionadas por el hijo. El hijo puede modificar pasos heredados del padre, pero al igual que con la anulación de métodos, esta capacidad debe utilizarse con cuidado puesto que se debe preservar la intención del padre.

La relación de generalización conecta un caso de uso hijo con un caso de uso padre. Un caso de uso hijo puede acceder y modificar atributos definidos por el caso de uso padre.

La capacidad de sustitución para los casos de uso significa que la secuencia de comportamiento de un caso de uso hijo debe incluir la secuencia de comportamiento de su padre. Sin embargo, los pasos de la secuencia del padre no tienen por qué ser contiguos; el hijo puede intercalar pasos adicionales entre los pasos de la secuencia de comportamiento heredada del padre.

La utilización de herencia múltiple con casos de uso requiere una especificación explícita de cómo se intercalan las secuencias de comportamiento de los padres para hacer la secuencia del hijo.

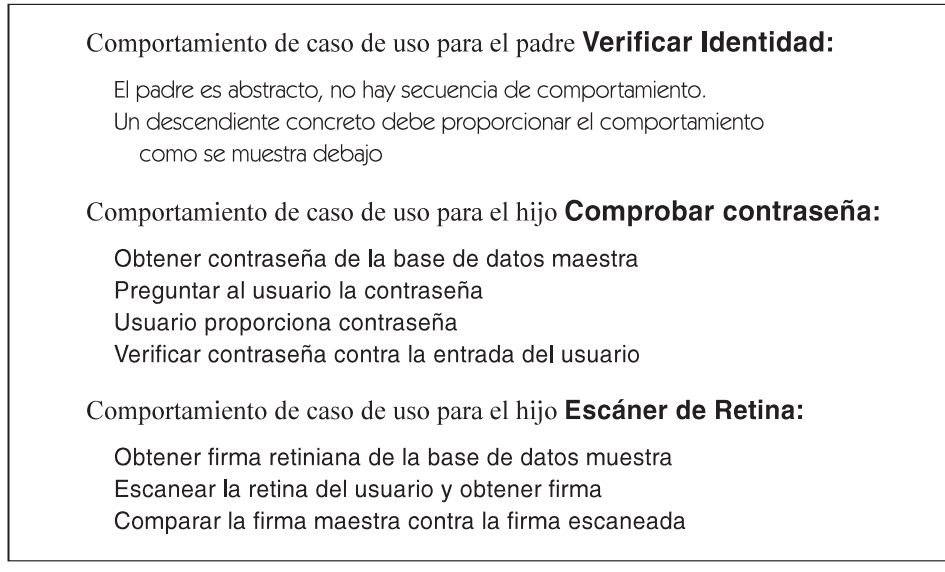


Figura 14.155 Secuencias de comportamiento para casos de uso padre e hijos

La generalización de casos de uso puede utilizar procedimientos para compartir la implementación de un caso de uso base sin capacidad de sustitución completa, pero esta capacidad debería usarse en poca cantidad.

Notación

Se utiliza el símbolo normal de generalización —una línea sólida desde el hijo al padre con una cabeza de flecha triangular hueca en la línea que toca al símbolo del padre.

Ejemplo

La Figura 14.154 muestra un caso de uso abstracto Verificar identidad y su especialización como dos casos de uso concretos, cuyo comportamiento se muestra en la Figura 14.155.

generalización de la asociación

Una relación de generalización entre dos asociaciones.

Véase también asociación, generalización.

Semántica

La generalización entre asociaciones está permitida, aunque es poco común. Como con cualquier relación de generalización, el elemento hijo debe añadir algo a la intención (reglas de definición) del padre y debe definir un subconjunto de extensión del padre. Añadir a la intención significa añadir restricciones adicionales. Una asociación hija está más restringida que su padre. Por ejemplo, en la Figura 14.156, si la asociación padre conecta las clases **Tema** y **Símbolo**, entonces la asociación hija

podría conectar las clases **Pedido** y **SímboloPedido**, donde **Pedido** es hija de **Tema** y **SímboloPedido** es hija de **Símbolo**. Definir un subconjunto de extensión significa que cada enlace de la asociación hija es un enlace de la asociación padre, pero no a la inversa. El ejemplo obedece a esta regla. Cualquier enlace que conecta **Pedido** y **SímboloPedido** también puede conectar **Tema** y **Símbolo**, pero no todos los enlaces que conectan **Tema** y **Símbolo** conectarán **Pedido** y **SímboloPedido.**

Notación

La asociación hija se conecta a la asociación padre mediante un símbolo de generalización (una flecha de línea continua y punta triangular hueca). La punta de flecha se sitúa en el padre. Debido a que las líneas conectan otras líneas, la notación de la generalización de asociaciones puede ser confusa y debe ser utilizada con cuidado.

Ejemplo

La Figura 14.156 muestra dos especializaciones de la asociación general **modelo-vista** entre **Tema** y **Símbolo**: La asociación entre **Pedido** y **SímboloPedido** es una especialización, como también lo es la asociación entre **Cliente** y **SímboloCliente.** Cada uno de ellos conecta una clase **Tema** a una clase **Símbolo**. La asociación general **Tema-Símbolo** se puede considerar una asociación abstracta donde las dos asociaciones hijas son concretas.

Este patrón en el que dos jerarquías de clases están conectadas mediante asociaciones es bastante común.

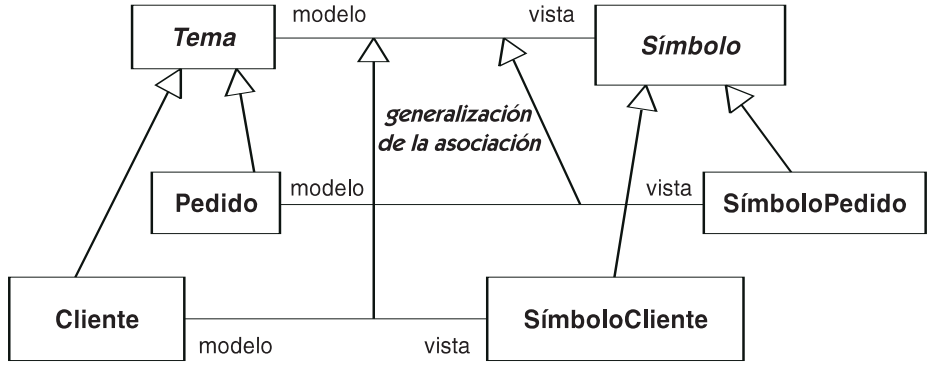


Figura 14.156 Generalización de la asociación

Discusión

La distinción entre definir subconjuntos y especializar una asociación no se encuentra claramente descrita en la especificación de UML2.

grupo de actividad

Un subconjunto arbitrario de nodos y edges dentro de una especificación de una actividad. Las particiones y los nodos estructurados son tipos de grupos de actividad.

habilitada

Transición cuyos prerrequisitos han sido satisfechos y es elegible para su ejecución.

Semántica

Antes de que se pueda disparar una transición, el objeto que la posee debe mantener el estado origen de la transición o uno de sus descendientes, y debe satisfacer el desencadenador de la transición. El desencadenador se satisface si el objeto ha recibido y no ha procesado todavía una ocurrencia de evento que concuerda con el desencadenador. Un desencadenador de señal se satisface por una señal que es descendiente del desencadenador de la transición. Si se producen dichas condiciones, se evalúa en el contexto actual la condición de guarda de la transición. Si la condición de guarda se evalúa a verdadero (o si no hay condición de guarda), se habilita la transición. Si la transición tiene múltiples estados origen (es decir, si es un nodo de unión), la configuración del estado actual debe incluirlos a todos.

Puede ocurrir que se habiliten varias transiciones con el mismo evento. En ese caso, se disparará una y sólo una transición. La elección puede ser no determinista. Cuando se elige una transición para su disparo, cualesquiera otras transiciones habilitadas dejan de estar habilitadas.

Si una transición contiene varios segmentos que no incluyen un pseudoestado de elección, entonces todos los desencadenadores y condiciones de guardas en todos los segmentos deben estar satisfechos para que se habilite la transición. La transición no se habilita con un segmento cada vez.

Un pseudoestado de elección divide una transición en dos transiciones separadas. La transición está habilitada si están satisfechos los desencadenadores y las condiciones de guarda de todos los segmentos anteriores al pseudoestado de elección. La transición permanece habilitada en el pseudoestado de elección. No se admite ningún desencadenador después del pseudoestado de elección. Al menos un segmento saliente debe satisfacer su condición de guarda de forma que la transición pueda trasladarse con éxito a un estado destino.

También se dice que una actividad está habilitada si sus prerrequisitos han sido satisfechos. En el caso de nodos condicionales o de conflictos sobre tokens de entrada, pueden habilitarse simultáneamente varias actividades, pero sólo una será elegida para su ejecución, a no ser que sus entradas sean disjuntas.

herencia

Mecanismo por el que los elementos más específicos incorporan estructura y comportamiento definidos por elementos más generales.

Véase también descriptor completo, generalización, redefinición.

Semántica

La herencia permite que se construya la descripción completa de un clasificador ensamblando fragmentos de declaración desde una jerarquía de generalización. Una jerarquía de generalización es un árbol (realmente, un orden parcial) de declaraciones de elementos del modelo, como

clases. Sin embargo, cada declaración no es la declaración de un elemento completo y utilizable. En su lugar, cada declaración es una declaración incremental que describe lo que añade la declaración del elemento a las declaraciones de sus antecesores en la jerarquía de generalización. La herencia es el proceso (implícito) de combinar esas declaraciones incrementales en descriptores completos que describen instancias reales.

Piense en cada clasificador como si tuviera dos descripciones, una declaración de segmento y un descriptor completo. La declaración de segmento es la lista incremental de características que el elemento declarar en el modelo —los atributos y operaciones declarados por una clase, por ejemplo. La declaración de segmento es la estructura adicional comparada con sus padres que declara un clasificador. El descriptor completo de un clasificador incluye la nueva estructura y la estructura de sus padres. El descriptor completo no aparece explícitamente dentro del modelo. Es la descripción completa de una instancia del elemento —por ejemplo, la lista completa de atributos y operaciones disponibles en un objeto de una clase. El descriptor completo es la unión de los contenidos de las declaraciones de segmento en un elemento y en todos sus antecesores.

Eso es la herencia. Es la definición incremental de un elemento. Otros detalles, como los algoritmos de búsqueda de métodos, tablas virtuales y demás, son simplemente mecanismos de implementación que hacen que funcione en un lenguaje particular, y no son parte de la definición esencial. Aunque esta descripción puede parecer extraña al principio, está libre de las implicaciones encontradas en la mayoría de las demás definiciones, y aún es compatible con ellas.

Conflictos

Si la misma característica aparece más de una vez entre el conjunto de segmentos heredados, puede haber un conflicto. A no ser que sea redefinido explícitamente, ningún atributo puede ser declarado más de una vez en un conjunto heredado. Si se produce una definición múltiple, las declaraciones entran en conflicto y el modelo está mal formado. (Esta restricción no es esencial por razones lógicas. Está presente para evitar la confusión que podría producirse si los atributos tuvieran que distinguirse mediante nombres de ruta.)

La misma operación se puede declarar más de una vez, siempre que la declaración sea exactamente la misma (sin embargo, los métodos pueden ser diferentes) o que la declaración de un hijo fortalezca una declaración heredada (por ejemplo, declarando que un hijo sea una consulta o incrementando su estado de concurrencia). Una declaración de método en un hijo reemplaza (anula) una declaración de método en un antepasado. No hay conflicto. Si se heredan métodos distintos desde dos antepasados diferentes que no tienen entre ellos relación de antepasado, entonces los métodos entran en conflicto y el modelo está mal formado.

Discusión

Las palabras *generalización* y *herencia* se usan a menudo de forma intercambiable, pero realmente son dos conceptos relacionados pero diferentes. La generalización es una relación taxonómica entre elementos de modelado. Describe qué *es* un elemento. La herencia es un mecanismo para combinar descripciones incrementales compartidas para formar una descripción completa de un elemento. En la mayoría de los sistemas orientados a objetos, la herencia se basa en la generalización, pero la herencia se puede basar en otros conceptos, como en el puntero de delegación del lenguaje Self. Basar el mecanismo de herencia en la relación de generalización permite factorizar y compartir descripciones y comportamiento polimórfico. Esta es la

aproximación elegida por la mayoría de lenguajes orientados a objetos y por UML. Pero tenga en cuenta que se podrían haber tomado otras aproximaciones y que se utilizan en algunos lenguajes de programación.

herencia de implementación

La herencia de la implementación de un elemento padre —es decir, su estructura (como atributos y operaciones) y su código (como métodos). En contraste, la herencia de interfaces implica herencia de especificaciones de interfaz (es decir, operaciones) pero ni métodos ni estructura de datos (atributos y asociaciones). La herencia de implementación (el significado normal de generalización en UML) incluye la herencia tanto de interfaz como de implementación.

Véase también generalización, herencia.

herencia múltiple

Punto de variación semántico de generalización en el que un elemento puede tener más de un padre. Esta es la asunción por defecto dentro de UML y es necesaria para el modelado apropiado de muchas situaciones, aunque los modeladores pueden elegir restringir su utilización para ciertos tipos de elementos. Algunos lenguajes de programación evitan o restringen su utilización. Contrastar con: herencia simple.

herencia simple

Una variación semántica de la generalización en la que un elemento puede tener un único padre. Si se permite una sola herencia o herencia múltiple es un punto de variación semántica. El valor predeterminado es permitir herencia múltiple.

hijo/a

El elemento más específico de una relación de generalización. Se denomina subclase de una clase. Una cadena de una o más relaciones hijas es un descendiente. Antónimo: padre.

Semántica

Un elemento hijo hereda las características de su padre (e indirectamente las de sus antepasados) y puede declarar características adicionales por sí mismo. También hereda cualquier asociación y restricción en la que participen sus antepasados. El elemento hijo obedece el principio de sustitución, es decir, una instancia de un descriptor satisface cualquier declaración variable clasificada como uno de los antepasados del descriptor. Una instancia de un hijo es una instancia indirecta del padre.

La relación de descendencia es el cierre transitivo de la relación hija. Una instancia de un descendiente es una instancia indirecta del antepasado.

Discusión

Obsérvese que *hijo*, *padre*, *antepasado* y *descendiente* no son términos oficiales de UML, pero los utilizamos en este libro por conveniencia ya que UML no parece tener buenos términos sencillos que cubran todos los usos de los conceptos.

hilo

(Hilo de control) Un camino simple de ejecución dentro de un programa, modelo dinámico, u otra representación de un flujo de control. También un estereotipo para la implementación de un objeto activo, como un proceso lijero.

Véase objeto activo, transición compleja, estado compuesto, máquina de estados y estado de sincronización.

hipervínculo

Conexión invisible entre dos elementos de notación que se puede atravesar mediante algún comando.

Véase también diagrama.

Notación

Una notación en un trozo de papel no contiene información oculta. Sin embargo, una notación en una pantalla de computadora puede contener hipervínculos invisibles adicionales que no son aparentes en una vista estática, pero que se pueden invocar de forma dinámica para acceder a alguna otra información, bien de forma gráfica o en una tabla de texto. Estos enlaces dinámicos son parte de una notación *dinámica* en la misma proporción que la información visible, pero la especificación UML no prescribe su forma. Son responsabilidad de las herramientas. La especificación UML intenta definir una notación *estática* para UML, con la comprensión de que cierta información útil e interesante podría aparecer de forma pobre o no aparecer en dicha vista. Por otro lado, no es la intención especificar el comportamiento de todas las herramientas dinámicas ni reprimir la innovación en la presentación dinámica. Con el tiempo, algunas notaciones dinámicas podrían llegar a estar lo suficientemente establecidas como para estandarizarlas, pero se necesita más experiencia.

hoja

Elemento generalizable que no tiene hijos en la jerarquía de generalización. Debe ser concreto (totalmente implementado) para tener cualquier utilidad.

Véase también abstracto/a, concreto/a.

Semántica

La propiedad de hoja declara que un elemento *debe* ser una hoja. El modelo está mal formado si declara un hijo de dicho elemento. El propósito es garantizar que una clase no puede ser modifi-

cada, por ejemplo, puesto que el comportamiento de la clase debe estar bien establecido para garantizar la fiabilidad. La declaración de hoja también permite separar la compilación de partes de un sistema asegurando que los métodos no pueden ser anulados y facilitando la expansión en línea del código de los métodos. Un elemento cuya propiedad hoja sea falsa podría ser una hoja en un momento dado, pero se le podrían añadir hijos en el futuro si se modifica el modelo. Ser una hoja o estar restringido a ser una hoja no es una propiedad semántica fundamental, sino que más bien son mecanismos de la ingeniería del software para controlar el comportamiento humano.

identidad

Propiedad inherente de un objeto de ser distinguible de todos los demás objetos.

Véase también valor dato, objeto.

Semántica

Los objetos son discretos y distinguibles de los demás. La identidad de un objeto es su manejador conceptual, la característica inherente que permite que otros objetos lo identifiquen y lo referencien. Conceptualmente, un objeto no necesita un valor interno para identificarse a sí mismo; dicho mecanismo no debería incluirse en modelos lógicos, puesto que se pueden generar automáticamente. En una implementación, la identidad se puede implementar mediante direcciones o claves, pero forman parte de la infraestructura de implementación subyacente y no se necesita incluir explícitamente como atributos en la mayoría de modelos. Los valores puros, por otro lado, no tienen identidad, y dos valores idénticos son indistinguibles en todos los sentidos.

La identidad sólo es significativa en un objeto que puede ser modificado. Dos referencias a un objeto tienen la misma identidad si el cambio al objeto utilizando una referencia es visible para la otra referencia. Un objeto de sólo lectura es por tanto indistinguible de un valor puro.

Hay una acción de prueba de identidad que determina si dos valores son el mismo objeto.

ignore

Etiqueta en un fragmento combinado de una interacción que indica que ciertos tipos de mensaje se suprimen en el modelo dentro del fragmento, independientemente de su ocurrencia real durante la ejecución. Esta etiqueta es equivalente a una etiqueta **consider** que lista todos los demás tipos de mensaje. *Véase* consider.

Semántica

Ciertas restricciones en las interacciones se expresan mejor centrándonos en un subconjunto de los tipos de mensaje que participan en la restricción. En dichos casos, puede ser útil suprimir otros tipos de mensaje en el modelo puesto que no son relevantes para la restricción dada o puesto que en la práctica se descartan. La etiqueta **ignore** en un fragmento combinado indica que los tipos de mensaje listados no se muestran dentro del fragmento y que cualesquiera restricciones los filtrarán antes de que sean aplicados.

Por ejemplo, una interacción podría fijar que un mensaje de tipo A siempre es seguido de inmediato por un mensaje de tipo B y por ningún otro de tipo A, pero los mensajes de ti-

po C podrían suceder independientemente sin afectar a los mensajes de tipo A o B. Este se puede modelar ignorando los mensajes de tipo C dentro del fragmento que requiere que B siga a A.

Notación

La palabra clave **ignore** es seguida por una lista separada por comas de nombres de tipo de mensaje. La cadena entera se coloca en la etiqueta rectangular en el límite del fragmento.

implementación

1. Definición de cómo algo es construido o calculado. Por ejemplo, un método es una implementación de una operación. Contrastar con: especificación. La relación de realización relaciona una implementación de una especificación.

Véase realización.

2. Etapa de un sistema que describe el funcionamiento del sistema en un medio ejecutable (como un lenguaje de programación, base de datos o hardware digital). Para la implementación, se deben tomar decisiones tácticas de bajo nivel para encajar el diseño con un medio de implementación particular y para evitar sus limitaciones (todos los lenguajes tienen ciertas limitaciones arbitrarias). Sin embargo, si el diseño se ha hecho bien las decisiones de implementación serán locales y ninguna de ellas afectará a una gran porción del sistema. Esta etapa se captura mediante modelos a nivel de implementación, especialmente la vista estática y el código. Contrastar con análisis, diseño y despliegue (fase de).

Véase proceso de desarrollo, etapas de modelado.

implementation (estereotipo de Componente)

Componente que proporciona la implementación para una especificación separada hacia la que tiene dependencia.

Véase implementación, especificación.

implementation class (estereotipo de Clase)

Clase que proporciona una implementación física, incluyendo atributos, asociaciones con otras clases y métodos para operaciones. Una clase de implementación está intencionada para un lenguaje orientado a objetos convencional con una clasificación única fija. Un objeto en esos sistemas debe tener exactamente una clase de implementación como su clase directa. Contrasta con **type**, un estereotipo para una clase que permite la clasificación múltiple. En un lenguaje convencional, como Java, un objeto puede tener una clase de implementación y muchos interfaces. La clase de implementación debe ser consistente con los interfaces.

Véase tipo.

importar

Relación dirigida que añade los nombres de elementos a un espacio de nombres.

Véase acceso, nombre calificado, visibilidad.

Semántica

Se pueden utilizar los nombres de los elementos definidos dentro de un paquete (u otro espacio de nombres) dentro de expresiones de texto (como restricciones OCL) encontradas en el paquete. Los elementos de otros paquetes se pueden referenciar utilizando sus nombres calificados, es decir, la secuencia de nombres de los espacios anidados que contienen al elemento empezando desde la raíz del sistema. En muchos casos, un modelador desea tratar un elemento o grupo de elementos como si se hubieran definido de forma local, de forma que se puedan utilizar los nombres sencillos en lugar de los calificados. Un paquete (el cliente) que importa un elemento desde otro paquete (el proveedor) puede usar el nombre sencillo del elemento importado para referirse a él dentro de expresiones. Los elementos se pueden importar sólo si son visibles al paquete que hace la importación. Por ejemplo, un elemento que es privado dentro de su paquete no puede ser importado por otro paquete. La relación de importación especifica la visibilidad de importación pública o privada dentro del paquete importador. Si es pública, el elemento importado es visible para cualquier elemento que pueda ver al paquete importador (y el elemento puede ser por tanto importado por otro paquete). Si la visibilidad es privada, el elemento importado no es visible fuera del paquete importador.

A un elemento importado se le puede dar un alias, que es el nombre mediante el que se puede referenciar directamente al elemento importado dentro del paquete importador. Si se da un alias, el nombre original del elemento importado no aparece en el paquete importador.

Un paquete gana acceso implícitamente a todos los paquetes importados por cualesquiera paquetes dentro de los que está anidado (es decir, los paquetes anidados pueden ver todo lo que ven sus paquetes contenedores). Si hay un conflicto de nombres, un nombre definido o importado por un paquete más interno oculta un nombre definido o importado por un paquete más externo. El nombre en el paquete más externo debe ser referenciado utilizando su nombre calificado.

Los elementos de diferentes tipos, como un atributo y una operación, no entran en conflicto. Algunos elementos utilizan parámetros adicionales frente al nombre para determinar si hay conflicto de nombres. Por ejemplo, operaciones con distintas signaturas no entran en conflicto aunque tengan nombres idénticos.

Si los nombres de dos elementos importados entran en conflicto, ninguno de los dos se añade al espacio de nombres. Si el nombre de un elemento importado entra en conflicto con un nombre definido dentro del espacio de nombres importador, el nombre interno tiene precedencia y el nombre importado no se añade al espacio de nombres.

Un elemento en un paquete tiene acceso a todos los elementos que están *visibles dentro del paquete* (u otro espacio de nombres). Las reglas de visibilidad se pueden resumir como sigue.

- Un elemento definido en un paquete es visible dentro del mismo paquete.
- Si un elemento es visible dentro de un paquete, entonces es visible dentro de todos los paquetes anidados dentro del paquete.

- Si un paquete importa otro paquete con visibilidad pública, entonces todos los elementos definidos con visibilidad pública en el paquete importado son visibles dentro del paquete importador.
- Si un paquete es hijo de otro paquete, entonces todos los elementos definidos con visibilidad pública o protegida en el paquete padre son visibles dentro del paquete hijo. Los elementos protegidos son invisibles fuera del paquete hijo.
- Las dependencias de acceso e importación no son transitivas. Si A puede ver a B y B puede ver a C, no significa necesariamente que A pueda ver a C.
- Un elemento definido en un espacio de nombres no es visible dentro del paquete que lo engloba más próximo, pero es invisible fuera de dicho paquete.

Una consecuencia es que un paquete no puede ver dentro de sus propios paquetes anidados a no ser que los importe y a no ser que el contenido de los paquetes anidados sea público.

Las siguientes son reglas adicionales sobre la visibilidad.

- Los contenidos de un clasificador, como sus atributos y sus operaciones así como sus clases anidadas, son visibles dentro del paquete si tienen visibilidad pública en el clasificador.
- Los contenidos de un clasificador son visibles dentro de un clasificador descendiente si tienen visibilidad pública o protegida en el clasificador.
- Todos los contenidos de un clasificador son visibles para elementos dentro del clasificador, incluyendo dentro de métodos o de máquinas de estados del clasificador.

El caso normal más sencillo implica a elementos en paquetes que son semejantes. En ese caso, un elemento puede ver todos los elementos en su propio paquete y todos los elementos con visibilidad pública en aquellos paquetes importados por su paquete. Una clase puede ver las características públicas en las demás clases que pueda ver. Una clase también puede ver características protegidas en sus antecesores.

Notación

Una dependencia de importación se representa mediante una flecha discontinua, dibujada con su cola en el paquete cliente y su cabeza en el paquete proveedor. La flecha utiliza la palabra clave **«import»** como etiqueta. Si la importación es privada, se usa en su lugar la palabra clave **«access»**.

Se puede mostrar antes o después de la palabra clave un nombre de alias.

Alternativamente, se puede representar una importación como una cadena de texto dentro del paquete importador utilizando la siguiente sintaxis:

{import **`nombreCalificado`**}

Ejemplo

La Figura 14.157 muestra un ejemplo de importación de un paquete entre dos paquetes semejantes. El paquete P importa al paquete Q. Las clases K y L en el paquete P pueden utilizar a M como un nombre sin calificar para referenciar a la clase pública M en el paquete Q, pero no pue-

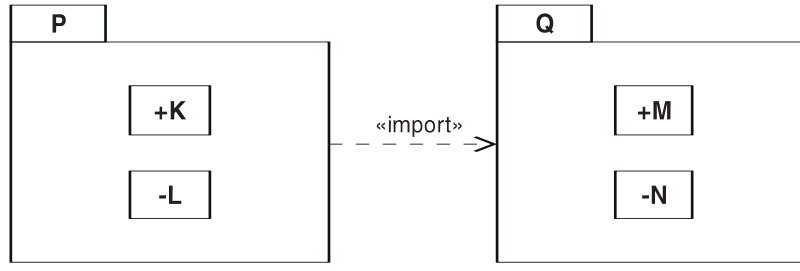


Figura 14.157 Importación de paquete

den ver a la clase privada N de ninguna forma. Las clases M y N en el paquete Q pueden usar el nombre calificado P::K para referenciar a la clase pública K en el paquete P, pero no puede ver a la clase L de ninguna forma.

La Figura 14.158 muestra un caso más complicado de visibilidad y declaraciones de acceso. El símbolo delante de un nombre de elemento representa la visibilidad del elemento fuera de su propio contenedor: + para acceso público, – para acceso privado (no visible desde fuera).

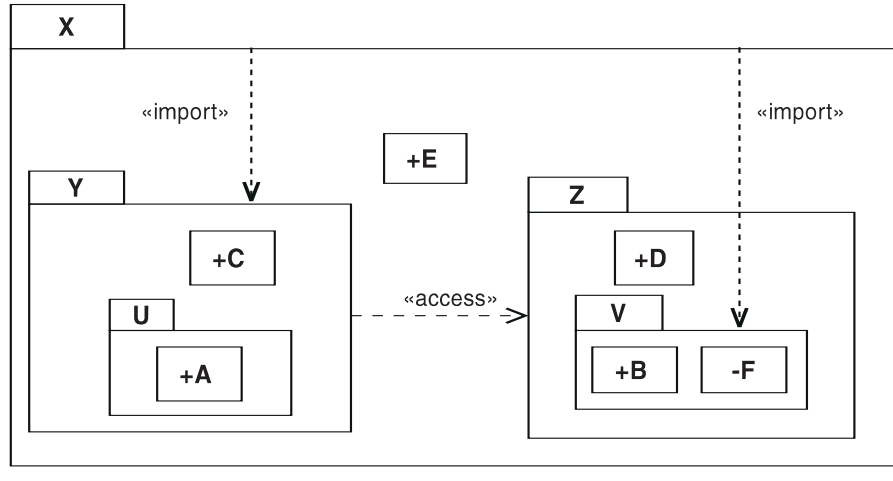


Figura 14.158 Reglas de acceso

La clase A puede ver a C y E y utilizar sus nombres sin calificar puesto que están en los paquetes que lo engloban Y y X.

Las clases C y A pueden ver a D y utilizar su nombre sin calificar puesto que el paquete Y importa al paquete Z. La clase A está anidada dentro del paquete Y y por tanto puede ver todo lo que ve Y.

Las clases A, C y E pueden ver B y utilizar su nombre sin calificar puesto que están anidadas en el paquete X, que importa al paquete V que contiene a B. Sin embargo no pueden ver a F, puesto que tiene visibilidad privada dentro del paquete V. La clase F, por tanto, no puede ser vista fuera del paquete V.

La clase E puede ver a D pero debe utilizar el nombre calificado Z::D, puesto que D está en el paquete Z, que no ha sido importado por el paquete X. La importación del paquete Z por el paquete Y tiene visibilidad privada (palabra clave access), y por tanto la importación del paquete Y por el paquete X no le da visibilidad sobre los contenidos del paquete Z.

Las clases C y E pueden ver a A pero deben usar el nombre calificado (U::A), puesto que la clase A está en el paquete U, que no ha sido importado por otro paquete. El paquete Y ha sido importado por el paquete X, de forma que la clase E no necesita incluir a Y en el nombre calificado.

Las clases B y F pueden ver a las clases D y E y utilizar su nombre no calificado, puesto que se encuentran en paquetes que las engloban. También pueden ver a C y utilizar su nombre no calificado, puesto que C está en el paquete Y, que es importado por el paquete que las engloba X, que finalmente contiene a B y F. El hecho de que F sea privada no la impide ver otras clases, pero otras clases no pueden ver a F.

Las clases B y F pueden verse entre ellas y utilizar sus nombres puesto que están en el mismo paquete. La clase F es privada a las clases en los paquetes externos, pero no a las clases en su propio paquete.

Historia

UML1 utilizaba las dependencias access e import para conceder permiso para referenciar elementos desde otros paquetes. Ese concepto sobreprotector se ha desechado. Ahora la importación sólo determina la aparición de un nombre importado en el espacio de nombres importador. Si se puede ver un elemento, puede ser referenciado directamente, y su nombre calificado siempre se puede utilizar dentro de expresiones.

Discusión

Observe que la importación es sólo sobre el uso de nombres en expresiones de texto. No se necesita para construir modelos por sí mismos. Un elemento desde otro paquete siempre puede ser referenciado de forma explícita en un diagrama, puesto que las relaciones son referencias directas que no dependen de los nombres. Observe también que la utilización de nombres en espacios de nombres es enormemente dependiente en las extensiones de UML utilizadas por las herramientas.

inactivo

Estado que no está activo; uno que no es sostenido por un objeto.

incluir

Relación desde un caso de uso *base* hacia un caso de uso de *inclusión*, que especifica que el comportamiento definido por el caso de uso de inclusión se inserta en el comportamiento definido por el caso de uso base. El caso de uso base puede ver la inclusión y puede depender de los efectos de realizar la inclusión, pero ni el caso de uso base ni el caso de uso de 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 de inclusión. El caso de uso de inclusión en esta relación no es un clasificador instanciable por separado. En cambio, describe explícitamente una secuencia adicional de comportamiento que se inserta en una instancia de un caso de uso que está ejecutando el caso de uso base. Se pueden aplicar varias relaciones de inclusión al mismo caso de uso base. El mismo caso de uso de inclusión se puede incluir en varios casos de uso base. Esto no indica ninguna relación entre los casos de uso base. Incluso puede haber varias relaciones de inclusión entre el mismo caso de uso de inclusión y caso de uso base, siempre que cada inserción se produzca en una ubicación diferente en la base.

El caso de uso de inclusión puede acceder a atributos u operaciones del caso de uso base. La inclusión representa comportamiento encapsulado que se puede reutilizar potencialmente en varios casos de uso base. El caso de uso base ve al caso de uso de inclusión, que puede establecer valores de atributos en el caso de uso base. Pero el caso de uso base no debe acceder a los atributos del caso de uso de inclusión, puesto que los casos de uso de inclusión habrán terminado cuando el caso de uso base vuelva a obtener el control.

Observe que se pueden anidar las adiciones (de cualquier tipo). Una inclusión, por tanto, puede servir como base de otra inclusión, extensión o generalización.

La inclusión es una sentencia explícita dentro de la secuencia de comportamiento del caso de uso base. Por tanto la ubicación es implícita, a diferencia de la situación que se produce con la relación de extensión.

Notación

Se dibuja una línea discontinua desde el símbolo del caso de uso base hacia el símbolo del caso de uso de inclusión con una cabeza de flecha en la inclusión. Se coloca la palabra clave **«include»** en la flecha (Figura 14.159). La ubicación se puede adjuntar a la flecha como una lista de propiedades entre llaves, pero normalmente se referencia como parte del texto del caso de uso base y no necesita representarse en el diagrama. La Figura 14.160 muestra las secuencias de comportamiento para esos casos de uso.

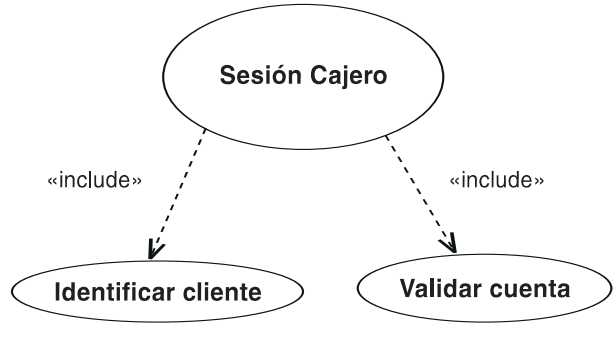


Figura 14.159 Relación de inclusión

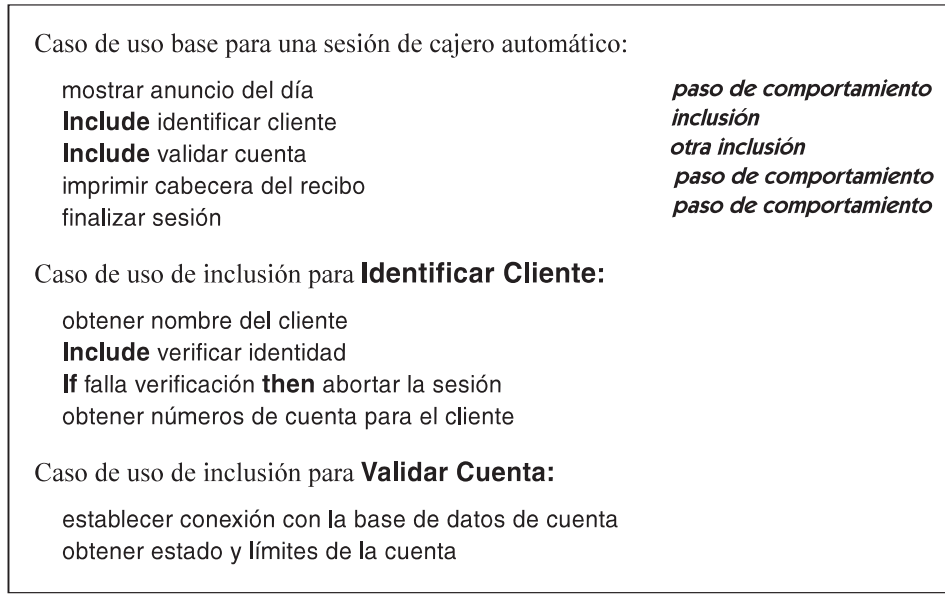


Figura 14.160 Secuencias de comportamiento para casos de uso

incomplete

Palabra clave que indica que los subtipos de un conjunto de generalizaciones no cubren todos los casos del supertipo.

Véase conjunto de generalizaciones.

indicador

Este término se utiliza en este libro para indicar un valor lógico que especifica una restricción estructural en un elemento del modelo. Dependiendo de la situación, a veces se modela un indicador como una sentencia verdadera-falsa y otras veces como una enumeración con dos posibles elecciones.

información de fondo

Cada aparición de un símbolo para una clase en un diagrama o en diferentes diagramas puede tener sus propias opciones de presentación. Por ejemplo, un símbolo para una clase puede mostrar sus atributos y operaciones, mientras que otro símbolo para la misma clase puede suprimirlos. Las herramientas proporcionan hojas de estilos vinculadas tanto a los símbolos individuales como a diagramas completos. Las hojas de estilos especifican opciones de presentación y se pueden aplicar a la mayoría de los símbolos, y no sólo a las clases.

No toda la información de modelado es más útil representada en su notación gráfica. Parte de la información se representa mejor en un formato de texto o tabulada. Por ejemplo, la información de programación detallada a menudo se presenta mejor mediante listas de texto. UML no

asume que toda la información en un modelo sea expresada como diagramas; parte de ella se puede expresar en forma de tablas. El Apéndice D de UML2 describe un formato tabulado para los diagramas de secuencia, pero no hay nada de particular, tanto en los diagramas de secuencia, como en el formato particular listado en el apéndice, que impida el uso de tablas para cualquier información de modelado. Esto es debido a que la información subyacente está adecuadamente descrita en el metamodelo de UML, y la responsabilidad de representar la información de forma tabulada es de la herramienta. Los enlaces perdidos pueden existir tanto en los elementos gráficos, como en los tabulados, dentro de las herramientas.

inicialización

Establecimiento del valor de un objeto recién creado —a saber, los valores de sus atributos, enlaces de asociaciones a las que pertenece y su control de estado.

Véase también instanciación.

Semántica

Conceptualmente, un objeto nuevo se crea completamente en un paso. Sin embargo, es más fácil pensar en el proceso de instanciación en dos pasos: creación e inicialización. Primero, se asigna una estructura vacía del objeto con la estructura propia de ranuras de atributos, y se le da identidad al nuevo objeto bruto. La identidad se puede implementar de varias maneras, como por la dirección del bloque de memoria que contiene al objeto o por un contador entero. En cualquier caso, es algo que es único a lo largo del sistema y que se puede utilizar como un manejador para encontrar y acceder al objeto. En este punto, el objeto no es todavía legal —puede violar las restricciones en sus valores y relaciones. El siguiente paso es la inicialización. Cualesquiera expresiones de valores iniciales declaradas para los atributos se evalúan, y los resultados se asignan a las ranuras de los atributos. El método de creación puede calcular de manera explícita los valores de los atributos, anulando por tanto los valores iniciales por defecto. Los valores resultantes deben satisfacer las restricciones establecidas en la clase. El método de creación también puede crear enlaces que contengan al nuevo objeto. Deben satisfacer la multiplicidad declarada de las asociaciones en las que participa la clase. Cuando se finaliza la inicialización, el objeto debe ser un objeto legal y debe cumplir las restricciones de su clase. Después de que la inicialización haya finalizado, los atributos o asociaciones cuya propiedad de variabilidad sea **frozen** o **addOnly** no se pueden alterar hasta que se destruya el objeto. El proceso de inicialización entero es atómico y no puede ser ni interrumpido ni intercalado.

La semántica de inicialización depende en gran medida en la concreción de la vista de modelado. En un nivel conceptual, el objeto sencillo adquiere un valor. En un nivel concreto cercano a la implementación, los mecanismos disponibles en un entorno de ejecución particular se vuelven importantes. No es posible imponer una única vista de la inicialización a través de todos los posibles usos.

inicio

Primera fase de un proceso de desarrollo de software, durante la que se conciben y evalúan las ideas iniciales sobre un sistema. Durante esta fase, se desarrolla algo de la vista de análisis y pequeñas partes de otras vistas.

Véase proceso de desarrollo.

instancia

Entidad individual con su propia identidad y valor. Un clasificador especifica la forma y comportamiento de un conjunto de instancias con propiedades similares. Una instancia tiene identidad y valores que son consistentes con la especificación del clasificador. Las instancias aparecen en modelos como especificaciones de instancia.

Véase también clasificador, enlace, especificación de instancia, identidad, objeto.

Semántica

El documento de especificación de UML no incluye una especificación clara del entorno en tiempo de ejecución que describe el modelo. Sin dicha especificación, los modelos quedan faltos de significado. La siguiente discusión probablemente debería haberse incluido en la especificación. Representa el trasfondo comúnmente aceptado sin el que los modelos tienen poco sentido.

Una instancia es una entidad en un sistema en ejecución. El propósito de los modelos es describir las posibles configuraciones y la historia de las instancias. Las instancias no aparecen en modelos —las instancias están "ahí fuera" en el sistema real. Un modelo contiene descripciones de instancias. Una descripción puede caracterizar a un conjunto de instancias sin considerar el contexto individual de cada instancia; un clasificador es dicha descripción. Una descripción puede describir a una instancia individual con más o menos detalle, incluyendo su valor, sus relaciones con otras instancias, y su historia; una especificación de instancia es dicha descripción.

Una instancia tiene identidad. En otras palabras, en diferentes momentos durante la ejecución de un sistema, la instancia puede ser identificada con la misma instancia en otros momentos, aunque el valor de la instancia cambie. En cualquier momento, una instancia tiene un valor que puede ser expresado en términos de valores de datos y referencias a otras instancias. Un valor de datos es un caso degenerado. Su identidad es la misma que su valor; o considerado desde un punto de vista diferente, no tiene identidad. La identidad representa a un manejador con el que se puede referenciar y manipular una instancia. Es distinto al valor de la instancia. Puede implementarse utilizando direcciones, claves u otros mecanismos, pero es un concepto abstracto distinto de sus posibles implementaciones.

Un valor es una entidad inmutable de algún tipo matemático. Los valores no se pueden cambiar; simplemente existen. Las operaciones se pueden definir sobre valores que producen otros valores, pero dichas operaciones no cambian los valores; sólo se pueden cambiar las instancias (mutables).

Además a la identidad y al valor, cada instancia tiene un descriptor que restringe los valores que puede tener la instancia. Un descriptor es un elemento del modelo que describe instancias. Es la dicotomía descriptor-instancia. La mayoría de los conceptos de modelado en UML tienen este carácter dual. El contenido principal de la mayoría de los modelos es descriptores de varios tipos. El propósito del modelo es describir los posibles valores de un sistema en términos de sus instancias y sus valores.

Cada tipo de descriptor describe un tipo 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 demás. Algunas instancias

no tienen nombres familiares y normalmente se pasan por alto excepto en escenarios muy formales, que sin embargo no suelen existir. Por ejemplo, un estado describe posibles ocurrencias del estado durante una traza de ejecución.

Un modelo describe los posibles valores de un sistema y su comportamiento en su progresión desde un valor a otro durante la ejecución. El valor de un sistema es el conjunto de todas sus instancias y los valores de dichas instancias. El valor de un sistema es válido si cada instancia es la instancia de algún descriptor del modelo, y si todas las restricciones implícitas y explícitas del modelo son satisfechas por el conjunto de instancias.

Los elementos de comportamiento en un modelo describen cómo el sistema y sus instancias progresan de valor en valor. El concepto de identidad de instancias es esencial para su descripción. Cada paso de comportamiento es la descripción del cambio de los valores de un pequeño número de instancias en términos de sus valores previos. El resto de las instancias del sistema mantienen sus valores sin cambios. Por ejemplo, una operación local sobre un objeto se puede describir mediante expresiones para los nuevos valores de cada atributo del objeto sin hacer cambios en el resto del sistema. Una función no local se puede descomponer en funciones locales sobre varios objetos.

Observe que las instancias en un sistema en ejecución no son elementos del modelo. No son parte del modelo en absoluto. Nada en un modelo es una *descripción* de algo en el sistema en ejecución. Una especificación de instancia es una descripción de una instancia o grupo de instancias. Puede ser más o menos específico —puede describir una sola instancia con su valor completo en un escenario muy específico o puede describir un conjunto de instancias con cierta ambigüedad en un rango de situaciones posibles.

Instancia directa. Cada objeto es la instancia directa de alguna clase y la instancia indirecta de los antecesores de la clase. Esto también es el caso de las instancias de otros clasificadores. Un objeto es una instancia directa de una clase si la clase describe la instancia y ninguna otra clase descendiente también describe al objeto. En el caso de clasificación múltiple, una instancia puede ser una instancia directa de más de un clasificador, ninguno de los cuales es antecesor de ninguno de los demás. Bajo cierta semántica de ejecución, uno de los clasificadores se designa como la clase de implementación y los otros se designan tipos o roles. El descriptor completo es la descripción completa implícita de una instancia —todos sus atributos, operaciones, asociaciones y otras propiedades— si fue obtenido por una instancia desde su clasificador directo o desde un clasificador antecesor mediante herencia. En caso de clasificación múltiple, el descriptor completo es la unión de las propiedades definidas por cada clasificador directo.

Creación. *Véase* instanciación para ver una descripción de cómo se crean las instancias.

Notación

Las instancias sólo existen en el “mundo real”, y por tanto no aparecen en los modelos. Las especificaciones de instancia aparecen en modelos. Es una distinción semántica sutil pero importante. Cuando se describen las instancias en un sistema en ejecución, se está construyendo un modelo de ello —un modelo de instancia. Las especificaciones de instancia son los elementos en dicho modelo. *Véase* especificación de instancia para ver su notación.

instancia de

Relación entre una instancia y su descriptor. Se modela en una especificación de instancia mediante un conjunto de clasificadores.

Véase instancia, especificación de instancia.

instancia de caso de uso

Ejecución de una secuencia de acciones especificadas en un caso de uso. Una instancia de un caso de uso. *Véase* caso de uso.

instancia directa

Instancia, como un objeto, cuyo descriptor más específico, como una clase, es una clase determinada. Utilizado en frases 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

Entidad que es una instancia de un elemento, como una clase, y también es una instancia de un hijo del elemento. Es decir, es una instancia pero no una instancia directa.

instanciable

Capaz de tener instancias directas. Sinónimo: concreto/a.

Véase también abstracto/a, instancia directa, elemento generalizable.

Semántica

Los elementos generalizables se pueden declarar como abstractos o instanciables. Si son instanciables se pueden crear instancias directas.

instanciación

Creación de nuevas instancias de elementos del modelo.

Véase también inicialización.

Semántica

Las instancias se crean en tiempo de ejecución como resultado de acciones de creación primitivas u operaciones de creación. En primer lugar se crea una identidad para la nueva instancia; a

continuación se asigna su estructura de datos como prescribe su descriptor; y por último se inicializan los valores de sus propiedades como prescribe el descriptor y el operador de creación. Conceptualmente, todo esto ocurre al mismo tiempo, pero es conveniente para el modelado y la implementación dividir la instanciación en varias etapas.

La dependencia de uso de instanciación muestra la relación entre una operación que crea instancias o una clase que contiene dicha operación y la clase de los objetos que se están instanciando.

Objetos. Cuando se instancia (crea) un nuevo objeto, se crea con identidad y memoria y se inicializa. La inicialización de un objeto define los valores de su atributo, sus asociaciones y su control de estado.

Normalmente, cada clase concreta tiene una o más operaciones de construcción de ámbito de clase (estáticas), cuyo propósito es crear nuevos objetos para la clase. Lo que hay por debajo de todas las operaciones de construcción es una operación primitiva implícita que crea una nueva instancia en bruto que luego se inicializa mediante las operaciones de construcción. Después de que se haya creado la instancia en bruto, tiene la forma prescrita por su descriptor, pero sus valores no han sido inicializados todavía, de forma que pueden ser semánticamente inconsistentes. Por tanto, una instancia no está disponible para el resto del sistema hasta que ha sido inicializada, lo que se produce inmediatamente después de la creación de la instancia en bruto.

Enlaces. De manera similar, los enlaces se crean mediante acciones u operaciones de creación, normalmente por operaciones de ámbito de instancia asociadas a una de las clases participantes, más que mediante operaciones constructoras sobre el propio elemento de asociación (aunque es una posible técnica de implementación bajo ciertas circunstancias). De nuevo, hay una operación primitiva implícita subyacente que crea un nuevo enlace entre una tupla de objetos. Esta operación no tiene efecto si ya existe un enlace de la misma asociación entre la tupla de objetos (a no ser que la asociación tenga un contenedor en uno de sus extremos). Con una asociación ordinaria, no hay nada más que hacer. Sin embargo, un enlace de una clase de asociación requiere la inicialización de sus valores de atributos.

Instancias de casos de uso. La instanciación de un caso de uso significa que se crea una instancia de un caso de uso, y la instancia del caso de uso comienza a ejecutarse al comienzo del caso de uso que la controla. La instancia del caso de uso temporalmente puede seguir otro caso de uso relacionado mediante relaciones de extensión o inclusión antes de que continúe ejecutando el caso de uso original. Cuando la instancia del caso de uso llega al final del caso de uso que está siguiendo, la instancia del caso de uso termina.

Otras instancias. Se pueden crear instancias de otros descriptores en un proceso similar de dos pasos: Primero realizar una creación en bruto para establecer la identidad y para asignar la estructura de datos, después inicializar los valores de la nueva instancia de forma que cumpla con todas las restricciones relevantes. Por ejemplo, una activación se crea de forma implícita como consecuencia directa de una llamada a una operación.

El mecanismo exacto de creación de instancias es responsabilidad del entorno de ejecución.

Notación

Una dependencia de instanciación se representa como una flecha discontinua desde la operación o clase que realiza la instanciación hacia la clase que está siendo instanciada; se adjunta el estereotipo «**instantiate**» a la flecha.

En un diagrama de secuencia, la instanciación de una nueva instancia se representa colocando un rectángulo en la cabeza de una flecha de mensaje. La línea de vida de la nueva instancia se dibuja por debajo de dicho rectángulo.

Discusión

La instanciación a veces se utiliza con el significado de ligar una plantilla para producir un elemento ligado, pero la ligadura es más específica para esta relación.

instanciar

Crear una instancia de un descriptor.

Véase instanciación.

instantánea

Una colección de objetos, enlaces, y valores que forman la configuración de un sistema en un instante durante su ejecución.

instantiate (estereotipo de Dependencia de uso)

Dependencia entre clasificadores que indica que las operaciones sobre el cliente crean instancias del proveedor.

Véase instanciación, uso.

intención[1]

Especificación formal de las propiedades estructurales y de comportamiento de un descriptor. Conocido en ocasiones como intensión. Contrastar con: extensión.

Véase también descriptor.

Semántica

Un descriptor, como una clase o asociación, tiene tanto una descripción (su intención) como un conjunto de instancias que lo describe (su extensión). El propósito de la intención es especificar las propiedades estructurales y de comportamiento de las instancias de una forma ejecutable.

---

[1] *N. del T.:* Palabra conocida en inglés como intent e intension.

interacción

Especificación de cómo se intercambian los mensajes a lo largo del tiempo entre roles dentro de un contexto para realizar una tarea. Una interacción especifica un patrón de comportamiento. El contexto es proporcionado por un clasificador o por una colaboración. En una instancia de una interacción, los objetos están vinculados a sus roles y una traza particular de mensajes entre objetos debe ser consistente con la especificación.

Semántica

Un conjunto reutilizable de objetos conectados se puede especificar utilizando un clasificador estructurado o una colaboración. Cada objeto es una parte dentro de un contexto bien definido.

Los objetos u otras instancias se comunican dentro de un contexto para llevar a cabo un propósito (como realizar una operación) mediante el intercambio de mensajes. Los mensajes pueden incluir señales y llamadas, así como interacciones implícitas a través de condiciones y eventos temporales. Un patrón de intercambio de mensajes para llevar a cabo un propósito específico se conoce como interacción.

Trazas. Una interacción describe un conjunto de trazas posibles. Una traza es una historia de ejecución particular. Comprende un conjunto parcialmente ordenado de ocurrencias de evento, incluyendo el envío o recepción de mensajes. En casos sencillos, las ocurrencias de evento (e incluso los mensajes) están normalmente totalmente ordenados; en casos más complejos, se pueden solapar.

Por ejemplo, una traza que implique una operación bancaria podría ser “insertar tarjeta de crédito, proporcionar la contraseña, solicitar retirada, recibir dinero, retirar tarjeta”. Una interacción puede describir varias trazas. Por ejemplo, otra traza para la misma interacción podría incluir “insertar tarjeta, proporcionar contraseña, banco rechaza la tarjeta debido a una contraseña incorrecta”. La semántica de una interacción se puede definir mediante las trazas que permite y las que prohíbe.

Estructura

Una interacción describe el comportamiento de un clasificador estructurado o una colaboración. Una interacción contiene un conjunto de líneas de vida. Cada línea de vida corresponde a una parte interna de un clasificador o de un rol de una colaboración. Cada línea de vida representa una instancia o conjunto de instancias sobre el tiempo. La interacción describe la actividad interna de las partes y los mensajes que intercambian.

Una interacción se estructura como un árbol de interacciones anidadas cuyas hojas son especificaciones de ocurrencia, especificaciones de ejecución y restricciones sobre el estado de la instancia destino y sus partes. Las construcciones que ensamblan interacciones más grandes fuera de otras más pequeñas incluyen construcciones de control estructuradas (bucle, condicional, paralelo y otras). Los nodos anidados del árbol se conocen como fragmentos de interacción. La mayoría de la estructura sintáctica de las interacciones se describe dentro de artículos sobre los distintos tipos de fragmentos de interacción.

Dentro de una interacción primitiva, las especificaciones de ocurrencia dividen una línea de vida en segmentos de tiempo sucesivos. Una especificación de ocurrencia es un punto en una

línea de vida que representa una pieza interesante de comportamiento que afecta a una instancia, como el envío o recepción de un mensaje o un cambio de estado de la instancia.

Los mensajes conectan pares de especificaciones de ocurrencia en diferentes líneas de vida (u ocasionalmente, en la misma línea de vida). Un mensaje se puede identificar mediante dos especificaciones de ocurrencia: una que representa el envío de un mensaje mediante una instancia y otra que representa la recepción de un mensaje por medio de la otra instancia. Un mensaje también tiene parámetros que describen el tipo de los mensajes y los valores de sus argumentos.

Las especificaciones de ejecución son regiones de una línea de vida que representa actividad de ejecución de las instancias descritas por la línea de vida. La actividad de ejecución incluye acciones primitivas así como la ejecución de operaciones que pueden abarcar varias capas de estructura anidada.

Para más detalles, *véase* fragmento combinado, fragmento de interacción, especificación del suceso, mensaje, especificación de una ejecución.

Notación

Las interacciones se muestran como diagramas de secuencia o como diagramas de comunicación.

Los diagramas de secuencia muestran explícitamente secuencias de especificaciones de sucesos, mensajes y representaciones de activaciones de métodos. Sin embargo, los diagramas de secuencia sólo muestran los objetos participantes y no sus relaciones con otros objetos o sus atributos. Por tanto, no muestran totalmente la vista contextual de una colaboración.

Los diagramas de comunicación muestran el contexto completo de una interacción, incluyendo los objetos y sus relaciones con otros objetos dentro de la interacción. Los diagramas de comunicación especifican secuencias de mensajes mediante etiquetas anidadas, de forma que no proporcionan una imagen visual de las secuencias de tiempo globales.

Los diagramas de secuencia a menudo son mejores para entender las interacciones de usuario y el diseño de los casos de uso, mientras que los diagramas de comunicación suelen ser mejores para entender algoritmos en estructuras de datos circulares y también para planear algoritmos que impliquen navegaciones dentro de redes de objetos.

Véase diagrama de interacción.

Historia

En UML2, las interacciones fueron fuertemente influenciadas por los Gráficos de Secuencia de Mensajes (MSC) de la Unión Internacional de Telecomunicaciones (ITU). Tienen construcciones muy similares pero no son completamente idénticos. La inclusión de estas construcciones puede influenciar al estándar ITU, y los dos conjuntos de construcciones pueden converger en el futuro.

Las interacciones de UML2 son una gran mejora respecto a las interacciones de UML1, que eran débiles a la hora de representar construcciones más complicadas, como condicionales, bucles e hilos concurrentes. La construcción de fragmento combinado de las interacciones de UML2 (y originalmente de MSC) maneja construcciones estructuradas de una forma limpia y potente. Las construcciones de UML1 están obsoletas y no se deberían utilizar.

intercalado semántico

Semántica de las interacciones en las que los eventos desde diferentes trazas pueden llegar en cualquier orden relativo cuando se fusionan las trazas, aunque los eventos en cada traza preserven su ordenación en la fusión. No tiene sentido que aquellos eventos no relacionados de diferentes trazas puedan ocurrir al "mismo tiempo"; el concepto de "mismo tiempo" no tiene significado para eventos independientes.

Véase interacción, traza.

interfaz

Declaración de un conjunto coherente de características y obligaciones públicas; contrato entre proveedores y consumidores de servicios.

Véase también clasificador, realización.

Semántica

Una interfaz es un clasificador para las propiedades, operaciones y recepciones visibles externamente de un clasificador de implementación, sin especificar su estructura interna. Una interfaz puede restringir la forma en que se invocan sus operaciones y puede imponer precondiciones y postcondiciones sobre sus operaciones. Cada interfaz a menudo especifica sólo una parte limitada del comportamiento de una clase real. Una interfaz debería definir un conjunto coherente de capacidades, mientras que una clase de implementación a menudo combina varios propósitos. Una clase puede dar soporte a varias interfaces, bien disjuntas o bien solapadas en su efecto. Una interfaz no tiene aspecto privado; todo su contenido es público.

Las interfaces pueden tener relaciones de generalización. Una interfaz hija incluye todo el contenido de sus antecesores pero puede añadir contenido adicional. Una interfaz es equivalente en esencia a una clase abstracta sin atributos y sin métodos, y sólo con operaciones abstractas. Todas las características de una interfaz tienen visibilidad pública (por el contrario, no habrá punto para incluirlas, ya que una interfaz no tiene nada "dentro" que pueda usarlas).

Una interfaz no tiene instancias directas. Una interfaz representa un contrato que debe ser cumplimentado por instancias de clasificadores que realizan la interfaz. Una interfaz que especifique el comportamiento de un clasificador disponible para otros clasificadores no especificados se conoce como interfaz proporcionado. Una interfaz que especifique el comportamiento solicitado por un clasificador hacia otro clasificador no especificado se conoce como interfaz obligatoria.

Un clasificador que implementa una interfaz no necesita tener la misma estructura exacta que la interfaz; simplemente debe proporcionar los mismos servicios a los solicitantes externos. Por ejemplo, un atributo en una interfaz puede ser implementado por operaciones en la implementación o los nombres de las operaciones en la implementación pueden ser específicos de la implementación (pero muchos modeladores esperarán que esté proporcionada por la implementación la operación exacta).

Una interfaz representa una declaración de servicios puestos a disposición o requeridos por clasificadores anónimos. El propósito de las interfaces es desacoplar el conocimiento directo de

los clasificadores que deben interactuar para implementar comportamiento. Las instancias pueden efectuar llamadas sobre instancias de clasificadores que implementan sus interfaces obligatorias, sin necesitar asociaciones directas entre los clasificadores que las implementan.

Estructura

atributos	Valores de estado que deben ser mantenidos por los clasificadores implementadores, no necesariamente como valores atributo.
operaciones	Servicios que pueden invocar objetos anónimos y no especificados sobre un objeto que los implemente. Una interfaz no puede proporcionar métodos para implementar sus operaciones.
recepciones	Señales que pueden enviar objetos anónimos a un objeto implementador.
restricciones	Precondiciones, postcondiciones u otras restricciones en la invocación de servicios de un objeto implementador.
anidadas	Interfaces anidadas que deben ser soportadas por clasificadores que implementen la interfaz. El concepto no está bien explicado en el documento oficial de UML.
protocolo	Máquina de estados de protocolo que especifica el orden en el que pueden ser invocados los servicios de los objetos implementadores.

Una interfaz puede tener asociaciones con otras interfaces. Esto significa que debe existir una asociación entre instancias de clasificadores que realizan las interfaces. Cada interfaz es una interfaz proporcionada para la otra interfaz.

Una interfaz puede tener asociaciones con un clasificador. Esto significa que debe existir una asociación entre instancias de clasificadores que realicen la interfaz e instancias del clasificador. La interfaz es una interfaz proporcionada para el clasificador.

Implementación

Si un clasificador implementa una interfaz, debe declarar o heredar todas las operaciones de la interfaz o proporcionar comportamiento equivalente. Puede contener operaciones adicionales (*véase* realización). Si el clasificador realiza más de una interfaz, debe contener cada operación encontrada en cualquiera de sus interfaces. La misma operación puede aparecer en más de una interfaz. Si sus signaturas coinciden, deben representar la misma operación ya que en caso contrario entran en conflicto y el modelo estará mal formado. (Una implementación puede adoptar reglas específicas del lenguaje para la concordancia de signaturas. Por ejemplo, en C++, los nombres de los parámetros y los tipos de retorno se ignoran.)

Los atributos deben ser implementados de alguna forma en el clasificador, pero UML es un tanto impreciso en qué compone la implementación. Presumiblemente una implementación que implementa atributos por alguna colección de operaciones también es libre de renombrar y reorganizar las operaciones durante el diseño. Si una interfaz se considera como una especificación de las operaciones exactas que deben ser proporcionadas por un clasificador, entonces es mejor o bien evitar la declaración de atributos en la interfaz o esperar que aparezcan también directamente en la implementación.

La especificación UML2 define la implementación como un caso especial de realización, pero la distinción entre realización e implementación parece mínima.

Notación

Una interfaz es un clasificador y puede representarse utilizando el símbolo rectangular con la palabra clave «**interface**». Se coloca una lista de atributos y operaciones definidas por la interfaz en el compartimento de atributos u operaciones. Todas las características son públicas. También se pueden incluir en el compartimento de operaciones las señales que lleven la palabra clave «**signal**», o pueden listarse en un compartimento separado con el nombre **signals**.

La relación de realización (interfaz proporcionado) se representa mediante una línea discontinua con una cabeza de flecha triangular sólida (un "símbolo de generalización discontinuo") desde un rectángulo del clasificador hasta un rectángulo de interfaz. También es posible usar una flecha de dependencia (una línea discontinua con una cabeza de flecha normal) con la palabra clave «**interface**». El clasificador proporciona la interfaz a otros.

La relación obligatoria se representa mediante una flecha de dependencia desde el rectángulo de un clasificador hasta el rectángulo de una interfaz. Se puede incluir opcionalmente la palabra clave «**use**», pero no es necesario.

Las asociaciones entre dos interfaces se pueden representar mediante una línea sólida de asociación. Si la asociación es bidireccional, la línea no tiene cabezas de flecha. Si la asociación es dirigida, se coloca una cabeza de flecha normal al final cerca de la interfaz de destino.

Hay disponible una notación más compacta para representar interfaces proporcionadas y obligatorias de clasificadores en aquellos lugares donde no es necesario mostrar el contenido de la interfaz. Para mostrar una interfaz proporcionada por un clasificador, la interfaz se puede representar como un pequeño círculo adjunto mediante una línea sólida al clasificador que proporciona la interfaz. El nombre de la interfaz se coloca debajo (o cerca) del círculo. La notación del círculo no muestra la lista de características que soporta la interfaz. Utilice el símbolo rectangular para mostrar la lista de características.

Para mostrar una interfaz obligatoria de un clasificador, una interfaz se puede mostrar como un pequeño semicírculo adjunto mediante una línea sólida a los clasificadores que requieren la interfaz. El nombre de la interfaz se coloca debajo (o cerca) del semicírculo.

Para mostrar que dos clasificadores comparten una interfaz, utilice el mismo nombre. Para más énfasis, se puede dibujar una flecha de dependencia desde una interfaz obligatoria a la interfaz proporcionada correspondiente.

En el caso de puertos externos de componentes, se puede conectar un símbolo de interfaz obligatoria o proporcionada directamente al símbolo del puerto. Un puerto complejo, es decir, uno que tenga tanto interfaces obligatorias como proporcionadas, se representa conectando varios símbolos de interfaz hacia un único símbolo de puerto.

Ejemplo

La Figura 14.161 muestra una vista simplificada de clases financieras que trabajan con precios de valores. El **PlanificadorFinanciero** es una aplicación de finanzas personal que realiza un seguimiento de inversiones, así como de gastos personales. Necesita tener la capacidad de actualizar los precios de los valores. El **AnalizadorDeFondosDeInversión** examina fondos de inver-

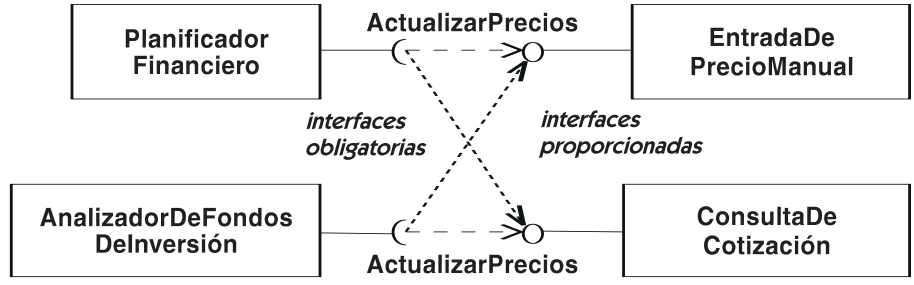


Figura 14.161 Proveedores de interfaces y clientes

sión en detalle. Necesita tener la capacidad de actualizar los precios de los valores subyacentes, así como los precios de los fondos. La capacidad de actualizar los precios de valores se representa mediante la interfaz **ActualizarPrecios**. Hay dos clases que implementan esta interfaz, mostradas por las líneas continuas que las conectan con el símbolo de la interfaz. La clase **EntradaDePrecioManual** permite a un usuario introducir precios de forma manual para los valores seleccionados. La clase **ConsultaDeCotización** recupera los precios de los valores desde un servidor de cotizaciones utilizando un módem o una conexión a Internet.

La Figura 14.162 muestra la notación completa para una interfaz con una palabra clave en un símbolo de clase. Vemos que esta interfaz implica dos operaciones —pedir el precio de un valor y obtener un valor, y enviar una lista de valores y recibir una lista de precios que han cambiado. En este diagrama, la clase **ConsultaDeCotización** está conectada a la interfaz utilizando una flecha de realización y **PlanificadorFinanciero** está conectado mediante una flecha de dependencia, pero es la misma relación mostrada en el diagrama previo, simplemente con una notación más explícita.

Este diagrama también muestra una interfaz nueva, **ActualizadorPeriódicoDePrecios**, que es un hijo de la interfaz original. Hereda las dos operaciones y añade una tercera operación que envía una petición para una actualización de precios periódica y automática. Esta interfaz es realizada por la clase **ServidorDeCotizaciones**, un servicio de suscripción. Implementa las mismas dos operaciones que **ConsultaDeCotización** pero de forma diferente. No comparte la implementación de **ConsultaDeCotización** (en este ejemplo) y por tanto no hereda la implementación de él.

La Figura 14.162 muestra la diferencia entre la herencia de interfaces y la herencia completa. La última implica la primera, pero un interfaz hijo puede ser implementado de forma diferente que la interfaz del padre. **ServidorDeCotizaciones** soporta la interfaz que implementa **ConsultaDeCotización**, llamada **ActualizarPrecios**, pero no hereda la implementación de **ConsultaDeCotización**. (En general, es conveniente heredar la implementación, así como las interfaces, de forma que las dos jerarquías a menudo son idénticas.)

Una dependencia entre una interfaz proporcionada y otra obligatoria se representa como una flecha discontinua desde la interfaz proporcionada a la interfaz obligatoria.

Una interfaz también puede contener una lista de las señales que maneja (Figura 14.163).

Las interfaces se utilizan para definir el comportamiento de las clases, así como de los componentes, sin restringir la implementación. Esto permite diferenciar la herencia de interfaces, como se declara en Java, de la herencia de implementación.

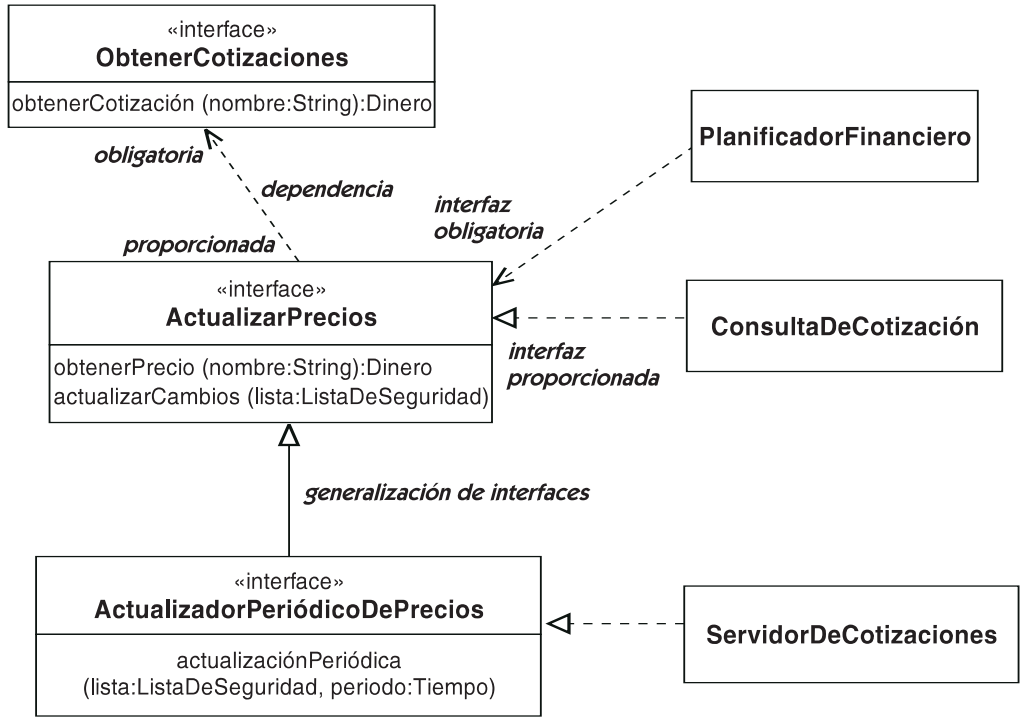


Figura 14.162 Notación completa de interfaces

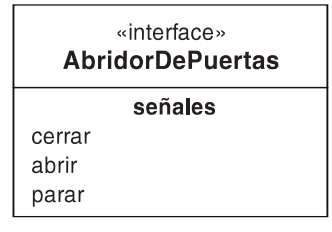


Figura 14.163 Interfaz con señales

interfaz obligatoria

Véase interfaz, puerto, interfaz proporcionada.

Semántica

Una interfaz requerida es una relación entre una interfaz y un clasificador que declara que el clasificador requiere los servicios descritos en la interfaz. Los servicios deben hacerse disponibles por otro clasificador, normalmente como uno de sus interfaces proporcionadas.

Lo requerido y las interfaces proporcionadas son complementarios. Si una clase declara una interfaz requerida y la otra clase requiere la misma interfaz, las clases pueden actuar recíprocamente sobre la interfaz sin la estructura adicional.

Notación

Una relación de la interfaz requerida se muestra por un medio círculo pequeño unido a un clasificador (o un puerto en un clasificador) por una línea. El medio círculo no es un símbolo separado y no representa la propia interfaz. El nombre de la interfaz se pone cerca del medio círculo.

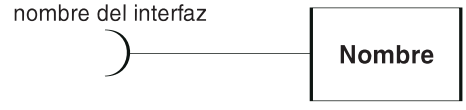


Figura 14.164 Diagrama de la estructura interna

interfaz proporcionado

Una interfaz que declara los servicios que un clasificador ofrece proporcionar a solicitantes anónimos.

Véase interfaz, puerto, interfaz requerida.

Semántica

Una interfaz proporcionada es una relación entre una interfaz y un clasificador (más bien un tipo de interfaz) que declara que la clase puede invocarse para proporcionar los servicios descritos en la interfaz. Una interfaz requerida es la relación complementaria que un clasificador requiere los servicios descritos en la interfaz. Si una clase declara una interfaz requerida y la otra clase requiere la misma interfaz, las clases pueden actuar recíprocamente sobre la interfaz, sin estructura adicional.

Notación

Una relación del tipo interfaz proporcionado se muestra por un círculo pequeño unido a un clasificador (o un puerto en un clasificador) por una línea (Figura 14.165). El círculo no es una parte separada del símbolo y no representa la propia interfaz. El nombre de la interfaz se pone cerca del círculo. Alternativamente, una interfaz proporcionada puede mostrarse usando la notación de la realización.

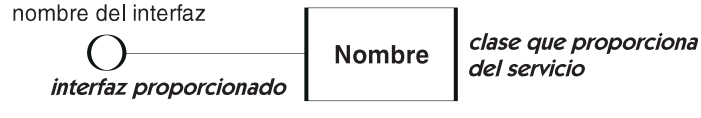


Figura 14.165 Notación de interfaz proporcionado

interrupción

Ocurrencia de un evento que termina una región de ejecución e inicia una ejecución que tiene como fin tratar con el suceso.

Semántica

Las interrupciones trabajan con aquellas situaciones en las que no se puede permitir siempre que una actividad termine su ejecución cuando se producen ciertos eventos, como sobrepasar un cierto límite de tiempo, una petición para abortar una transacción, o la ocurrencia de un evento que convierte un cálculo parcial en irrelevante.

Una región de actividad interrumpible es un conjunto de nodos y transiciones dentro del que se puede interrumpir una actividad si se produce el evento especificado. El desencadenador del evento que interrumpe se especifica como parte de la región de actividad interrumpible. También se especifica un nodo de actividad destino fuera de la región. Dicho nodo representa el manejador de la interrupción (este no es un término oficial de UML). Si se produce el evento especificado mientras hay actividad en la región, se termina toda la actividad de la región y se transfiere el control al nodo de actividad que maneja la interrupción. No hay posibilidad de reanudar la ejecución original después de una interrupción. El manejador de interrupción reemplaza la ejecución del nodo de actividad original y puede transferir el control donde quiera cuando finalice. La actividad fuera de la región de actividad interrumpible no se ve afectada por la interrupción y continúa concurrentemente junto con la ejecución del manejador de interrupción.

En el documento de UML2 no se especifican detalles de bajo nivel, como si las actividades terminadas tienen una última oportunidad de limpieza antes de su finalización y qué detalles del evento de interrupción se comunican al manejador de interrupción, aunque dichos detalles se pueden añadir en perfiles.

Notación

Una región de actividad interrumpible se representa en un diagrama de actividad como un rectángulo discontinuo con los bordes redondeados (Figura 14.166). El manejador de interrupción se representa como un símbolo de actividad separado de la región interrumpible. Una transición de interrupción se representa como una flecha dentada tipo “rayo” desde dentro de la región interrumpible hasta el límite del manejador de interrupción. El nombre del evento de interrupción se coloca cerca de la flecha dentada. Tenga cuidado de empezar la flecha dentro de la región interrumpible, o de lo contrario puede ser confundida con una excepción.

Discusión

UML2 ha proporcionado soporte rudimentario para las interrupciones, pero se podría esperar una capacidad mucho más amplia en el futuro, quizás en un perfil.

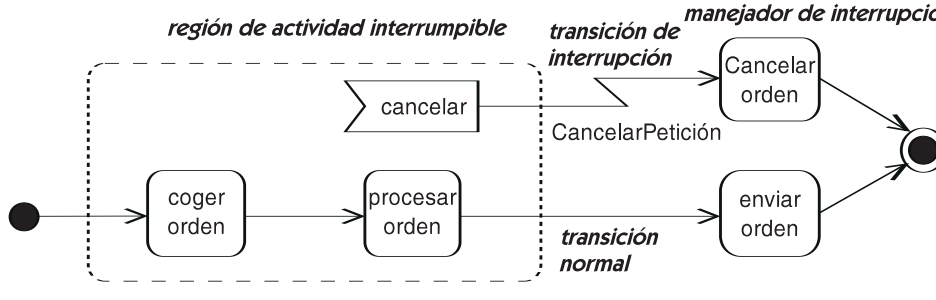


Figura 14.166 Interrupción

intervalo

Rango de valores entre dos valores designados.

Semántica

Los intervalos se utilizan a menudo para las multiplicidades (utilizando enteros) y el tiempo (utilizando expresiones de temporales).

Notación

Un intervalo se representa como una cadena de texto con el valor mínimo o una expresión seguida por un par de puntos (..) y seguida de nuevo por el valor máximo u otra expresión. Por ejemplo:

```
1..3
```

```
31 de Mayo de 2002..30 de Junio de 2002
```

invariante

Restricción que debe ser verdadera en todo momento (o al menos en todo momento en que ninguna operación esté incompleta), o al menos en todo momento durante un intervalo de tiempo especificado o cuando las condiciones especificadas son verdaderas.

Semántica

Una invariante es una expresión lógica que debe ser verdadera en todo momento cuando las condiciones especificadas son verdaderas. No se espera que sea cierta bajo una cierta granularidad, como por ejemplo durante la ejecución de una operación que actualice los valores sobre los que depende la restricción. Más que establecida formalmente, la granularidad exacta a menudo es implícita. Una invariante es una aserción, no una sentencia ejecutable. Si falla, el modelo está mal formado; no tiene la intención de especificar una acción correctiva (aunque el software de comprobación de errores podría tomar acciones de emergencia para evitar en una implementación daños en el sistema). Dependiendo de la forma exacta de la expresión, podría o no ser posible verificarla automáticamente por adelantado.

Véase también precondición, postcondición.

Estructura

Una invariante se especifica como una restricción. Su utilización depende de su contexto dentro de otra construcción de modelado.

invariante del estado

Una condición que debe ser verdad cuando se activa un estado.

Semántica

Un invariante del estado es una aserción de una restricción dada que debe ser verdad cuando se activa un cierto estado. Si no es verdad, el modelo incurre en un error. La restricción puede depender del objeto en estado activo, así como de los valores accesibles del objeto o de los valores globales.

También se puede poner un estado invariante en una línea de vida dentro de una interacción. El intervalo entre las especificaciones de la ocurrencia es equivalente a un estado. En este uso, la restricción se evalúa en el punto cuando el siguiente evento ocurre en la línea de vida.

Notación

Un invariante del estado puede ser denotado uniendo un comentario que contiene el texto de la restricción al símbolo del estado por una línea discontinua. Un invariante del estado en una línea de vida puede denotarse sobreponiendo el texto de la restricción (entre llaves) sobre la línea de vida o poniendo un símbolo del estado (rectángulo pequeño con las esquinas redondeadas) que contiene el nombre de un estado en la línea de vida (Figura 14.167).

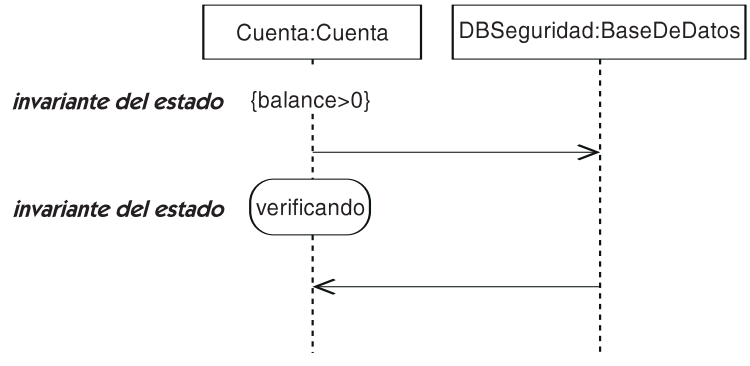


Figura 14.167 Invariante del estado en una línea de vida

invocación

Petición de ejecución de un elemento de comportamiento parametrizado designado o de una característica de comportamiento.

Semántica

Una solicitud de invocación es una acción que especifica un elemento de comportamiento o una característica de comportamiento y una lista de valores de argumentos. Para muchos tipos de invocación, también se designa un objeto destino. Los tipos de los valores en la lista de argumentos deben concordar con los parámetros correspondientes en el elemento de comportamiento. Dependiendo del tipo de invocación, la solicitud es o bien síncrona o asíncrona. Para una solicitud asíncrona, la acción forma el objeto destino, la designación del elemento de comportamiento y la lista de valores de argumentos en un paquete de solicitud que

se transmite al objeto destino (o a los destinos implicados por la acción específica). El hilo invocador entonces es libre de continuar sin esperar a la finalización, o incluso al inicio de la ejecución invocada.

Para una solicitud síncrona, el hilo invocador es bloqueado mientras la solicitud de invocación se transmite al destino y procede la ejecución solicitada. El paquete de solicitud de invocación incluye suficiente información para permitir una subsiguiente acción de retorno para devolver el control al hilo invocador; esta información de retorno es opaca e inaccesible para la ejecución destino excepto para realizar un retorno. Cuando la ejecución invocada finaliza, cualesquiera valores de retorno son transmitidos a la ejecución invocadora, que se desbloquea y se le permite continuar su ejecución.

Las acciones de invocación tienen las siguientes variedades, cada una con sus propios parámetros:

difusión	Envía una señal a un conjunto de objetos destino implícito. Los objetos están intencionados para representar todos los objetos dentro de una región especificada de un sistema. El tipo de la señal y la lista de argumentos son parámetros. Siempre asíncrona.
operación de llamada	Ejecuta un método encontrado mapeando una operación utilizando reglas de resolución específicas del sistema. El objeto destino, la operación, la lista de argumentos y la sincronía son parámetros.
actividad de llamada	Invoca una actividad directamente desde otra actividad. La actividad invocada, la lista de argumentos y la sincronía son parámetros. También se pueden invocar otros tipos de comportamiento, como máquinas de estados.
envío de señal	Envía una señal a un objeto destino. El objeto destino, el tipo de la señal y la lista de argumentos son parámetros. Siempre asíncrona.
envío de objeto	Envía un objeto señal a un objeto destino. El objeto destino y el objeto señal son parámetros. Siempre asíncrona.

Notación

No hay definida ninguna sintaxis para las acciones, pero puede valer para todos los usos informales (y para muchos formales) usar el nombre de la señal o de la operación seguida por una lista de argumentos entre paréntesis. Normalmente las palabras clave **broadcast**, **call** o **send** clarificarán situaciones ambiguas.

layer

Patrón de arquitectura cuyo cometido es agrupar paquetes con el mismo nivel de abstracción en un modelo. Cada capa representa un mundo virtual con un nivel de realidad determinado.

lenguaje de restricción de objetos

Véase OCL.

library (estereotipo de Artefacto)

Artefacto que representa una librería estática o dinámica.

Véase artefacto.

ligadura

La correspondencia de valores con parámetros para producir un elemento individual a partir de un elemento parametrizado. La relación de ligadura es una relación dirigida entre un elemento del modelo y un parámetro dentro del contexto de una invocación o uso de una plantilla.

Véase también elemento ligado, plantilla.

Semántica

Una definición parametrizada, como una operación, una señal o una plantilla, define la forma de un elemento. Un elemento parametrizado no se puede utilizar directamente, porque sus parámetros no tienen valores específicos. La ligadura es una relación dirigida que asigna valores a parámetros para producir un elemento nuevo y usable. La ligadura sirve en las operaciones para producir llamadas, en las señales para producir mensajes y en las plantillas para producir nuevos elementos del modelo. Los dos primeros se ligan durante la ejecución para producir entidades de tiempo de ejecución. Normalmente no figuran en los modelos, salvo como ejemplos o resultados de simulación. Los valores de los argumentos se definen dentro del sistema ejecución.

Sin embargo, una plantilla se liga en tiempo de modelado para producir nuevos elementos del modelo para su uso dentro del modelo. Los valores de los argumentos pueden ser otros elementos del modelo, como clases, además de los valores de datos, como los enteros o las cadenas. La relación de ligadura liga valores a una plantilla, produciendo un elemento real del modelo que puede ser utilizado directamente dentro del modelo.

Una relación de ligadura tiene un elemento proveedor (la plantilla), un elemento cliente (el elemento ligado recién generado) y una lista de valores para ligar a los parámetros de la plantilla. El elemento ligado se define sustituyendo cada valor del argumento por su correspondiente parámetro dentro de una copia del cuerpo de la plantilla. La clasificación de cada elemento debe ser la misma, o una descendiente, que la clasificación declarada de sus parámetros.

La ligadura no afecta a la propia plantilla. Cada plantilla se puede ligar muchas veces, cada una de las cuales produce un nuevo elemento ligado.

Notación

La ligadura de las plantillas se indica mediante la palabra clave **«bind»** vinculada a una flecha de línea discontinua que conecta el elemento generado (en la cola de la flecha) con la plantilla (en

la punta de la flecha). *Véase* la Figura 14.168. Los valores de los argumentos reales se muestran como una lista de expresiones de texto separadas por comas y encerradas entre los símbolos mayor y menor que, después de la palabra clave **«bind»** y sobre la flecha:

«bind»<**argumento**$_{\text{lista,}}$>

y en el que cada argumento tiene la siguiente forma:

nombre-parámetro → **valor**

Si los parámetros están ordenados (como es habitual con las operaciones), sólo se necesita incluir el valor. Obsérvese que el valor puede incluir tipos, si es apropiado para el parámetro.

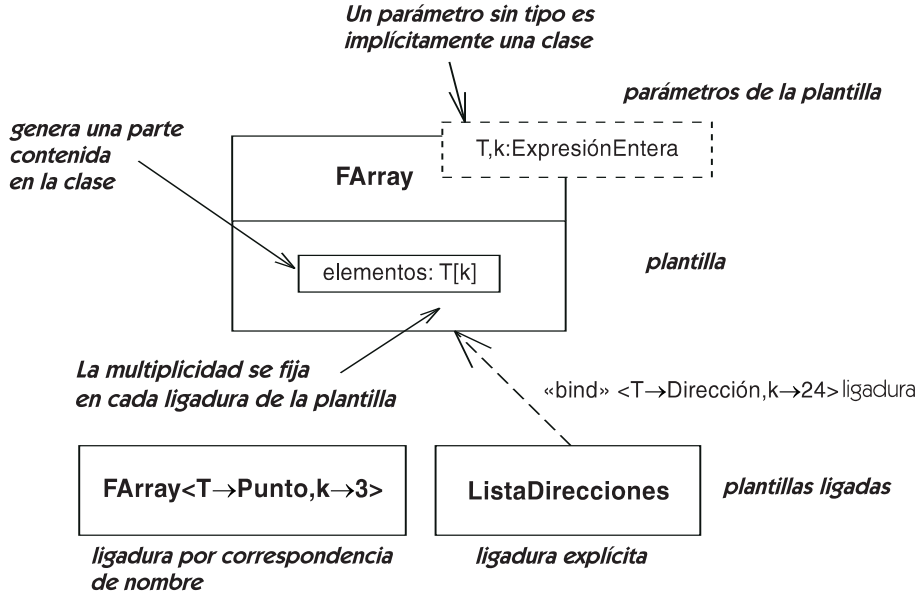


Figura 14.168 Plantillas: declaración y ligadura

Una notación alternativa y más compacta para la ligadura utiliza la correspondencia de nombres para evitar las flechas. Para indicar un elemento ligado (generado), el nombre de la plantilla va seguido de una lista, separada por comas, de expresiones de texto encerradas entre los símbolos mayor y menor que:

nombre-plantilla <**argumento**$_{\text{lista,}}$>

En cualquier caso, cada argumento se indica mediante una cadena de texto que se evalúa estáticamente durante la construcción del modelo. No se evalúa dinámicamente como en los argumentos de una operación o de una señal.

En la Figura 14.168, la forma explícita que utiliza la flecha declara una nueva clase **ListaDirecciones**, cuyo nombre se puede utilizar en modelos y expresiones. La forma implícita de una

línea **FArray<T→Punto,k→3>** declara una "clase anónima" sin nombre propio. Esta sintaxis de una línea se puede utilizar en expresiones.

Se pueden añadir atributos y operaciones adicionales a la clase ligada; la interpretación es que se crea una clase anónima mediante la ligadura, y la nueva clase con las características adicionales es una subclase de la clase anónima.

Historia

La notación de la ligadura ha cambiado en UML2.

Discusión

Los parámetros de las plantillas y los parámetros de las operaciones se encuentran en distintos niveles semánticos, pero representan conceptos semánticos relacionados, al menos conceptualmente.

La especificación utiliza una flecha compuesta por un guión y un símbolo mayor que (->) pero se puede considerar como un tipo de cobardía tipográfica, a la luz de la disponibilidad de conjuntos de caracteres modernos y de la muerte de las tarjetas perforadas.

La notación para la ligadura de plantillas ha cambiado ligeramente en UML2 y se han restringido las plantillas a elementos para los cuales tienen sentido.

línea de vida

Rol en una interacción que representa a un participante a lo largo de un periodo de tiempo y, por extensión, al participante mismo. En un diagrama de secuencia, se representa como una línea vertical, paralela al eje temporal, con un símbolo en la parte superior que muestra su nombre y su tipo.

Semántica

Una línea de vida representa a un participante un una interacción. Una interacción se basa en un contexto en el que interactúan los objetos. El contexto puede ser un clasificador estructurado o una colaboración. Las partes del clasificador o los roles de la colaboración tienen relaciones entre ellos mediante las que intercambian mensajes. Una línea de vida representa una de esas partes o roles.

Puesto que una parte o un rol pueden tener multiplicidad mayor que uno, puede representar varios objetos durante la ejecución. Una línea de vida representa sólo uno de esos objetos, y por tanto un rol general puede mapearse en varias líneas de vida en una interacción particular. Si la multiplicidad es mayor que uno, una expresión selectora opcional en cada línea de vida especifica a qué objeto del conjunto representa.

Una línea de vida indica el periodo durante el que existe un objeto. Un objeto es activo si posee un hilo de control —es decir, si es la raíz del hilo. Un objeto pasivo está activo temporalmente durante el tiempo en el que tiene un hilo de control pasando por él —es decir, durante el periodo de tiempo durante el que tiene una llamada a un procedimiento pendiente. Esto último se conoce como

especificación de ejecución (o activación). Incluye el tiempo durante el que un procedimiento está llamando a un procedimiento de nivel inferior. Esta distinción sólo es útil al considerar llamadas a procedimientos anidados; para sistemas de objetos concurrentes que interactúan de manera asíncrona, todos ellos deben ser considerados activos todo el tiempo en la mayoría de los casos.

Una línea de vida contiene una lista ordenada de especificaciones de suceso, cada una de las cuales modela el suceso de un evento. El orden representa la secuencia temporal en la que se producen los eventos. La ordenación relativa de las especificaciones de suceso en diferentes líneas de vida sólo tiene sentido si los mensajes las conectan o si se insertan dependencias de secuencia explícitas, o de otro modo se consideran concurrentes y puede aparecer en cualquier orden relativo real.

Notación

En un diagrama de secuencia, una línea de vida se representa como una línea discontinua vertical. (La línea puede ser sólida pero normalmente es discontinua.) La línea vertical representa la existencia del objeto en un periodo de tiempo particular (Figura 14.169).

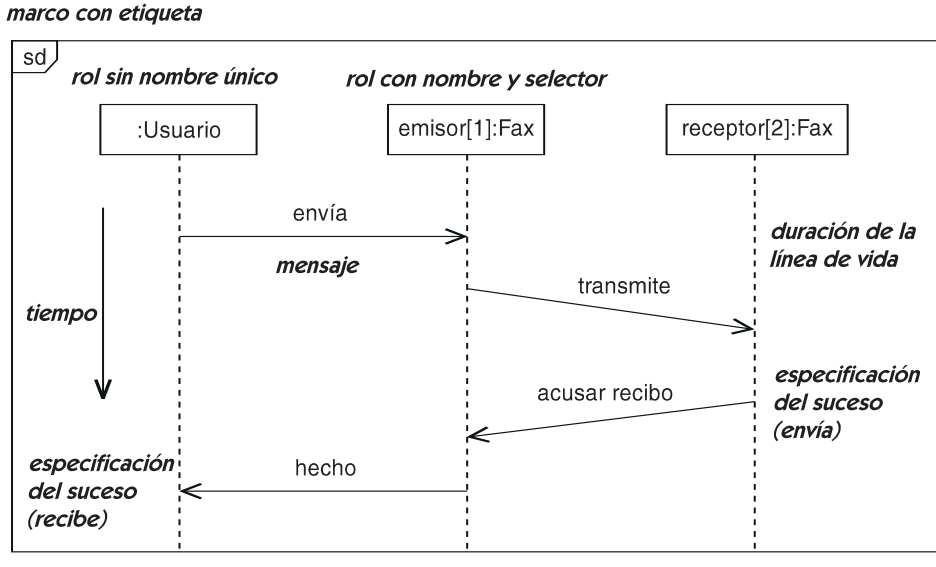


Figura 14.169 Líneas de vida

La creación del objeto se representa con un símbolo al principio de la línea de vida, un rectángulo en el punto en el que se crea el objeto. Si el rectángulo está en la parte superior del diagrama de secuencia, entonces se entiende que el objeto ya estaba creado cuando empezó la interacción. El nombre y el tipo del participante se muestra como una cadena de texto en el rectángulo, con la siguiente sintaxis:

`nombre`$_{\text{opc}}$ ⌊ **`[selector]`** ⌋$_{\text{opc}}$ ⌊ **`:Tipo`** ⌋$_{\text{opc}}$

donde **nombre** es el nombre del participante individual (normalmente asignado por el modelador de la interacción para su conveniencia), **selector** es una expresión que identifica un objeto

particular cuando el rol tiene multiplicidad mayor que uno, y **Tipo** es el tipo del objeto. El nombre a menudo se omite cuando sólo hay un objeto de un tipo dado en una interacción.

Se puede utilizar la cadena **self** para denotar la instancia del clasificador que posee la interacción.

Las flechas entre líneas de vida indican mensajes entre objetos. La línea de vida que se encuentra en la cola de la flecha está enviando el mensaje; la línea de vida que se encuentra en la cabeza de la flecha recibe el mensaje (por ejemplo, el proveedor de una operación). La intersección de una flecha de mensaje con una línea de vida es una especificación de suceso, el envío o recepción de un mensaje. La ordenación vertical de las especificaciones de suceso de una línea de vida indica el orden temporal relativo de envío o recepción de mensajes. La recepción de un mensaje por el receptor sigue a su envío por el emisor. No hay ordenación entre especificaciones de suceso en líneas de vida diferentes a no ser que haya un camino de mensaje entre ellos. Las especificaciones de suceso que no están relacionadas son concurrentes y pueden producirse en cualquier orden relativo durante la ejecución.

Otros tipos de especificaciones de suceso incluyen restricciones de estado, creación, destrucción y el comienzo y fin de la ejecución (aunque lo último a menudo corresponde con la recepción de los mensajes).

Si el objeto se crea o se destruye durante el periodo de tiempo mostrado en el diagrama, entonces su línea de vida comienza o concluye en el punto apropiado. De otro modo, va desde la parte superior del diagrama hasta la inferior. Se dibuja un símbolo de objeto en la parte superior de la línea de vida. Si el objeto se crea durante el tiempo representado en el diagrama, entonces el símbolo de objeto se dibuja en la cabeza de la flecha del mensaje que lo crea. En caso contrario, el símbolo de objeto se dibuja por encima de todas las flechas de mensaje.

Si el objeto se destruye durante el diagrama, su destrucción se marca con una gran *X*, bien en la cabeza de la flecha del mensaje que causa la destrucción o (en el caso de autodestrucción) justo por debajo del último mensaje de retorno enviado desde el objeto destruido. Un objeto que existe cuando comienza la transacción se muestra en la parte superior del diagrama (por encima de la primera flecha). Un objeto que existe cuando la transacción termina tiene su línea de vida continuando por debajo de la última flecha.

El periodo de tiempo durante el que un objeto está permanente o temporalmente activo se puede representar con un rectángulo fino (una línea doble sólida) que oculta la línea de vida. Se puede sobreponer una segunda doble línea, ligeramente desplazada, para representar la recursión. *Véase* especificación de una ejecución para más detalles. Mostrar las especificaciones de las ejecuciones sólo es útil para entender el flujo de control en las llamadas a procedimientos. Puesto que todos los objetos en un sistema asíncrono están siempre activos, las dobles líneas normalmente se omiten puesto que no añaden información en tales casos.

Véase la Figura 14.117 para ver un ejemplo que muestra la creación, destrucción y activaciones recursivas.

En un fragmento combinado, como un fragmento condicional, las líneas de vida pueden solapar varios operandos. En este caso, los diferentes operandos no representan secuencia temporal. La parte de una línea de vida dentro de un operando particular, sin embargo, representa una secuencia temporal de especificaciones de suceso.

En una construcción condicional o de repetición, se puede colocar una condición de guarda en la parte superior de una línea de vida dentro de un operando en un fragmento combinado. La condición de guarda determina si la bifurcación dada es elegida o si se realiza otra iteración. La

condición es una expresión en el objeto representado por la línea de vida. Se representa como una expresión de texto entre corchetes, con la forma:

```
[Expresión-lógica]
```

Véase la Figura 14.150 para ver un ejemplo de una condicional y la Figura 14.141 para ver un ejemplo de un bucle.

Una restricción de estado es una condición que debe ser verdadera un tiempo determinado durante una interacción. Se representa como una expresión de texto entre llaves sobre la línea de vida en el punto relativo en que se aplica. Debe ser satisfecha inmediatamente antes del siguiente evento de la línea de vida.

Una línea de vida puede ser interrumpida por un símbolo de estado para representar un cambio de estado. Se puede dibujar una flecha hasta el símbolo de estado para indicar el mensaje que causó el cambio de estado, o el símbolo de estado puede simplemente aparecer en la línea si representa un cambio durante un procesamiento interno. *Véase* la Figura 14.101 para ver un ejemplo.

Una corregión indica un conjunto de eventos que pueden suceder en cualquier orden, independientemente de su posicionamiento real en una línea de vida. *Véase* la Figura 14.89 para ver un ejemplo.

línea de vida del objeto

Véase línea de la vida.

lista

Colección ordenada de tamaño variable de elementos del modelo pertenecientes a otro elemento del modelo y anidados dentro de él; bolsa ordenada.

Véase también clasificador, multiplicidad, estado.

Semántica

La palabra lista es un atajo para una bolsa ordenada, es decir, una secuencia ordenada de valores en los que el mismo valor puede aparecer más de una vez (formalmente, un mapeo desde un subconjunto de enteros positivos a los elementos de un conjunto). Corresponde a establecer las propiedades de multiplicidad **isUnique =false e isOrdered=true**. Un conjunto sin duplicados cuyos elementos tienen una ordenación relativa se conoce como *conjunto ordenado*. La palabra *lista* implica que puede haber valores duplicados.

Un clasificador contiene varias listas de elementos subordinados, incluyendo atributos, operaciones y métodos. Un estado contiene una lista de transiciones internas. Otros tipos de elementos contienen listas de otros elementos. Cada tipo de lista se describe de forma individual. Este artículo describe las propiedades de las listas embebidas en general. Además de listas de atributos y operaciones, listas opcionales pueden mostrar otros valores predefinidos o definidos por el usuario, como responsabilidades, reglas o historias de modificaciones. UML no define esas listas opcionales. La manipulación de listas definidas por el usuario es dependiente de las herramientas.

Una lista embebida y los elementos de la lista pertenecen exclusivamente a la clase que los contiene u otro elemento contenedor. La propiedad no es compartida entre varios contenedores. Otras clases pueden ser capaces de acceder a los elementos de la lista —por ejemplo, mediante herencia o asociación— pero la propiedad de las listas contenidas para la edición del modelo pertenece al contenedor inmediato. Los elementos poseídos son almacenados, copiados y destruidos junto con sus contenedores.

Los elementos de una lista tienen un orden determinado por el modelador. El orden puede ser útil para el modelador —por ejemplo, puede ser utilizado por un generador de código para generar una lista de declaraciones en un lenguaje de programación. Si el modelador no presta atención al orden, puede ser porque el modelo está en la etapa de análisis o porque el lenguaje ignora la ordenación, luego el orden todavía existe en el modelo pero sencillamente puede ser ignorado por ser irrelevante.

Notación

Como componente de un especificador de multiplicidad, utilice la palabra lista entre llaves: **{list}**, **{seq}**, o **{sequence}**.

Una lista embebida en un rectángulo de clasificador aparece dentro de su propio compartimento como una lista de cadenas, una cadena por línea para cada elemento de la lista. Cada cadena es la representación codificada de una característica, como un atributo, operación, transición interna y demás. La naturaleza de la codificación se describe en el artículo para cada clase de elemento.

Ordenación. El orden canónico de las cadenas es el mismo que para los elementos de la lista dentro del modelo, pero la ordenación interna puede ser anulada opcionalmente y en ese caso las cadenas se ordenan de acuerdo con alguna propiedad interna, como el nombre, visibilidad o estereotipo. Sin embargo, observe que los elementos mantienen su orden original en el modelo subyacente. Simplemente se suprime en la vista la información de ordenación.

Puntos suspensivos. Unos puntos suspensivos (…) como elemento final de una lista o como elemento final de una sección delimitada de una lista indica que hay elementos adicionales en el modelo que cumplen el criterio de selección pero que no se muestran en la lista. En una vista diferente de la lista, dichos elementos pueden aparecer.

Estereotipo. Un estereotipo se puede aplicar a un elemento de lista. Se coloca una palabra clave de estereotipo entre comillas (« ») precediendo a la cadena del elemento.

Cadena de propiedad. Una cadena de propiedad puede especificar una lista de propiedades de un elemento. Se coloca a continuación del elemento una lista separada por comas de propiedades o restricciones, encerradas entre llaves ({ }).

Propiedades de grupo. También se pueden aplicar estereotipos y otras propiedades a grupos de elementos de lista. Si un estereotipo, palabra clave, cadena de propiedad o restricción aparecen en una línea sólo, entonces la línea no representa un elemento de lista. En cambio, las restricciones se aplican a cada elemento de la lista sucesivo como si se hubieran colocado directamente en cada línea. Se aplicará hasta que aparezca en la lista otra propiedad de grupo. Todas las propiedades de grupo se pueden cancelar insertando una línea con una palabra clave vacía («»), pero normalmente es más claro colocar todas las entradas no sujetas a propiedades de grupo al principio de la lista. La Figura 14.170 muestra la aplicación de estereotipos a varios elementos de lista.

Observe que las propiedades de grupo son simplemente una comodidad rotacional y que cada elemento del modelo tiene su propio valor distinto para cada propiedad.

Nombre del compartimento. Un compartimento puede mostrar un nombre que indique qué tipo de compartimento es. El nombre se muestra en una fuente distintiva (como en negrita o en un tamaño menor) centrado en la parte superior del compartimento. Esta capacidad es útil si se omiten algunos compartimentos o si se añaden compartimentos definidos por el usuario. Para una clase, los compartimentos predefinidos se llaman atributos y operaciones. Un ejemplo de compartimento definido por el usuario podría ser requisitos. El compartimento de nombre en una clase debe estar siempre presente y por tanto no requiere ni permite un nombre de compartimento. La Figura 14.170 y la Figura 14.171 muestran compartimentos con nombre.

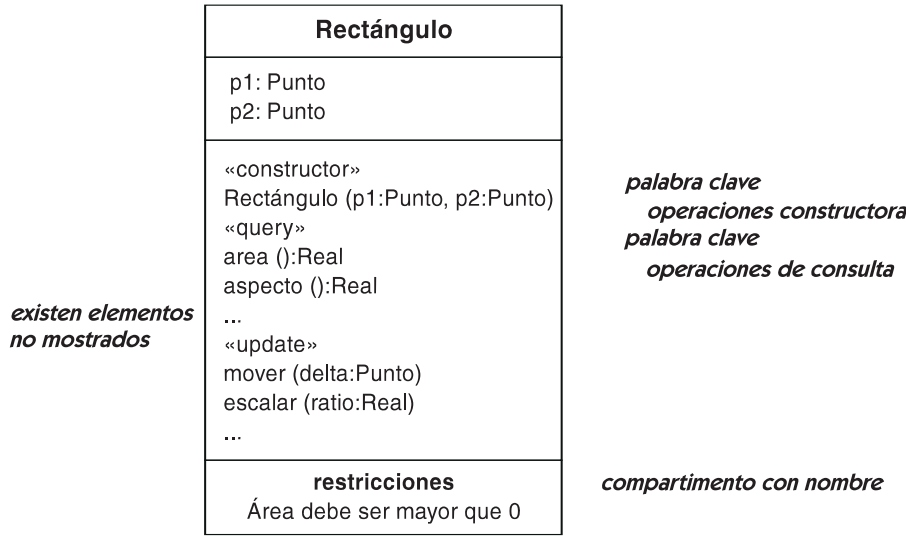


Figura 14.170 Palabra clave de estereotipo aplicada a grupos de elementos de lista

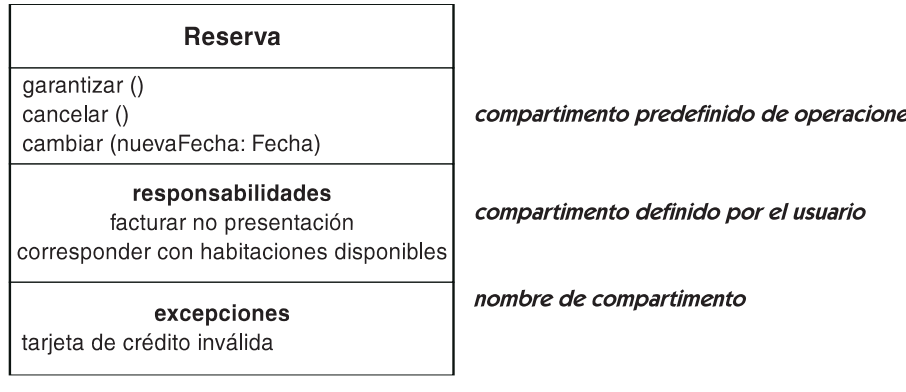


Figura 14.171 Compartimentos con nombres

Opciones de presentación

Ordenación. Una herramienta puede presentar los elementos de la lista de una forma ordenada. En ese caso, la ordenación inherente de los elementos no es visible. Una ordenación se basa en alguna propiedad interna y no indica información del modelo adicional. Las reglas típicas de ordenación incluyen ordenación alfabética, ordenación por estereotipo (como constructores, destructores, métodos ordinarios), ordenación por visibilidad (pública, protegida, privada) y demás.

Filtrado. Los elementos de la lista pueden ser filtrados de acuerdo a alguna regla de selección. La especificación de las reglas de selección es responsabilidad de las herramientas. Si una lista filtrada no muestra elementos, entonces no hay ningún elemento que cumpla el criterio de filtrado, pero la lista original puede contener (o puede que no) otros elementos que no cumplen el criterio y por tanto son invisibles. Es responsabilidad de las herramientas si se indica y cómo se indica la presencia de algún filtrado local o global, aunque un diagrama autónomo debería tener alguna indicación de dicho filtrado, si se quiere que sea entendible.

Si se suprime un compartimento, no se puede inferir nada acerca de la presencia o ausencia de sus elementos. Un compartimento vacío indica que ningún elemento cumple el filtro de selección (si hay alguno).

Observe que los atributos también se pueden mostrar por composición (*véase* la Figura 14.73).

lista de parámetros

Una especificación de los valores que una operación o la plantilla recibe. Una lista de parámetros es una lista ordenada de declaraciones de parámetro. La lista puede estar vacía, en cuyo caso la operación se llama sin parámetros.

Véase parámetro.

Notación

Una lista de parámetros es una lista separada por comas de declaraciones de parámetro entre paréntesis.

(**parametro**$_{\texttt{lista}}$)

Los paréntesis se muestran aún cuando la lista está vacía.

( )

literal de una enumeración

Valor de instancia de un tipo enumerado. Tiene un nombre y una posición relativa dentro de la lista de literales de su tipo enumerado. *Véase* enumeración.

llamada

La invocación de una operación o de un comportamiento.

Véase también acción, disparador de llamada.

Semántica

Una llamada es una acción que invoca una operación sobre un objeto destino. Una llamada proporciona valores reales para los parámetros de la operación. Cada valor debe ser compatible con su correspondiente parámetro, es decir, debe ser del mismo tipo o de un tipo descendiente. Si la operación tiene valores por defecto, estos se pueden omitir en la llamada, proporcionándose automáticamente los valores por defecto. La acción de operación de llamada tiene la operación como parámetro estático de la acción y tiene valores de tiempo de ejecución para los objetos destino y los argumentos.

La información sobre la llamada, incluyendo los valores de los argumentos, y en las llamadas síncronas información suficiente para devolver el control al llamador, se transmite al objeto destino. La forma de transmitir la información no está restringida y puede ser dependiente de la implementación. En particular, el objeto destino puede estar en el mismo sistema o en un sistema remoto, dado que no hay distinción entre las llamadas locales y remotas. En el caso de las llamadas concurrentes o asíncronas, no se puede presuponer el orden en el que se reciben las diferentes llamadas. Los perfiles de sistema son libres de restringir algunas de estas posibilidades.

La operación invocada se utiliza para causar un efecto en el objeto destino, utilizando las reglas de resolución de operación del sistema subyacente. Por ejemplo, una regla de búsqueda de método orientado a objetos examinaría la clase del objeto destino buscando un método vinculado a la operación; si no se encuentra el método, se examinarían las clases antecesoras hasta que se encontrara el método. Sin embargo, la especificación de UML2 no obliga a una determinada regla de resolución de operación. Otros posibles tipos de búsqueda de métodos son la delegación (como en el lenguaje Self), métodos antes-y-después, diversas formas de herencia y el disparo de transiciones de máquinas de estados (*véase* más adelante). Desafortunadamente, la especificación de UML no lista ni nombra distintas reglas de resolución. La mayoría de los modeladores probablemente utilizarán las familiares reglas orientadas a objetos de Smalltalk y Java, por lo que cualquier otra aproximación debería vincular comentarios al modelo.

Efecto como actividad. En este estilo, los métodos se modelan como actividades. Utilizando la regla de resolución orientada a objetos, si la operación no se puede resolver en una actividad, hay un error. La especificación de UML2 no establece si tal error se debe manejar en tiempo de ejecución como una excepción, o si el modelo se considera mal formado. Esta opción es un punto de variación semántica que es una opción de implementación.

Una vez que se determina el método, se crea una ejecución de él a la que se le proporcionan el objeto destino y los valores de los argumentos. La ejecución de la actividad continúa hasta que termina. Durante la ejecución de la actividad, se conserva cualquier valor de retorno que se produzca.

La llamada puede ser asíncrona o síncrona. En una llamada asíncrona, la ejecución del llamador continúa inmediatamente después de que se realice la llamada, y la ejecución de la operación invocada continúa concurrentemente con la ejecución del llamador. Cuando se completa la ejecución de la llamada asíncrona, simplemente termina. Si hay valores de retorno, son ignorados. No hay un flujo de control de vuelta al llamador.

En una llamada síncrona, la ejecución del llamador se bloquea durante la llamada. Se transmite, junto con los valores de los argumentos, suficiente información sobre la ejecución del llamador para permitir que se despierte al llamador cuando se completa la ejecución de la operación invocada. No se especifica la forma de esta información de retorno del control y una

implementación es libre de utilizar cualquier aproximación que funcione. La información de retorno de control no se encuentra explícitamente disponible para la operación invocada. Cuando se completa la ejecución del procedimiento invocado, se transmite al llamador el hecho de la finalización junto con cualquier valor de retorno generado por la operación invocada, al cual se despierta y se le permite continuar con su ejecución.

Semántica alternante de métodos. La especificación de UML2 permite la definición de semánticas arbitrarias para el efecto de recibir una llamada de operación, pero no proporciona ninguna forma formal de definir tales semánticas, ni proporciona muchas directrices sobre reglas significativas. Las siguientes directrices parecen razonables, pero no se encuentran en el documento de especificación. Una regla de resolución puede depender del objeto destino y de la operación, y podría no depender de los valores de los argumentos. En el caso convencional de la orientación a objetos, el método depende del tipo del objeto y de nada más. En este caso, se podrían vincular a clases y no a objetos. En una aproximación de delegación (como la que se utiliza en el lenguaje Self), los métodos se pueden vincular directamente a los objetos, y la búsqueda de métodos utiliza punteros de delegación en lugar de punteros de superclases. Para las llamadas síncronas es necesario que el efecto de resolución alcance un punto de retorno explícito, en el cual se puede devolver el control de tiempo al llamador, aunque no es necesario que el efecto invocado cese su ejecución inmediatamente. Se deben especificar de alguna manera los valores de retorno y deben estar disponibles en el punto de retorno. Es posible, para una llamada, invocar a más de una actividad. En ese caso, las reglas deben dejar claro en qué orden se ejecutan las actividades, incluyendo la posibilidad de ejecución concurrente. Incluso en las llamadas síncronas, el punto de retorno debe estar claro, bien porque se encuentra explícitamente en una de las actividades, bien porque sigue implícitamente la finalización de un conjunto de actividades designado. Si hay múltiples actividades para una llamada síncrona con valores de retorno, debe estar claro cómo se producen los valores de retorno. Probablemente la aproximación más sencilla es designar a una única actividad como la actividad principal, cuya finalización proporcionará valores de retorno y provoque un retorno del control al llamador. También son posibles reglas de retorno más complejas, pero tienen el riesgo de provocar semánticas carentes de significado e imposibilidad de implementación. Lo más seguro es que la mayoría de los modeladores utilizarán la tradicional semántica de resolución de un único método, basado en clases y orientado a objetos para la mayoría del modelado.

Efecto como transición de máquina de estados. La recepción de una petición de llamada por un objeto con una máquina de estados activa puede causar el disparo de una transición provocado por la operación. Los parámetros de la operación se encuentran disponibles para cualquier efecto vinculado a la transición. Para una llamada es posible desencadenar tanto un método, como una transición, si ambos se encuentran definidos para una clase dada. En ese caso, el método se ejecuta primero. La ejecución del método puede afectar al estado del objeto, y el nuevo estado estará disponible para la máquina de estados para evaluar las condiciones de guarda y ejecutar las actividades vinculadas. Si la llamada es síncrona, se devuelve cualquier valor de retorno del método al llamador cuando se complete el método, momento en el cual el llamador es desbloqueado y continúa su ejecución. En ese punto (o si no hay método), la llamada puede habilitar las transiciones desencadenadas por la operación. A pesar de que la llamada sea síncrona, la ejecución de las transiciones de la máquina de estados es asíncrona y no puede devolver valores. Si no hay método, se realiza inmediatamente la ejecución de una llamada síncrona, pero cualquier transición desencadenada por ella se realiza concurrentemente con la subsiguiente ejecución del llamador.

Aceptación explícita de una llamada. Hay una acción de aceptación que explícitamente espera la recepción de una llamada de una determinada operación. Cuando el objeto propietario recibe una llamada de una operación dada, la acción receptora recibe una copia de los valores de los argumentos de la llamada. Si la llamada fuera asíncrona, el llamador se bloquea y la acción receptora entrega un valor de salida opaco que contiene suficiente información para, más tarde, devolver el control al llamador. El procedimiento receptor no puede examinar o manipular esta información de ninguna forma, pero se puede copiar, almacenar y pasar. Cuando una acción posterior ejecuta una acción de respuesta en la misma operación, debe proporcionar una copia de la información de retorno opaca como un argumento, junto con los valores de retorno especificados por la operación. En ese momento se transmiten los valores de retorno al llamador, permitiéndosele continuar su ejecución. La ejecución del procedimiento receptor continúa después de la acción de respuesta hasta que realiza otra acción de recepción, termina o realiza una llamada por sí mismo. Si la llamada es asíncrona, el llamador continúa inmediatamente y no es necesaria ninguna acción de respuesta. Sin embargo, si se intenta una respuesta utilizando la información de retorno proporcionada en la acción de recepción sobre la llamada asíncrona, no hay ningún efecto y la ejecución continúa sin errores.

Si la clase tiene un método en una operación dada, la ejecución del método podría tomar precedencia sobre la acción de recepción (la cual podría no ejecutarse nunca). Si dos o más acciones de recepción sobre la misma operación están pendientes para un único objeto, no está determinado cuál de ellas recibe la llamada, aunque una y sólo una lo hará.

Puertos. Una llamada a un clasificador estructurado puede incluir un puerto en el objeto destino. El objeto que recibe la llamada puede distinguir qué puerto recibe la llamada y utilizar esta información dentro de un método o eligiendo entre métodos.

Llamada directa de un procedimiento. Hay una acción para llamar directamente a un comportamiento. La mayor parte del tiempo el comportamiento es un procedimiento (una actividad en términos de UML). No hay un objeto destino. La acción tiene valores de entrada para los argumentos del comportamiento. La llamada puede ser asíncrona o síncrona, en cuyo caso puede tener valores de retorno.

Una llamada directa representa una forma de modelar una llamada no orientada a objetos a un procedimiento específico, y también una forma de estructurar actividades extensas en piezas manejables.

Dependencias. Una dependencia de uso de llamada modela una situación en la cual una operación de la clase cliente (o la propia operación) llama a una operación de la clase proveedora (o a la propia operación). Se representa con el estereotipo **«call»**.

Notación

En un diagrama de secuencia o en un diagrama de comunicación, una llamada se muestra como un mensaje dirigido al objeto o clase destino.

En un diagrama de actividad, una llamada se muestra como una caja de esquinas redondeadas que contiene el nombre de la operación. Se puede mostrar, de forma opcional, una lista de argumentos entre paréntesis. Una llamada a un puerto utiliza la sintaxis:

`nombre-operación` via **`nombre-puerto`**

Una dependencia de llamada se muestra con una flecha de línea discontinua desde el llamador a la operación o clase llamada con el estereotipo **«call»**.

La mayoría de las llamadas se representarán como parte del código de los procedimientos en un lenguaje de programación.

Discusión

Véase resolución para una discusión de los diversos efectos que pueden invocar las llamadas. Entre ellas se incluyen la ejecución de procedimientos y la habilitación de transiciones de máquinas de estados.

Muchos lenguajes de programación tratan las llamadas a procedimientos remotos de forma diferente a las llamadas a procedimientos locales, principalmente por razones históricas de rendimiento y por limitaciones hardware. Ya no hay razones para distinguirlas en los modelos y UML no lo hace. No es necesario que un procedimiento invocado se ejecute en el mismo entorno que el procedimiento llamador, mientras se pase alguna información de control entre entornos.

Obsérvese que la acción de aplicar función no es una llamada, sino la ejecución de una función matemática primitiva y predefinida, como un cálculo numérico.

localización

Ubicación física de un artefacto, como un archivo, dentro de un entorno distribuido. En UML, la localización es discreta y las unidades de localización son nodos.

Véase también artefacto, nodo.

Semántica

El concepto de localización requiere el concepto de un espacio dentro del que pueden existir las cosas. UML no modela la complejidad total del universo tridimensional. A cambio, soporta un modelo topológico de espacios conectados por rutas de comunicación. Un nodo es un recurso de computación en el que puede vivir una entidad de tiempo de ejecución. Los nodos están conectados mediante rutas de comunicaciones modeladas como asociaciones. Los nodos pueden contener despliegues de artefactos, lo que significa que una copia del artefacto se almacena en el nodo o se ejecuta en él. El despliegue se puede especificar a nivel de tipo o de instancia. Un modelo a nivel de tipo especificaría que ciertos tipos de nodos tienen ciertos tipos de artefactos. Un modelo a nivel de instancia especificaría que ciertas instancias del nodo tendrían ciertas instancias de artefactos. Los despliegues pertenecen a los nodos, y el mismo artefacto puede ser desplegado en varios nodos.

Los despliegues y las localizaciones de los despliegues son para artefactos físicos. Los elementos del modelado lógico pueden ser implementados mediante artefactos físicos. Esta relación se conoce como manifestación (*véase* la entrada). Podríamos tener la siguiente cadena de relaciones: Un elemento del modelado lógico, como un componente, es manifestado por un artefacto, como un archivo de programa, que es desplegado en un nodo, como una computadora. No hay relación directa entre el componente y la computadora, pero hay una relación indirecta.

Notación

La localización de un artefacto (como un archivo) desplegado en un nodo se puede representar anidando el símbolo del artefacto dentro del símbolo del nodo, como se muestra en la Figura 14.172. El despliegue también se puede representar mediante una flecha de dependencia (línea discontinua) desde el artefacto hasta el nodo, con la palabra clave «**deploy**». La relación de despliegue puede mostrarse a nivel de tipo o a nivel de instancia utilizando respectivamente símbolos de clasificador (sin subrayado) o símbolos de especificación de instancia (cadena de nombre subrayado con nombre y tipo).

La Figura 14.172 muestra las localizaciones de despliegue a nivel de tipo. Los archivos PostScript pueden vivir en impresoras. La Figura 14.173 muestra una versión a nivel de instancia del diagrama previo. En este ejemplo, la impresora pr1 tiene dos instancias de archivos PostScript, un archivo de cabecera y un archivo de trabajo.

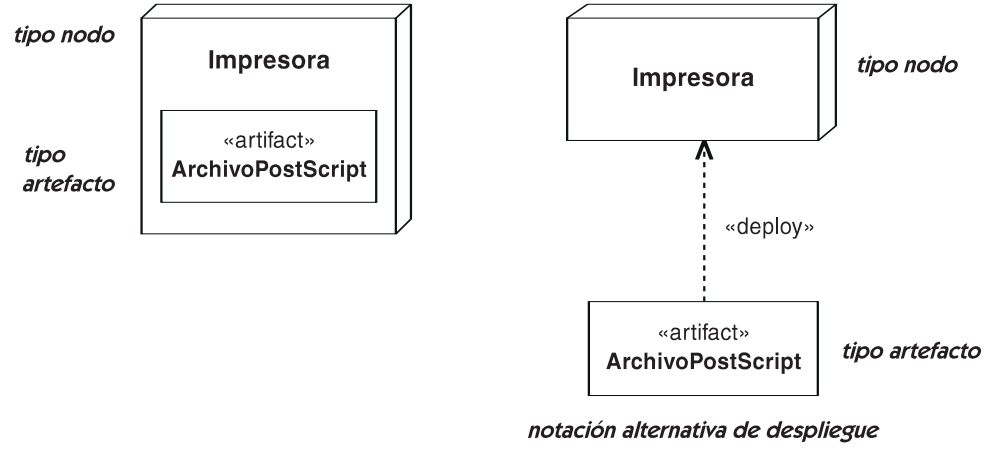


Figura 14.172 Localizaciones de despliegue a nivel de tipo

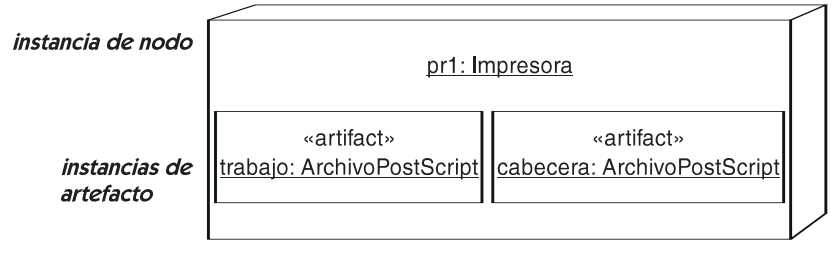


Figura 14.173 Localizaciones de despliegue a nivel de instancia

Historia

En UML2, la localización se ha restringido a los artefactos, en vez de a todos los elementos del modelo.

mal formado

Designación dada a un modelo que está construido incorrectamente, que viola una o más reglas o restricciones especificadas por el modelo o predefinidas. Antónimo: bien formado.

Véase también conflicto, restricción.

Semántica

Un modelo que viola las reglas y restricciones de buena formación no es un modelo válido y por tanto tiene semántica inconsistente. Intentar la utilización de dicho modelo puede producir resultados sin sentido. Es responsabilidad de las herramientas de modelado detectar los modelos mal formados y prevenir su utilización en situaciones que pudieran ser problemáticas. Debido a la utilización de ciertas construcciones que extienden a la semántica propia de UML, la verificación automática puede no ser posible en todos los casos. Además, no se puede esperar que la comprobación automática verifique la consistencia de las operaciones, puesto que implicaría la resolución del problema de la parada. Por tanto a efectos prácticos es necesaria una combinación de verificación automática y humana.

Aunque un modelo terminado debe estar bien formado, las versiones intermedias de un modelo probablemente esté mal formado en determinados momentos durante el desarrollo, puesto que podrían ser fragmentos incompletos de un modelo final. Editar un modelo válido para producir otro modelo válido puede requerir pasar a través de modelos intermedios que estén mal formados. Esto no es diferente de la edición de programas informáticos —el programa final pasado a un compilador debe ser válido, pero las copias de trabajo en un editor de texto son a menudo inválidas. Por tanto los modelos mal formados deben ser editables y almacenables por las herramientas de soporte.

manejador de excepción

Nodo de actividad ejecutable responsable de asumir el control si se produce una excepción de un tipo dado durante la ejecución del nodo de actividad adjunto al manejador de excepción.

Semántica

Un manejador de excepción está adjunto a un nodo de actividad para proteger el nodo contra excepciones de un tipo dado que podrían ocurrir durante la ejecución del nodo protegido. Si se produce una excepción que sea del mismo tipo o descendiente del tipo dado, se abandona la ejecución del nodo protegido y comienza la ejecución del manejador de excepción. Se crea un token de excepción cuyos valores de atributos capturan los parámetros de la excepción. El manejador de excepción tiene una ubicación de entrada designada, que recibe como valor el token de excepción. El manejador de excepción comparte el ámbito con el nodo al que protege y tiene acceso a todos los valores y objetos a los que podía acceder el nodo protegido. Un manejador de excepción debe tener el mismo número y tipos de pines de salida que el nodo al que protege. Cuando la ejecución del manejador finaliza, los valores de salida producidos por el manejador de excepción sustituyen a los valores de salida originales del nodo protegido, y se continúa la ejecución de los nodos subsecuentes como si la excepción no se hubiera producido. El propósito de un manejador de excepción es producir situaciones en las que la ejecución puede continuar sin afectar si se produjo una excepción.

Notación

Véase excepción para detalles de notación.

manejador de interrupción

Actividad que gana el control cuando se produce una interrupción. (Este no es un término oficial de UML.) *Véase* interrupción.

manifestación

Implementación física de un elemento del modelo como un artefacto.

Semántica

En software, los modelos al final se implementan como un conjunto de artefactos, como archivos de distintos tipos. Los artefactos son las entidades que se despliegan en nodos físicos, como computadoras o unidades de almacenamiento. Es importante hacer un seguimiento del mapeo de los elementos del modelo y su implementación en artefactos. Una manifestación es la relación entre un elemento del modelo y el artefacto que lo implementa. Esta es una relación muchos a muchos. Un elemento del modelo puede ser implementado por varios artefactos, y un artefacto puede implementar varios elementos, aunque algunos enfoques de diseño intentan más que un mapeo uno a uno (pero sólo para elementos del modelo "importantes", como quiera que se defina "importante").

Notación

Una relación de manifestación se representa con una flecha de dependencia —una línea discontinua con una cabeza de flecha sencilla— desde un artefacto a un elemento del modelo. Se coloca la pala-

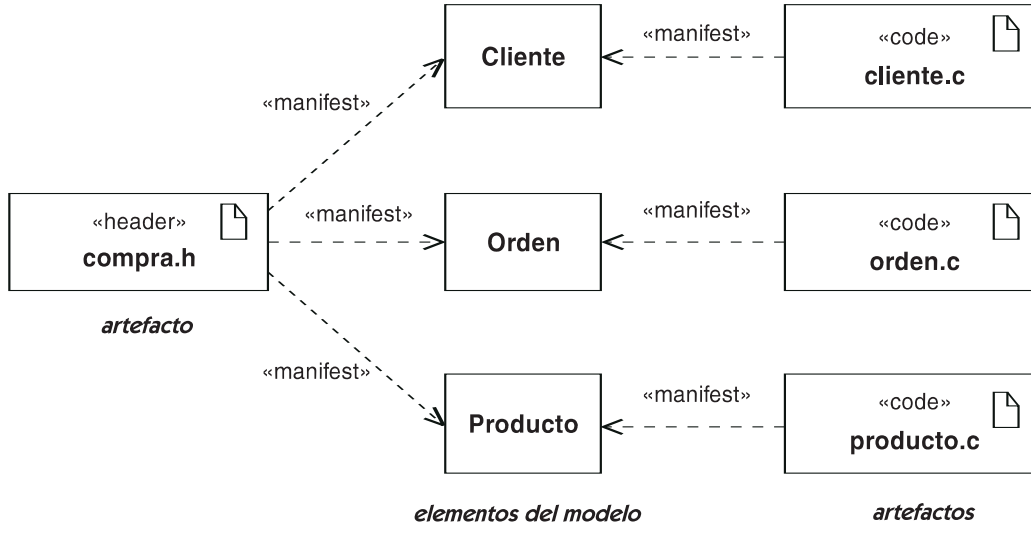


Figura 14.174 Manifestación de elementos por artefactos

bra clave «manifest» en la flecha. Un artefacto puede tener varias flechas de manifestación abandonándolo y un elemento del modelo puede tener varias flechas de manifestación entrando en él.

En la Figura 14.174 se manifiestan tres clases por un archivo de cabecera, **compra.h**, pero cada clase es manifestada por su propio archivo de código.

máquina de estados

Una especificación de las secuencias de los estados por los que un objeto o una interacción pasa en respuesta a eventos durante su vida, junto con sus efectos correspondientes (acción y actividad). Una máquina de estados se une a una clase, a una colaboración, o a un método del origen y especifica el comportamiento de las instancias del elemento origen.

Véase también acción, actividad, estado compuesto, acontecimiento, pseudoestado, estado, transición.

Semántica

Una máquina de estados es un gráfico de estados y de transiciones que describe la respuesta de una instancia de un clasificador a la recepción de eventos. Las máquinas de estados se pueden unir a clasificadores, tales como clases y casos de uso, así como a colaboraciones y métodos. El elemento al que va unida la máquina de estados se llama el dueño de la máquina del estado.

Una máquina de estados completa es un estado compuesto que se ha descompuesto recurrentemente en subestados. Los estados simples internos no tienen ningún subestado. Una máquina de estados puede incluir una referencia a otra máquina de estados usando un estado de la submáquina. Las máquinas de estados pueden ser redefinidas. *Véase* redefinición (máquina de estados).

Semántica de la ejecución de la máquina

La semántica de la ejecución de la máquina de estados se discute en las secciones siguientes. Hay muchas maneras de implementar esta semántica de ejecución, muchas de las cuales se implementan de formas muy distintas a algunos de los pasos explícitamente descritos aquí. La mayor parte de esta semántica está descrita en otros artículos, pero se recolectan aquí por conveniencia.

En todo momento, uno o más estados están activos en la configuración activa del estado de la máquina de estados de un objeto o de otra instancia. Si un estado está activo, entonces una transición que sale del estado puede dispararse, causando la ejecución de una acción y la activación de otro estado o estados en lugar del estado original. Más de un estado activo indica concurrencia interna. La estructura de la máquina de estados y sus transiciones imponen restricciones sobre los estados que pueden estar activos concurrentemente. Brevemente, si un estado compuesto está activo, exactamente un subestado directo debe ser activo en cada región del estado compuesto. Ya que, un estado compuesto puede tener regiones múltiples, pueden estar activos un número indeterminado de subestados indirectos de la máquina jerarquizada del estado.

Disparo de transiciones y acciones

La asunción básica es que una máquina de estados procesa un evento a la vez y termina todas las consecuencias de ese acontecimiento antes de procesar otro evento. Es decir los eventos no obran

recíprocamente con otros eventos durante el proceso del evento. Esto se conoce como proceso de ejecutar-hasta-terminar. No significa que todo el cómputo sea no interrumpible. Un cómputo extendido ordinario se puede descomponer en una serie de pasos de ejecutar-hasta-terminar, y el cómputo se puede interrumpir por un evento exterior entre cualquiera de sus pasos. Esto está muy cerca de la situación física real dentro de una computadora, donde las interrupciones pueden ocurrir en un tiempo discreto, pequeños pasos.

El corolario de esta asunción es que los eventos son asincrónicos. Dos eventos nunca ocurren en exactamente el mismo tiempo, más exacto, si dos acontecimientos ocurren en exactamente a la misma hora, es una coincidencia y pueden ser procesados como si hubieran ocurrido en cualquier orden, sin pérdida de generalidad. Los resultados de las diversas secuencias de ejecución pueden ser diferentes, esta es una de las condiciones características esenciales de la concurrencia, pero no se puede asumir la simultaneidad en un mundo distribuido. Cualquier cómputo en que se haga tal asunción se estropea lógica y físicamente. La ejecución concurrente requiere independencia en un mundo distribuido.

Acciones y actividades. Conceptualmente, las acciones son instantáneas y los eventos nunca son simultáneos. En una puesta en práctica, la ejecución de acciones requiere una cierta hora, pero lo importante es que las acciones son (conceptualmente) atómicas y no interrumpibles. Una actividad se puede descomponer en subactividades y en última instancia, acciones. Las actividades unidas a las transiciones se ejecutan usando la semántica de ejecutar-hasta-finalizar. Si un objeto reconoce un evento mientras está ejecutando un paso de ejecutar-hasta-terminar, la ocurrencia del evento se pone en una cola de eventos hasta que la ejecución del paso de ejecutar-hasta-finalizar se completa. Las ocurrencias del evento se quitan de la cola de eventos únicamente cuando no se está ejecutando ninguna acción. Si la acción de un objeto envía una señal a otro objeto, la recepción de la señal no es síncrona. Se coloca en la cola de eventos y después se gestiona como cualquier otro evento, después de la terminación de cualquier actividad actual de la forma ejecutar-hasta-finalizar unida a la transición actual. Una llamada a una operación suspende al llamador hasta que se ha ejecutado la operación. Puede ser puesta en ejecución, en la opción del receptor, como método o como evento de la llamada que accione la máquina de estados del receptor. Para evitar problemas con los períodos largos durante los cuales los eventos no pueden ser procesados, las consecuencias unidas a las transiciones deben ser breves. Las actividades de la transición no están pensadas para modelar regiones protegidas o los cómputos interrumpibles largos, que se pueden modelar como actividades dentro de subestados. Esto permite el procesado de eventos y permite que los cómputos anidados sean interrumpidos. Si se incluyen actividades largas en sistemas reales, los eventos no podrán ser procesados de manera oportuna. Ésta es una consecuencia de un mal modelo. Las actividades deben ser cortas, comparadas al tiempo de respuesta requerido por los eventos que puedan ocurrir.

Cuando se dispara una transición, se ejecuta cualquier actividad unida a ella. Una actividad puede utilizar los argumentos del evento disparado, así como atributos del objeto o de los valores accesibles desde él. Una actividad se termina antes de que se procese cualquier evento adicional. Si una transición tiene segmentos múltiples, los parámetros del evento disparador están disponibles como el evento implícito actual.

Durante la ejecución de un paso de ejecutar-hasta-terminar, todas las actividades tienen acceso a un suceso implícito actual, que es el acontecimiento que accionó la primera transición en la secuencia de ejecutar-hasta-terminar. Debido a que puede haber más de un evento que podría dar lugar a la ejecución de una actividad, la actividad puede necesitar discriminar en el tipo del evento actual para extraer sus valores.

Cola de eventos. Las nuevas instancias del evento se ponen en una cola de eventos para un objeto. Si un objeto está ocioso y no hay ocurrencias del evento en la cola, el objeto espera hasta que recibe un evento y después lo gestiona. Conceptualmente, un objeto maneja una sola instancia del evento a la vez. En una puesta en práctica real, los acontecimientos pueden hacer cola en un orden definido. La semántica de UML, sin embargo, no especifica un orden para procesar los eventos concurrentes, y un modelador no debe asumir ninguno. Si los acontecimientos deben ser procesados en cierto orden, la máquina de estados se debe construir para hacer cumplir el orden. Una puesta en práctica real probablemente seleccionaría una cierta regla de ordenación simple.

Disparadores. Por cada estado activo de un objeto, las transiciones salientes del estado son candidatas a ser disparadas. Se acciona una transición candidata si se maneja un evento cuyo tipo es igual al del evento disparador de la transición. Una señal que es un descendiente de una señal en un disparador de la transición también accionará la transición. Una transición no se dispara por una señal antepasada. Cuando un acontecimiento se maneja y acciona una transición, se evalúa la condición de guarda de la transición. Si el valor de la condición de guarda es verdadera, entonces la transición se permite. La expresión lógica de la condición de guarda puede implicar a los parámetros del evento del disparador, así como los atributos del objeto. Observe que las expresiones de guarda no deben producir efectos secundarios. Es decir, no pueden no alterar el estado del objeto o del resto del sistema. Por lo tanto, el orden en el que se evalúan es indiferente al resultado. Una condición de guarda se evalúa solamente cuando se maneja un evento. Si la condición se evalúa como falsa, no se reevalúa si algunas de sus variables cambian de valor más adelante. Un evento puede ser candidato para accionar varias transiciones con diversas condiciones de guarda, pero si todas las condiciones de guarda son falsas, se desecha el evento y no se dispara ninguna transición. A continuación se selecciona otro evento de la cola.

Pseudoestados. Para estructurar condiciones complejas, una transición se puede modelar con segmentos múltiples. El primer segmento tiene un evento de disparo y es seguido por un árbol de ramificación de segmentos con condiciones de guarda. Los nodos intermedios en el árbol son pseudoestados, los estados simulados que están presentes para estructurar las transiciones pero que no pueden seguir estando activos en el final de un paso ejecutar-hasta-terminar. Cada trayectoria posible a través del árbol de segmentos se mira como transición separada y es independiente para la ejecución. Un segmento individual no se puede disparar de manera aislada. Todas las condiciones de guarda a lo largo de una serie de segmentos deben ser verdades o la transición (cualesquiera incluyendo sus segmentos) no se dispara. En la práctica, la guarda condiciona en un punto de la rama los resultados posibles. Por lo tanto, una puesta en práctica podía procesar un solo paso de la transición multisegmento a la vez, pero no siempre.

El disparador se pone generalmente en el primer segmento de una transición, pero puede también seguir a la unión o a la conjunción de un pseudoestado. Una trayectoria puede tener a lo más un disparador.

Hay una diferencia leve si uno de los pseudoestados es un pseudoestado de elección. En ese caso, la cadena de los segmentos de la transición se evalúa hasta la elección, sin incluirla. Si las condiciones de la guarda son verdaderas, los disparadores de la transición y las actividades unidas a los segmentos hasta el pseudoestado de elección se ejecutan y pueden cambiar los valores de los objetos en el sistema. Las condiciones del protector en los segmentos de la transición que salen del pseudoestado de elección entonces se evalúan y uno de ellos se selecciona para ser disparado, basándose en los valores que pueden haber sido modificados por actividades anteriores en la transición. Una de las transiciones debe dispararse; si todas las condiciones de guarda son

falsas, el modelo está mal formado. Una opción permite que los resultados de algunas acciones en la transición afecten decisiones consecuentes sobre la misma transición, pero el modelador debe cubrir todos los resultados posibles.

Indeterminismo. Si no se permite ninguna transición, un evento simplemente se ignora. Esto no es un error. Si sólo está activada una transición, se dispara. Si más de una transición de un solo estado está activada, entonces solamente una de ellas se disparará. Si no se especifica ninguna restricción, entonces la opción es no determinista. No se debe hacer ninguna asuncion sobre si la opción será justa, fiable, o al azar. Una puesta en práctica real puede proporcionar las reglas para la resolución de los conflictos, pero se aconseja a los modeladores hacer uso explícito en lugar de confiar en tales reglas. Tanto si una transición se dispara como si no, el evento se consume.

Transiciones en estados compuestos. Las transiciones que dejan un estado activo son elegibles para ser disparadas. Además, las transiciones en cualquier estado compuesto que contiene un estado activo son candidatas a ser disparadas. Esto se puede mirar como una característica similar a la herencia de transiciones por los subestados (aunque no se modela como generalización). Una transición en un estado externo es elegible para dispararse solamente si no se dispara ninguna transición en los estados internos. Si no, es enmascarada por la transición interna.

Estados concurrentemente activos. Cuando un objeto maneja un evento, su configuración activa puede contener uno o más estados. Cada estado recibe una copia separada del evento y actúa en él independientemente. Las transiciones en estados concurrentemente activos se disparan independientemente. Un subestado puede cambiar sin afectar a los otros, al menos en el caso de una transición compleja, tal como una bifurcación o unión (descrito más adelante). Si un objeto tiene estados concurrentes, no deben obrar recíprocamente con memoria compartida. Los estados concurrentes se definen para ser independientes y deben actuar en espacios de valores distintos. Cualquier interacción debe ser explícita enviando señales. Si dos estados concurrentes deben tener acceso a un recurso compartido, deben enviar explícitamente señales al recurso, que puede entonces actuar como árbitro. Una puesta en práctica puede funcionar de forma muy distinta a tal comunicación explícita, pero se deben tener cuidado en tal caso para asegurarse de que no sobrevienen conflictos inconsistentes o peligrosos. Si las acciones concurrentes tienen acceso a valores compartidos, el resultado es no determinista.

Si la configuración activa del estado contiene estados múltiples, el mismo proceso puede ocurrir independientemente para cada estado. En muchos casos, sin embargo, solamente un estado del sistema puede ser afectado por una transición dada.

Actividades de la entrada y de la salida. Si una transición cruza el límite de un estado compuesto, la actividad de la entrada o la actividad de la salida en el estado compuesto puede ser ejecutada. El límite se cruza cuando el estado origen y el estado destino de la transición están en diversos estados compuestos. Observe que una transición interna no causa un cambio del estado, así que nunca invoca actividades de la entrada o de la salida. Para determinar las actividades de salida y de entrada que se ejecutan, encuentre el estado activo actual del objeto (éste se pudo anidar dentro del estado compuesto que es origen de la transición) y el estado destino de la transición. A continuación encuentre el estado compuesto interno que incluye el estado actual y el estado destino. Llame a esto antepasado común. Las actividades de salida del estado actual y cualquier estado incluido hasta él, pero no incluyendo al antepasado común, se ejecutan, los más internos primero. Entonces la actividad en la transición se ejecuta. Después de eso, se ejecuta la actividad de entrada del estado destino y cualquier otro incluido hasta él, pero no incluyendo al

antepasado común, siempre se ejecutará la actividad de entrada exterior primero. Es decir los estados se abandonan uno a la vez hasta que alcanzan al antepasado común, y entonces los estados se incorporan hasta que se alcanza el estado destino. Las actividades de salida y de entrada en el antepasado común no son ejecutadas, porque el estado no ha cambiado. Este procedimiento asegura que cada estado está fuertemente encapsulado. La actividad en la transición se ejecuta después de que se haya ejecutado cualquier actividad de la salida y antes de que se realice cualesquiera actividades de entrada. Observe que el disparo de una autotransición (una transición de un estado a sí mismo) causará la salida de cualquier estado anidado dentro del estado origen que puede estar activo (la transición pudo haber heredado de un estado compuesto que incluye). También causa la ejecución de la actividad de salida del estado origen seguido por la ejecución de su actividad de entrada. Es decir se sale del estado y después se entra de nuevo. Si no se desea este efecto, se puede usar una transición interna. Esto no causará un cambio del estado, incluso si el estado activo se anida dentro del estado con la transición.

Cadenas de segmentos. Una transición se puede estructurar con varios segmentos unidos por nodos intermedios. Cada segmento puede tener su propia actividad. Las actividades se pueden interpolar con la actividad de la entrada y de la salida para la transición total. Con respecto a actividades de la entrada y de la salida, cada acción en un segmento de la transición ocurre donde ocurriría si el segmento fuera una transición completa. *Véase* la Figura 14.82 para un ejemplo.

Estado siguiente. Después de que se realicen todas las actividades, el estado actual original pasa a inactivo (a menos que sea el estado destino), el estado destino de la transición pasa a ser activo, y los eventos adicionales pueden entonces ser procesados.

Las transiciones internas

Una transición interna tiene un estado de origen pero ningún estado destino. Su disparador no causa un cambio del estado, incluso aunque la transición que la dispara esté heredada de un estado que incluye. Debido a que el estado no cambia, no se realiza ninguna actividad de salida o de entrada. El único efecto de una transición interna es la ejecución de su actividad. Las condiciones para disparar una transición interna son exactamente iguales que para una transición externa.

Observe que el disparar una transición interna puede enmascarar una transición externa que use el mismo evento. Por lo tanto, puede tener sentido definir una transición interna sin actividad. Como se declaró anteriormente, el disparar solamente una transición por evento dentro de una región secuencial, y una transición interna tiene prioridad sobre una transición externa. Las transiciones internas son útiles para los acontecimientos de proceso sin cambios en el estado.

Estados iniciales y finales

La encapsulación de estados es deseable para separar el interior de un estado del exterior. Es también deseable conectar transiciones a un estado compuesto, sin tener conocimiento sobre la estructura interna del estado. Esto se puede lograr usando estados iniciales y estados finales dentro de un estado compuesto.

Un estado puede tener un estado inicial y final. Un estado inicial es un pseudoestado —un estado simulado que se conecta normalmente a un estado normal— y un objeto puede no permanecer en un estado inicial. Un objeto puede permanecer en un estado final, pero un estado final puede no tener ninguna transicion accionada explícitamente; su propósito es permitir a una transición de la terminación en un estado interno. Un estado inicial no puede seguir siempre activo; debe tener una

transición saliente. Si hay más de una transición saliente, todas deben carecer de disparadores y sus condiciones de guarda deben representar todas las posibilidades. Es decir una transición saliente debe disparar cuando se invoca el estado inicial. Un objeto nunca puede permanecer en el estado inicial, por lo tanto, transitará inmediatamente a un estado normal.

Si un estado compuesto tiene un estado inicial, después las transiciones se pueden conectar directamente con el estado compuesto como destino. Cualquier transición al estado compuesto implica una transición al estado inicial dentro del estado compuesto. Si un estado compuesto carece un estado inicial, después el estado compuesto no puede ser destino de las transiciones; deben ser conectadas directamente con los subestados. Un estado con un estado inicial puede también tener transiciones conectadas directamente con los estados internos, así como al estado compuesto. Si un estado compuesto tiene un estado final, después puede ser el origen de una o más transiciones salientes de terminación, es decir, las transiciones que carecen acontecimiento explícito que las accione. Una transición de terminación es realmente una transición sobre la que está permitida la transición por la terminación de la actividad dentro del estado. Una transición a un estado final es por lo tanto una declaración de que la ejecución del estado compuesto está completa. Cuando un objeto pasa a un estado final, las transiciones de terminación que salen de su estado compuesto pueden dispararse si sus condiciones de guarda están satisfechas.

Un estado compuesto puede tener transiciones salientes etiquetadas —esto es, transiciones con disparadores explícitos de eventos. Si ocurre un evento que cause el disparo de tal transición, entonces cualquier actividad en curso dentro del estado (en cualquier profundidad del anidamiento) se termina, las acciones de salida de los estados jerarquizados terminados se ejecutan, y se procesa la transición. Tales transiciones son de uso frecuente para modelar excepciones y condiciones de error.

Transiciones complejas

Una transición en un estado ortogonal implica una transición en todas sus regiones ortogonales. Esto puede suceder de dos maneras.

Una transición puede tener múltiples estados destino, uno dentro de cada región de un estado ortogonal. Observe que una transición que se bifurca tiene un solo evento como disparador, condición de guarda y actividad. Esto es una transición explícita en un estado ortogonal que especifica cada destino directamente. Esto representa una bifurcación explícita del control en subestados concurrentes.

Alternativamente, una transición puede omitir destinos dentro de una o más regiones de un estado ortogonal, o puede tener el propio estado compuesto como destino. En este caso, cada región ortogonal omitida debe tener un estado inicial dentro de ella para indicar su estado inicial por defecto. Si no, la máquina de estados está malformada. Si se dispara la transición compleja, los subestados explícitos destino pasan a ser activos, al igual que los estados iniciales de las regiones ortogonales sin destinos explícitos. En definitva, cualquier transición en cualquier región ortogonal implica una transición a los estados iniciales de cualquier otra región del par que no contenga estados explícitos destino. Una transición a un estado compuesto implica en sí misma una transición a los estados iniciales de cada una de sus regiones ortogonales. Si un estado compuesto está activo, un subestado directo de cada una de sus subregiones estará también activo.

De forma similar, una transición de un estado en cualquier región ortogonal implica una transición de todas las regiones similares. Si la ocurrencia de un evento causa el disparo de una transi-

ción, la actividad en las otras regiones ortogonales se termina, las actividades de salida de las regiones se ejecutan, la propia actividad de la transición misma se ejecuta, y el estado destino pasa a ser activo, y de esta forma reduce el número de estados concurrentemente activos. La transición al estado final de una región ortogonal no fuerza la terminación de la actividad en otras regiones ortogonales similares (esto no es una transición fuera de la región). Cuando todas las regiones ortogonales han alcanzado sus estados finales, el estado compuesto que incluye se considera terminado y cualquier transición de la terminación que sale del estado compuesto se puede disparar. Una transición compleja puede tener estados múltiples de origen y estados múltiples de destino. En ese caso, su comportamiento es la combinación de la bifurcación y la unión descrita anteriormente.

Estado de historia

Un estado compuesto puede contener un estado de historia, que es un pseudoestado. Si una transición heredada causa una salida automática del estado compuesto, el estado "recuerda" el subestado que estaba activo cuando ocurrió la salida forzada. Una transición al pseudoestado de la historia dentro del estado compuesto indica que el subestado recordado se debe restablecer. Una transición explícita a otro estado o al estado que se incluye a sí mismo no permite el mecanismo de la historia, y en él se aplican las reglas generales de la transición. Sin embargo, el estado inicial del estado compuesto se puede conectar con el estado de historia. En ese caso, una transición al estado compuesto (indirectamente) invoca el mecanismo de historia. El estado de historia puede tener una sola transición saliente de terminación sin condición de guarda; el destino de esta transición es el estado de la historia por defecto. Si la región del estado nunca no ha incorporado o si salió normalmente, después una transición al estado de la historia va al estado de la historia por defecto.

Hay dos clases de estado de historia: un estado superficial de historia y un estado profundo de historia. El estado superficial de la historia restaura los estados contenidos directamente (profundidad una) en el mismo estado compuesto que el estado de historia. El estado profundo de historia restaura el estado o los estados que estaban activos antes de la transición explícita pasada que hizo el estado compuesto. Puede incluir los estados anidados dentro del estado compuesto a cualquier profundidad. Un estado compuesto puede tener a lo más uno de cada clase de estado de la historia. Cada uno puede tener su propio estado de historia por defecto.

El mecanismo de historia debe ser evitado siempre que la situación se pueda modelar más directamente, pues es complicado y no necesariamente tiene una buena correspondencia con mecanismos de implementación. El mecanismo profundo de historia es particularmente problemático y se debe evitar a favor de mecanismos más explícitos (y más realizables).

Notación

Un diagrama de máquina de estados muestra una máquina de estados o una porción jerarquizada de una máquina de estados. Los estados son representados por los símbolos del estado (rectángulos con las esquinas redondeadas), y las transiciones son representadas por las flechas que conectan los símbolos del estado. Los estados pueden contener también subdiagramas para la contención y el ensamblado físicos. Un diagrama completo de la máquina de estados se pone en un marco rectangular con su nombre puesto en una etiqueta pequeña pentagonal en la esquina izquierda superior. Los puntos de la entrada y de la salida se pueden poner en el límite. La Figura 14.175 muestra un ejemplo.

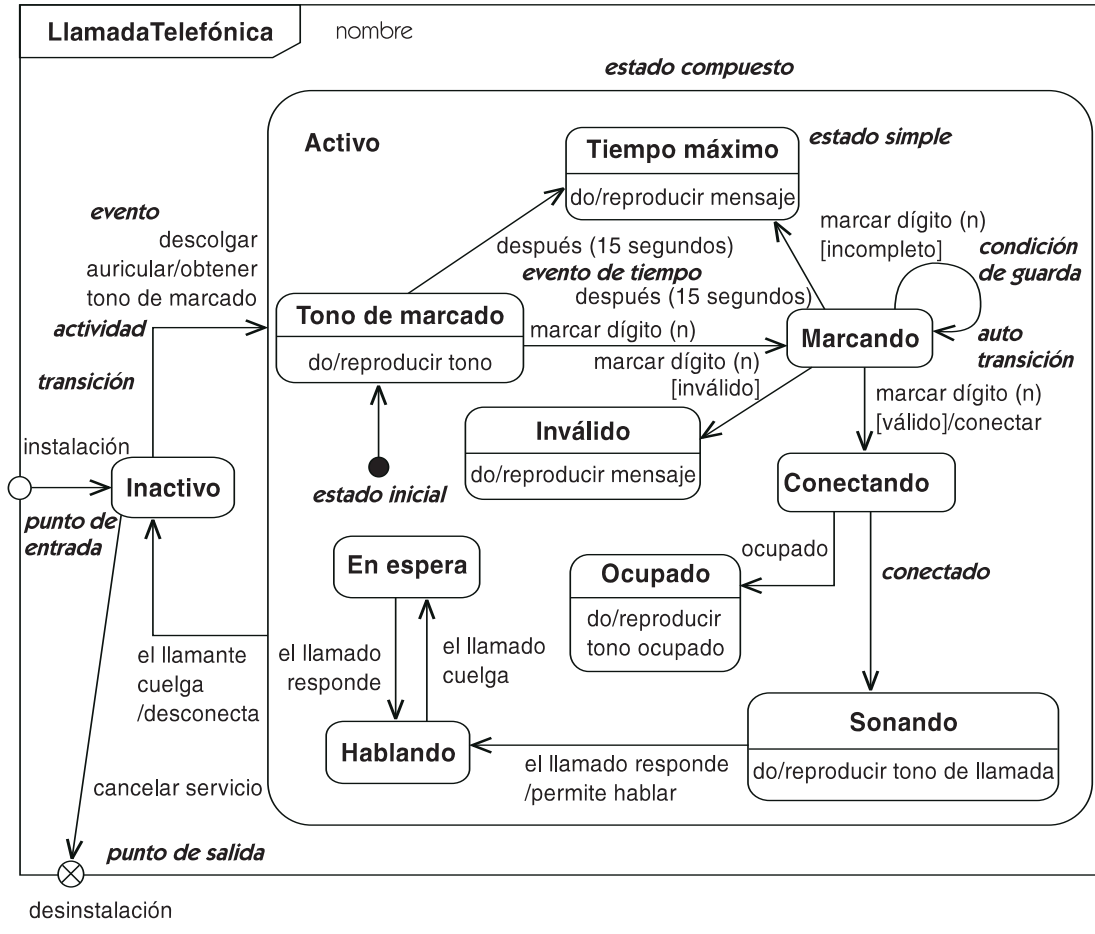


Figura 14.175 Diagrama de estado

La notación del diagrama de estados fue inventada por David Harel e incorpora aspectos de las máquinas de Moore (acciones en entrada) y de las máquinas de Mealy (acciones en transiciones), así como la adición de los conceptos de estados anidados y de estados concurrentes.

Para más detalles, *véase* estado, redefinición (máquina) del estado, estado compuesto, submáquina, pseudoestado, actividad de la entrada, actividad de la salida, transición, transición interna, hacer la actividad, y redefinición (máquina del estado). *Véase también* nodo del control para algunos símbolos opcionales previstos para el uso dentro de diagramas de la actividad, pero que también pueden ser utilizados en diagramas de estados.

Discusión

Las máquinas de estados se pueden utilizar de dos maneras. Por lo tanto, su significado se puede entender de cualquiera de las dos. En un caso, la máquina del estado puede especificar comportamiento ejecutable de su elemento-típico principal, una clase. En ese caso, la máquina de estados describe la respuesta del elemento principal mientras recibe acontecimientos del resto del universo. La respuesta es descrita por transiciones, que indican qué sucede cuando el elemento principal recibe un evento mientras está en un estado dado. El efecto se expresa como una

acción y cambio del estado. Las acciones pueden incluir enviar señales a otros objetos, que accionan sus máquinas de estados. Las máquinas de estados proporcionan una especificación simplificada del comportamiento de un sistema.

En el segundo caso, la máquina del estado se puede utilizar como especificación del protocolo, demostrando el orden legal en el cual las operaciones pueden invocarse en una clase o un interfaz. En tal máquina del estado, las transiciones son accionadas por acontecimientos de llamada y sus acciones invocan la operación deseada. Esto significa que se permite a un llamador invocar la operación en ese punto. La máquina de estados del protocolo no incluye acciones para especificar el comportamiento de la operación en sí mismo. Demuestra qué operaciones pueden ser invocadas en un orden particular. Tal máquina de estados especifica secuencias válidas de operación. Éste es un uso de una máquina del estado como generador de secuencias en un lenguaje (de teoría de lenguajes de la información). Tal máquina se significa como restricción del diseño del sistema. No es directamente ejecutable y no indica qué sucede si ocurre una secuencia ilegal porque se supone que no debe ocurrir. Es responsabilidad del diseñador de sistemas asegurarse de que solamente ocurren las secuencias legales. Este segundo uso es más abstracto que el primero, que especifica, en una forma ejecutable, qué sucede en todos los casos. Pero a menudo resulta muy conveniente, especialmente en la codificación de alto nivel. Las máquinas de estados han sido parte de la teoría de la informática por muchos años, pero el modelo de la máquina de estados en UML se basa en gran parte en las extensiones hechas por David Harel, que agregó el concepto de estados anidados y de regiones ortogonales. Esta extensión permite que las máquinas de estados crezcan hasta problemas más grandes sin las cantidades excesivas de duplicidad que serían necesarios de otra manera. Harel ha desarrollado una teoría exacta, formal de sus máquinas de estados. El modelo de UML es menos formal que Harel y contiene algunas modificaciones previstas para realzar la utilidad de las máquinas de estados en los modelos orientados a objetos.

máquina de estados de comportamiento

Una máquina de estados utilizada para expresar el comportamiento de los objetos de una clase (incluyendo un sistema completo), a diferencia de la especificación de una secuencia legal de invocaciones a operaciones.

Véase máquina de estados. Compárese con máquina de estados de protocolo.

Semántica

Una máquina de estados de comportamiento es un comportamiento ejecutable pensado para especificar la ejecución de los objetos de una clase como respuesta a la realización de un evento. Una máquina de estados de comportamiento debe estar preparada para aceptar cualquier secuencia de eventos legales. Por el contrario, una máquina de estados de protocolo es la especificación de las secuencias legales de invocación de operaciones. No es directamente ejecutable y no está pensada para manejar secuencias arbitrarias de eventos; más bien está pensada para restringir al sistema en su conjunto para evitar secuencias ilegales.

Discusión

El mecanismo utilizado en UML1 está basado (pero no es idéntico) en los formalismos de los diagramas de estados inventados por el informático David Harel. Este tipo de especificación se puede transformar en código de varias formas para su implementación.

máquina de estados de protocolo

Una máquina de estado usada para especificar las secuencias legales de llamadas de operación y señales recibidas por un objeto.

Semántica

Una máquina de estado conductual es una especificación ejecutable que convierte los eventos reconocidos por un objeto en una sucesión de efectos. Cualquier sucesión de eventos produce un resultado. Por contraste, una máquina de estado protocolar especifica las sucesiones legales de eventos, que pueden ocurrir dentro del contexto de un clasificador. No es responsable de asegurar que una sucesión legal ocurre; la responsabilidad puede convivir con el llamador o puede distribuirse sobre el diseño de un sistema. Una máquina de estado protocolar meramente define las secuencias legales, a menudo para facilitar el plan del sistema total que asegura que esas sucesiones válidas ocurran. En términos teóricos del lenguaje, una máquina de estado protocolar es un aceptador o una gramática para sentencias válidas.

Una máquina de estado protocolar difiere de una máquina de estado conductual como sigue:

- Las transiciones no tienen efectos. El propósito de una transición es especificar los mensajes legales. Deben especificarse efectos en otra parte del posible plan, como operaciones sobre clases o como transiciones de una máquina de estado conductual que podría ser construida después en el proceso de diseño.
- La transición puede tener una precondición que debe ser verdad si el evento ocurre. Esto se interpreta como una precondición en la operación asociada, si cualquiera.
- Una transición puede tener una postcondición. La postcondición indica el estado del objeto poseído después de que la transición está completa. Aunque las transiciones protocolares no tenga los efectos, las postcondiciones pueden imponer restricciones en los resultados de la operación llamada o en efectos que pueden necesitarse como parte de la aplicación eventual.

Si una sucesión de primacías de eventos (operación llama o signos) pesados lleva a un camino válido a través de la máquina de estado protocolar, esa sucesión de eventos es legal y debe aceptarse por el diseño del sistema. Las postcondiciones pueden imponer restricciones en la implementación.

Si una sucesión de eventos no lleva a un camino válido, no puede ocurrir. Hay dos posibles consecuencias para el diseñador: Si la creación de los eventos es bajo el control del resto del sistema, el diseñador debe asegurar que las secuencias inválidas no se generen, pero entonces la clase designada no necesitaría el asidero tal como secuencias inválidas. Por otra parte, la máquina de estado protocolar puede ser una aserción que la secuencia inválida no pueda ocurrir (para cualesquiera razones) y que la necesidad del sistema no los maneje. (Un diseñador sabio verificará sin embargo para las sucesiones no válidas y generará fracasos de error elegantes si es necesario.)

Notación

Una máquina de estado protocolar es mostrada por la palabra clave {**protocol**} después del nombre de la máquina de estado.

Una transición protocolar tiene la sintaxis siguiente para su etiqueta:

⌊[**precondición**]⌋$_{opc}$ **nombre-mensaje** ⌊(**lista-parámetros**)⌋$_{opc}$
⌊[**postcondición**]⌋$_{opc}$

El nombre del mensaje debe ser el nombre de una operación o signo.

Ejemplo

La Figura 14.176 muestra las reglas para ofrecer un puente de contrato como una máquina de estado protocolar.

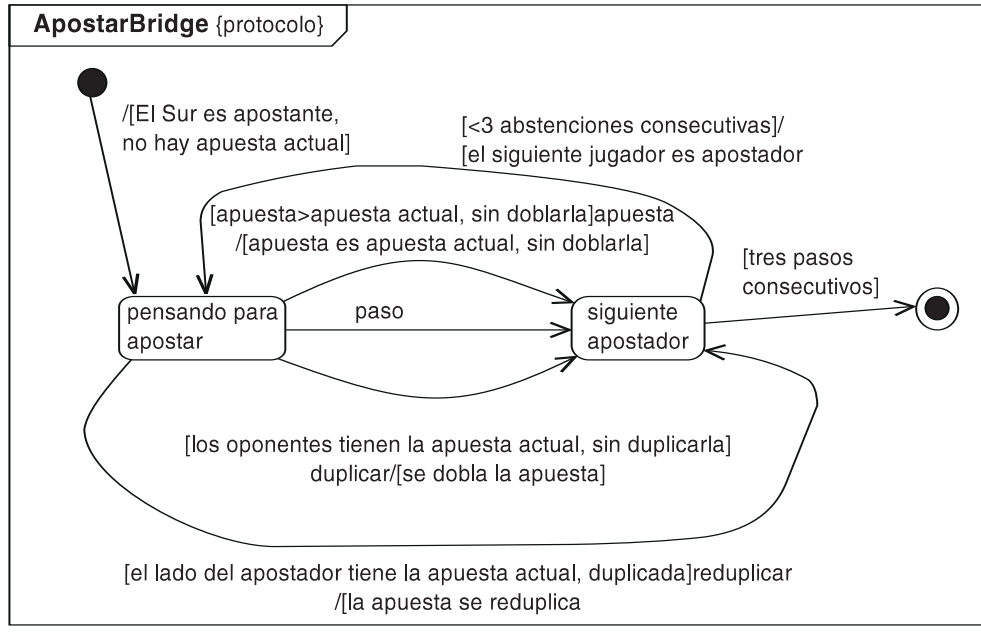


Figura 14.176 Máquina de estados protocolar para las apuestas en el bridge

Historia

En UML2, el concepto previamente informal de máquina de estado protocolar ha sido formalizado.

marca de tiempo

Indicación del tiempo en el que se produce un evento. Las marcas de tiempo se pueden utilizar en restricciones.

Semántica

Una marca de tiempo es el tiempo en que se produce un evento.

Notación

Una marca de tiempo se representa como una pequeña marca (una línea horizontal corta) incluso con un evento en un diagrama de secuencia. El nombre de la marca de tiempo se coloca cerca de la marca y se puede utilizar en expresiones de tiempo. *Véase* la Figura 14.264 para ver un ejemplo.

Discusión

Observe que una acción de vigilancia de tiempo no es una sustituta satisfactoria para una marca de tiempo. Aunque las marcas de tiempo no son parte de la especificación oficial de UML2, recomendamos su utilización puesto que de otro modo la capacidad es escasa.

marcador gráfico

Elemento notacional, como una geometría, textura, patrón de relleno, tipografía, color y demás.

Véase también uso de la tipografía.

Notación

Los símbolos de notación se construyen con diversos marcadores gráficos. Ningún marcador gráfico tiene significado semántico por sí mismo, pero el objetivo de la notación es utilizar marcadores gráficos de forma consistente y ortogonal tanto como sea posible.

Algunos marcadores gráficos se utilizan para construir símbolos UML predefinidos, mientras que otros marcadores gráficos no se usan en la notación canónica. Por ejemplo, no se ha asignado ningún significado al color puesto que muchas impresoras no son capaces de imprimirlo y ciertas personas no pueden distinguir todos los colores. Los marcadores gráficos no asignados, como los colores, pueden ser utilizados dentro de herramientas de edición para cualquier propósito que desee el modelador o la herramienta. A menudo esa utilización se puede modificar y no tiene un significado fijo. Los modeladores deberían usar dichas capacidades con cuidado si el modelo se puede usar en otro contexto donde el marcador gráfico pudiera no estar disponible (por ejemplo en una copia monocroma).

UML permite extensión gráfica limitada para su notación. Se puede asociar un icono gráfico o un marcador gráfico (como una textura o color) con un estereotipo. EML no especifica la forma de la especificación gráfica. Pero existen muchos mapas de bits y formatos que podrían ser usados por un editor gráfico (aunque su portabilidad es un problema difícil).

Se puede concebir formas de especificación de iconos y sustitución más generales, aunque las dejamos para la inventiva de los constructores de herramientas —con la advertencia de que el uso excesivo de la capacidad de extensibilidad puede llevar a la pérdida de portabilidad entre herramientas.

materialización

El acto de materializar alguna cosa.

Véase materializar.

materializar

Tratar como un objeto alguna cosa que no es vista usualmente como un objeto

Discusión

Materializar tiene un amplio rango de significados filosóficos y literarios. Se usa para describir la caracterización de conceptos abstractos como cosas o personas, en mitología y en poesía. Por ejemplo, el dios Thor era una materialización del trueno (dios del trueno). La teoría de Platón de ideas convertidas en cosas en torno a la percepción prevalece. Considera los conceptos puros, tales como Belleza, Bondad, y Coraje, como, realidad eterna verdadera, y considera la presentación física como copia imperfecta, materialización lleva esto a su máximo límite.

Materialización es una de las ideas más utilizadas en la orientación de objetos, y suyace en casi todos los aspectos del modelado.

Al construir un modelo, en primer lugar se requiere la imposición de objetos dentro de un mundo continuo. Los humanos hacen esto de modo natural en muchas frases que pronuncian, un nombre es una materialización de una cosa y un verbo es una materialización de una acción. La materialización se usa especialmente cuando se aplica a cosas en modelos o programas que no llegan a ser tratados como objetos, tales como comportamientos dinámicos. Muchas personas piensan en una operación como un objeto, pero ¿qué pasa con la ejecución (la palabra misma es una materialización) de una operación? Generalmente, la gente piensa en eso como un proceso. Pero materialícelo y déle un nombre —llámelo activación— y puede repentinamente darle propiedades, relaciones formales a otros objetos, manipularlos, y guardarlos. La materialización de comportamiento transforma un proceso dinámico en estructuras de datos que pueden ser almacenadas y manipuladas. Este es un concepto de gran alcance para el modelado y la programación.

mensaje

Transporte de información desde un rol a otro como parte de una interacción dentro de un contexto; a nivel de instancia, comunicación desde un objeto a otro. Un mensaje puede ser una señal o la llamada a una operación. El envío y la recepción de un mensaje son especificaciones de sucesos.

Véase también llamada, colaboración, interacción, operación, enviar, señal.

Semántica

Un mensaje es la transmisión de una señal desde un objeto (el emisor) a uno o más objetos (los receptores), o es la llamada a una operación en un objeto (el receptor) por otro objeto (el emisor o llamador). La implementación de un mensaje puede tomar diversas formas, como una llamada a procedimiento, elevado explícito de eventos, comunicación intraprocesos entre hilos activos, y demás. A nivel lógico, el envío de una señal y la llamada a una operación son similares. Ambos implican una comunicación desde un emisor a un receptor que pasa información por valor que el receptor utiliza para determinar qué hacer. Una llamada se puede considerar como un patrón de señales que implican un envío con un argumento puntero implícito de retorno que más tarde es utilizado para enviar una señal de retorno al llamador. Una llamada se puede modelar como

dos mensajes, un mensaje de llamada y más tarde un mensaje de retorno. A nivel de implementación, las señales y las llamadas tienen diferentes propiedades y comportamiento detallado, de forma que se distinguen como elementos UML. Un mensaje pertenece a una interacción, y el emisor y el receptor son líneas de vida de la interacción.

La recepción de una señal puede desencadenar una transición de máquina de estados. El envío de una señal es asíncrono; el emisor continúa la ejecución después de enviarla.

La recepción de una llamada de operación puede invocar un procedimiento o puede desencadenar una transición de máquina de estados o ambas cosas. Una llamada puede ser asíncrona o síncrona. Si la llamada es síncrona, el llamador espera hasta que la ejecución invocada responde. Una operación síncrona puede devolver valores como parte de la respuesta. Cuando se completa la ejecución del procedimiento, el llamador recupera el control y recibe cualesquiera valores de retorno.

Observe que una llamada directa de una actividad es simplemente una construcción de flujo de control dentro del modelo y no corresponde con un mensaje, puesto que sólo está implicado un único objeto.

Un mensaje representa una sola transmisión entre un emisor y un receptor. Un mensaje tiene una especificación de suceso de envío y una especificación de suceso de recepción. La acción de difusión envía un conjunto de mensajes, uno por cada objeto en el conjunto destino implícito. Otras acciones envían un mensaje, pero pueden aparecer en bucles o regiones de expansión para enviar conjuntos de mensajes.

Un mensaje tiene argumentos, que deben concordar en número y tipo con los parámetros de la operación especificada o con los atributos de la señal especificada. Cualesquiera valores de retorno para una operación llamada de forma asíncrona se ignoran y no generan un mensaje de respuesta.

Se coloca una restricción temporal en las especificaciones de sucesos que identifican un mensaje, más que al propio mensaje. Se puede colocar una restricción de duración en el mensaje.

No hay una escala temporal absoluta que relacione eventos en diferentes líneas de vida. A no ser que se restrinja, los eventos en diferentes líneas de vida son concurrentes. Un mensaje establece una ordenación temporal entre el envío y la recepción de eventos. Los eventos en una sola línea de vida están ordenados temporalmente. Cualquier cadena de mensajes que vuelva a la misma línea de vida debe terminar en un punto posterior que el principio de la cadena (es decir, los viajes al pasado están prohibidos excepto en las novelas de ciencia ficción).

Estructura

interacción	Interacción propietaria del mensaje.
evento de envío	Especificación del suceso de enviar el mensaje. La línea de vida que posee esta especificación del suceso es la emisora del mensaje.
evento de recepción	Especificación del suceso de recepción del mensaje. La línea de vida que posee esta especificación del suceso es la receptora del mensaje.
signatura	Señal u operación que define el tipo del mensaje.
argumentos	Lista de argumentos compatible con la signatura. Los argumentos deben ser accesibles al emisor; es decir, deben ser propiedades del emisor o del propietario de la interacción, parámetros de la invoca-

	ción de la interacción, constantes o valores alcanzables por los elementos precedentes.
conector	(opcional) Conector sobre el que se transmite el mensaje.
sincronicidad	Bien una señal (siempre asíncrona), una llamada síncrona o una llamada asíncrona. Debe concordar con la signatura.
clase	Si el mensaje se completa (por defecto, incluye al emisor y al receptor), perdido (sin receptor), o encontrado (no emisor). Los mensajes perdidos y encontrados se utilizan en situaciones de modelado avanzado para sistemas bulliciosos o sistemas con conocimiento incompleto. En la mayoría de los modelos no son necesarios.

Un mensaje en el modelo no puede cruzar los límites de un fragmento combinado, como una rama condicional o un bucle. Para modelar una instancia de mensaje que atraviese un límite, utilice una puerta en el límite para combinar dos mensajes, cada uno en un fragmento. Durante la ejecución, una instancia de mensaje corresponderá a la cadena de mensajes del modelo.

Notación

La notación para los diagramas de secuencia y de comunicación es diferente.

Diagramas de secuencia

En un diagrama de secuencia, un mensaje se representa como una flecha sólida desde la línea de vida de un objeto, el emisor, a la línea de vida de otro objeto, el destino (Figura 14.177). Oficialmente, las escalas temporales en las distintas líneas de vida son independientes, de forma que el ángulo de la flecha no tiene ningún significado semántico. Sin embargo, algunos modeladores dan por hecha una escala temporal uniforme entre todas las líneas de vida, de forma que una flecha perpendicular a las líneas de vida implica transmisión de mensajes instantánea (o al menos rápida), y una flecha inclinada implica que la duración de la transmisión es perceptible. Dichas asunciones no se reflejan en el modelo semántico a no ser que se inserten restricciones temporales. En caso de un mensaje de un objeto a sí mismo, la flecha debe empezar y terminar en la misma línea de vida. Las flechas de mensaje se colocan en orden secuencial desde la parte superior a la inferior, de forma vertical. A no ser que el punto de envío de un mensaje siga al punto de recepción de otro mensaje en la misma línea de vida (o que haya una cadena de mensajes intermedios que conecten a los dos), los dos mensajes son concurrentes, y su orden relativo no es significativo. Los mensajes pueden tener números de secuencia, pero debido a que el orden relativo de los mensajes se muestra de forma visual, los números de secuencia se suelen omitir. (Además, los números de secuencia no son útiles si hay mensajes concurrentes.)

Una flecha de mensaje se puede etiquetar con la sintaxis:

⌊ **atributo** = ⌋$_{\text{opc}}$ **nombre** ⌊ (**argumentos**$_{\text{lista,}}$) ⌋$_{\text{opc}}$ ⌊ : **valor-de-retorno** ⌋$_{\text{opc}}$

donde nombre es el nombre de la señal u operación y atributo es el nombre opcional de un atributo (de la línea de vida) para almacenar el valor de retorno.

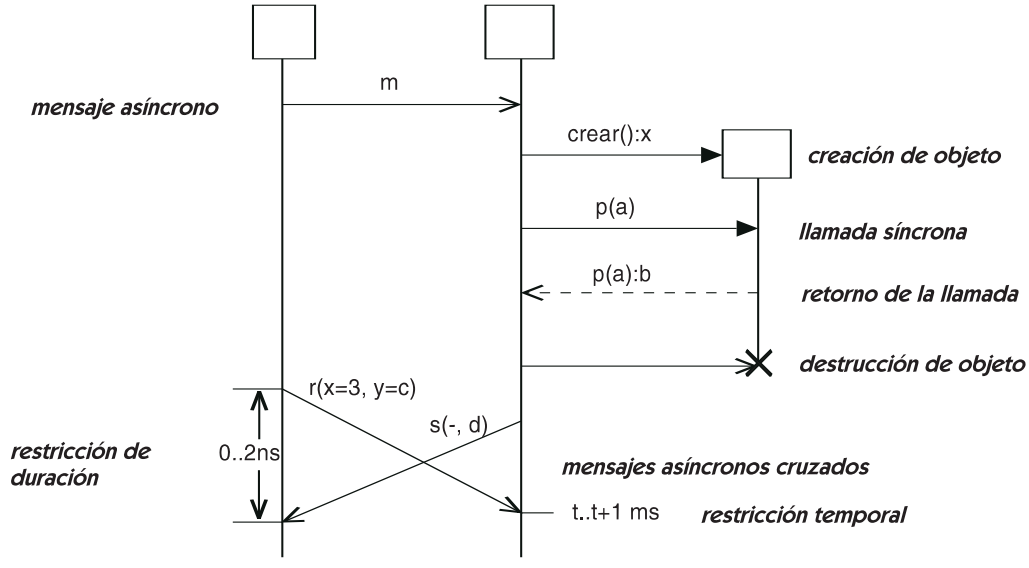


Figura 14.177 Notación de mensajes en diagramas de secuencia

Los valores de los argumentos se pueden reemplazar por un guión (-), que indica que cualquier valor de argumento es compatible con el modelo. La cadena de nombre entera se puede reemplazar por un asterisco (*), que indica que cualquier mensaje es compatible con el modelo.

Como opción, se puede incluir un nombre de parámetro con un valor de argumento, de forma que un argumento puede tener una de las siguientes sintaxis:

```
valor-de-argumento
nombre-de-parámetro = valor-de-argumento
-
```

Si se utilizan nombres de parámetro y no aparecen algunos parámetros, cualquier valor es aceptable para los parámetros desaparecidos.

Sincronicidad. Un mensaje asíncrono se representa mediante una cabeza de flecha abierta. Una llamada síncrona se representa mediante una cabeza de flecha rellena negra. El retorno de una llamada síncrona se representa mediante una línea discontinua con una cabeza de flecha abierta.

Si un diagrama contiene principalmente llamadas síncronas mostrando regiones de activación, los mensajes de retorno se pueden omitir y los valores de retorno se muestran en los mensajes de llamada originales. El retorno es implícito al final de la región de activación llamada. Esta convención reduce el número de flechas en el diagrama pero origina más esfuerzo mental para el lector, de forma que muchos modeladores prefieren mostrar siempre los retornos.

Un mensaje que crea (o causa la creación de) un objeto se representa colocando la caja de cabecera de una línea de vida en la cabeza de la flecha. Un mensaje que destruye (o que causa la destrucción de) un objeto se representa colocando una gran *X* en la cabeza de flecha. Los mensajes pueden ser síncronos o asíncronos.

Los mensajes perdidos o encontrados se representan colocando un pequeño círculo negro al final de la flecha en el que el objeto es desconocido.

Retardo de transmisión. Normalmente las flechas de mensaje se dibujan de forma horizontal, indicando que el tiempo requerido para enviar el mensaje es atómico —es decir, es breve en comparación con la granularidad de la interacción y que nada más puede "suceder" durante la transmisión del mensaje. Esta es la asunción correcta dentro de muchas computadoras. Si entregar el mensaje requiere algún tiempo, durante el que algo más puede suceder (como un mensaje en el sentido contrario), entonces la flecha de mensaje se puede inclinar hacia abajo de forma que la cabeza de flecha esté por debajo de la cola de la flecha. Sin embargo, a no ser que dos mensajes se crucen, no se pueden sacar conclusiones reales de los mensajes inclinados.

Diagramas de comunicación

En un diagrama de comunicación, un mensaje se representa como una pequeña flecha etiquetada adjunta a una ruta entre los objetos emisor y receptor. La ruta es el conector utilizado para acceder al objeto destino. La flecha discurre a lo largo de la ruta apuntando en la dirección del objeto destino. En el caso de un mensaje desde un objeto a sí mismo, el mensaje aparece en una ruta que vuelve al mismo objeto y aparece la palabra clave «**self**» en el extremo de destino. Se puede adjuntar más de un mensaje a un enlace, en la misma o en distintas direcciones. La flecha de mensaje se etiqueta con el nombre del mensaje (operación o nombre de la señal) y sus argumentos.

La ordenación relativa de los mensajes se representa mediante los números de secuencia en la etiqueta del mensaje. A diferencia de los diagramas de secuencia, en los que la secuencia se muestra gráficamente, los números de secuencia son necesarios en los diagramas de comunicación. Un mensaje también se puede etiquetar con una condición de guarda.

Sincronicidad. Se pueden utilizar los mismos tipos de flecha que en los diagramas de secuencia.

Etiqueta de mensaje. La etiqueta tiene la siguiente sintaxis:

`expresión-de-secuencia`$_{\text{opc}}$ **`mensaje`**

donde **`mensaje`** es la sintaxis de mensajes descrita previamente bajo los diagramas de secuencia, y la expresión de secuencia combina ordenación y condicionamiento, como se describe debajo.

Expresión de secuencia. La expresión de secuencia es una lista separada por puntos de términos de secuencia seguidos por dos puntos (':'). Cada término representa un nivel de anidamiento procedimental dentro de la interacción global. Si todo el control es asíncrono, entonces no se produce el anidamiento y se utiliza un sistema de numeración sencillo de un solo nivel. Cada término de secuencia tiene la siguiente sintaxis:

`etiqueta expresión-de-iteración`$_{\text{opc}}$

donde **`etiqueta`** es

`entero`

o

`nombre`

El entero representa la ordenación secuencial del mensaje dentro del siguiente nivel superior de llamada procedimental. Los mensajes que se diferencian en un término entero están relacionados secuencialmente a ese nivel de anidamiento. Un ejemplo es: 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 se diferencian en el nombre final son concurrentes a ese nivel de anidamiento. Un ejemplo es: El mensaje 3.1.ª y el mensaje 3.1b son concurrentes dentro de la activación 3.1. Todos los hilos de control son iguales dentro de la profundidad de anidamiento.

La expresión de iteración representa ejecución condicional o iterativa. Esto representa que se ejecutan cero o más mensajes, dependiendo de las condiciones. Las elecciones son

* [**`cláusula-de-iteración`**]	una iteración
[**`cláusula-de-condición`**]	una bifurcación

Una iteración representa una secuencia de mensajes en el nivel de anidamiento dado. La **`cláusula-de-iteración`** se puede omitir (en cuyo caso, las condiciones de iteración no están especificadas).

La **`cláusula-de-iteración`** está pensada para expresarse en pseudocódigo o en un lenguaje de programación real; UML no impone su formato. Un ejemplo podría ser:

```
*[i := 1..n].
```

Una condición representa un mensaje cuya ejecución está sujeta a la verdad de la cláusula de condición. La **`cláusula-de-condición`** está pensada para ser expresada en pseudocódigo o en un lenguaje de programación real; UML no impone su formato. Un ejemplo podría ser: [x > y].

Observe que una bifurcación se representa igual que una iteración sin asterisco. Uno podría pensar en ella como en una iteración restringida a una sola ocurrencia.

La notación de iteración asume que los mensajes de la iteración se ejecutarán secuencialmente. También existe la posibilidad de ejecutarlos concurrentemente. La notación para esto es escribir a continuación del asterisco una línea vertical doble, de paralelismo (*||).

Observe que en una estructura de control anidada, el símbolo de repetición no se repite en los niveles internos. Cada nivel de estructura especifica su propia iteración dentro del contexto que la engloba.

Ejemplo

Los siguientes son ejemplos de sintaxis de etiquetas de control de mensajes.

2: mostrar(x, y)	**`Mensaje sencillo`**
1.3.1: p=encontrar(especificaciones):estado	**`Llamada anidada con valor de retorno`**
1b.4 [x < 0]: invertir (x, color)	**`Condicional con un segundo hilo concurrente`**
3.1*[i:=1..n]: actualizar( )	**`Iteración`**

Diagramas de actividad

Recepción de señal. La recepción de una señal puede representarse como un pentágono cóncavo que parece un rectángulo con una muesca triangular a su lado (cualquier lado). La signatura de la señal se representa dentro del símbolo. Se dibuja una flecha de transición sin etiquetar desde el estado de acción previo hasta el pentágono, y otra flecha de transición sin etiquetar desde el pentágono hasta el siguiente estado de acción. Este símbolo reemplaza la etiqueta de evento en la transición, que se dispara cuando la actividad previa ha finalizado y sucede el evento (Figura 14.178).

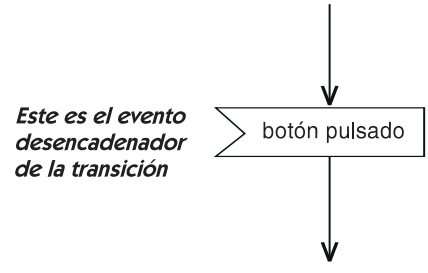


Figura 14.178 Recepción de señal

Envío de señal. El envío de una señal se puede representar como un pentágono convexo que parece un rectángulo con un punto triangular en un lado (cualquier lado). La signatura de la señal se representa dentro del símbolo. Se dibuja una flecha de transición sin etiquetar desde el estado de acción previo hasta el pentágono, y otra flecha de transición sin etiquetar desde el pentágono hasta el siguiente estado de acción. Este símbolo reemplaza la etiqueta de envío de señal en la transición (Figura 14.179).

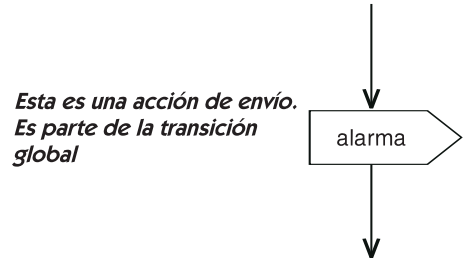


Figura 14.179 Envío de señal

Ejemplo

En la Figura 14.180, **IntroducirDatosDeTarjetaDeCrédito** y **CargoTarjeta** son actividades. Cuando finalizan, el procesamiento va al siguiente paso. Después de que finalice **IntroducirDatosDeTarjetaDeCrédito**, hay una división en base a la cantidad de la petición; si es mayor de 25 €, se debe obtener autorización. Se envía una señal **petición** al centro de crédito. En una máquina de estados plana, esto se podría representar como una acción adjunta a la transición que abandona **IntroducirDatosDeTarjetaDeCrédito**; significan lo mismo. **EsperarAutorización**, sin embargo,

es un estado de espera real. No es una actividad que se completa internamente. Debe esperar a una señal externa del centro de crédito (autorizar). Cuando se produce la señal, se dispara una transición normal y el sistema va a la actividad **CargoTarjeta**. El evento desencadenador se podía haber representado como una etiqueta en la transición desde **EsperarAutorización** hasta **Cargo-Tarjeta**. Es simplemente una notación variante que significa lo mismo.

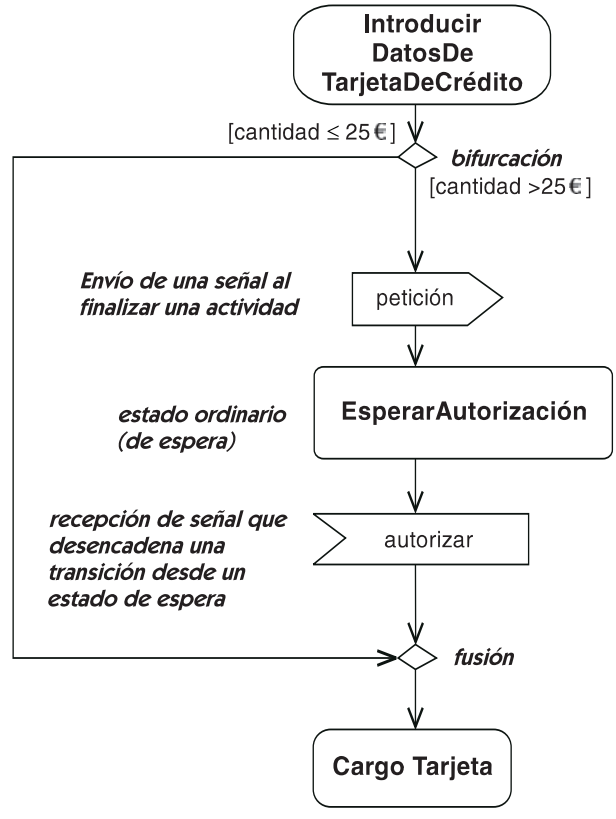


Figura 14.180 Diagrama de actividad que muestra el envío y recepción de señales

Historia

El concepto de mensaje ha sido ampliado considerablemente en UML2. Ahora el contenido puede ser cualquier objeto. El envío de llamadas y señales se ha traído dentro de un enfoque global común.

metaclass (estereotipo de clase)

Clase cuyas instancias son clases. Las metaclases se utilizan típicamente para construir metamodelos. Una metaclase se puede modelar como un estereotipo de una clase utilizando la palabra clave «**metaclass**». *Véase* la Figura 14.181.

Véase supratipo, perfil, estereotipo.

Figura 14.181 Metaclase

metametamodelo

Modelo que define el lenguaje para expresar un metamodelo. La relación entre un metametamodelo y un metamodelo es análoga a la relación entre un metamodelo y un modelo. Este nivel de indirección normalmente es relevante sólo para los constructores de herramientas, constructores de bases de datos y cosas por el estilo. UML se define en términos de un metametamodelo, conocido como Servicio de metaobjetos (MOF, Meta-Object Facility).

metamodel (estereotipo de Modelo)

Modelo que define el lenguaje para expresar otros modelos; una instancia de un metametamodelo. El metamodelo UML define la estructura de los modelos de UML. Los metamodelos normalmente contienen metaclases.

metaobjeto

Término genérico para todas las entidades en un lenguaje de metamodelado. Por ejemplo, metatipos, metaclases, metaatributos y metaasociaciones.

metarrelación

Término que agrupa relaciones que conectan descriptores con sus instancias. Incluye la relación de instancia y la relación de supratipo.

método

Implementación de una operación. Especifica el algoritmo o procedimiento que produce los resultados de una operación.

Véase también concreto/a, operación, realización, resolución.

Semántica

Un método es la implementación de una operación. Si una operación no es abstracta, debe tener un método o desencadenar una transición de máquina de estados, bien definida en la clase con la operación o heredado de un antecesor. Un método se especifica como una expresión procedimental, una cadena lingüística en un lenguaje designado (como C++, Smalltalk, o un lenguaje

humano) que describe un algoritmo. Por supuesto, debe hacerse concordar el lenguaje con el propósito. Un lenguaje humano, por ejemplo, puede ser adecuado en el análisis temprano, pero no es apropiado para generación de código.

Una declaración de una operación implica la presencia de un método a no ser que la operación se declare abstracta. En una jerarquía de generalización, cada declaración repetida de la operación implica un nuevo método que anula cualquier método heredado de la misma operación. Dos declaraciones representan la misma operación si sus signaturas concuerdan.

Observe que un método es un procedimiento ejecutable —un algoritmo— no simplemente una especificación de resultados. Por ejemplo, una especificación antes-y-después no es un método. Un método es un compromiso de implementación y se dirige a temas de algoritmos, complejidad computacional y encapsulación.

En ciertos aspectos, un método puede tener propiedades más estrictas que su operación. Un método puede ser una consulta incluso cuando la operación no esté declarada como una consulta. pero si la operación es una consulta, entonces el método debe ser una consulta. De forma similar, un método puede fortalecer la propiedad de concurrencia. Una operación secuencial se puede implementar como un método con guarda o como un método concurrente. En esos casos, el método es consistente con la declaración de sus operaciones. Simplemente fortalece las restricciones.

Notación

La presencia de un método se indica mediante una declaración de operación que no tiene la propiedad abstracta (Figura 14.182). Si la operación es heredada, el método se puede representar 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 como una nota adjunta a la entrada de la lista de operaciones, pero normalmente los cuerpos de los métodos no se muestran en absoluto en los diagramas. Permanecen ocultos en un editor de texto para mostrar bajo demanda.

Discusión

La especificación de UML no define realmente cómo se mapean las operaciones en métodos o llamadas a operación. La especificación asume que hay algún mecanismo de resolución, pero la

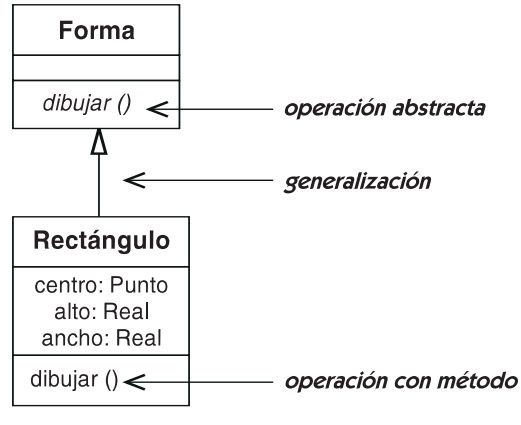


Figura 14.182 Método en una operación no abstracta

especificación no define el mecanismo preciso. Esto permite que los modelos UML se utilicen con un amplio rango de lenguajes de programación con diferentes formas de búsqueda de métodos, pero desafortunadamente deja al modelo un tanto ambiguo. En la práctica, se debe asumir para la mayoría de los modelos de una determinada organización un mecanismo de resolución específico. En muchos casos, la resolución será la búsqueda de métodos tradicional en la orientación a objetos, en la que la búsqueda de un método para una operación dada se hace comenzando en la clase destino y trabajando hacia los niveles más generales de la jerarquía de clases. Sin embargo, son posibles otros mecanismos de resolución, y deben ser soportados por las herramientas apropiadas.

Véase resolución.

miembro

Nombre de un componente estructural heredable y con nombre, de un clasificador, o bien un atributo, extremo de la asociación poseído, operación, recepción o método. Cada clasificador puede tener una lista de cero o más de cada tipo de miembro. Una lista de miembros de un tipo dado se representa como una lista de cadenas dentro de un compartimento del símbolo del clasificador.

Véase también lista.

modelLibrary (estereotipo de Paquete)

Paquete que contiene elementos del modelo con la intención de ser reutilizados por otros paquetes, pero sin definir estereotipos para extender el metamodelo (como en un perfil).

modelo

Descripción semánticamente completa de un sistema.

Véase también paquete, subsistema.

Semántica

Un modelo es una abstracción de un sistema desde un punto de vista particular. Describe el sistema o entidad al nivel de precisión y punto de vista elegido. Los distintos modelos proporcionan puntos de vista más o menos independientes que se pueden manipular por separado.

Un modelo puede comprender una jerarquía contenedora de paquetes en la que el paquete de más alto nivel corresponde al sistema entero. Los contenidos de un modelo son el cierre transitivo de sus relaciones de contención (propiedad) desde los paquetes de nivel superior hasta los elementos del modelo.

Un modelo también puede incluir partes relevantes del entorno del sistema, representadas, por ejemplo, por actores y sus interfaces. En particular se puede modelar la relación del entorno con los elementos del sistema. Un sistema y su entorno forman un sistema más grande a un nivel de ámbito mayor. Por tanto, es posible relacionar elementos a varios niveles de detalle de una forma elegante.

Los elementos en modelos diferentes no se afectan directamente unos a otros, pero a menudo representan los mismos conceptos a diferentes niveles de detalle o a diferentes etapas de desarrollo. Por tanto, las relaciones entre ellos, como la dependencia de traza y refinamiento, son importantes para el mismo proceso de desarrollo y a menudo captura decisiones de diseño importantes.

Notación

Un modelo se puede representar con un símbolo de paquete (rectángulo con una pequeña etiqueta rectangular en la parte superior izquierda) con la palabra clave «**model**». En lugar de la palabra clave, se puede colocar un pequeño triángulo en el cuerpo del símbolo o en la etiqueta. *Véase* la Figura 14.183.

Sin embargo, hay poco detalle de notación que mostrar sobre los modelos. Las herramientas pueden mostrar listas de modelos, pero los modelos tienen pocas relaciones entre ellos mismos. Es más útil la capacidad de cruzar desde un nombre de modelo a su paquete superior o a un mapa de su contenido global.

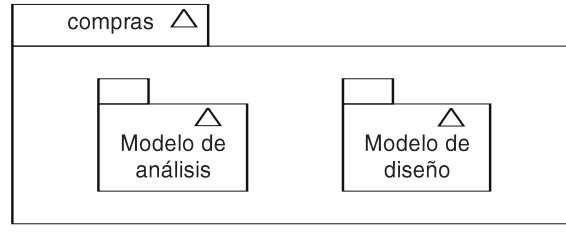


Figura 14.183 Notación de modelo

Discusión

Ninguna vista de un sistema, ni de hecho el propio sistema, está nunca completo por sí mismo. Siempre hay conexiones con el ancho mundo, y un modelo nunca es tan bueno como la realidad. Por tanto, el concepto de un modelo cerrado siempre es una aproximación en la que se pueden dibujar líneas arbitrarias para el trabajo práctico.

Un modelo UML se representa como una jerarquía de paquetes que enfatiza una vista de un sistema. Cada modelo puede tener su propia jerarquía de niveles que puede ser similar o diferente a 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 como se manifiesta a los actores externos que interactúan con el sistema. Un modelo de casos de uso se representa en un diagrama de casos de uso.

modelo de diseño

Modelo construido para explorar alternativas de arquitectura e implementación, en oposición a entender los requisitos del dominio. Contrastar con: modelo de análisis, modelo de implementación. (Este no es un término oficial de UML.)

módulo

Unidad software de almacenamiento y manipulación. Los módulos incluyen módulos de código fuente, módulos de código binario y módulos de código ejecutable. La palabra no corresponde con una construcción UML única, sino que incluye varias construcciones.

Véase componente, paquete, subsistema.

MOF

Servicio de metaobjetos (Meta-Object Facility), una especificación del Object Management Group (OMG) creado con la intención de su utilización en la especificación de lenguajes de modelado, como UML y CWM (Common Warehouse Metamodel —Metamodelo de Almacén Común). El MOF tiene mecanismos de depósito apropiados para almacenar y acceder a modelos en diversos lenguajes. MOF define un lenguaje de modelado idéntico a un subconjunto de UML. Un usuario de UML no necesita aprender el MOF para construir modelos UML, puesto que los conceptos UML son suficientes para su utilización. MOF puede ser útil para desarrolladores que escriban herramientas para intercambiar modelos. Los usuarios ordinarios simplemente utilizarán dichas herramientas y no necesitan entender su formato interno.

muchos

Abreviatura para la multiplicidad 0..* —es decir, cero o más (ilimitado). En otras palabras, la cardinalidad de una colección con esta multiplicidad puede ser cualquier entero no negativo.

Véase multiplicidad.

multiobjeto

Este concepto de UML1 se ha eliminado en UML2.

Discusión

Un multiobjeto tenía la intención de permitir modelar un conjunto de dos maneras complementarias: como un solo objeto que puede tener operaciones sobre el conjunto entero y como un con-

junto de objetos individuales que tienen sus propias operaciones. El concepto se puede modelar en UML2 utilizando una clase estructurada en vez de un multiobjeto. La clase estructurada contiene un conjunto de objetos. Otra clase puede tener asociaciones tanto con la clase estructurada como con sus partes.

La Figura 14.184 muestra un ejemplo.

Para realizar una operación en cada objeto en un conjunto de objetos asociados requiere dos mensajes: una iteración sobre el grupo (el multiobjeto) para extraer enlaces a los objetos individuales, y luego un mensaje enviado a cada objeto utilizando el enlace (temporal).

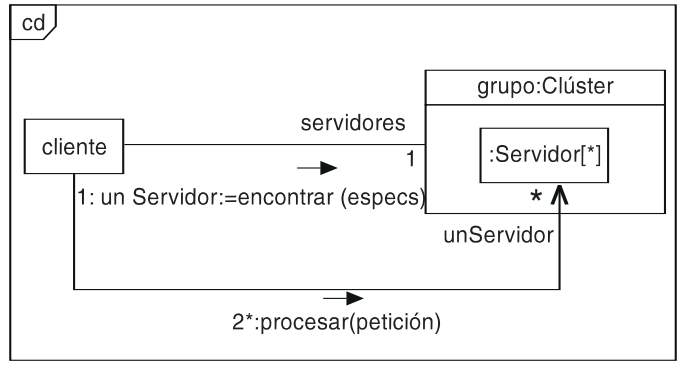


Figura 14.184 Diagrama de comunicación con multiobjeto

Esto se puede evitar en un diagrama combinando los mensajes en uno que incluya una iteración y una aplicación a cada objeto. El nombre de rol de destino toma un indicador *muchos* (*) para mostrar que están implicados varios enlaces. Aunque esto se puede escribir como un solo mensaje, en el modelo subyacente (y en cualquier código real) requiere las dos capas de estructura (iteración para encontrar enlaces y mensaje utilizando cada enlace) mencionadas previamente.

multiplicidad

Especificación del rango de valores de cardinalidad permisibles —los tamaños— que puede asumir una colección. Las especificaciones de multiplicidad pueden ser dadas por extremos de asociación, atributos, partes dentro de clases compuestas, repeticiones de mensajes y otros propósitos. En principio, una multiplicidad es un subconjunto (posiblemente infinito) de enteros no negativos. En la práctica, es un intervalo entero. Si la multiplicidad es mayor que uno, incluye una indicación de si los elementos están ordenados y son únicos.

Contrastar con: cardinalidad.

Véase también multiplicidad (de asociación), multiplicidad (de parte).

Semántica

Rango de cardinalidad. La multiplicidad es una restricción sobre la cardinalidad (tamaño) de una colección. En principio, es un subconjunto de enteros no negativos. En la práctica,

normalmente es un solo intervalo con un valor mínimo y otro máximo. Cualquier colección (en UML) debe ser finita, pero el límite superior de todas las colecciones puede ser finito o sin límite (una multiplicidad sin límite se conoce como "muchos"). El límite superior debe ser mayor que cero; o, por lo menos una multiplicidad que comprenda sólo cero no es muy útil, puesto que permite solamente el conjunto vacío. La multiplicidad se codifica como una cadena.

En UML, el rango de multiplicidad se especifica como intervalos enteros. Un intervalo es un conjunto de enteros contiguos caracterizados por sus valores mínimo y máximo. Algunos conjuntos infinitos no se pueden especificar de esta forma —por ejemplo, el conjunto de los enteros pares— pero normalmente se pierde poco cuando simplemente se incluyen los intervalos. Para la mayoría de propósitos de diseño, un propósito principal de la multiplicidad es limitar la cantidad de almacenamiento que se podría necesitar.

Véase multiplicidad (de asociación) y multiplicidad (de parte) para detalles específicos sobre la utilización de la multiplicidad con esos elementos.

Ordenación y unicidad. Si el límite superior es mayor que uno, la multiplicidad también incluye indicadores para la ordenación y la unicidad. Un conjunto es ordenado si sus elementos se pueden atravesar en una secuencia fija. Los elementos son únicos si ningún par de elementos tiene el mismo valor: de otro modo se permiten elementos duplicados. Una colección que permite duplicados se conoce como bolsa.

Notación

El rango de multiplicidad se especifica mediante una expresión de texto para un intervalo entero de la forma

```
mínimo..máximo
```

donde **`mínimo`** y **`máximo`** son enteros, o **`máximo`** puede ser un "*", lo que indica un límite superior ilimitado. Una expresión como 2..* se lee "2 o más".

Un intervalo también puede tener la forma

```
número
```

donde **`número`** es un entero que representa un intervalo de un solo tamaño. Es equivalente a la expresión **`número..número`**.

La expresión de multiplicidad consistente en una sola estrella

```
*
```

es equivalente a la expresión 0..* —es decir, indica que la cardinalidad no está restringida ("cero o más, sin límite"). Esta multiplicidad tan común se lee como "muchos".

La ordenación y la unicidad se representan mediante palabras clave entre llaves. La ordenación puede ser **ordered** o **unordered**. Por defecto es unordered y no necesita mostrarse. La unicidad puede ser **unique** o **nonunique**. Por defecto es unique y no necesita mostrarse.

Se pueden utilizar las siguientes palabras clave para la combinación de propiedades:

set	sin ordenar, elementos únicos (por defecto)
bag	sin ordenar, elementos no únicos
ordered set	ordenado, elementos únicos
list (o sequence)	ordenado, elementos no únicos

Ejemplo

0..1	**valor opcional**
1	**exactamente uno**
0..*	**cualquier número de elementos, sin ordenar, únicos**
*{list}	**cualquier número de elementos, ordenados, duplicados**
1..* {ordered}	**uno o más, ordenados, únicos**
1..6 {bag}	**entre 1 y 6, sin ordenar, duplicados**

Historia

En UML1, la ordenación era una propiedad separada de la multiplicidad. Puesto que los conjuntos ordenados no pueden contener duplicados, muchos modelos de listas UML eran ligeramente incorrectos.

En UML2, se reconoció que la ordenación y la unicidad están íntimamente relacionadas con la multiplicidad, de forma que fueron combinados en un concepto.

Discusión

Una expresión de multiplicidad puede incluir variables, pero deben evaluarse a valores enteros cuando el modelo está completo —es decir, deben ser parámetros o constantes. La multiplicidad no está destinada a ser evaluada dinámicamente dentro de un ámbito de ejecución como un límite de array dinámico. Está destinada a especificar el posible rango de valores (peor caso) que un conjunto puede asumir y que la aplicación por tanto debe acomodar en sus estructuras de datos y operaciones. Es una constante de tiempo de modelado. Si los límites son variables en tiempo de ejecución, entonces la elección de multiplicidad apropiada es muchos (0..*).

La multiplicidad se puede suprimir en un diagrama, pero existe en el modelo subyacente. En un modelo terminado, no tiene sentido una multiplicidad “no especificada”. No conocer la multiplicidad no es diferente de decir que es muchos, puesto que en ausencia de cualquier conocimiento, la cardinalidad podría tomar cualquier valor, que es justo el significado de muchos.

Véase valor no especificado.

multiplicidad de un atributo

El posible número de valores de un atributo en cada objeto.

Semántica

La multiplicidad asociada a un atributo declara cuántos valores pueden existir por cada objeto que tiene ese atributo. Si la multiplicidad máxima es mayor que uno, también indica si los elementos están ordenados y duplicados. La multiplicidad usual es exactamente uno (**1..1**), significando que cada objeto tiene un valor para el atributo. Otras multiplicidades comunes son cero o uno (un valor optativo o "anulable"); cero o más, sin límite (un conjunto de valores); y uno o más, sin límite (un conjunto no vacío de valores). La frase "cero o más sin límite", normalmente se llama muchos.

Notación

La multiplicidad se muestra por un adorno de rango de multiplicidad entre paréntesis después del tipo del atributo (Figura 14.185). Si no hay ningún paréntesis, entonces la multiplicidad es exactamente uno (un valor escalar, el valor predeterminado). Las palabras claves que indican la clasificación y unicidad del atributo, se ponen entre llaves después de los paréntesis.

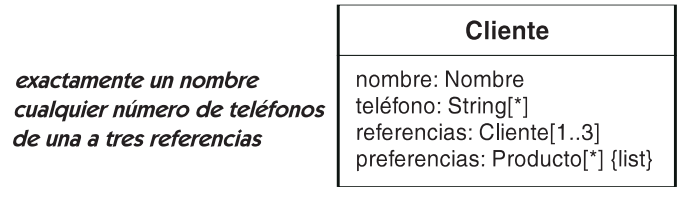


Figura 14.185 Multiplicidad de atributos

multiplicidad de una asociación

La multiplicidad especificada en un extremo de una asociación.

Véase multiplicidad.

Semántica

La multiplicidad junto a un extremo de una asociación declara cuántos objetos pueden referenciarse desde la posición definida por el extremo de la asociación.

Para una asociación binaria, la multiplicidad en el extremo designado restringe cuántos objetos de la clase designada pueden asociarse con un solo objeto dado perteneciente al otro extremo. La multiplicidad se define típicamente como un rango de enteros. (*Véase* multiplicidad para una definición más general.) Las multiplicidades más comunes son uno, cero o uno; cero o más, sin límite; y uno o más, sin límite. La frase "cero o más, sin límite" se llama normalmente muchos. Si la multiplicidad es mayor que uno, también contiene indicadores de si la colección de asociados están ordenados o si son únicos.

En una asociación de orden *n*, la multiplicidad se define con respecto al otro extremo *n*-1. Por ejemplo, dada una asociación ternaria entre las clases (A, B, C), entonces la multiplicidad del extremo C dice cuantos objetos de C pueden aparecer en asociación con un par específico de

objetos A y B. Si la multiplicidad de esta asociación es (muchos, muchos, uno), hay un único valor de C entonces para cada posible pareja (A, B).

Para un par dado (B, C), puede haber muchos valores de A, sin embargo, hay muchos valores de A, B y C que pueden participar en la asociación.

Véase asociación *n*-aria para una discusión sobre la multiplicidad de orden *n*.

Notación

La multiplicidad se muestra mediante un adorno junto a la asociación a la que se aplica (Figura 14.186). Un rango de números tiene la forma **n1..n2**. Se pueden usar palabras claves para indicar su clasificación y singularidad.

Véase multiplicidad para más detalles sobre la sintaxis y las formas más generales de especificación (aunque éstos son probablemente los casos más generales y necesarios en la práctica).

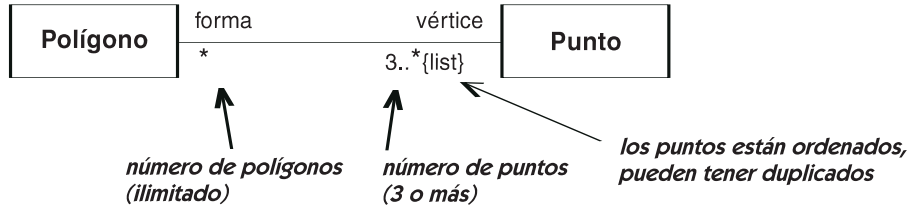


Figura 14.186 Multiplicidad de asociación

multiplicidad de una parte

El rango de posibles cardinalidades de las instancias dentro de un contexto. Esto es, cuántas instancias pueden existir legítimamente a la vez.

Semántica

Cuando es aplicado a una parte dentro de un clasificador estructurado o colaboración, la multiplicidad declara cuántas instancias de la parte pueden existir dentro de una instancia del contenedor.

El valor predeterminado usual es uno, pero pueden declararse otros valores.

Figura 14.187 Multiplicidad de parte

Notación

El indicador de multiplicidad puede ponerse alineado con el nombre, usando la sintaxis de multiplicidad (de atributo) o puede ponerse en la esquina superior derecha del rectángulo de la parte (Figura 14.187). El indicador puede omitirse si la multiplicidad es exactamente uno.

natural ilimitado

Especificación de cardinalidad que puede ser un entero no negativo o la especificación “muchos” indicando que no hay límite superior (aunque es finito). Utilizado en la especificación de la multiplicidad.

navegabilidad

La navegabilidad en un extremo de la asociación es una cualidad Booleana que indica si es posible usar una asociación en tiempo de ejecución para encontrar el valor o valores (dependiendo de la multiplicidad del extremo) del tipo especificado por el extremo, dando un valor para cada uno de los otros extremos. Un atributo siempre es navegable.

Vease también eficiencia de la navegación.

Semántica

La navegabilidad describe si es posible cruzar una asociación en tiempo de ejecución. En tiempo de ejecución, la extensión de una asociación es un juego de tuplas (enlaces), conteniendo cada tupla un valor que corresponde a cada extremo de la asociación. Navegabilidad significa que, a partir de unos valores iniciales para todos los extremos de la asociación excepto uno, es posible obtener todas las tuplas para esos valores; el resultado normalmente se define como el conjunto para el extremo restante de la asociación. La cardinalidad del conjunto de tuplas y el conjunto de valores equivalente se restringe por la multiplicidad en ese extremo de la asociación.

En el caso de una asociación binaria, navegabilidad en un extremo (el extremo destino) significa que es posible obtener el conjunto de valores para el extremo asociado con un valor único del tipo específico del otro extremo (el extremo origen).

La falta de navegabilidad no significa que no haya ninguna manera de cruzar la asociación. Si es posible cruzar la asociación en la otra dirección, es posible buscar todas las instancias de la otra clase para encontrar a esas instancias que le conduzcan a un objeto, y por lo tanto inviertan la asociación. Este acercamiento puede ser práctico en casos pequeños.

Un atributo puede modelarse como una asociación o no, y viceversa. En cualquier caso, los atributos siempre son navegables. En otras palabras, dada una instancia de una clase, es posible obtener el valor o los valores de un atributo específico.

Notación

Una dirección de la asociación navegable se muestra con una punta de flecha en el extremo de la línea de la asociación enlazada a la clase destino. La flecha indica la dirección transversal

(Figura 14.188). El adorno de navegabilidad puede suprimirse (normalmente en todas las asociaciones en un mismo diagrama). Las puntas de flecha pueden aplicarse a cero, uno, o ambos extremos de una asociación binaria o a cualquier número de extremos junto a las clases de una asociación *n*-aria. Nunca se ponen adornos en los extremos de la línea junto al diamante en una asociación *n*-aria.

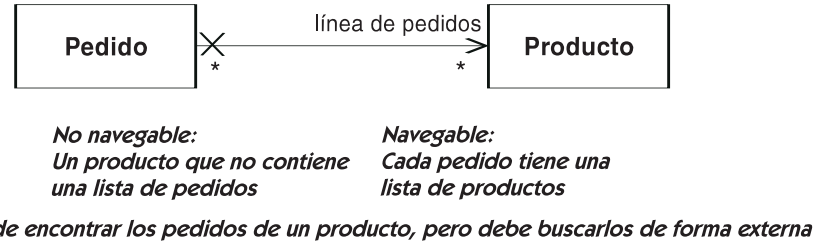


Figura 14.188 Navegabilidad

Una dirección no navegable de una asociación se puede mostrar con una pequeña X en el extremo de una asociación junto a la clase que no puede ser alcanzada.

Por conveniencia, las puntas de flecha pueden omitirse en asociaciones que son navegables en todas las direcciones. En teoría, esto puede confundirse con una asociación que es no navegable en cualquier dirección, pero tal asociación es improbable en la práctica y puede anotarse explícitamente si ocurre.

Alternativamente una falta de anotación indica la navegabilidad "incierta" y toda la navegabilidad debe mostrarse como flechas o Xs. La interpretación de los extremos sin marca es una convención que debe establecerse por los modeladores y las herramientas de modelado.

La Figura 14.189 muestra varias combinaciones de signos de navegabilidad en asociaciones binarias. La notación explícita exige mostrar la navegabilidad explícitamente.

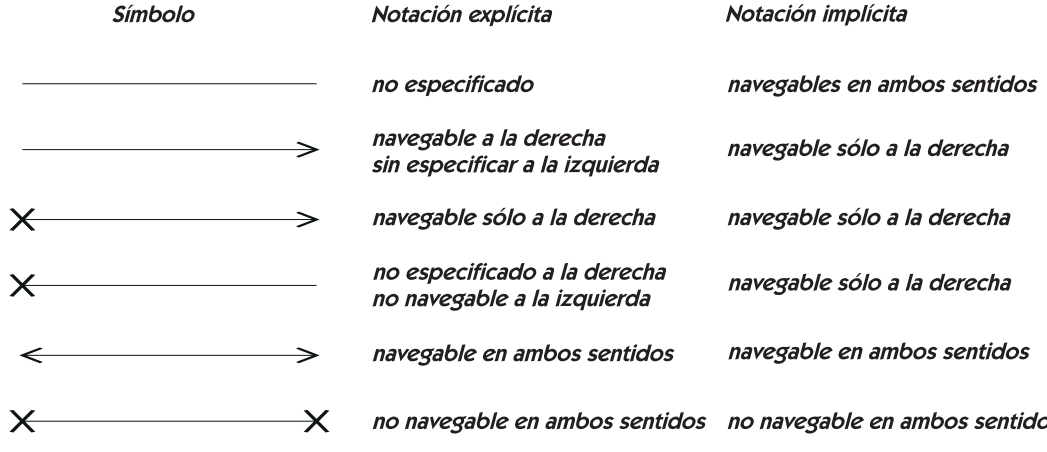


Figura 14.189 Notación para navegación en asociaciones binarias

La notación implícita trata una sola flecha como sentido único navegable y la ausencia de flechas como asociaciones bidireccionalmente navegables.

Discusión

La frase "obtiene los valores" es deliberadamente algo vaga. A menudo implica la navegación eficiente (*véase* entrada). A menudo también implica que el transversal puede realizarse usando una acción (*véase* acción de lectura), usando la sintaxis de un lenguaje de programación o usando una expresión de OCL (como `objeto.nombre`).

La navegabilidad es distinta de la propiedad de una característica de una clase o una asociación. Un extremo de la asociación que pertenece a una clase está reflejado en un atributo de la clase en el extremo opuesto de la asociación y puede usarse por consiguiente en métodos de la clase para cruzar la asociación. Sin embargo, el extremo puede ser navegable aún cuando él no pertenezca a una clase. En ese caso, si la asociación no es visible por la propia clase, el extremo no sería visible dentro de la clase y un método de la clase no pueda cruzar la asociación, pero sería posible cruzar la asociación con un método (de alguna otra clase) que tenga visibilidad sobre la asociación.

navegable

Una asociación o enlace que puede cruzarse en una expresión. Su propiedad de navegabilidad es verdadera. Tal enlace se implementa a menudo como un puntero o conjunto de punteros, pero puede implementarse de otras formas, como una tabla en una base de datos.

Véase también navegabilidad, eficiencia de la navegación.

navegación

Para atravesar las conexiones en un gráfico, sobre todo para atravesar los enlaces y atributos en un modelo de objetos para trazar un objeto a un valor. En el último caso, el camino de la navegación puede expresarse como una sucesión de nombres de atributos o nombres de roles.

Véase también navegabilidad.

neg

Palabra clave para la construcción negativa en una interacción.

negativo

Un fragmento combinado en una interacción que especifica las secuencias de la ejecución que no deben ocurrir.

Semántica

Un fragmento negativo describe secuencias que no pueden ocurrir, es decir, aquéllas que están prohibidas por la interacción. Es evidentemente inútil si todas las secuencias válidas se descri-

ben explícitamente. Es útil si la descripción general está incompleta pero ciertas secuencias deben ser excluidas explícitamente.

Notación

La palabra clave **neg** se pone en la etiqueta de un fragmento combinado. El cuerpo del fragmento describe las secuencias prohibidas.

no determinista

Propiedad por la que una ejecución podría producir más de una posible salida.

Véase determinista.

no interpretado

Marcador de posición para un tipo o tipos cuya implementación no está especificada por UML. Cada valor no interpretado tiene una representación de cadena correspondiente. En cualquier implementación física, como un editor de modelos, la implementación debería ser completa, de forma que no debería haber valores no interpretados.

nodo

Un nodo es un objeto físico en tiempo de ejecución que representa un recurso computacional que generalmente tiene por lo menos capacidad de proceso y memoria. Es posible desplegar artefactos ejecutables en los nodos.

La palabra *nodo* también se usa (confusamente) para denotar una estructura de control compuesta dentro de una actividad. *Véase* nodo de actividad.

Véase también artefacto, despliegue, localización.

Semántica

Los nodos incluyen los dispositivos informáticos pero también (en un modelo comercial, por lo menos) recursos humanos y recursos de proceso de mecánico. Pueden representarse nodos como tipos y como instancias. Un nodo define una localización en la que reside un artefacto.

Los nodos físicos tienen muchas propiedades adicionales, como capacidad, producción y fiabilidad. UML no predefine estas propiedades, ya que hay gran número de posibilidades, pero pueden modelarse en UML usando los estereotipos y los valores etiquetados.

Los nodos se pueden conectar mediante asociaciones para mostrar los medios de comunicación. Las asociaciones pueden tener estereotipos para distinguir varios medios de comunicación caminos o diferentes implementaciones de ellos.

Un nodo es inherentemente parte de la vista de implementación y no de la vista de análisis.

En los modelos de despliegue generalmente aparecen instancias de los nodos en lugar de tipos.

Aunque los tipos de nodo son potencialmente significativos para mostrar los tipos de artefactos que pueden desplegarse, los tipos de los nodos individuales pueden permanecer anónimos en muchos casos.

Un nodo es un clasificador y puede tener atributos. En la mayoría de los casos, las instancias de nodos se muestran en diagramas de despliegue. Los tipos de nodo tienen un uso más limitado.

Hay varios tipos de nodos pero en muchos casos están faltos de semántica, y para distinguirlo es necesario usar algo más que comentarios.

Notación

Un nodo se muestra como una figura que se parece a una proyección descentrada de un cubo (Figura 14.190).

Un tipo del nodo tiene un nombre como un clasificador:

`Tipo-nodo`

donde **`Tipo-nodo`** es el nombre del clasificador.

Una instancia de nodo tiene un nombre y un nombre de tipo. El nodo puede tener una línea de subrayado en el nombre o debajo de él. La cadena del nombre tiene la sintaxis

`nombre: Tipo-nodo`

El **`nombre`** es el nombre del nodo individual (si lo hay). El **`Tipo-nodo`** dice qué tipo de nodo es. Cualquiera o ambos elementos son optativos.

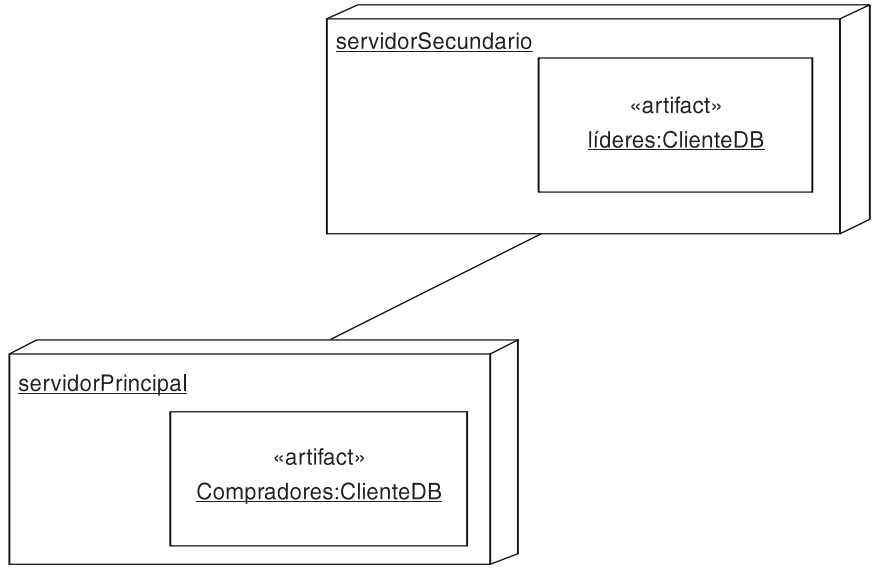


Figura 14.190 Nodos y artefactos

Se pueden utilizar las flechas de la dependencia (flechas punteadas con la punta de flecha en el componente) para mostrar la habilidad de un tipo del nodo para dar soporte a un tipo del componente. Se puede utilizar un tipo de estereotipo para declarar el tipo preciso de dependencia.

Se pueden mostrar los tipos de los artefactos dentro de los símbolos de tipo del nodo. Esto indica las instancias de los artefactos que pueden residir en las instancias de los nodos de los tipos dados.

Se pueden mostrar instancias de artefactos dentro de los símbolos de instancia de las instancias de los nodos.

Los nodos pueden ser conectados por símbolos de asociación a otros nodos. Una asociación entre dos nodos indica un camino de comunicación entre ellos. La asociación puede tener un estereotipo para indicar la naturaleza del medio de comunicación (por ejemplo, el tipo de canal o red).

Ejemplo

La Figura 14.190 muestra dos nodos conectados que contienen dos bases de datos.

nodo almacén de datos

Memoria intermedia central para la información persistente.

Semántica

Un nodo almacén de datos es un nodo objeto que puede aceptar valores de entrada desde varios orígenes y puede enviar valores de salida a diversos destinos. Normalmente los flujos de entrada y de salida están desconectados. En otras palabras, un valor se deposita en el almacén de datos por un hilo de control y extraído después por otro hilo de control diferente. En un almacén de datos, la persistencia de lo datos puede sobrepasar la vida del hilo que creó los datos. Si la actividad que lo contiene termina o ejecuta un nodo final de actividad, los tokens del almacén de datos son destruidos.

Notación

Un nodo almacén de datos se representa como un rectángulo que contiene la palabra clave **«datastore»** encima del nombre del almacén de datos. Los nodos instancia de almacén de datos también pueden representarse subrayando el nombre del nodo.

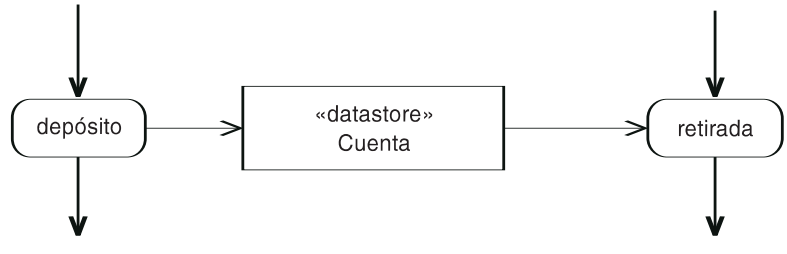


Figura 14.191 Nodo almacén de datos

Ejemplo

La Figura 14.191 muestra una cuenta bancaria que acepta depósitos y retiradas. Las dos acciones se producen de forma independiente en hilos de control diferentes en la misma actividad. No necesitan coincidir.

nodo central de almacenamiento temporal

Un nodo objeto es una actividad que acepta entradas de múltiples nodos objeto o produce salidas sobre múltiples nodos objeto o ambos. Los flujos desde los nodos centrales de almacenamiento temporal no se conectan directamente con acciones.

Semántica

Un nodo central de almacenamiento temporal modela un buffer tradicional que puede albergar valores de varias fuentes o enviar los valores a varios destinos. No hay una ordenación predefinida de valores dentro del nodo, pero dado que es un tipo de nodo objeto, se pueden aplicar las distintas opciones de ordenación de los nodos objeto.

Notación

Se dibuja mediante el símbolo del nodo objeto (un rectángulo) con la palabra clave **«centralBuffer»** (Figura 14.192).

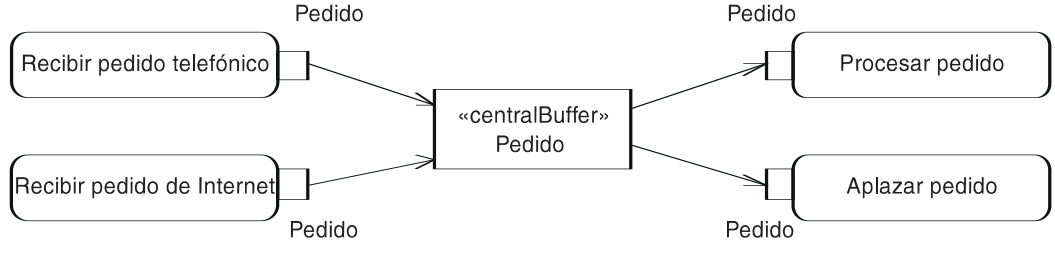


Figura 14.192 Nodo central de almacenamiento temporal

nodo condicional

Nodo de control estructurado en una actividad que representa una elección dinámica entre varias cláusulas.

Semántica

Un nodo condicional es una construcción de control dentro de una actividad, que contiene una o más cláusulas. Cada cláusula tiene una sección de prueba y una sección de cuerpo. Cuando se ejecuta el nodo condicional, se ejecutan las secciones de prueba de las cláusulas en un orden no especificado, posiblemente de forma concurrente. Si una o más de las secciones de prueba producen

valores verdaderos, se ejecuta la sección de cuerpo correspondiente a una de las secciones de prueba verdaderas. Si más de una sección de prueba produce un valor verdadero, sólo se ejecuta una sección de cuerpo. La elección entre las diversas pruebas verdaderas es no determinista.

No se garantiza que se ejecuten todas las secciones de prueba antes de que se elija una opción. Es posible que las secciones de prueba se ejecuten de forma concurrente y que algunas puedan estar parcialmente completas cuando se hace una elección. Los modeladores deberían evitar pruebas que creen efectos laterales, o de lo contrario puede haber indeterminación adicional como resultado de la indeterminación en el orden de ejecución de las pruebas.

La secuencia de pruebas se puede controlar especificando prioridades entre cláusulas. No se probará ninguna cláusula a no ser que todas las cláusulas de mayor prioridad se hayan evaluado a falsas. Puede haber varias cláusulas de la misma prioridad, en cuyo caso puede haber indeterminación entre ellas. Si las cláusulas están totalmente ordenadas, se elimina la potencial indeterminación. Obsérvese sin embargo que la indeterminación puede ser algo útil de modelar en algunas ocasiones.

Los valores creados en una prueba se pueden utilizar en la sección cuerpo correspondiente. Su utilización fuera de la condicional implica restricciones más severas y está desaconsejado.

Una cláusula else se ejecuta si ninguna de las otras cláusulas se evalúa a verdadero. Se modela como una cláusula con una condición verdadera cuya prioridad es menor que todas las demás cláusulas.

Véase disparador cualquiera, else.

Estructura

Un nodo condicional tiene dos subnodos que representan la prueba y el cuerpo. Se designa como resultado de la prueba un pin de salida del nodo de prueba.

Una cláusula puede tener cláusulas predecesoras y sucesoras en la misma condicional. La relación predecesor-sucesor define la prioridad de evaluación de las pruebas.

Si el cuerpo de cualquier cláusula tiene pines de salida, entonces cada cláusula y la misma condicional entera debe tener un conjunto idéntico de pines de salida (idéntico en número, tipo y multiplicidad). Una vez completada la ejecución del cuerpo elegido, el valor de cada uno de sus pines de salida se copia a los correspondientes pines de salida de la condicional. El conjunto de pines en cada cláusula debe coincidir, puesto que de otro modo alguno de los pines de salida de la condicional podría carecer de valores tras la ejecución de ciertas cláusulas.

El flag **asegurado** es una aserción del modelador por la que al menos una prueba tendrá éxito. Esto es necesario si, por ejemplo, acciones posteriores utilizan las salidas de la condicional.

El flag **determinado** es una aserción del modelador por la que sólo una prueba finalizará con éxito al mismo tiempo. Esto se puede utilizar para declarar un comportamiento determinista.

Los flags **asegurado** y **determinado** son aserciones de modelado, y no sentencias ejecutables. Son útiles cuando la inteligencia humana sobrepasa la capacidad de validadores de teoremas, pero si son incorrectos pueden llevar a errores.

Notación

No se define ninguna notación en el documento de UML. Se espera que las actividades estructuradas utilicen normalmente sintaxis textual en algún lenguaje.

Para situaciones sencillas, se puede representar una condicional mediante un nodo de decisión con varias salidas. Hacerlo de esta manera no captura totalmente la estructura anidada de una condicional, pero es satisfactoria para situaciones sencillas.

Para situaciones más complejas, se podría usar una variación de la notación de fragmento condicional: Introducir la condicional en un rectángulo con una etiqueta **alt**, dividirlo en regiones con líneas discontinuas, y empezar cada región con una expresión de prueba entre paréntesis, mientras que el resto de la región será un gráfico de actividad del cuerpo. Aunque una prueba también se puede mostrar como un gráfico de actividad, gráficamente llega a ser confuso mostrar tanto las pruebas como los cuerpos de una forma gráfica.

nodo de actividad

Un tipo de elemento en una actividad que se puede conectar mediante flujos. Es un tipo de elemento abstracto cuyas variedades específicas incluyen acciones, nodos de control, nodos objeto (incluyendo patillas y nodos parámetro) y nodos estructurados.

Semántica

Una descomposición de una actividad comprende nodos de actividad conectados mediante flujos. Los distintos tipos de nodos de actividad se describen dentro de sus propias entradas.

Los nodos de actividad se pueden agrupar en nodos estructurados y particiones. Los nodos estructurados proporcionan una organización jerárquica de nodos que incluye construcciones de control estructuradas.

Los nodos se pueden redefinir como parte de la especialización de la definición de una actividad.

El indicador *mustIsolate* especifica que las acciones en un nodo estructurado se ejecutan sin entrar en conflicto con acciones de otros nodos. Un conflicto es un intento de acceso a la misma información, como a la ranura de atributos de un objeto, cuando al menos una de las que acceden modifica la información, llevando a la posibilidad de una indeterminación. Si existe la posibilidad de conflicto, el nodo se puede ejecutar sin intercalar ninguna ejecución de los nodos en conflicto. Es aceptable cualquier técnica de implementación que suporte este objetivo.

Notación

La notación depende del tipo específico de nodo de actividad. Normalmente es un rectángulo con las esquinas redondeadas o alguna variante de ésta.

nodo de control

Un nodo de actividad en una actividad cuyo propósito es coordinar flujos de control y flujos de objetos entre otros nodos; una construcción específica de flujo-de-control.

Véase también actividad.

Semántica

En una actividad, el flujo de control simple desde un nodo a otro se modela utilizando transiciones de flujos de control y transiciones de flujos de datos. Una acción comienza su ejecución cuando recibe tokens en todas sus transiciones de entrada; cuando completa su ejecución, produce tokens en todas sus transiciones de salida. Las formas de control más complejas se modelan utilizando nodos de control, algunos de los cuales no requieren o no producen tokens en todas sus transiciones. Los nodos de control modelan decisiones, divisiones concurrentes y el inicio y parada de una actividad. Las condicionales estructuradas y bucles se modelan utilizando nodos de actividad estructurados. (No se llaman nodos de control en el metamodelo UML2, pero también modelan flujo de control complejo.)

Tipos de nodos de control

Los tipos de nodos de actividad que modelan control complejo son los siguientes:

Decisión. Un nodo de decisión tiene una transición de entrada y varias transiciones de salida. Normalmente las transiciones de salida tienen guardas. Cuando se activa la transición de entrada, se activa una transición de salida cuya guarda sea verdadera. Sólo se activa una salida cuando se satisfaga más de una guarda. Por comodidad, una salida de la decisión se puede etiquetar con la palabra clave **else.** Este flujo se toma si ninguna otra guarda se satisface. Si ninguna guarda es verdadera, el modelo está mal formado.

Normalmente una decisión va emparejada con una siguiente fusión (para formar una condicional) o con una fusión anterior (para formar un bucle), aunque las decisiones y las fusiones se pueden utilizar para formar patrones de control que no son posibles utilizando estructuras de control totalmente anidadas. Sin embargo, se debe tener cuidado para evitar configuraciones sin sentido.

Una decisión se representa con un diamante con una flecha de entrada y dos o más flechas de salida. Cada salida se etiqueta con una condición de guarda (Figura 14.193).

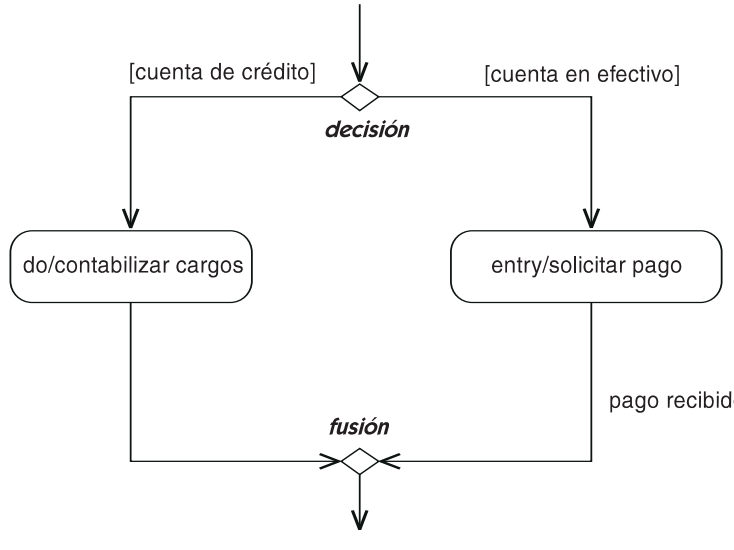


Figura 14.193 Decisión y fusión

Fusión. Una fusión es un lugar en el que dos o más caminos de control alternativos se juntan. Es la inversa de una decisión. Cuando se activa cualquier transición de entrada, se activa la transición de salida.

El diamante es el símbolo tanto para una decisión como para una fusión. Es una decisión si hay varias flechas de salida; es una fusión si hay varias flechas de entrada (Figura 14.193).

División. Un nodo de división tiene una transición de entrada y varias transiciones de salida. Cuando la transición de entrada está activa, todas las transiciones de salida se activan. En otras palabras, una división incrementa la cantidad de actividad concurrente.

Normalmente una división va emparejada con un nodo de unión posterior de forma que la cantidad de concurrencia al final se equilibra, aunque son posibles, y útiles, situaciones más complicadas. Sin embargo, se debe tener cuidado de evitar situaciones sin sentido en las que se crean tokens de actividad en exceso y no se unen adecuadamente.

Una división se representa con una línea gruesa con una flecha de entrada y dos o más flechas de salida (Figura 14.194).

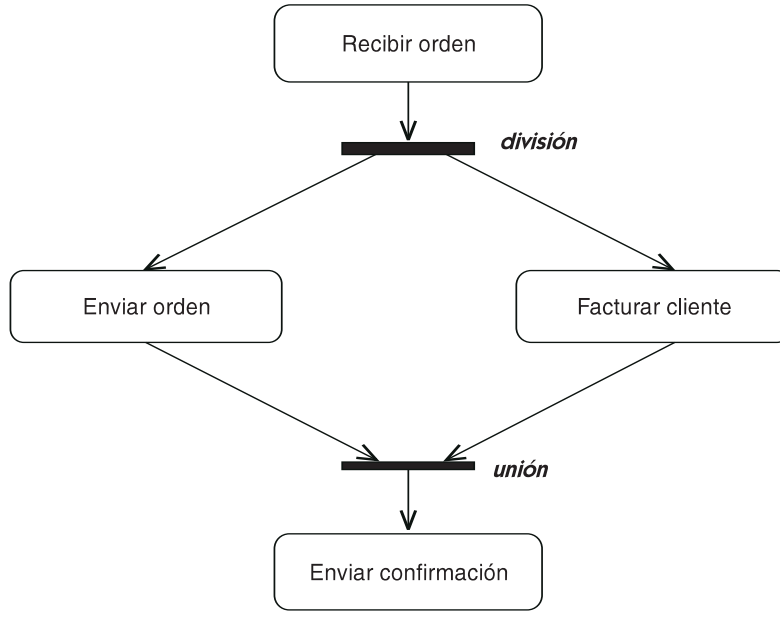


Figura 14.194 División y unión

Unión. Un nodo de unión tiene dos o más transiciones de entrada y una transición de salida. Cuando todas las transiciones de entrada están activas, se activa la transición de salida. En otras palabras, una fusión decrementa la cantidad de concurrencia.

Una fusión se representa con una línea gruesa con dos o más flechas de entrada y una flecha de salida (Figura 14.194).

Se pueden combinar una unión y una división dibujando una línea gruesa con varias flechas de entrada y varias de salida.

Las divisiones y las uniones no son necesarias en las situaciones más sencillas, puesto que las distintas transiciones entrando o abandonando un nodo son equivalentes a una unión o división, pero utilizándolos hacemos obvia la concurrencia. (Observe también que la convención para las diversas transiciones en un nodo ha cambiado desde UML1, por lo que mostrar de forma explícita las divisiones/uniones y las decisiones/fusiones evita cualquier peligro de falta de entendimiento.)

Inicial. Un nodo inicial tiene una transición de salida y ninguna de entrada. Un nodo inicial representa el punto de inicio por defecto para una transición en la actividad contenedora. Cuando se invoca la actividad, se activa la salida del nodo inicial.

Un nodo inicial se representa con un pequeño círculo relleno y una flecha abandonándolo. (Figura 14.195).

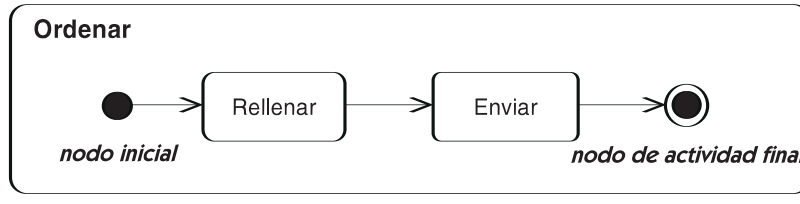


Figura 14.195 Nodos inicial y de actividad final

Actividad final. Un nodo de actividad final tiene una o más transiciones de entrada y ninguna transición de salida. Si cualquiera de las transiciones de entrada está activa, se termina la ejecución de la actividad y cualesquiera nodos o flujos activos se abortan. Cualesquiera salidas producidas por la actividad se organizan en un paquete de retorno. Si la actividad fue invocada por una llamada síncrona, se transmite un mensaje de retorno al emisor con los valores de salida.

Es posible tener varios nodos de actividad final. El primero de ellos en activarse termina toda la actividad.

Una actividad final se representa como un pequeño círculo hueco que contiene un círculo relleno aún menor (un ojo de buey o símbolo de destino) con flechas entrando en él (Figura 14.195).

Flujo final. Un nodo de flujo final tiene una o más transiciones de entrada y ninguna de salida. Si cualquiera de las transiciones está activa, se consumen sus tokens. Esto proporciona una forma de eliminar actividad a la salida de un bucle autónomo. Sin embargo, en la mayoría de los casos es más limpio unir explícitamente los tokens en otro nodo, de forma que esta construcción debería ser utilizada con mucho cuidado.

Un flujo final se representa con un pequeño círculo vacío con una X atravesando el círculo. Tiene una o más flechas entrando en él (Figura 14.196).

Condicional. Un nodo condicional es una construcción de control estructurada con una o más transiciones de entrada, una o más transiciones de salida y dos o más cláusulas embebidas, que son los fragmentos de actividad subordinados. Cuando todas las transiciones de entrada están activas, se evalúan las pruebas de las cláusulas y se elige una sola cláusula para su ejecución. Cuando la cláusula completa su ejecución, sus valores de salida pasan a ser los valores de

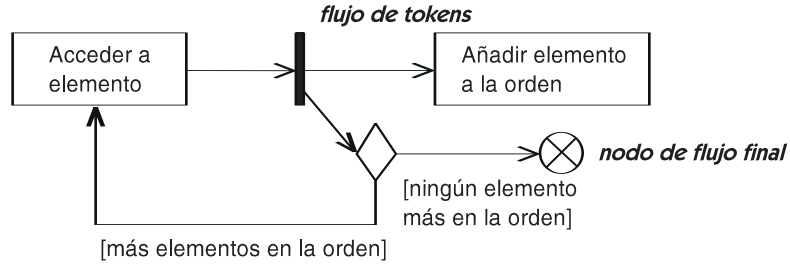


Figura 14.196 Nodo de flujo final

salida de la propia condicional, y se activan las transiciones de salida de la condicional. Para más detalles, *véase* nodo condicional.

Bucle. Un nodo de bucle es una construcción de control estructurada con una o más transiciones de entrada, una o más transiciones de salida y tres fragmentos de actividad embebidos, uno de inicialización, otro de prueba y finalmente uno del cuerpo del bucle. Cuando se activan todas las transiciones de entrada, se ejecuta la parte de inicialización, luego se ejecuta repetidamente la parte del cuerpo mientras que la parte de prueba produzca un valor de salida verdadero. Cuando la prueba es falsa, los valores de salida del cuerpo pasan a ser los valores de salida del bucle en sí, y se activan las transiciones de salida de la condicional. *Véase* nodo repetitivo para ver los detalles completos.

Otras construcciones. Además de las anteriores, otras construcciones que no son consideradas nodos de control realizan funciones que tienen aspectos de control.

Véase manejador de excepción, región de expansión, región de actividad interrumpible.

nodo de decisión

Nodo de control en una actividad que pasa control y datos a una o varias salidas.

Véase también bifurcación, fusión.

Semántica

Un nodo de decisión tiene una entrada y dos o más salidas. El valor de entrada se utiliza para evaluar condiciones de guarda en cada una de las salidas. Si una condición de guarda se evalúa a verdadera, la salida correspondiente es candidata para ser elegida. Una sola salida candidata es elegida para recibir una copia del valor de entrada. Si más de una condición de guarda se evalúa a verdadera, la elección de la salida es no determinista. Si ninguna condición de guarda es verdadera, el modelo está mal formado.

Una de las condiciones de guarda puede tener la condición especial **else**. Se elegirá una salida con una condición else si ninguna otra condición es verdadera.

A un nodo se le puede adjuntar comportamiento de decisión de entrada. Este comportamiento tiene una entrada y una salida. La entrada debe ser del mismo tipo que la entrada del nodo de decisión. Cuando el nodo de decisión recibe un valor, se invoca el comportamiento de decisión

de entrada con el valor de entrada del nodo de decisión como su argumento. El valor de salida del comportamiento debe ser referenciado en condiciones de guarda en las salidas del nodo de decisión. El comportamiento puede ser invocado varias veces, por tanto no debe tener ningún efecto lateral.

Notación

Un nodo de decisión se representa como un diamante con una flecha de entrada y dos o más flechas de salida. Normalmente la mayoría de las flechas de salida tienen condiciones de guarda entre corchetes. Se puede colocar la condición de guarda especial **[else]** en una flecha de salida.

Se muestra un comportamiento de decisión de entrada como un símbolo de nota (rectángulo con la esquina superior izquierda doblada) con la palabra clave **«decisionInput»** encima de una cadena con la expresión para el comportamiento.

Ejemplo

La Figura 14.197 muestra un nodo de decisión operando sobre la salida de una actividad previa. La Figura 14.198 muestra el mismo ejemplo usando una expresión de decisión de entrada para evitar repetir un cálculo en cada bifurcación.

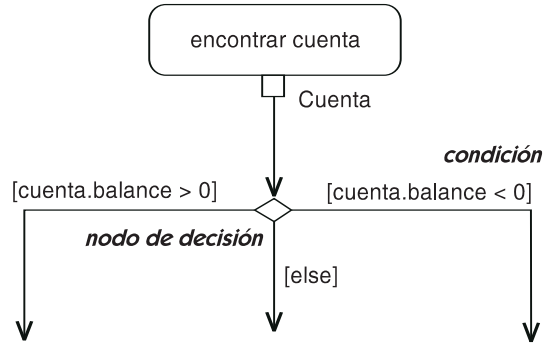


Figura 14.197 Nodo de decisión

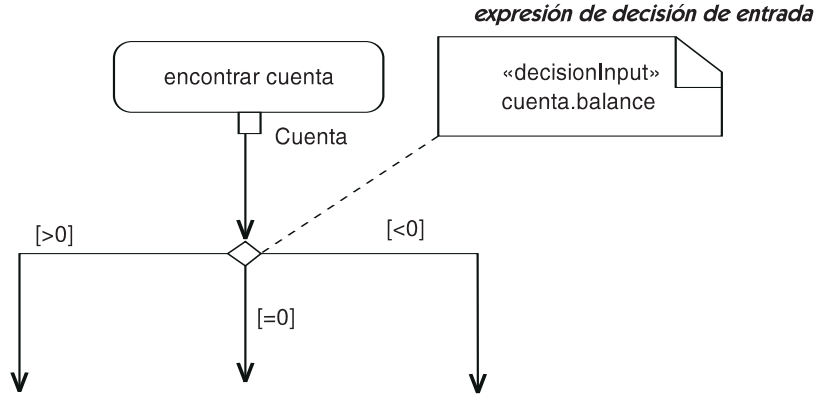


Figura 14.198 Nodo de decisión con expresión de decisión de entrada

nodo de división

Tipo de nodo de control en una actividad que copia una sola entrada en varias salidas concurrentes.

Véase división.

nodo de unión

Nodo de control en una actividad que sincroniza varios flujos.

Semántica

Un nodo de unión tiene varias transiciones de entrada y una transición de salida. Cuando un token está disponible en cada transición de entrada, se consumen los tokens de entrada y se coloca un token en la transición de salida. Esto tiene el efecto de reducir el número de tokens concurrentes en la ejecución.

Se puede colocar una especificación de unión en el nodo para especificar condiciones bajo las que el nodo puede dispararse sin esperar a que los tokens estén presentes en todos los nodos. La especificación de unión tiene semántica complicada e introduce un serio peligro de construir modelos mal formados con tokens sueltos. El peligro puede ser muy reducido colocando el nodo dentro de una región interrumpible y tratar la salida del nodo como una transición de interrupción.

Los tokens de datos se unen sacándolos como una secuencia sobre la transición de salida.

Notación

Una unión se muestra como una barra gruesa con dos o más flechas de flujo de entrada y una flecha de flujo de salida (Figura 14.199).

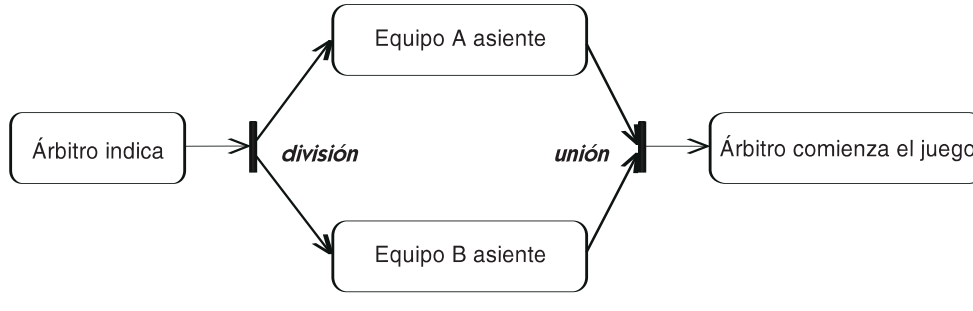


Figura 14.199 Nodos división y unión

Observe que una división y una unión tienen la misma notación, pero diferentes números de entradas y salidas. Es posible combinarlos en un símbolo con varias flechas de entrada y salida.

Discusión

La unión de valores de datos es un área controvertida y podría manejarse de otras maneras. Unir un solo valor de datos con uno o más valores de control es sencillo, aunque la especificación no lo permite explícitamente.

nodo ejecutable

Nodo de actividad que se puede ejecutar y que puede tener un manejador de excepción.

Semántica

Los nodos ejecutables son aquellos que pueden tener manejadores de excepción que pueden capturar excepciones originadas por el nodo o uno de los nodos anidados dentro de él. Los nodos ejecutables incluyen acciones, nodos condicionales, nodos repetitivos y regiones de expansión.

nodo final

Tipo abstracto de nodo en un gráfico de actividad que para la actividad. Los subtipos de nodo final son nodo final de actividad, que termina toda la actividad dentro del gráfico, y nodo final de flujo, que termina el hilo de actividad que contiene al nodo pero no afecta a la actividad concurrente.

nodo final de actividad

Un nodo en una especificación de una actividad cuya ejecución causa la terminación forzada de todos los flujos de la actividad y la terminación de la ejecución de la actividad.

Véase nodo final.

Semántica

Un nodo final de actividad representa la finalización de la ejecución de una actividad. Si hay cualquier ejecución concurrente cuando un token alcanza el nodo, todas las demás ejecuciones finalizan a la fuerza y se eliminan los tokens. En el caso más común, el token que alcanza el nodo es el único activo.

Las salidas de la actividad que se han generado previamente se liberan para su entrega.

Si la actividad fue invocada por otra actividad, se devuelve un token a la actividad que la invocó. Si la actividad representa la vida de un objeto, el objeto se destruye.

Notación

Un nodo final de actividad se muestra como un pequeño círculo con un pequeño punto negro en el interior (el símbolo del ojo de buey).

Ejemplo

La Figura 14.200 muestra una actividad sencilla. La actividad comienza con un nodo inicial y termina con un nodo final de actividad.

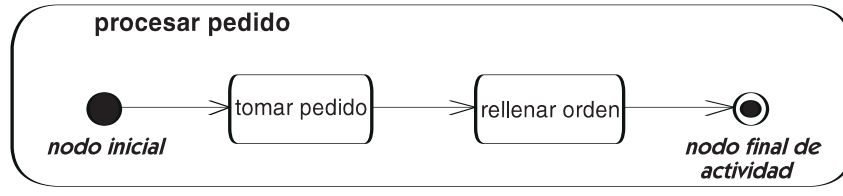


Figura 14.200 Nodo final de actividad y nodo inicial

nodo final de flujo

Nodo en una actividad que destruye todos los tokens que alcanza. No tiene ningún otro efecto en la ejecución de la actividad.

Semántica

Un nodo final de flujo destruirá todos los tokens que alcance. Esto podría ser útil para terminar un hilo de control que puede ejecutar independientemente de que no tendrá mayor efecto en el resto de la actividad.

Notación

Un nodo final de flujo se representa como un círculo con una X dentro (Figura 14.201).

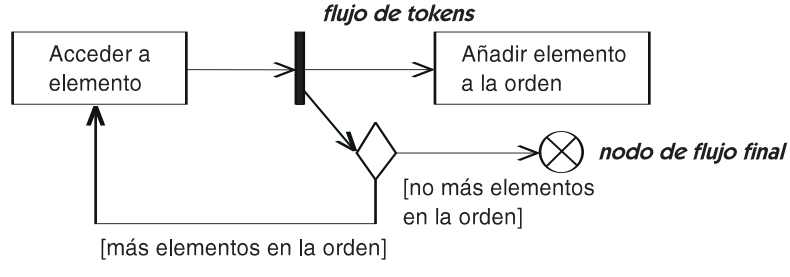


Figura 14.201 Nodo final de flujo

Discusión

El documento UML2 muestra un ejemplo de un flujo final utilizado para terminar un bucle que había estado lanzando ejecuciones concurrentes de una subactividad. Esta construcción proba-

blemente debería evitarse, si es posible, a favor de un bucle estructurado con una fusión de control explícita, puesto que su semántica es sospechosa.

nodo inicial

Nodo de control que indica el lugar donde empieza la ejecución cuando se invoca a la actividad.

Semántica

Un nodo inicial es un nodo de control con ninguna entrada y una salida. Cuando se invoca la actividad que contiene al nodo inicial, se coloca un token de control en el nodo inicial y se permite la ejecución de la actividad.

Una actividad puede tener varios nodos iniciales, en cuyo caso se coloca un token de control en cada uno de ellos. Representaría una situación con concurrencia.

Notación

Un nodo inicial se representa como un pequeño círculo negro con ninguna flecha de entrada y una flecha de salida. *Véase* la Figura 14.200.

nodo objeto

Un tipo de nodo de actividad que representa la existencia de un objeto producido por una acción en una actividad y usado por otras acciones.

Véase también flujo de control, flujo de objeto.

Semántica

Una actividad modela el flujo de control entre las acciones y las estructuras de control. También puede modelar el flujo de valores entre las acciones.

Un nodo objeto representa un valor de objeto que existe a un punto dentro de un cálculo. El objeto puede ser la salida de un nodo de actividad y la entrada de otro nodo de actividad. Un nodo objeto tiene un flujo de objeto de entrada conectado a una acción origen (u otro nodo de actividad). Cuando la acción de origen se completa, se genera un token del objeto y coloca en nodo objeto. Esto representa la creación de un objeto de la clase o un nuevo valor del objeto como resultado de un cómputo. Si un nodo objeto tiene múltiples flujos de entrada, los valores pueden venir de cualquiera de ellos.

Un nodo objeto tiene una conexión a un pin de la salida en una acción de origen (u otro nodo de actividad). Siempre que la acción se ejecute se pone en el nodo objeto. Si el nodo objeto se conecta a los pines de entrada en acciones múltiples, pueden luchar por los valores en el nodo objeto.

Un nodo objeto tiene un tipo. La ausencia de un tipo significa que se permite cualquier tipo. El tipo de los objetos en el nodo debe ser el mismo o descendientes de los tipos del nodo. Si un nodo del objeto es la entrada o la salida de una acción, los tipos deben ser idénticos.

Un nodo objeto puede tener un límite superior en el número de valores que puede contener. Si el nodo está lleno, no aceptará los tokens adicionales de flujos de entrada que pueden a su vez bloquear las acciones precedentes. Si no se especifica ningún límite, la capacidad del nodo es ilimitada.

Por defecto, los valores en un nodo objeto se ordenan según su llegada. Un objeto nodo puede tener un comportamiento que permita seleccionar qué valor se seleccionará.

Por simplicidad, existe la posibilidad de especificar los comportamientos predefinidos FIFO (First In First Out) y LIFO (Last In First Out), pero pueden definirse otras políticas mediante un comportamiento en destino. El comportamiento seleccionador debe estar libre de efectos lado. Un símbolo de nodo objeto representa la existencia de un valor del objeto en un punto particular dentro de la ejecución de una actividad. Nodos objeto diferentes pueden representar el mismo valor en las diferentes fases de su vida. Los nodos objetos también pueden representar los valores de cálculos intermedios, incluso aquéllos que nunca son guardados en memoria.

Notación

Un nodo objeto se representa en un diagrama de actividad como un rectángulo con flechas de flujo de objetos como entradas y salidas. El nombre del tipo del valor de objeto se pone en el rectángulo. El objeto puede restringirse para estar en un estado específico o un conjunto de estados ortogonales. El nombre de un estado o lista de estados puede adjuntarse dentro del cuadrado y puesto después o bajo el nombre del tipo. Las restricciones adicionales en el nodo se ponen entre llaves, dentro o debajo del rectángulo. La sintaxis completa es:

`nombreClase` [**`nombreEstado`**] {**`la restricción`**}

Además, las formas especiales siguientes pueden ponerse fuera del rectángulo:

{upperBound = **`integer`**}

{ordering = **`regla`**}

donde la regla puede ser el nombre de un comportamiento de selección definido por el usuario o puede tener uno de los siguientes:

FIFO }

LIFO

A menos que se especifique, un nodo objeto es ilimitado y FIFO.

Ejemplo

La Figura 14.202 muestra nodos objeto en un diagrama de actividad. Un valor del objeto se crea por la realización de una acción o una actividad invocada. Por ejemplo, un valor con clase y estado **Orden[Presentada]** se crea por la realización de actividad **Solicitar Servicio**. Porque esa actividad tiene dos actividades sucesoras, el flujo de objeto **Orden[Presentada]** es una salida de un símbolo de bifurcación. El estado **Orden[Introducida]**, por otra parte, es el resultado de completar la actividad **TomarOrden**, que no tiene otra actividad sucesora así que no es necesaria una bifurcación, no tiene ningún otro sucesor.

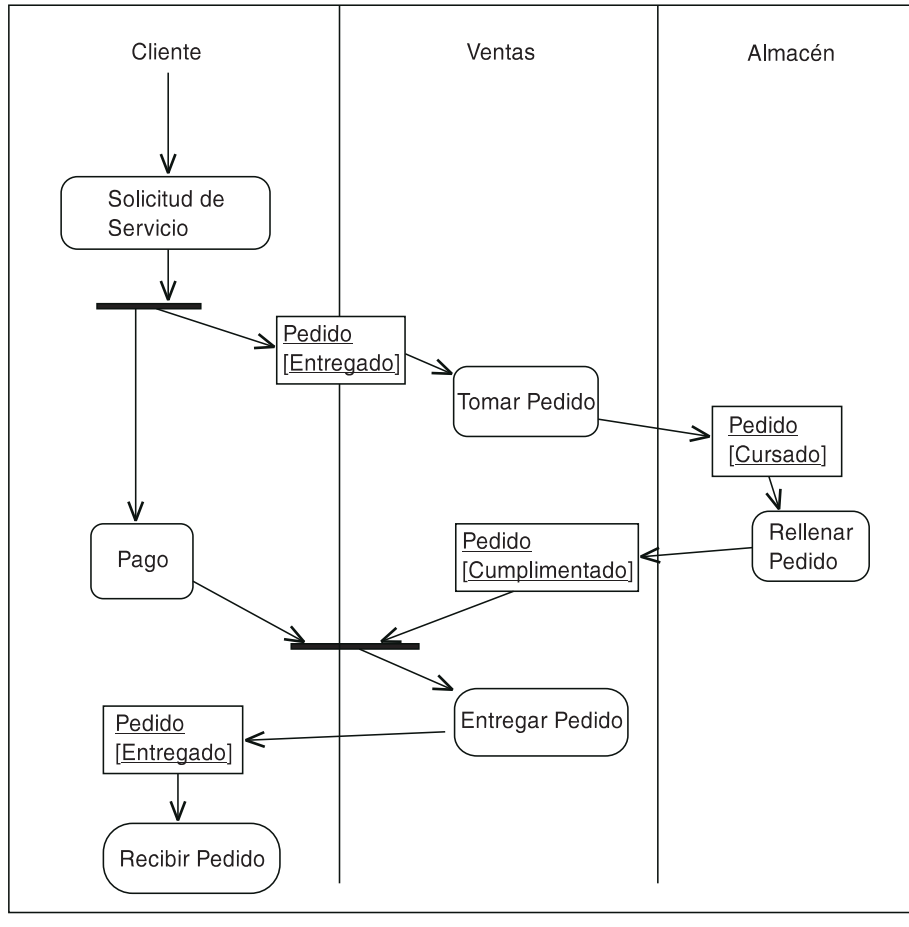


Figura 14.202 Nodos objeto en un diagrama de actividad

En este ejemplo, todos los nodos objeto se trazarán probablemente en un solo objeto cuyo valor es modificado por cada invocación de la actividad.

La Figura 14.203 muestra una porción de un diagrama de actividad relacionada con la construcción de una casa. Cuando se ha construido el marco, el carpintero es libre para trabajar en el tejado y la casa está lista para que la fontanería sea instalada. Estos eventos se diseñan como señales —**Carpintero Libre** y **Marco Preparado**— desde un nodo de actividad a los otros. El diagrama muestra la sintaxis especial para una señal. Esto no implica que los procesos se comuniquen directamente; puede haber comunicación con otros objetos que no se muestran en el modelo. Como resultado de estas señales, se puede construir el tejado e instalar la fontanería. La notación de las señales sólo se necesita para modelar la comunicación entre objetos distintos o los procesos que se ejecutan en ellos.

Discusión

Un nodo objeto representa la vista de los flujos de datos de un cómputo. Al contrario de los tradicionales donde fluyen datos, existe un punto definido para el flujo de control. Esto pone su

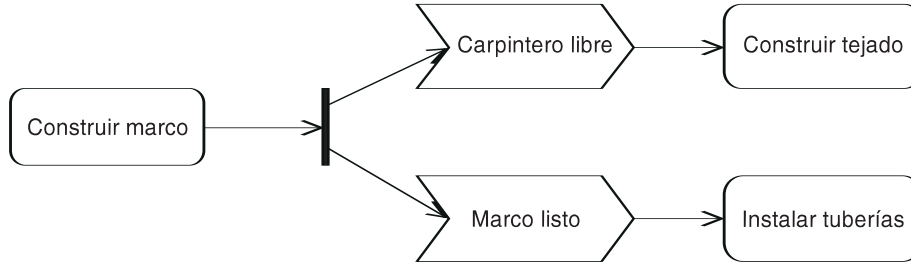


Figura 14.203 Señales en diagramas de actividad

ángulo recto en un armazón objeto-orientado. La orientación del objeto une los puntos de vista de las estructuras de datos, el flujo de control y los flujos de datos en un solo modelo.

nodo repetitivo

Nodo de actividad estructurado que se ejecuta de forma repetida mientras que una condición se evalúe a verdadera. Esta entrada discute los bucles en los modelos de actividad. *Véase también* bucle para los bucles en los modelos de interacción.

Semántica

En un modelo de actividad, un nodo repetitivo es un tipo de nodo de control estructurado. Tiene tres partes que son fragmentos de modelo de actividad más pequeños:

inicialización	Fragmento que computa los valores iniciales para las variables del bucle antes de la primera ejecución del cuerpo.
prueba	Fragmento que evalúa una condición para realizar otra iteración del bucle antes de la primera ejecución del cuerpo (opcionalmente) y antes de cada una de las siguientes ejecuciones. Este fragmento no debería producir efectos laterales. Se designa como resultado de la prueba un valor lógico en la subactividad de prueba. El nodo repetitivo también tiene un indicador que indica si la prueba se debe realizar antes de la primera iteración del cuerpo o si hay al menos una iteración del cuerpo.
cuerpo	Fragmento que realiza el trabajo principal del bucle en cada iteración. Puede tener efectos laterales.

Para permitir la ejecución de bucles sin efectos laterales utilizando un enfoque de flujo de datos, un bucle también puede definir y actualizar un conjunto de variables de repetición:

variables de bucle	Conjunto de pines en el bucle que tienen valores que son computados inicialmente, disponibles dentro del cuerpo de prueba y del cuerpo del bucle, y que se actualizan después de cada iteración del cuerpo del bucle.

entradas de bucle	Conjunto de valores de fuera del bucle utilizados para inicializar las variables del bucle.
actualizaciones de bucle	Conjunto de resultados del cuerpo del bucle que se utilizan para actualizar los valores de las variables del bucle para la siguiente iteración. Dentro de una iteración, las variables del bucle no cambian sus valores. Sus nuevos valores se calculan en la actualización de los valores del bucle pero no se colocan en las variables del bucle hasta la finalización del cuerpo del bucle.
resultados	Conjunto de valores del cuerpo del bucle que representan los valores de salida de todo el bucle. Pueden incluir variables del bucle. Si la prueba se realiza antes de la primera iteración (es decir, si el bucle pudiese tener cero iteraciones), los valores de salida sólo pueden venir de las variables del bucle, que son los únicos valores del bucle que inicialmente tienen valores.

Notación

No hay una notación gráfica oficial para un nodo de repetición. La impresión de los desarrolladores de UML era que esta construcción en la mayoría de los casos se expresa mejor de manera textual. Indudablemente se pueden proponer diversas notaciones, y finalmente una de ellas podría ser adoptada. Hasta entonces, se puede colocar una descripción textual en un símbolo de actividad (rectángulo con esquinas redondeadas).

nombre

Una cadena usada para identificar a un elemento del modelo.

Véase también el espacio de nombres.

Semántica

Un nombre es un identificador —secuencia de caracteres de un finito, alfabeto predefinido en algún lenguaje. La implementación puede imponer restricciones respecto a la forma de los nombres, como la exclusión de ciertos caracteres (por ejemplo, signos de puntuación), restricciones en caracteres iniciales, y así sucesivamente. En particular, es de suponer que los nombres son utilizables como seleccionadores y claves de las búsquedas dentro de los varios conjuntos de datos.

Por ejemplo, los nombres del alfabeto romano normalmente incluyen mayúsculas y minúsculas; números; y uno o más separadores, como subrayado y guión, mientras que otros signos de puntuación son dependientes de la implementación.

Las herramientas y lenguajes pueden imponer límites razonables en la longitud de las cadenas y el conjunto de caracteres usado para los nombres, posiblemente más restrictivo que aquéllos para cadenas arbitrarias, como comentarios.

Se definen nombres dentro de un espacio de nombres, como un paquete o clase. Dentro de un espacio de nombres, un nombre debe ser único dentro de su propio grupo semántico, como cla-

sificadores estados, atributos, y así sucesivamente, pero los nombres de grupos diferentes pueden coincidir (aunque esto debe evitarse para impedir la confusión). Cada espacio de nombres, excepto el sistema completo, se contiene dentro de otro espacio de nombres. Los nombres de todos los espacios de nombres anidados y el último elemento pueden componerse en una única cadena calificada.

Observe que la ausencia de un nombre no es equivalente a una cadena vacía y a veces implica un elemento anónimo. Dos elementos sin nombre no son necesariamente el mismo elemento.

nombre calificado

Una cadena compuesta encadenando los nombres de espacios de nombres que contienen un elemento, empezando desde el espacio de nombres anónimo implícito que contiene el sistema entero y acabando con el nombre del propio elemento.

Semántica

Un nombre calificado identifica un elemento ejemplar singularmente, como un atributo o un estado, dentro de un sistema y puede usarse dentro de una expresión para hacer referencia a un elemento.

No todas las clases de elementos tienen nombre.

Notación

Un nombre calificado se despliega como una lista de espacio de nombres anidados y nombres del elemento separados por dos puntos dobles (::). Un espacio de nombres es un paquete o un elemento con anidadas declaraciones. Por ejemplo:

```
Accounting::Personnel::Employee::address
```

Un nombre calificado referencia un elemento en el paquete nombrado por el prefijo del camino.

nombre de clase

Cada clase (o, en general, clasificador) debe tener un nombre no nulo que es único entre los clasificadores dentro de su contenedor (como un paquete o una clase). El ámbito de un nombre es el paquete que lo contiene y otros paquetes que pueden ver al paquete que lo contiene.

Véase nombre para una discusión completa sobre el nombrado y las reglas de unicidad.

Notación

Un nombre de clase se muestra en el compartimento superior del rectángulo que representa la clase. El compartimento del nombre también puede contener una palabra clave o un nombre de estereotipo (Figura 14.204).

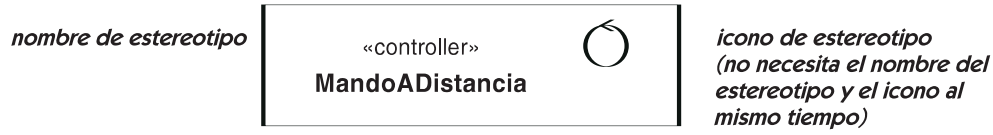


Figura 14.204 Compartimento de nombre

Se puede colocar una palabra clave opcional de estereotipo sobre el nombre de la clase dentro de comillas o ángulos dobles («»), y/o se puede colocar un icono de estereotipo en la esquina superior derecha del compartimento. El nombre de estereotipo no se debe corresponder con una palabra clave predefinida, como **enumeration**.

El nombre de la clase aparece a continuación. El nombre de la clase se centra horizontalmente en negrita. Si la clase es abstracta, el nombre aparece en cursiva, aunque cualquier especificación explícita del estado de generalización entre llaves (como **{abstract}** o **{concrete}**) tiene precedencia sobre la fuente empleada para el nombre.

Por defecto, se asume que una clase que se muestra dentro de un paquete se define dentro de ese paquete. Para mostrar una referencia a una clase definida en otro paquete utilice la sintaxis:

```
Nombre-paquete::Nombre-clase
```

como cadena de nombre en el compartimento de nombre (Figura 14.205). Se puede especificar un nombre completamente cualificado encadenando los nombres de paquetes separados por una pareja de dos puntos (::). Se puede utilizar el mismo nombre de clase para diferentes clases en diferentes paquetes, proporcionando nombres cualificados para distinguirlas, pero esta duplicidad en los nombres puede llevar a error y debe ser utilizada con cuidado.

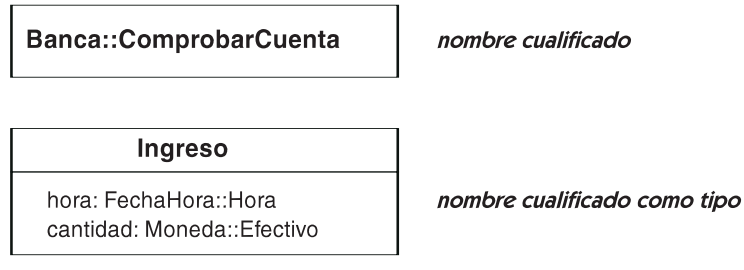


Figura 14.205 Nombres cualificados para clases de otros paquetes

Las referencias a clases también aparecen en expresiones de texto, con mayor incidencia en la especificación de tipos para los atributos variables y las variables. En expresiones de texto, una referencia a una clase se indica simplemente utilizando el nombre de la propia clase, incluyendo un posible nombre de paquete, sujeto a las reglas sintácticas de la expresión.

nombre del rol

Un nombre para un extremo de una asociación particular dentro de una asociación.

El término *nombre de rol* de UML1 realmente no se usa en UML2, pero como el término oficial *asociación y nombre* es algo pesado, continuaremos usando el término UML1 por conveniencia, como tantos usuarios lo harán indudablemente.

Semántica

Un nombre de rol proporciona un nombre para identificar un extremo de la asociación dentro de una asociación así como navegar de un objeto a otro usando la asociación. Ya que un nombre de rol puede usarse de estas dos maneras complementarias, el nombre debe ser único en dos espacios de nombres simultáneamente.

Todos los nombres de rol en una asociación deben ser diferentes. Dentro de una misma-asociación (una asociación que envuelve la misma clase más de una vez), los nombres de rol son necesarios para eliminar la ambigüedad de fines unidos a la misma clase. Si hay múltiples asociaciones sin nombre entre un par de clases, los nombres de rol se necesitan para distinguir las asociaciones y se necesitan en cualquier caso para la navegación. Por otra parte, el nombre de rol es optativo, porque los nombres de la clase pueden usarse para eliminar la ambigüedad de fines. Sin embargo son necesarios para escribir las expresiones de navegación en lenguajes tales como OCL.

Un nombre de rol es también usado para navegar de un objeto a los objetos relacionados vecinos. Cada clase "ve" las asociaciones unidas a ella y puede usarlas para encontrar los objetos relacionados a uno de sus casos o instancias. Por convención, el nombre de rol en el fin de la asociación unido a una clase vecina se usa para formar una expresión de navegación para acceder al objeto o colección de objetos relacionados por esa asociación. En la Figura 14.206 considere la clase **B** que está asociada a la clase **A** por una asociación *uno-a-muchos* y a la clase **C** por una asociación *un-a-uno*. Dado un objeto **bb** de clase **B**, la expresión **bb.elA** rinde un conjunto de objetos de clase **A** y la expresión **bb.elC** rinde un objeto de clase **C**. En efecto, el nombre de rol en el lado lejano de una asociación navegable es un atributo de una clase —que puede usarse como un término en una expresión de acceso atravesando la asociación.

Porque un nombre de rol puede usarse como un nombre de atributo para extraer valores, un nombre de rol introduce el espacio de nombres de la clase en el lado lejano de la asociación.

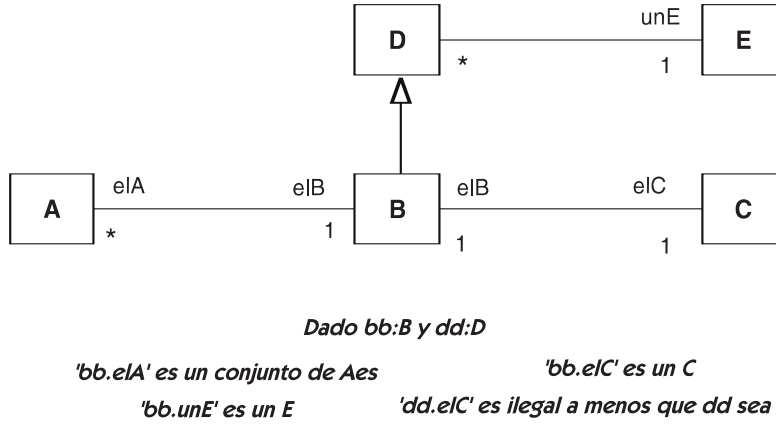


Figura 14.206 Navegación sobre asociaciones

Entra el mismo espacio de nombres como nombres del atributo. Ambos, nombres de atributo y nombres de rol deben ser únicos dentro de ese espacio de nombres. Se heredan atributos y nombres de rol de la asociación, y el nombre de atributo y los nombres del pseudoatributo deben ser únicos entre nombres heredados también. Un nombre de rol enlazado a una clase del antepasado puede usarse para navegación en un descendiente. En la Figura 14.206, la expresión **bb.unE** es legítima, porque la clase **B** hereda en **unE** el nombre de rol de la clase **D**.

Nombres de rol y nombres de la asociación son optativos si cada asociación puede ser singularmente identificada. Un nombre de asociación o los nombres de rol en sus fines pueden identificar una asociación. No es necesario tener ambos, aunque se permite. Si es la única asociación entre dos clases, ambos el nombre de la asociación y los nombres de rol, pueden omitirse. En principio, un nombre de rol se requiere para una expresión de navegación. En la práctica, una herramienta puede proveer una regla predefinida para crear un nombre implícito de rol de los nombres de las clases asociadas.

Notación

Un nombre de rol se muestra mediante una cadena gráfica puesta cerca del fin de un camino de asociación en que se encuentra una caja de la clase. Si existe, el nombre de rol no puede suprimirse.

El nombre de rol pueden llevar un marcador de visibilidad —an arrowhead— que indica si el elemento al extremo lejano de la asociación puede ver el elemento enlazado al nombre de rol.

nombre de ruta

Véase nombre cualificado.

nota

Un símbolo para desplegar un comentario u otra información textual, tal como el cuerpo de un método o una restricción.

Notación

Una nota es un rectángulo con su esquina superior-derecha plegada hacia el frente. Contiene texto o el texto extendido (como un documento incrustado) que no se interpreta por UML. Una nota puede presentar la información de los varios tipos de elementos del modelo, tales como un comentario, una restricción o un método. La nota no indica normalmente de forma explícita el tipo de elemento que representa, pero normalmente es evidente por su forma y uso. Dentro de una herramienta de modelado, el elemento subyacente será explícito en el modelo. Una nota puede ligarse al elemento que describe con una línea punteada. Si la nota describe elementos múltiples, cada uno de ellos estará ligado con una línea.

Una nota puede tener una palabra clave entre llaves para clarificar su significado. La palabra clave «**constraint**» indica una restricción.

Ejemplo

La Figura 14.207 muestra notas usadas para varios propósitos, una restricción de funcionamiento, una restricción en una clase y un comentario.

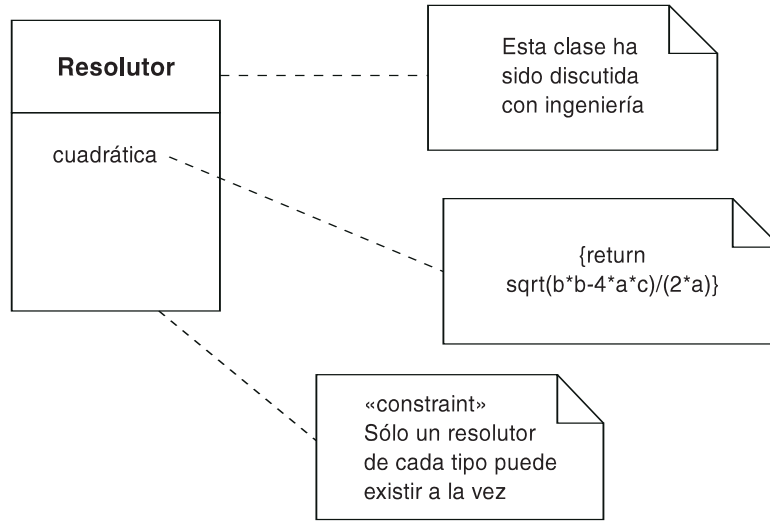


Figura 14.207 Notas

notación canónica

UML define una notación canónica que utiliza dibujos de líneas monocromáticas y texto para mostrar cualquier modelo. Éste es el "formato de publicación" estándar de los modelos de UML, aplicable a los diagramas impresos.

Las herramientas de edición gráfica pueden extender la notación canónica a conveniencia y proporcionar capacidades de interacción. Por ejemplo, una herramienta podría proporcionar la capacidad de resaltar los elementos seleccionados en la pantalla. Otras capacidades de interacción incluyen la navegación dentro del modelo y el filtrado del modelo que se muestra de acuerdo a las propiedades seleccionadas. Este tipo de formato es efímero y no es obligatorio en UML. En una visualización interactiva, existe poco riesgo de ambigüedad, ya que el usuario puede simplemente pedir una aclaración. Por lo tanto, el estándar de UML se centra en el formato canónico impreso, que debe ser soportado por cualquier herramienta, a sabiendas de que una herramienta interactiva puede y debería proporcionar extensiones interactivas.

notación tabular

La especificación de UML2 define una notación tabular para los diagramas de secuencia.

Discusión

El concepto de expresar información del modelo como tablas no es original en absoluto, y la forma concreta de hacerlo es preferible que la dé una herramienta de implementación. Su

inclusión en la especificación es muy vaga, ya que sólo está definido para un único diagrama UML.

nulo

La falta explícita de un valor, o bien porque no existe o bien porque no se muestra.

número de secuencia

Una parte del texto de una etiqueta del mensaje en un diagrama de comunicación que indica el orden de ejecución relativa de los mensajes en una interacción. Un número de sucesión puede mostrar la situación del mensaje dentro de una secuencia llamada anidada, el nombre de una traza de control y una especificación de condicional y la ejecución reiterativa.

Véase colaboración, mensaje.

Object Management Group

Véase OMG.

objeto

Una entidad discreta con un límite bien-definido e identidad que encapsula estado y comportamiento; una instancia de una clase.

Véase también clase, identidad, instancia, flujo de objeto.

Semántica

Un objeto es una instancia de una clase que describe el conjunto de posibles objetos que puede existir. Un objeto puede verse desde dos perspectivas relacionadas: como una entidad en un momento particular con un valor específico y como un poseedor de identidad que tiene valores diferentes con el tiempo. Ambas vistas pueden coexistir en un modelo, pero no en el mismo objeto o clase. La primera vista es apropiada para una instantánea que representa un sistema en un punto del tiempo. Una instantánea de un objeto tiene los valores de cada uno de sus atributos. Un objeto está ligado a una colección de enlaces que lo conectan a otros objetos.

Cada objeto tiene su propia identidad única y puede ser referenciado por un identificador único que proporciona acceso a él. La vista de un objeto como una identidad es apropiada para una instancia de la colaboración en la que el objeto tiene relaciones en tiempo de ejecución con otros objetos que usa para intercambiar instancias de mensajes.

Un objeto contiene una sección para cada atributo en su descriptor completo —es decir, para cada atributo declarado en su clase directa y en cada clase antecesora. Cuando la instanciación e inicialización de un objeto se completan, cada sección contiene un valor que es una instancia del clasificador declarado como el tipo del atributo. Cuando el sistema se ejecuta, el valor de una

sección del atributo puede cambiar, a menos que la propiedad de mutabilidad del atributo le prohiba que cambie. En todo momento entre la ejecución de varias operaciones, los valores en un objeto deben satisfacer las restricciones tanto implícitas como explícitas impuestas por el modelo. Durante la ejecución de una operación, las restricciones pueden violarse de manera temporal.

Si en un ambiente de ejecución está permitida la clasificación múltiple, entonces un objeto puede ser instancia directa de más de una clase. El objeto contiene una sección para cada atributo declarado en cualquiera de sus clases directas o de sus antepasados. El mismo atributo no puede aparecer más de una vez, pero si dos antecesores directos son descendientes de un antepasado común, sólo una copia de cada atributo del antepasado se hereda, sin tener en cuenta los múltiples caminos a él.

Si se permite la clasificación dinámica, un objeto puede cambiar su clase directa durante la ejecución. Si se ganan atributos en el proceso, entonces sus valores deben ser especificados por la operación que cambia la clase directa.

Si se permiten la clasificación múltiple y la clasificación dinámica, entonces un objeto puede ganar y puede perder sus clases directas durante la ejecución. Sin embargo, el número de las clases directas nunca puede ser menor a uno (debe tener alguna estructura, aun cuando sea transitoria).

Se puede llamar a un objeto para ejecutar cualquier operación que aparezca en el descriptor completo de cualquier clase. Esto es, tiene tanto las operaciones directas como las heredadas.

Un objeto puede usarse como el valor de cualquier variable o parámetro que declare como tipo el mismo que la clase o un antepasado de la clase directa del objeto. En otras palabras, una instancia de cualquier descendiente de una clase puede aparecer como el valor de una variable cuyo tipo se declara de esa clase. Éste es el principio de sustitución. Este principio no es una necesidad lógica pero existe para simplificar la implementación de lenguajes de programación.

Notación

Diagramas del objeto

Un diagrama de objetos muestra una configuración de objetos. Un objeto es una instancia de una clase. Un objeto se diseña como una especificación de una instancia que puede representar a un objeto aislado o un conjunto de objetos que satisfacen las condiciones dadas. La regla general para la notación de instancias por ejemplo es usar el mismo símbolo geométrico como el descriptor pero subrayando el nombre de la especificación de la instancia para distinguirlo como un individuo. La especificación de la instancia puede incluir valores o restricciones para los atributos, pero las propiedades compartidas por todas las instancias sólo se transcriben en el descriptor.

Por ejemplo, las operaciones sólo aparecen en la clase; no hay ninguna necesidad de mostrarlos para cada objeto de la clase, porque todos los objetos comparten las mismas operaciones.

La notación canónica para una especificación de objeto es un rectángulo con dos compartimientos.

El compartimiento de la cima contiene el nombre del objeto y la clase y el compartimiento inferior contiene una lista con los nombres de los atributos y los valores (Figura 14.208).

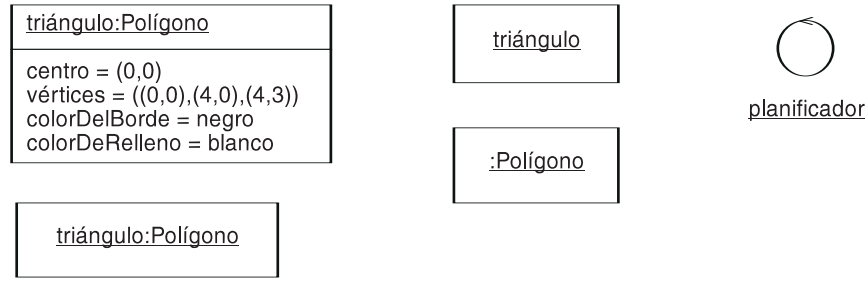


Figura 14.208 Notación de objetos

No hay ninguna necesidad de mostrar las operaciones porque son las mismas para todos los objetos de una clase.

El compartimiento superior muestra el nombre del objeto y su clase, todos subrayados usando la sintaxis

`nombreObjeto: nombreClase`

Los nombreClase pueden incluir el camino completo incluyendo el paquete si es necesario.

Los nombres del paquete preceden a el nombreClase y se separan por dos puntos. Por ejemplo

displayWindow: WindowingSystem::GraphicWindows::Window

Se puede mostrar un estereotipo para la clase de forma textual (entre llaves sobre la cadena del nombre) o como un icono en la esquina superior-derecha. El estereotipo para un objeto debe ajustarse al estereotipo de su clase.

Para mostrar las múltiples clases de las que el objeto es una instancia, se utiliza la coma como separador lista de nombres de clase. Algunas de las clases pueden ser los roles transitorios que el objeto toma durante una colaboración. Por ejemplo

unaPersona: Profesor, esquiador

El tipo de una especificación del objeto puede ser abstracto. Cualquier objeto instanciado debe tener una clase concreta que es un descendiente de la clase abstracta.

El segundo compartimiento muestra los atributos para el objeto y sus valores como una lista. Cada línea tiene la sintaxis

`nombreAtributo: tipo = valor`

El tipo es redundante con la declaración del atributo en la clase y puede omitirse.

El valor se especifica como una cadena que representa el valor. El nombre del atributo no se subraya.

El nombre del objeto puede omitirse. En este caso, los dos puntos deben conservarse junto con el nombre de la clase. Esto representa un objeto anónimo de la clase, que adquiere identidad

a partir de sus relaciones. Cada símbolo que contiene un objeto anónimo denota un objeto distinto, esta distinción se logra por sus relaciones con otros objetos.

La clase del objeto puede suprimirse (junto con los dos puntos), pero el nombre debe mostrarse cuando sea posible para evitar la confusión.

El compartimiento del valor de atributo puede suprimirse en conjunto.

Pueden suprimirse los atributos cuyos valores no son de interés.

El valor del atributo puede ser una restricción, en lugar de un solo valor. Esto significa que el valor del objeto debe satisfacer la restricción. Se puede ligar una restricción en un símbolo de nota al nombre del atributo mediante una línea punteada.

Diagramas de secuencia

En un diagrama de secuencia, una línea de la vida representa un objeto o un conjunto de objetos de un tipo. La línea de vida se muestra como una línea vertical o un rectángulo estrecho encabezado por un rectángulo conteniendo el nombre y tipo del objeto. El rectángulo del título tiene la misma sintaxis que el diagrama de objetos. *Véase* la línea de la vida para la sintaxis. La Figura 14.169 muestra un ejemplo.

Para mostrar la presencia de un objeto de una clase en un estado particular, se puede poner una restricción de estado en la línea de vida. Esto se muestra mediante el nombre del estado en un rectángulo redondeado pequeño. También se puede usar una lista de nombres de estados separada por comas. La restricción tiene efecto cuando se recibe el siguiente evento.

La Figura 14.51 muestra un ejemplo.

Para mostrar un cambio de clase (clasificación dinámica), debe haber por lo menos dos restricciones estatales en una línea de la vida, separadas por lo menos por un evento (con la recepción de un mensaje que cambia de estado).

También se puede poner una restricción en los valores de atributos. Esto se muestra como una cadena entre llaves junto a la línea de vida. La restricción debe validarse después de que se reciba el evento siguiente.

Diagramas de actividad

En un diagrama de actividad, puede mostrarse el flujo de un objeto como un valor o como un objeto de flujo. Esto tiene un énfasis diferente de los objetos en objeto o los diagramas de secuencias, en que los objetos son los objetos de interés. *Véase* flujo de objetos.

objeto activo

Un objeto que pertenece a un hilo de control y que puede iniciar una actividad de control; una instancia de una clase activa.

Véase también objeto pasivo.

Semántica

Un objeto activo no se ejecuta dentro de otro hilo de control, marco de pila o máquina de estados. Tiene un lugar independiente de control dentro de la ejecución global del sistema. En cier-

to sentido, *es* el hilo. Cada objeto activo es un lugar de ejecución distinto; los objetos activos no son reentrantes, y la ejecución recursiva no es posible sin la creación de objetos adicionales.

Un objeto activo es la raíz de una 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 y continúa hasta que la acción de la transición se ejecuta por completo y el objeto espera una entrada externa. Por tanto, un objeto activo no se ejecuta dentro del ámbito de otro objeto. Puede ser creado por la 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. Las operaciones sobre un objeto activo desencadenadas por otros objetos deberían implementarse en el objeto activo como eventos de llamada.

Un objeto pasivo se puede crear como parte de una acción de otro objeto. Tiene su propio espacio de direcciones. Un objeto activo no tiene hilo de control. Sus operaciones se pueden llamar dentro del marco de pila de un objeto activo. Sin embargo, se puede modelar como una máquina de estados para mostrar los cambios en su estado causados por la ejecución de operaciones sobre él.

Un proceso de un sistema operativo convencional es muy similar a un objeto activo. Un hilo de un sistema operativo se puede, o no, implementar mediante un objeto activo.

La distinción entre activo y pasivo es fundamentalmente una decisión de diseño y no restringe la semántica de los objetos. Tanto los objetos activos, como los pasivos, pueden tener máquinas de estados y pueden intercambiar eventos.

Notación

Un rol de colaboración para un objeto activo se muestra en un diagrama de colaboración mediante un rectángulo con líneas verticales dobles en el lado izquierdo y en el derecho. Es frecuente mostrar los objetos activos como compuestos con partes empotradas.

Un objeto activo también se muestra con el símbolo de un objeto con los bordes de línea doble, con el nombre subrayado, pero los objetos activos sólo aparecen dentro de ejemplos de ejecución y, por lo tanto, no son demasiado comunes.

También se puede utilizar la palabra clave {**active**} en las propiedades para indicar que un objeto es activo.

Ejemplo

La Figura 14.44 muestra tres objetos activos en un sistema de automatización de una fábrica: un robot, un horno y un gestor de fábrica que es un objeto de control. Los tres objetos existen y se ejecutan concurrentemente. El gestor de fábrica inicia el hilo de control en el paso 1, para dividirse después en dos hilos concurrentes de control (2a y 2b) que los ejecutan el horno y el robot respectivamente. Cuando cada uno ha finalizado su ejecución, los hilos se unen en el paso 3, en el gestor de fábrica. Cada objeto continúa vivo y conserva su estado hasta que le llega el siguiente evento.

Historia

La notación ha cambiado en UML2.

objeto compuesto

Instancia de una clase compuesta.

Véase composición, clasificador estructurado.

Semántica

Un objeto compuesto tiene una relación de composición con todas sus partes compuestas. Esto significa que es responsable de su creación y destrucción, y que ningún otro objeto es igualmente responsable. En otras palabras, no hay recolección de basura con las partes; el objeto compuesto puede y debe destruirlas cuando muere, o bien debe traspasar la responsabilidad de hacerlo a otro objeto.

La relación de composición se implementa a menudo por medio de contención física dentro de la misma estructura de datos que el propio objeto compuesto (habitualmente un registro). La contención física asegura que el tiempo de vida de las partes coincide con el tiempo de vida del objeto compuesto.

Notación

Se puede conectar un objeto compuesto a cada una de sus partes por medio de un enlace de composición, es decir, una línea con un diamante relleno al final, adjunto al objeto compuesto (Figura 14.209).

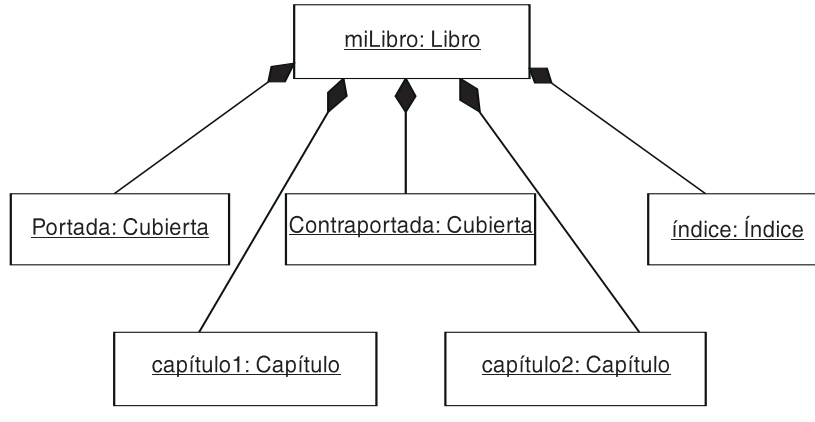


Figura 14.209 Objeto compuesto

Una instancia de una clase estructurada se representa como un rectángulo cuyos objetos internos y conectores pueden estar anidados dentro de un compartimento gráfico dentro del símbolo del objeto. El compartimento gráfico se puede representar como un compartimento adicional debajo del compartimento de atributos, aunque normalmente el compartimento de atributos se suprime.

Ejemplo

La Figura 14.210 muestra un objeto estructurado, la ventana de una computadora, compuesta de diversas partes. Contiene varias instancias de la clase **BarraDeDesplazamiento**. Cada instancia

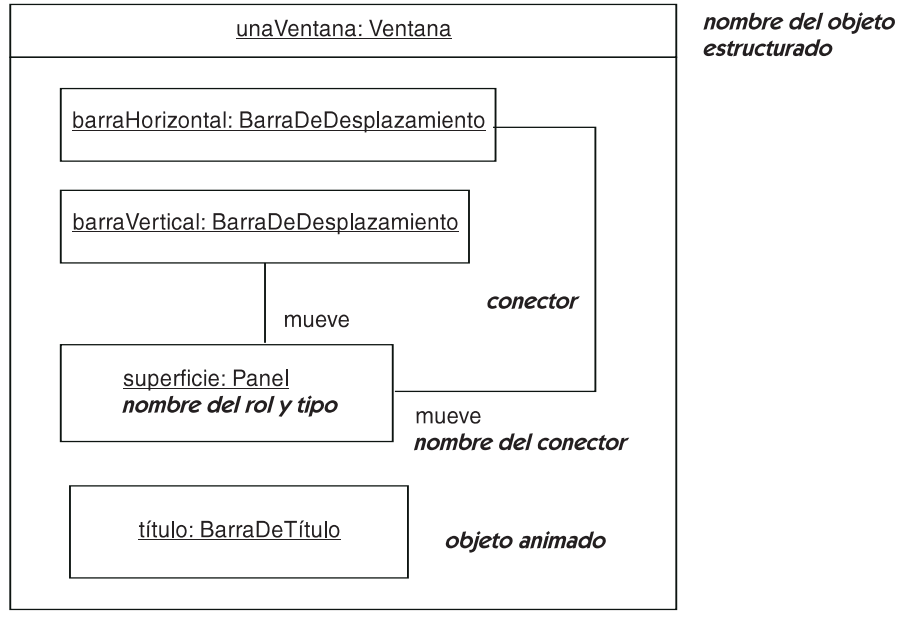


Figura 14.210 Objeto estructurado

tiene su propio nombre y rol dentro del objeto compuesto. Por ejemplo, tanto **barraHorizontal** como **barraVertical** son barras de desplazamiento, pero se comportan de forma diferente dentro de la composición. Se etiquetan por sus roles dentro del objeto estructurado.

objeto pasivo

Un objeto que no tiene su propio hilo de control. Sus operaciones ejecutan bajo un hilo de control fijado en un objeto activo.

Véase también clase activa, objeto activo.

Semántica

Un objeto activo es uno que posee un hilo de control y puede comenzar la actividad del control.

Un objeto pasivo es uno que tiene un valor pero no comienza el control. Sin embargo una operación en un objeto pasivo puede enviar mensajes mientras se procesa una demanda que ha recibido de un hilo existente.

Una clase pasiva es una clase cuyas instancias son objetos pasivos. Una clase pasiva puede no declarar recepciones porque sus objetos no reciben señales.

Notación

Una clase activa se muestra duplicando los lados derecho e izquierdo del rectángulo de la clase. Una clase pasiva se muestra por la ausencia de duplicidad. Las mismas distinciones pueden mostrarse en símbolos del objeto.

En muchos casos, no se muestra la presencia de clases activas. No debe extraerse ninguna conclusión de los símbolos, a menos que se muestre por lo menos alguna clase activa.

OCL

Lenguaje de Restricción de Objetos, un lenguaje para especificar restricciones y preguntas.

OCL no está pensado para escribir acciones o código ejecutable.

Semántica

El Object Constraint Language (OCL) es un lenguaje de texto para escribir las expresiones de navegación, expresiones Booleanas y otras consultas. Puede usarse para construir expresiones para restricciones, condiciones de guarda, acciones, precondiciones y postcondiciones, aserciones y otros tipos de expresiones de UML. El OCL ha sido definido por la OMG en una especificación hermana a UML. Se puede encontrar una descripción completa de la sintaxis de OCL y la semántica en la especificación en el sitio Web de OMG.

Se puede encontrar un resumen en [Caluroso-99]. El siguiente resumen selectivo contiene la sintaxis de OCL más útil para crear las expresiones de navegación y condiciones Booleanas. El lenguaje completo contiene un gran número de operaciones y operadores predefinidos, así como colecciones y tipos primitivos.

Notación

Abajo se muestra la sintaxis para algunas expresiones de navegación comunes. Estas formas pueden usarse encadenadas. El más a la izquierda debe ser una expresión para un objeto o una colección de objetos. Las expresiones están creadas para trabajar con colecciones de valores cuando sea posible. Para más detalles y sintaxis, vea la descripción de OCL.

`ítem.selector`
: el **`selector`** es el nombre de un atributo en el artículo o el nombre de un rol en el extremo destino de un enlace conectado al artículo. El resultado es el valor del atributo o de los objetos relacionados. El resultado es un valor o una colección de valores dependiendo de las multiplicidades del artículo y la asociación.

`ítem.selector` (**`lista de argumentos`**)
: el **`selector`** es el nombre de una operación en el artículo. El resultado es el valor de retorno de la operación aplicada al artículo.

`ítem.selector` [**`valor calificado`**]
: el **`selector`** designa una asociación calificada que califica el artículo. El **`valor-calificado`** es un valor para el atributo calificador. El resultado es el objeto relacionado o colección seleccionado por el calificador. Observe que esta sintaxis es aplicable para indexar arrays como una forma de calificación.

`colección ->` **`propiedad-de-la-colección`**
: **`propiedad-de-la-colección`** es el nombre de una función de OCL incorporada en las colecciones. El resultado es la propiedad de la co-

lección. Es ilegal si **`propiedad-de-la-colección`** no es una función predefinida de OCL. Algunas de las propiedades se enumeran a continuación.

`colección ->` select (**`expresión-booleana`**)
: la **`expresión-booleana`** se escribe en términos de objetos dentro de la colección. El resultado es el subconjunto de objetos en la colección para la que la expresión es verdadera.

`colección ->` size
: El número de elementos en la colección.

self
: Denota el objeto actual (puede omitirse si el contexto está claro).

`operador`
: La aritmética usual y los operadores Booleanos:

  = < > < = > = < > +. * / **not**

Ejemplo

Vuelo.piloto.horasEntrenamiento >= vuelo.avión.horas_Mínimas
`La colección de pilotos que tiene suficientes horas de entrenamiento.`

compañía.empleados->select (titulo = "Jefe" and self.reportes->size > 10)
`El número de jefes que tiene más de 10 informes.`

ocurrencia de la colaboración

Véase uso de la colaboración.

Discusión

El término era utilizado para uso de la colaboración, pero esta utilización confunde el significado de ocurrencia, que se suele utilizar habitualmente con el significado de suceso temporal que acostumbra a significar acontecimiento temporal. Para describir la aparición de una entidad estática en un contexto es preferible la palabra *uso*

ocurrencia de la interacción

Véase uso de la interacción.

Discusión

Se utilizaba este término, pero entra en conflicto con la utilización de ocurrencia con el significado de instancia de un evento. La palabra *uso* aparece en otros lugares para indicar la referencia a un elemento definido que será usada dentro de un contexto particular. El término *referencia de la interacción* debería funcionar bien igualmente.

ocurrencia de un evento

Véase ocurrencia, especificación del suceso.

Discusión

Este término se utilizaba tanto para ocurrencia como para suceso, que son diferentes. Se puede utilizar como sinónimo de *ocurrencia*, por lo que la palabra *evento* es redundante.

ocurrencia de una ejecución

Véase especificación de una ejecución.

Discusión

Se utilizaba el término ocurrencia de una ejecución, pero entraba en conflicto con la utilización de ocurrencia para referirse a la instancia de un evento. El término especificación de una ejecución es consistente con el uso de los términos especificación de objeto, especificación del suceso y especificación de valor para modelar grupos de entidades en tiempo de ejecución dentro de un contexto.

OMG

El Grupo de Gestión de Objetos, Inc., una corporación sin ánimo de lucro pensada para facilitar la especificación de tecnología del software a través de un consorcio abierto de compañías interesadas, universidades, asociaciones y otros participantes. La OMG es el dueño de las especificaciones de UML, MOF, OCL y otras tecnologías relacionadas. Puede encontrar más información sobre la OMG, las características técnicas de UML oficiales y otros trabajos en el sitio web *www.omg.org*.

opcional

Un fragmento combinado en una interacción sobre la que representa una opción dinámica sobre si ejecutar el operando no.

Semántica

Una estructura optativa tiene un operando con una condición de guarda. El operando se ejecuta si la guarda es verdad y no se ejecuta en otro caso. Esta estructura es equivalente a un condicional con una cláusula else vacía.

Notación

Un fragmento optativo se muestra como una región rectangular con la etiqueta **opt** dentro de un pentágono pequeño en la esquina izquierda superior. La región contiene opcionalmente el cuerpo del operando ejecutado (Figura 14.211).

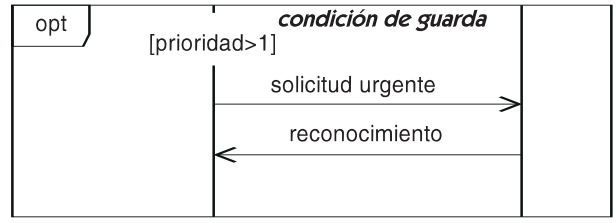


Figura 14.211 Fragmento opcional (opt)

operación

Una operación es una especificación de una transformación o consulta que un objeto puede llamar para ejecutar.

Véase también llamada, método, resolución.

Semántica

Una operación especifica una transformación en el estado del objeto destino (y posiblemente el estado del resto del sistema alcanzable del objeto designado) o una consulta que devuelve un valor a la visita de la operación. Tiene un nombre y una lista de parámetros incluyendo los parámetros del retorno. Puede imponer las restricciones en parámetros o el objeto destino antes de o después de la invocación.

Una operación es invocada por una llamada que suspende al llamador hasta que la ejecución de la operación está completa, después de que el llamador recupera su control más allá del punto de la llamada, recibiendo un valor de retorno si lo proporciona la operación. Una llamada puede ser síncrona o asíncrona. Sólo una llamada síncrona puede recibir un valor de retorno.

Una operación especifica el resultado de un comportamiento, no el propio comportamiento. Una llamada a una operación sufre un paso de resolución para escoger el comportamiento a ser invocado basado en el funcionamiento y características del objeto designado, como su tipo. El comportamiento puede ser un método, una transición de la máquina de estados, o algo más.

Un método es un procedimiento que lleva a cabo una operación. Tiene un algoritmo o descripción del procedimiento. Una llamada de la que se resuelve causa la ejecución del procedimiento.

Una transición de la máquina de estados puede tener una operación como su disparador. Una llamada que se resuelve causa el disparo de la transición.

Una operación declara en una clase. La declaración es heredada por los descendientes de la clase. Un funcionamiento puede redefinirse en una clase descendiente. La redefinición debe tener el mismo número de argumento y parámetros de resultado y sus tipos deben conformar (se permiten descendientes). Si dos operaciones que no están relacionadas por redefinición tienen las firmas concordantes (incluir conformando las firmas) las declaraciones chocan y el modelo se forma mal. Dos operaciones pueden tener el mismo nombre si sus firmas no coinciden. La definición de conformidad puede variar entre los sistemas.

La declaración de la operación, que es el antepasado común de todas las otras declaraciones, se llama *origen* (después de Bertrand Meyer). Representa la declaración gobernante de la operación que es heredada por los otros.

Una operación puede ser abstracta o concreta. Una operación concreta debe resolverse a un comportamiento (como método o disparador de la transición) en la propia clase. Una operación abstracta no necesita y usualmente no tendrá un comportamiento.

Estructura

Un funcionamiento tiene los siguientes elementos principales.

abstracto
: Una operación puede ser abstracta o concreta. Una operación concreta debe resolverse a un comportamiento. Una operación abstracta debe resolverse a comportamientos en descendientes concretos sólo.

concurrencia
: La semántica de llamadas coexistentes a la misma instancia pasiva, una enumeración. Los posibles valores son:

  secuencial
  : Los llamadores se deben coordinar para que sólo uno llame a el objeto (en cualquier funcionamiento secuencial), puede ejecutarse sólo una llamada a la vez. Si ocurren llamadas coexistentes, entonces la semántica y la integridad del sistema no pueden garantizarse.

  guardada
  : Las llamadas múltiples de los hilos coexistentes pueden ocurrir simultáneamente a un objeto (en cualquier operación guardada) pero sólo se permite comenzar a uno a la vez. Los otros se bloquean hasta que la ejecución de la primera operación esté completa. Es responsabilidad del modelador asegurar que no se produzcan abrazos mortales debido a los bloqueos simultáneos. Las operaciones guardadas deben realizarse correctamente (o se bloquea) en el caso de una ejecución secuencial simultánea o la semántica de guarda no puede exigirse.

  concurrente
  : Las múltiples llamadas de hilos concurrentes pueden ocurrir simultáneamente a un objeto (en operaciones concurrentes). Todos ellos pueden proceder concurrentemente con semántica correcta. Las operaciones coexistentes deben ser diseñadas para que realicen correctamente en el caso de una operación concurrente, secuencial o guardada en el mismo objeto. Por otra parte, la semántica concurrente no puede exigirse.

restricciones
: Las operaciones pueden tener precondiciones, postcondiciones y condiciones del cuerpo. Las precondiciones deben ser verdad cuando se llama a una operación, de lo contrario la llamada es errónea y la operación podría fallar. El llamador debe asegurar que se reúnen las precondiciones. Las postcondiciones se pueden afirmar como verdad cuando el comportamiento invocado completa su ejecución, de lo contrario la implementación del comportamiento estaría en

	error. El desarrollador debe asegurar que las postcondiciones se cumplen. Las condiciones del cuerpo son las postcondiciones en los valores de retorno. Las precondiciones y postcondiciones se heredan por subclase y no pueden sobrecargarse. Las condiciones del cuerpo son heredadas por las subclases pero pueden sobrecargarse en la subclase.
hoja	Si el funcionamiento puede ser redefinido por las clases descendientes. Si es verdadero, la aplicación no puede redefinirse. El valor predeterminado es falso.
excepciones	Una lista de excepciones que la operación puede levantar durante la ejecución.
consulta	Si la ejecución de la operación deja el estado del sistema sin cambio. Esto es, si es una consulta. Si verdadero, la operación devuelve un valor, pero no tiene ningún efecto de lado. Si falso, puede alterar el estado del sistema, pero no se requiere un cambio. El valor predeterminado es falso.
nombre	El nombre de la operación, una cadena. El nombre, junto con la lista de tipos de los parámetros (no incluyendo el nombre de los parámetros), se llama la firma concordante de la operación. La inclusión de los tipos del retorno varía entre las aplicaciones. La firma concordante debe ser única dentro de la clase y sus antecesores. Si hay una duplicación, debe ser una redefinición de la operación; por otra parte el modelo está mal formado.
lista de parámetros	La lista de declaraciones de los parámetros de la operación. *Véase* lista de parámetros.
resultados del retorno	Una lista de los tipos de los valores devueltos por una llamada de la operación, si los hay. Si la operación no devuelve los valores entonces esta propiedad tiene el valor nulo. Observe que muchos lenguajes no soportan los valores de retorno múltiples, pero prevalece un concepto de modelado válido que puede llevarse a cabo de varias maneras, como por ejemplo tratando uno o más de los parámetros como valores de salida.
estática	Si la operación se aplica a los objetos individuales (no estático) o a la propia clase (estática). Las operaciones estáticas se usan a menudo para constructores. El valor predeterminado es el no estático.
visibilidad	La visibilidad de la operación por otras clases diferente a la que la define. *Véase* visibilidad.

Un método es un procedimiento de comportamiento que lleva a cabo una operación. Un método se liga a una clase. La operación debe definirse en la misma clase o un antepasado de la clase que contiene el método. Los parámetros del comportamiento deben emparejar los parámetros de la operación que lleva a cabo. Un método puede tener precondiciones y postcondiciones. Puede debilitar las precondiciones o puede fortalecer las postcondiciones con respecto a la ope-

ración que lleva a cabo. Un disparador de la llamada es un disparador que habilita una transición basado en la recepción de una llamada.

El disparador de la llamada se define para la clase que posee la declaración de la máquina de estados. El disparador de la llamada hace disponible sus argumentos a cualquier comportamiento asociado a la transición. Cualquier parámetro de retorno de la operación es ignorado por el disparador de la llamada.

Un funcionamiento puede redefinirse. *Véase* redefinición (operación).

Notación

Un funcionamiento se muestra como una cadena de texto que puede analizarse para obtener las propiedades de la operación.

La sintaxis predefinida es:

⌊**«estereotipo»**⌋$_{opc}$ **visibilidad**$_{opc}$ **nombre** (**parametro-lista**) ⌊**: volver-tipo**⌋$_{opc}$
⌊**{ propiedad-cordón }**⌋$_{opc}$

El estereotipo, visibilidad, tipo de retorno y la cadena de propiedad son opcionales (junto con sus delimitadores). La lista de parámetros puede estar vacía. La Figura 14.212 muestra algunas operaciones típicas.

+mostrar(): Ubicación
+ocultar()
«constructor»+crear ()
-vinculaXWindow(xwin:Xwindow*)

Figura 14.212 Diagrama de la estructura interna

Nombre. Una cadena que es el nombre de la operación (no incluidos los parámetros).

Lista del parámetros. Una lista separada por comas de declaraciones de parámetros, cada uno comprende dirección, nombre y tipo. La lista completa se encierra entre paréntesis (incluyendo una lista vacía). *Véase* lista de parámetros y parámetro para más detalles.

Tipo de retorno. Una cadena que contiene una lista separada por comas de nombres de clasificadores (clases, tipos de datos o interfaces). La cadena del tipo sigue a los dos puntos (:) que siguen a la lista del parámetro de la operación. Si el funcionamiento no devuelve el valor, se omiten los dos puntos de la cadena del tipo de retorno (por ejemplo, void de C++). Algunos, pero no todos, los lenguajes de programación soportan valores de retorno múltiples.

Excepciones. Las excepciones potencialmente elevadas por la operación pueden mostrarse como una cadena de propiedad de la forma:

excepción **nombre**$_{lista}$x

Visibilidad. La visibilidad se muestra como una de las marcas de la puntuación '+', '#', '–' o '~' para representar público, protegido, privado o paquete. Alternadamente, la visibilidad puede ser mostrada como una palabra clave dentro de la cadena de propiedad (por ejemplo, {**visibilidad=privado**}).

Esta forma debe usarse para las opciones definidas por el usuario o las opciones dependientes del lenguaje.

Abstracto. Una operación abstracta se muestra poniendo el nombre en cursiva. Por otra parte la operación es concreta. Lo abstracto también puede ser indicado por la cadena propiedad **abstract**.

Consulta. La opción se muestra por una cadena de propiedad de la forma **Query=true** o **Query=false**. La opción verdadera también puede ser mostrada por la consulta de la palabra clave. La ausencia de una opción explícita indica lo **falso**. Esto es, la operación puede que altere el estado del sistema (pero no se garantiza que lo altere).

Hoja. La opción se muestra por una cadena propiedad de la forma **isPolymorphic=true** (sobrecargable) o **isPolymorphic=false** (no es sobrecargable). La ausencia de una opción explícita indica **verdadero**. Esta es, sobrecargable. La cadena **isPolymorphic** aislada indica un valor verdadero.

Estática. Una operación de ámbito de instancia (no estática) se indica no subrayando la cadena de la operación. Una operación de ámbito de clase (estática) se indica subrayando la cadena del nombre. Una función estática se puede indicar de manera alternativa por una cadena de propiedad **static**.

Concurrencia. La opción es una cadena de propiedad de la forma **concurrency=`valor`** donde **`valor`** es una de las palabras claves **sequential**, **guarded** o **concurrent**.

Restricciones. Una restricción se puede mostrar como una cadena de texto entre llaves dentro de una nota. El símbolo de la nota se conecta a la operación por una línea punteada. Las palabras claves "**precondition**", "**postcondition**", y los "**bodyCondition**" pueden usarse para distinguir el tipo de restricción.

Método. No hay ninguna anotación especificada para indicar la presencia de un método. Se puede usar la convención siguiente para indicar la presencia de métodos: Una declaración de una operación concreta necesariamente implica la presencia de un comportamiento para llevar a cabo el comportamiento. Una declaración de un funcionamiento abstracto no implica el comportamiento.

Para mostrar la presencia de un comportamiento concreto en una subclase, redefina la operación en la subclase como una operación concreta.

Cuerpo del método. El cuerpo de un método puede mostrarse como una cadena dentro de una nota atada a una declaración del funcionamiento. La palabra clave "**method**" puede usarse para indicar una declaración de método.

Señales. Para indicar que una clase acepta una señal (una recepción), la palabra clave "**signal**" se posiciona al frente de la declaración de la operación dentro de la lista de operaciones. Los parámetros son los atributos de la señal. La declaración no debe tener un tipo de retorno. La contestación del objeto a la recepción de las señales se muestra dentro de una máquina de estados asociada a una clase.

Redefinición. *Véase* redefinición (funcionamiento).

Opciones de la presentación

Pueden suprimirse (juntos, no separadamente) la lista del argumento y tipo del retorno.

Una herramienta puede mostrar la indicación de visibilidad de una manera diferente, como usando un icono especial u ordenando los elementos por grupo.

La sintaxis de la cadena de la signatura de una operación se puede especificar en un lenguaje de programación concreto, como C++ o Smalltalk. Pueden ser incluidas propiedades etiquetadas específicas en la cadena.

Pautas de estilo

El funcionamiento empieza la carta con un escriba en letras minúsculas.

operación abstracta

Una operación que carece de una implementación —es decir, una que tiene la especificación, pero no el método. Alguna de sus clases descendientes concretas debe proporcionarle una implementación.

Véase abstracto/a, elemento generalizable, herencia y polimórfico/a.

Semántica

Si una operación se declara como abstracta en una clase, carece de implementación (un método o un disparador) en la clase, y la propia clase es, necesariamente, abstracta. Un descendiente concreto debe proporcionar una implementación para la operación. Si la clase hereda una implementación para la operación, pero declara la operación como abstracta, la declaración de abstracción invalida la implementación heredada por la clase. Si una operación se declara como concreta en una clase, entonces la clase debe proporcionar una implementación (un método o un disparador) o heredarla de un antecesor. Si una operación no se declara en una clase, entonces hereda la declaración de la operación y la implementación (o la carencia de implementación) de sus antecesores.

Una operación se puede implementar como un método o como un disparador de una máquina de estados. Cada clase puede declarar su propia implementación para una operación o heredar una definición de sus antecesores.

Notación

El nombre de la clase abstracta se muestra en cursiva (Figura 14.213). De manera alternativa se puede colocar la palabra clave **abstract** en la lista de propiedades después de la signatura de la operación.

Discusión

El uso más importante del concepto de herencia es permitir que las operaciones abstractas puedan ser implementadas de forma distinta para cada descendiente concreto de una clase. Una operación abstracta permite al que realiza la llamada invocar una operación sin saber exactamente

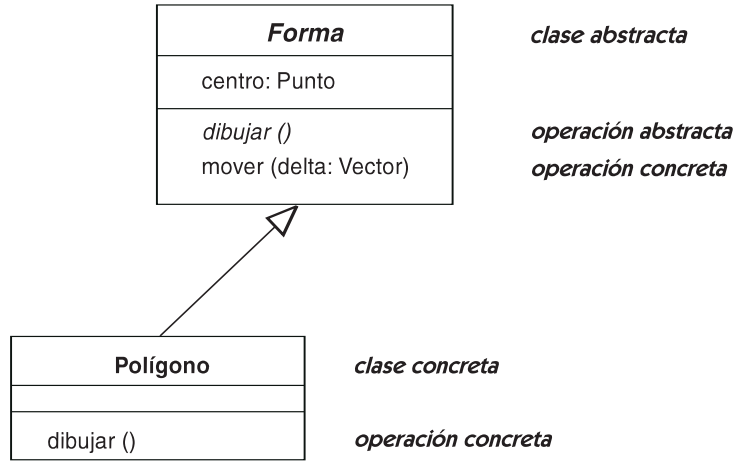


Figura 14.213 Clases y operaciones abstractas

cuál es la clase de objeto destino, siempre que el objeto destino proporcione la operación al ser una instancia indirecta de la clase abstracta que tiene la declaración de la operación abstracta. El significado de tal operación polimórfica es que la responsabilidad de determinar el tipo de objeto se ha desplazado del que realiza la llamada al mecanismo de herencia. No sólo se libera al que realiza la llamada de la molestia y el coste de escribir múltiples sentencias de selección, sino que, además, no necesita estar al tanto de que posibles subclases existen de una clase abstracta. Esto significa que las subclases adicionales se pueden añadir posteriormente con nuevas implementaciones para la operación. De ese modo, las operaciones abstractas, el polimorfismo y la herencia facilitan la actualización de los sistemas cuando se añaden nuevos tipos de objetos y comportamientos, sin tener que modificar el código que invoca al comportamiento genérico. Esto reduce enormemente el tiempo necesario para actualizar un sistema y, lo que es más importante, reduce la posibilidad de inconsistencias accidentales.

operación de clase

Una operación cuyo acceso está vinculado a una clase, en lugar de a una instancia de una clase.

Véase también característica estática.

Semántica

Una operación de clase es una operación global en el espacio de nombres de una clase. Se puede invocar sin una instancia de la clase —sus métodos tienen acceso a las características de la clase.

Las operaciones de clase se utilizan normalmente para construir nuevas instancias de una clase. Algunas veces también son utilizadas para acceder a los atributos de clase de una clase.

Una operación de clase se modela como una operación para la cual el indicador **isStatic** es verdadero.

Notación

Una operación de clase se muestra en la sección de operaciones de una clase utilizando la sintaxis de una clase normal con la cadena completa de la operación subrayada. *Véase* la Figura 14.29.

operando

Uno de los subfragmentos anidados de un fragmento combinado en una actividad. Cada fragmento combinado tiene una configuración específica de operandos.

Véase fragmento combinado.

operando de la interacción

Pieza estructural de un fragmento combinado; un subfragmento.

Semántica

Un fragmento combinado es una construcción estructurada de interacción. Cada fragmento comprende uno o más operandos de interacción, cada uno de ellos un subfragmento de dicha interacción. El número de operandos depende del tipo del fragmento combinado. Por ejemplo, un bucle tiene un operando (el cuerpo del bucle) y una condicional tiene uno o más operandos (las ramas de la condicional). Un operando es un fragmento anidado de una interacción. Cada operando abarca las líneas de vida cubiertas por el fragmento combinado o subconjunto de ellas.

Notación

Un fragmento combinado se representa como un rectángulo dividido en varios subfragmentos mediante líneas horizontales. En la mayoría de los casos la ordenación de los subfragmentos en el diagrama no implica ni secuencia temporal ni ordenación de prueba.

Véase fragmento combinado.

opt

Palabra clave para un fragmento combinado optativo en una interacción.

ordenación

Una propiedad de un conjunto de valores, tales como el conjunto de objetos relacionados con un objeto por una asociación, declarando si el conjunto se pide o no ordenado.

Véase también asociación, fin de la asociación, multiplicidad.

Semántica

Si la multiplicidad límite superior en un atributo o el extremo de la asociación es mayor que uno, entonces un conjunto de objetos está asociado con la sección del atributo o el extremo de la asociación.

La propiedad de la clasificación declara si el conjunto se pide o no ordenado. Si es no ordenado, los objetos en el conjunto no tienen ningún orden explícito; forman un conjunto ordinario. Si se pide, los elementos en el conjunto tienen un orden explícitamente impuesto. El orden de los elementos es parte de la información representada por la asociación —esto es, es información adicional más allá de la información en los propios elementos. Los elementos pueden obtenerse en ese orden. Cuando un nuevo eslabón se agrega a la asociación, su posición en la sucesión debe ser especificada por la operación que lo agrega. La posición puede ser un argumento de la operación o puede ser implícito. Por ejemplo, una operación dada puede poner un nuevo eslabón al final de la lista existente de eslabones, pero la situación del nuevo eslabón debe especificarse de algún modo.

Observe que un conjunto pedido no es igual que un conjunto cuyos elementos se ordenan por uno o más atributos de los elementos. Un ordenamiento es totalmente determinado por los valores de los objetos en el conjunto. Por consiguiente, no agrega la información, aunque puede ciertamente ser útil para los propósitos de acceso. La información en una asociación pedida por otro lado, es adicional a la información en los propios elementos.

Una relación pedida puede llevarse a cabo de varias maneras, pero la aplicación normalmente se declara como una propiedad de generación de código del lenguaje-especificado. Una extensión de aplicación podría sustituir los datos que estructuran para sostener los elementos para la especificación genérica **ordenada**.

Un conjunto ordenado requiere una especificación separada de la propia regla de ordenación, la cual es mejor dada como una restricción. Un conjunto clasificado no debe declararse como ordenado.

Notación

Una ordenación es especificada por una palabra clave entre llaves cerca del extremo de la asociación a la cual se aplica (Figura 14.214). La ausencia de una palabra clave indica no ordenado. La palabra clave {**ordered**} indica un conjunto ordenado. Para los propósitos del diseño, la palabra clave {**sorted**} puede usarse para indicar un conjunto colocado por sus valores internos.

Figura 14.214 Conjuntos ordenados y desordenados

Para un atributo con multiplicidad mayor que uno, una de las palabras claves de la clasificación puede ponerse después de la cadena del atributo, entre llaves, como parte de una cadena de propiedad.

Si un conjunto es ambos, ordenado y puede tener valores duplicados (una bolsa), la palabra clave **list** o **sequence** pueden usarse.

Si la palabra clave de la clasificación se omite, entonces el conjunto es no ordenado.

Discusión

Un conjunto ordenado tiene la información en la clasificación, información que le es adicional a las entidades en el propio conjunto. Ésta es la información real. Por consiguiente, es no derivable pero debe especificarse cuando una entidad se agrega. En otros términos, en cualquier operación que agrega una entidad, su posición dentro de la lista de entidades debe especificarse. Por supuesto, puede llevarse a cabo una operación para que la nueva entidad se inserte en una situación implícita, como el principio o el fin de la lista. Y simplemente porque un conjunto es ordenado no significa que cualquier clasificación de entidades se permitirá. Éstas son decisiones que los diseñadores deben tomar. En general, la posición de la nueva entidad dentro de la lista es un parámetro de una operación de creación.

Observe que la clasificación de una asociación binaria debe ser especificada independientemente para cada dirección. La ordenación carece de sentido a menos que la multiplicidad en una dirección sea mayor que uno. Una asociación puede ser completamente desordenada, puede ser ordenada en una dirección y no en otra, o puede ser ordenada en ambas direcciones.

Asuma una asociación entre las clases UN y B que está ordenada en la dirección de B.

Normalmente, entonces un nuevo eslabón se añadirá como una operación en un objeto UN, especificando un objeto de B y una posición en la lista de objetos de B existentes para el nuevo eslabón.

Frecuentemente, una operación en un objeto UN crea un nuevo objeto B y también crea un *únase* entre UN y B. La lista debe agregarse a la lista de eslabones mantenida por A. Es posible crear un nuevo eslabón a partir del lado de B, pero generalmente el nuevo eslabón se inserta en una posición predefinida en la lista A-to-B , porque la posición dentro de esa lista tiene el significado pequeño del fin de B. Claro, un programador puede llevar a cabo más situaciones complicadas si lo necesita.

Una asociación ordenada en ambas direcciones es algo rara, porque puede ser poco práctica para especificar el punto de la inserción en ambas direcciones. Pero es posible sobre todo si los nuevos eslabones son incluidos a las situaciones predefinidas en cualquier dirección.

Observe que un conjunto ordenado no contiene información extra más allá de la información en el conjunto de entidades. La ordenación ahorra tiempo en un algoritmo, pero no agrega información. Puede considerarse como una optimización del diseño y no hay necesidad de incluirla en el análisis del modelo. Puede especificarse como un valor de la propiedad de la clasificación pero no requiere que una operación especifique una situación para una nueva entidad agregada al conjunto. La situación de la nueva entidad debe determinarse automáticamente por el método de examinar los atributos en los que la lista se ordena.

ordenación general

Restricción de que el tiempo de una especificación de suceso precede al tiempo de otra especificación de suceso en una interacción.

Semántica

Normalmente las especificaciones de suceso en las interacciones se ordenan por su posición relativa en las líneas de vida y por la ordenación de las restricciones implicadas por mensajes (el evento origen de un mensaje ocurre antes que el evento destino). Las especificaciones de sucesos que no están restringidas por líneas de vida y mensajes (incluyendo restricciones indirectas de varios eventos) se asume que son concurrentes. A veces un modelador quiere especificar una restricción de ordenación entre dos especificaciones de suceso que de otra forma no tendrían orden. Una ordenación general es una restricción de que una especificación de suceso precede a la otra.

Notación

Una ordenación general se muestra como una línea punteada entre dos especificaciones de suceso (normalmente las intersecciones entre los mensajes y las líneas de vida). La línea contiene una cabeza de flecha sólida en algún lugar a lo largo de su camino (no al final). La flecha apunta hacia la especificación de suceso que ocurre más tarde.

Ejemplo

La Figura 14.215 muestra un ejemplo en el que dos usuarios intercambian mensajes con un servidor. En ausencia de una ordenación general, se pueden asumir las siguientes restricciones de ordenación (y sus cierres):

a < b, c < d, b < d, d < e, e < f, e < g, g < h

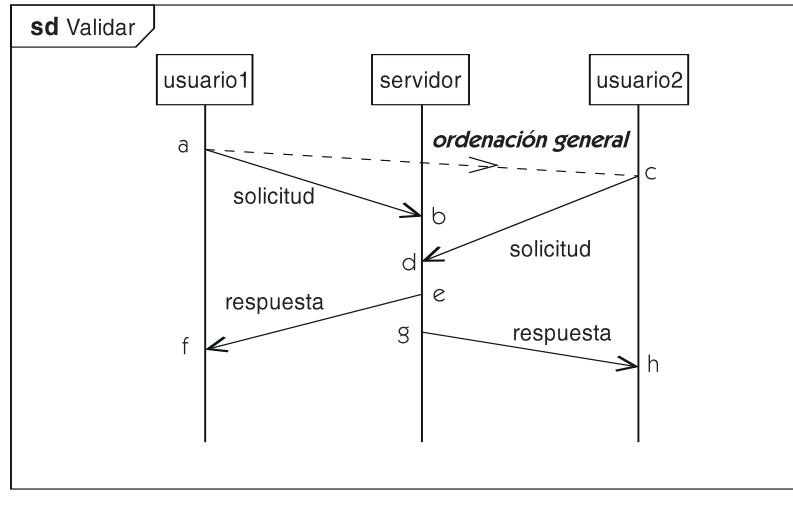


Figura 14.215 Ordenación general de especificaciones de suceso

Aplicando varias restricciones, podemos inferir que, por ejemplo, **c < f**. No es posible determinar la ordenación relativa de **a** y **c**, porque no están relacionadas por una cadena de restricciones. La adición de una ordenación general entre **a** y **c** especifica de forma explícita que **a < c**.

Sin embargo, la ordenación relativa de **b** y **c** todavía está sin especificar. De forma similar, la ordenación relativa de **f** y **g** está sin especificar.

Discusión

La ordenación explícita de restricciones debería evitarse siempre que sea posible. A menudo representan la ausencia (en el modelo o en el sistema real) de mensajes de sincronización necesarios.

otherwise

Véase else.

overlapping

Palabra clave que indica que los subtipos de un conjunto de generalización son compatibles; eso es, un objeto puede ser una instancia de más de uno simultáneamente.

Véase conjunto de generalización.

padre

El elemento más general en una relación de generalización. Superclase llamada por una clase. Una cadena de una o más relaciones de padre (es decir, el cierre transitivo) es un antecesor. El contrario es el hijo.

Véase generalización.

Discusión

Observe que hijo, padre, antecesor y descendiente no son términos oficiales de UML, pero nosotros los usamos en este libro por conveniencia porque UML no parece tener buenos términos simples que cubran todos los usos de los conceptos.

palabra clave

Adorno textual que categoriza un elemento del modelo que no tiene su propia sintaxis distintiva. No son una categoría semántica.

Véase también marcador gráfico, estereotipo.

Semántica

Las palabras clave no son una categoría semántica. Se utilizan para la notación de estereotipos, pero también se utilizan para la notación de otros elementos.

Notación

Las palabras clave se utilizan para elementos del modelo incorporados que no tienen una notación única, así como para los estereotipos definibles por el usuario. La notación general para la utilización de una palabra clave es englobarla entre comillas (« »).

```
«palabra-clave»
```

Cuando la palabra clave es parte de un símbolo de área, como un rectángulo de clase, la palabra clave se coloca dentro de los límites del símbolo.

Algunas palabras claves predefinidas son descritas en el texto de este documento y tratadas como palabras reservadas en la notación. Están disponibles otros nombres para que los usuarios los empleen como nombres de estereotipos. No se permite la utilización de un nombre de estereotipo que concuerde con una palabra clave predefinida.

Discusión

El número de símbolos visuales fácilmente distinguibles es limitado. La notación UML por tanto hace uso de palabras clave de texto para distinguir variaciones sobre un tema común, incluyendo subclases metamodelo de una clase base, estereotipos de una clase base metamodelo, y grupos de listas de elementos. Desde la perspectiva del usuario, la distinción del metamodelo entre subclases metamodelo y estereotipos a menudo no es importante, aunque sí lo es para los constructores de herramientas y otras personas que implementan el metamodelo.

paquete

Un mecanismo de uso general para organizar elementos en grupos, estableciendo propiedades de elementos, y proveyendo nombres únicos para hacer referencia a los elementos.

Véase también acceso, dependencia, importación, modelo, espacio de nombres, subsistema.

Semántica

Un *paquete* es una agrupación de elementos del modelo y diagramas. Cada elemento del modelo que no es parte de otro elemento del modelo, debe declararse dentro de exactamente un espacio de nombres; se dice que el espacio de nombres que contiene la declaración de un elemento posee el elemento. Un paquete es un espacio de nombres de uso general, que puede poseer cualquier clase de elemento del modelo, que no se restringe a un tipo particular de elemento. Un paquete puede contener paquetes anidados y los elementos del modelo ordinarios. Normalmente hay un paquete raíz que posee al modelo completo de un sistema.

Los paquetes son la base para el control de la configuración, almacenamiento y control de acceso.

Cada elemento o es poseído por otro elemento del modelo o por un solo paquete, para que la jerarquía de propiedad sea un árbol estricto. Los elementos del modelo (incluso los paquetes) pueden hacer referencia a otros elementos en otros paquetes, para que la red de uso sea un grafo dirigido en lugar de un árbol.

Un modelo es un tipo de paquete.

Los paquetes pueden tener las relaciones de dependencia con otros paquetes. En la mayoría de los casos, éstos sumarizan las dependencias entre los contenidos de los paquetes. Una dependencia de uso entre dos paquetes significa que allí existe al menos una dependencia de uso entre los elementos de los dos paquetes (no que cada par de elementos tiene la dependencia).

Un paquete es un espacio de nombres para sus elementos. Un elemento nombrado puede ser determinado únicamente por su nombre calificado, que es la serie de nombres de paquetes u otros espacios de nombres de raíz al elemento particular. Para evitar la necesidad de calificar nombres, un paquete puede importar elementos o volúmenes de otro paquete en su propio espacio de nombres. Un elemento dentro del paquete importador puede usar entonces el nombre del elemento importado como si se hubiera definido directamente en el paquete.

Un paquete anidado tiene acceso directamente a cualquier elemento contenido en paquetes exteriores (a cualquier grado de anidación), sin necesitar importarlos. Un paquete debe importar sus paquetes contenidos para agregarlos a su espacio de nombres directos, sin embargo. En general, el paquete contenido es un límite de la encapsulación.

Un paquete especifica la visibilidad de sus elementos. La visibilidad indica si otros elementos pueden tener acceso al elemento o a sus contenidos. Cuando un paquete importa elementos de otros paquetes, puede restringir la visibilidad de los elementos importamos a sus propios clientes.

Un paquete define la visibilidad de sus elementos contenidos como **privado** o **público**.

Los elementos públicos están disponibles a otros elementos de su propio paquete o uno de sus paquetes anidados y a la importación de paquetes el paquete. Los elementos privados no están disponibles en absoluto fuera del propio paquete.

Los contenidos de otros tipos de elemento (como atributos y operaciones) también pueden tener la visibilidad **protegida** o **paquete**. Los elementos protegidos sólo están disponibles a descendientes del clasificador que posee la característica. Los elementos del paquete están disponibles a todos los elementos dentro del mismo paquete como el clasificador que posee el rasgo.

La aplicación del perfil es una relación entre un paquete y una indicación de perfil que los estereotipos definieron en el perfil pero pueden ser aplicados a los elementos dentro del paquete.

Véase importación para una descripción completa de las reglas de visibilidad para los elementos que están en varios paquetes.

Notación

Un paquete se muestra como un rectángulo grande con un rectángulo pequeño (una “etiqueta”) unido en una esquina (normalmente, la cima izquierda del rectángulo grande). Ello tiene el significado de sugerir una carpeta del archivo. Pueden mostrarse los contenidos del paquete dentro del rectángulo grande (Figura 14.216).

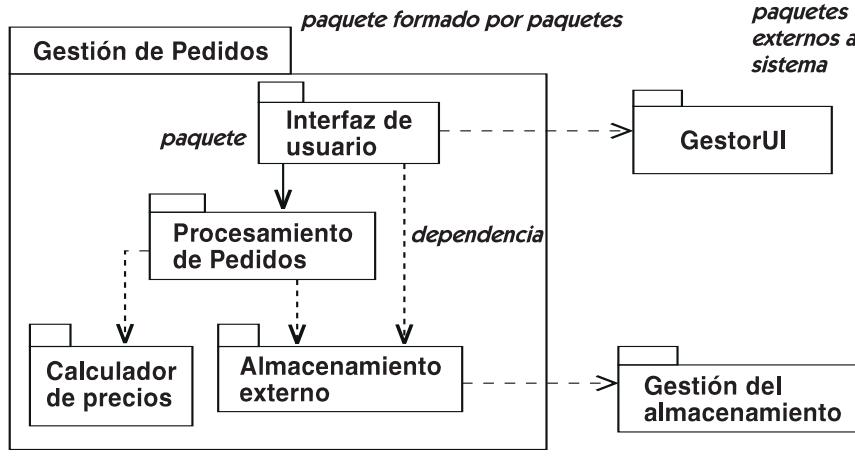


Figura 14.216 Paquetes y sus relaciones

Si no se muestran los contenidos del paquete, entonces el nombre del paquete se pone dentro del rectángulo grande. Si se muestran los contenidos del paquete, entonces el nombre del paquete puede ponerse dentro de la etiqueta.

Una cadena de la palabra clave puede ponerse sobre el nombre del paquete. Las palabras clave pueden incluir modelo. También se transcriben estereotipos usuario-definidos con palabras clave, pero ellos no deben crear conflicto con cualquier palabra clave predefinida.

La visibilidad de un elemento del paquete fuera del paquete puede ser indicada precediendo el nombre del elemento por un símbolo de visibilidad (‘+’ para el público, ‘–’ para el privado).

Pueden dibujarse flechas discontinuas entre los símbolos del paquete para mostrar las relaciones, eso ocurre entre por lo menos algunos de los elementos en los paquetes. Por ejemplo, una dependencia con la palabra clave «**use**» implica una dependencia de uso entre por lo menos un elemento en un paquete y un elemento en otro paquete.

La aplicación del perfil es mostrada por una flecha discontinua de un paquete a un perfil (embale el símbolo con el «**profile**» de la palabra clave). La flecha tiene la palabra clave «**apply**».

Opciones de la presentación

Una herramienta también puede mostrar la visibilidad desplegando esos elementos que se encuentran selectivamente en un nivel de visibilidad escogido, por ejemplo, todos los elementos públicos sólo.

Una herramienta puede mostrar la visibilidad por un marcador gráfico, como color o fuente.

Es de esperar que se mostrarán paquetes con grandes contenidos como iconos simples con nombres, en los que los contenidos pueden ser accedidos dinámicamente por “zooming” a una vista detallada. En lugar de mostrar los contenidos de un paquete como iconos anidados, se puede usar una línea de ramificación desde el paquete, a los iconos de sus elementos. Un pequeño cruce en un círculo se une a la línea en el fin del paquete. *Véase* la Figura 14.217. (La misma anotación puede usarse para cualquier espacio de nombres anidado, incluso clases declaradas en el alcance de otras clases.)

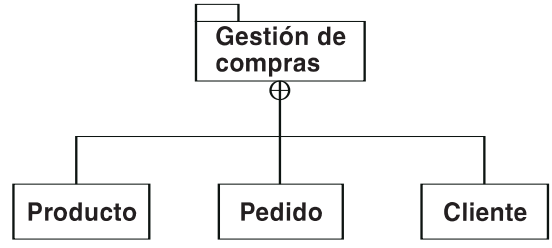


Figura 14.217 Notación externa para el contenido de los paquetes

Ejemplo

La Figura 14.216 muestra la estructura del paquete de un subsistema de orden-proceso. El paquete de nivel-superior contiene varios paquetes de bajo nivel. Las dependencias entre los paquetes son mostradas por flechas discontinuas. La figura también muestra algún paquete externo de los que el subsistema depende. Éstos pueden ser los componentes de off-the-shelf o elementos de la biblioteca.

Discusión

Se piensan los paquetes principalmente como acceso y mecanismos de control de configuración para permitir desarrollos, particularmente en grupos de trabajo grandes, para organizar a los modelos grandes y desarrollarlos sin entrar en la forma de cada uno de los otros. Inherentemente, quiere decir eso que los diseñadores quieren que ellos signifiquen. Más prácticamente, los paquetes deben seguir algunos tipos de límite semántico si han de ser útiles. Porque son considerados como unidades de mando de configuración, deben contener elementos que probablemente evolucionarán juntos.

Los paquetes también agrupan elementos que deben compilarse juntos. Si un cambio a un elemento fuerza la recopilación de otros elementos, entonces también podrían ser puestos en un paquete.

Cada elemento modelo debe ser poseído por exactamente un paquete u otro elemento modelo.

Por otra parte, el mantenimiento del modelo, hacer versión y control de la configuración se vuelve imposible. El paquete que posee un elemento modelo controla su definición.

Puede hacerse referencia y puede usarse en otros paquetes, pero un cambio en él, requiere permiso de acceso y derechos de actualización al paquete que lo posee.

par

La palabra clave que indica un fragmento combinado paralelo en un diagrama de interacción.

Véase paralelo.

paralelo

Un fragmento combinado que especifica los operandos de la interacción múltiple (subfragmentos) de quien el comportamiento será ejecutado concurrentemente (en paralelo).

Semántica

Un solo fragmento de la interacción impone una clasificación lineal en los eventos que ocurren en cada línea de la vida. No hay ninguna restricción en general, en la clasificación de eventos de diferentes líneas de vida. Aunque los diagramas de la secuencia parecen tener una dimensión de tiempo vertical no se sincronizan los horarios de líneas de vida diferentes y la conclusión no puede deducirse de la posición relativa de dos eventos en líneas de vida diferentes.

Los mensajes entre las líneas de vida imponen una clasificación: Los eventos que preceden al evento enviando en una línea de vida preceden los eventos que siguen al evento receptor en la otra línea de la vida.

Una estructura paralela tiene dos o más operandos (subfragmentos). Dentro de cada operando los eventos se piden en líneas de vida y por mensajes. Eventos en operandos paralelos sin embargo, no tienen ninguna clasificación relativa y pueden entrelazarse en cualquier orden consistente con la clasificación impuesta separadamente por cada operando.

Notación

Un fragmento combinado paralelo en un diagrama de sucesión se muestra como un perfil rectangular con un pentágono pequeño en la esquina izquierda superior que contiene la equivalencia de la etiqueta (Figura 14.218). El rectángulo cubre las líneas de vida dentro de las que intercambian los mensajes dentro del fragmento. El rectángulo es verticalmente dividido en dos o más secciones por líneas horizontales discontinuas. Cada sección contiene un fragmento de diagrama de secuencia conteniendo mensajes entre las líneas de vida. Dentro de cada sección, la clasificación de eventos a lo largo de una línea de vida es significante. La clasificación de las secciones dentro del rectángulo total no es significativa y ninguna clasificación es expresada o implica eventos de secciones diferentes.

Ejemplo

La Figura 14.218 muestra una interacción con dos usuarios y un servidor. Cada usuario hace una demanda, el servidor realiza el servicio y respuestas al usuario. En este ejemplo no hay ninguna clasificación especificada entre las dos sucesiones de la interacción. La primera secuencia puede preceder a la segunda o seguirla o las dos secuencias pueden ser entrelazadas de varias maneras. Por ejemplo, la segunda secuencia podría empezar primero pero terminar en último lugar. El proceso de las dos demandas puede o no solaparse. Ninguna asunción en absoluto puede hacerse sobre las dos sucesiones.

Si dos secuencias de la interacción pueden ocurrir en cualquier orden, pero ningún entrelazando de las secuencias puede ocurrir, la estructura de la región crítica puede usarse junto con la estructura paralela.

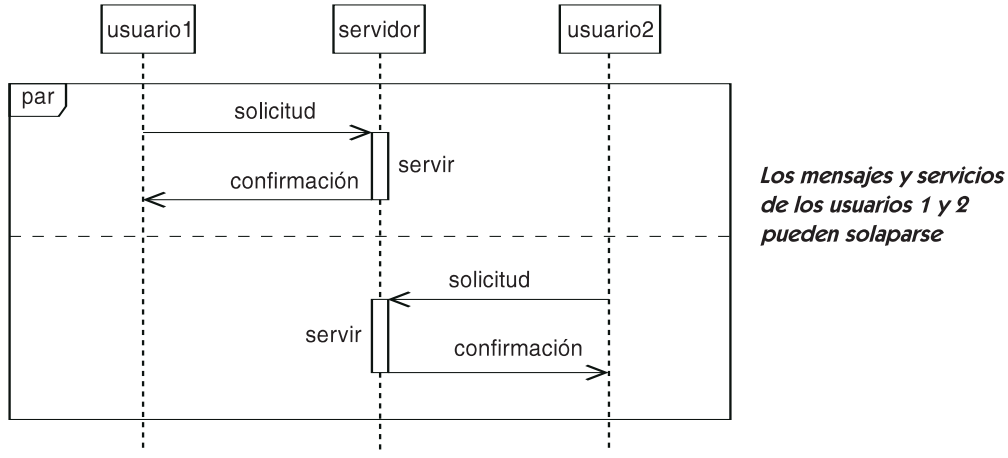


Figura 14.218 Fragmento paralelo combinado en un diagrama de secuencia

parámetro

La especificación de una variable que puede ser cambiada, pasada o retornada. Un parámetro puede incluir nombre, tipo y dirección. Se usan parámetros para operaciones, mensajes, eventos y plantillas. Contraste: argumento.

Una dependencia de uso de parámetro relaciona una cierta operación que tiene un parámetro o una clase que contiene una cierta operación para la clase del parámetro.

Véase también argumento, ligadura.

Semántica

Un parámetro es un marcador de posición para un valor que se liga a él cuando el elemento adjuntado se usa. Restringe los valores que el argumento puede tomar. Tiene las siguientes partes.

valor predefinido — Una expresión para un valor para ser usado si ningún argumento se proporciona para el parámetro. La expresión se evalúa cuando la lista de parámetros se liga a los argumentos.

dirección — La dirección de flujo de información del parámetro, una enumeración con los valores siguientes:

- **in** — Un parámetro de la entrada pasado por valor. Los cambios al parámetro no están disponibles para ser llamados. Esto es el valor predeterminado.
- **out** — Un parámetro de salida. No hay valor de entrada. El valor final está disponible para ser llamado.
- **inout** — Un parámetro de entrada que puede modificarse. El valor final está disponible para ser llamado.

return	Un valor de retorno de una llamada. El valor está disponible para ser llamado. Semánticamente, no difieren de un parámetro de salida, pero el resultado está disponible para el uso en una línea de expresión.

Las opciones precedentes no están disponibles en cualquier lenguaje de programación, pero el concepto detrás de cada una de ellas tiene sentido en la mayoría de los lenguajes y puede trazarse en una implementación.

nombre	El nombre del parámetro. Debe ser único dentro de su lista de parámetros.
tipo	Una referencia a un clasificador (una clase, tipo de datos, o una interfaz en la mayoría de los procedimientos). Un límite del argumento para el parámetro debe ser una instancia del clasificador o de uno de sus descendientes.

Notación

Cada parámetro se muestra como una cadena de texto que puede descomponerse en varias propiedades de un parámetro. La sintaxis predefinida es:

dirección$_{opc}$ **nombre:tipo**⌊**multiplicidad**⌋$_{opc}$ ⌊=**valor-por-defecto**⌋$_{opc}$

Dirección. La dirección se muestra como una palabra clave que precede al nombre de la operación. Si la palabra clave está ausente, entonces la dirección es dentro. Las opciones son dentro, fuera, dentrofuera y retorno. Los parámetros de retorno se muestran usualmente seguidos de una coma en la firma de la operación, donde necesitan ser marcados por dirección.

Nombre. El nombre se muestra como una cadena.

Tipo. El tipo se transcribe como una cadena que es el nombre de una clase, una interfaz, o un tipo de datos.

Valor predefinido. El valor se muestra como una cadena de la expresión. El lenguaje de la expresión sería conocido por (y especifiable a) una herramienta pero no se mostraría en el formato canónico.

Estática. Una operación estática (uno que aplica a la clase entera, en lugar de un objeto simple) se indica subrayando la cadena de la firma. Si la cadena no se subraya, la operación aplica al objeto designado (el valor predeterminado).

Dependencia de parámetro. Una dependencia de parámetro se muestra como una flecha discontinua desde la operación que tiene el parámetro o desde la clase que contiene la operación, a la clase del parámetro; la flecha tiene el «**parámetro**» del estereotipo atado.

Ejemplo

Matriz::transformar (**in** distancia: Vector, **in** ángulo: Real = 0): **retur** Matriz

Aquí, todas las etiquetas de dirección pueden omitirse.

parámetro de entrada y salida

Parámetro utilizado para proporcionar argumentos al procedimiento llamado y también para devolver valores al llamante utilizando efectos laterales en el propio parámetro.

Semántica

Un parámetro puede tener una dirección, que puede incluir entrada, salida, entrada y salida, y regreso. Un parámetro de entrada y salida es un parámetro intencionado para representar directamente parámetros en ciertos lenguajes de programación que pueden tanto proporcionar valores de argumentos como permitir asignaciones a variables parámetro dentro de los cuerpos de los procedimientos, con el valor asignado disponible para el llamante.

Notación

Se puede colocar la palabra clave **inout** antes del nombre de un parámetro de salida. Los parámetros sin palabras clave son parámetros de entrada.

Véase parámetro de salida.

parámetro de retorno

Un parámetro que representa la información producida durante la ejecución de un comportamiento y llevada al llamador del comportamiento después de completar la ejecución.

Semántica

Los tipos del parámetro predefinidos son en (only) y retorno. Cada uno de estos pasos de información en modo de sólo-lectura o por-valor.

Discusión

El uso de los parámetros de entrada y salida es de conveniencia para lenguajes de programación específicos y debe evitarse en el modelo, porque causa fuertes dependencias entre los detalles internos de un comportamiento y el entorno de la llamada.

parámetro de salida

Un parámetro que comunica valores al llamador que usa los efectos colaterales en el mecanismo de llamada del parámetro.

Véase también parámetro inout.

Semántica

Un parámetro puede tener una dirección que puede incluir entrada, salida, entrada y salida, y regreso. Un parámetro de salida es un parámetro pensado para representar directamente pará-

metros en ciertos lenguajes de programación dentro de los que permiten las asignaciones a las variables del parámetro los cuerpos de procedimientos, con los valores asignados disponibles al llamador. Un parámetro de entrada y salida es un parámetro que puede usarse para ambos, para entrada y valores de salida.

Notación

La palabra clave **out** puede ponerse antes del nombre de un parámetro de salida. La palabra clave **inout** puede ponerse antes del nombre de un parámetro de entrada y salida. Los parámetros sin palabras clave están en parámetros.

Discusión

El uso de parámetros out y los parámetros del inout sólo están presentes para proporcionar compatibilidad de bajo nivel con programación detallada. No tienen ningún lugar en el modelado y debe ser evitado por todo diseñador prudente. Una distinción limpia entre un parámetro y los parámetros de retorno son comunes en la mayoría de los lenguajes de programación modernos con excepciones desafortunadas.

parámetro real

Véase argumento

parte

Véase parte estructurada.

parte estructurada

Dentro de un clasificador estructurado, un elemento que representa un objeto o conjunto de objetos dentro de una relación contextual.

Véase clasificador estructurado.

partición

La división de un conjunto en subconjuntos disjuntos. *Véase* partición de actividad para un importante tipo de partición usado en actividades.

partición de actividad

Una partición en grafos de actividad que permite la organización de las responsabilidades de las actividades. Las particiones de actividad no tienen un significado fijo, pero a menudo se corresponden con unidades organizacionales en un modelo de negocio. También se pueden utilizar

para especificar la clase responsable de implementar un conjunto de tareas. Algunas veces se denominan *calles*[1] debido a su notación.

Véase también actividad.

Semántica

Los nodos de actividad se pueden organizar dentro de una actividad en particiones, que a menudo se suelen denominar *calles* debido a su notación. Las particiones de actividad son agrupacio-

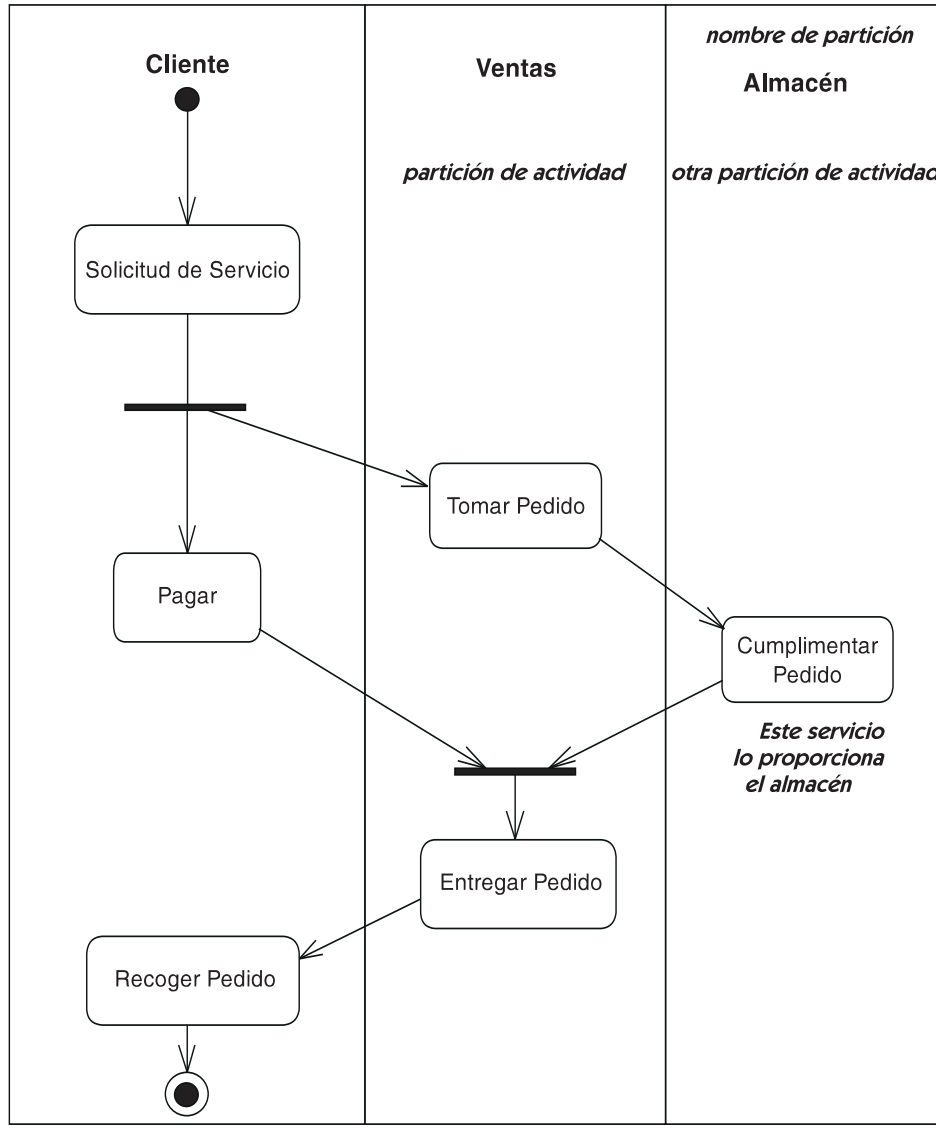


Figura 14.219 Participaciones de actividad en un diagrama de actividad

[1] Se alude al término inglés *swimlanes*, que hace referencia a las calles de una piscina. *(N. del T.)*

nes de nodos de actividad que permiten organizar el grafo de actividad. Cada partición de actividad representa alguna característica con significado de los nodos —por ejemplo, la organización de negocio responsable de un paso de un flujo de trabajo. Las particiones de actividad se pueden utilizar de cualquier forma que le venga bien al modelador. Si las particiones están presentes, dividen los nodos de un grafo de actividad entre ellas.

Si se vincula una clase a una partición, la clase es responsable de implementar el comportamiento de los nodos contenidos dentro de la partición.

Cada partición de actividad tiene un nombre distinto del resto de las particiones.

Notación

Un diagrama de actividad se puede dividir en secciones, cada una separada de sus vecinas por líneas continuas (Figura 14.219). Aunque lo más habitual es que sean verticales y rectas, también pueden ser horizontales o curvas, o formar una rejilla. Cada sección se corresponde con una partición de actividad. Cada partición representa responsabilidades de alto nivel de parte de la actividad completa, que se puede implementar finalmente mediante un objeto o más. El orden relativo de las particiones de actividad en un diagrama no tiene significado semántico pero podría indicar alguna relación del mundo real. Cada nodo de actividad se asigna a una partición, y su símbolo se coloca dentro de su sección. Los flujos pueden cruzar las líneas; no tiene ningún significado especial la dirección de las flechas de flujo.

Dado que las particiones de actividad son categorías arbitrarias, se pueden indicar mediante otros métodos si la distribución geométrica no es posible. Entre las formas alternativas de representación de las particiones se encuentra el uso de colores o simplemente el uso de valores etiquetados. De forma alternativa, se pueden etiquetar los nodos con sus particiones.

Historia

A diferencia de UML1, las particiones en UML2 pueden ser bidimensionales.

participa

Término informal para el añadido de un elemento del modelo a una relación o a una relación reified. Por ejemplo, una clase participa en una asociación y un rol clasificador participa en una colaboración.

patrón

Una colaboración parametrizada que representa un conjunto de roles para clasificadores parametrizados, relaciones, y comportamientos que pueden ser aplicados a múltiples situaciones mediante el enlace de elementos del modelo (usualmente clases) a los roles del patrón. Es una plantilla de colaboración.

Semántica

Un patrón representa una colaboración paremetrizada que se puede utilizar varias veces dentro de uno o más sistemas. Para ser un patrón, la colaboración debe ser usable en un amplio rango

de situaciones para justificar el que se le dé nombre. Un patrón es una solución que se ha mostrado que funciona en un conjunto de situaciones. No es necesariamente la única solución al problema, pero es una solución que ha sido efectiva en el pasado. La mayoría de los patrones tienen ventajas y desventajas dependiendo de varios aspectos del sistema. El modelador debe considerar esas ventajas o desventajas antes de tomar la decisión de usar un patrón.

Una colaboración parametrizada de UML representa las vistas estructurales y del comportamiento de ciertos tipos de patrones. Los patrones implican otros aspectos que no son modelados por UML, tales como la lista de ventajas y desventajas o los ejemplos de usos anteriores. Muchos de esos otros aspectos se expresan mediante palabras. *Véase* [Gamma-95] para una descripción completa sobre patrones, así como un catálogo de patrones de diseño probados.

Generando colaboraciones a partir de patrones. Una colaboración puede ser usada para especificar la implementación de construcciones de diseño. El mismo tipo de colaboración se puede utilizar muchas veces parametrizando sus constituyentes. Un patrón es una colaboración parametrizada. Generalmente, las clases de los roles en la colaboración son parámetros. Un patrón se instancia como una colaboración por la ligadura de los valores, usualmente clases, a sus parámetros. Para el caso común de roles parametrizados, la plantilla se completa especificando las clases para cada rol. Típicamente, los conectores en un patrón no están parametrizados. Cuando la plantilla ha sido ligada, representan asociaciones implícitas entre las clases ligadas a la colaboración —eso es, la ligadura de la plantilla para hacer que la colaboración genere asociaciones adicionales.

Notación

La Figura 14.220 muestra el patrón Observer de [Gamma-95]. Hemos modificado la estructura que muestra el libro de patrones de diseño para especificar un patrón como una colaboración para indicar la naturaleza colaborativa de la mayoría de los patrones.

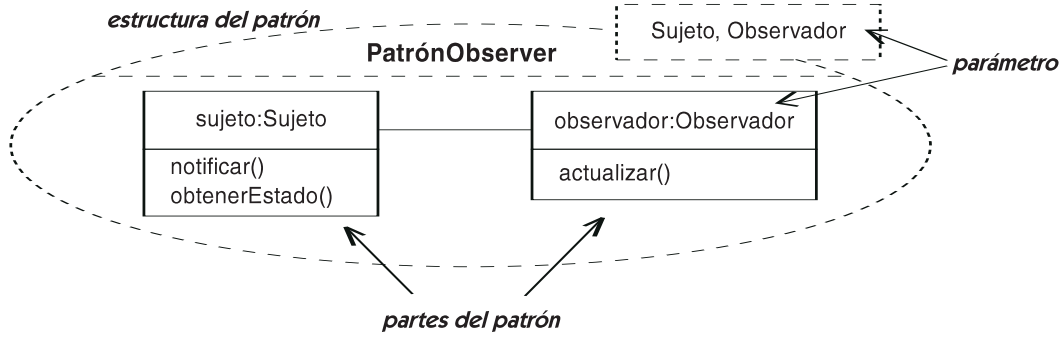


Figura 14.220 Definición del patrón

La ligadura de un patrón para producir una colaboración se muestra como una elipse punteada que contiene el nombre del patrón (Figura 14.221). Una línea discontinua se dibuja desde el símbolo de ligadura del patrón a cada una de las clases (u otros elementos del modelo) que participan en la colaboración. Cada línea se estiqueta con el nombre del parámetro.

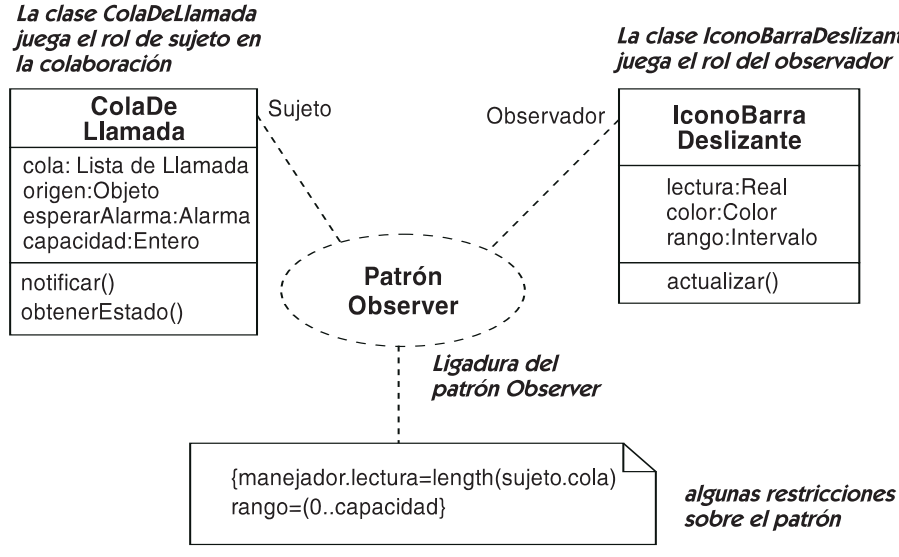


Figura 14.221 Ligadura de un patrón para hacer una colaboración

En la mayoría de los casos, el nombre del rol en la colaboración se puede usar como nombre de parámetro. Por lo tanto, el símbolo de ligadura del patrón puede mostrar el uso del patrón de diseño, conjuntamente con las clases reales del uso del patrón. La ligadura del patrón por lo general no muestra la estructura interna de la colaboración, se genera mediante la ligadura. Esto viene dado por el propio símbolo de la ligadura.

perfil

La definición de un conjunto de elementos adicionales limitados a un metamodelo base para adaptarlo a una plataforma específica o dominio.

Semántica

Un perfil es un paquete que identifica un subconjunto de un metamodelo base existente (posiblemente incluyéndolo todo) y define los estereotipos y restricciones que pueden aplicarse al subconjunto del metamodelo seleccionado. El propósito es permitir las modificaciones limitadas de metamodelos de UML sin requerir o permitir cambios arbitrarios al metamodelo.

Si se requiere extensibilidad completa de primera clase, un nuevo metamodelo puede definirse usando MOF, pero el resultado puede o no puede ser como UML.

Un perfil también puede contener los elementos modelo o puede tener una dependencia a un paquete que contenga de elementos modelo. Un paquete tal en una biblioteca de modelos. Los elementos en el paquete vienen a estar disponibles a un paquete que aplica el perfil.

Un perfil designa un metamodelo base (como el metamodelo de UML normal) y selecciona los elementos de él usando importación del elemento o importación del paquete. Los elementos seleccionados pueden ser un subconjunto que son significativos para una plataforma o dominio,

específico, de modo que esas herramientas o modeladores no necesitan el trato con capacidades de UML no deseadas.

Un perfil contiene un conjunto de estereotipos y declaraciones de restricción. Los estereotipos pueden ser extensiones de elementos del metamodelo en el conjunto seleccionado. La declaración de un estereotipo define etiquetas —esencialmente metaatributos adicionales— de elementos metamodelo. En un modelo del usuario, el estereotipo puede aplicarse a un elemento de metatipo dado y los metaatributos adicionales pueden ser valores dados por el diseñador. Los valores modelo pueden usarse para comunicación entre los humanos o pueden ser usados por herramientas para generar código. Un estereotipo puede agregar metaatributos pero no puede quitar los existentes.

Una restricción puede aplicarse a una metaclase dada. Elementos de la metaclase en los modelos usuario-definidos están sujetos a restricción. Las restricciones pueden agregarse en perfiles, pero no pueden quitarse restricciones existentes o pueden debilitarse. Un modelo con restos del perfil modelo de UML debe satisfacer todas las restricciones de UML.

Un perfil se hace disponible para un modelo de usuario por aplicación del perfil.

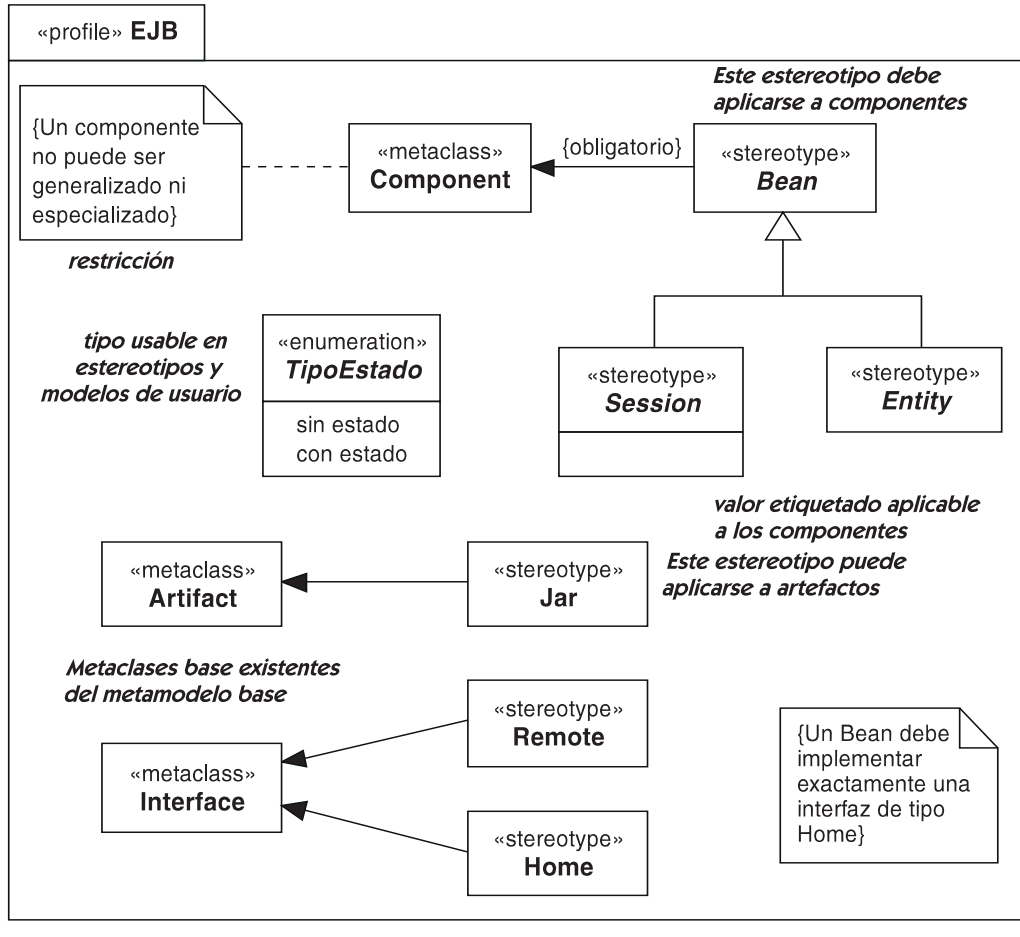


Figura 14.222 Definición de perfil

Notación

Un perfil se muestra como un símbolo del paquete (rectángulo con etiqueta grande) con la palabra clave «**profile**» sobre el nombre del paquete. El símbolo del paquete contiene elementos del metamodelo seleccionados en el perfil, estereotipos declarados con flechas de extensión a sus elementos del metamodelo base, y restricciones que aplican a los elementos del metamodelo.

Un perfil puede importar declaraciones de tipo que pueden usarse como los tipos de etiquetas del estereotipo. Cualquier tipo importado también puede usarse en un modelo del usuario, similar a tipos predefinidos como Entero y Cadena.

Un perfil puede tener una flecha de dependencia a una biblioteca de modelos, es decir, un paquete conteniendo declaraciones del elemento modelo. Los elementos modelo están disponibles para los paquetes que aplica el perfil.

Ejemplo

La Figura 14.222 (del documento de especificación de UML) muestra la definición de un perfil simple para un ambiente de EJB.

permiso

Un tipo de dependencia que concede el permiso al elemento de cliente para usar el contenido del elemento del proveedor, sin tener en cuenta las declaraciones de visibilidad de los elementos contenidos.

Semántica

El permiso es un tipo de dependencia que hace los elementos interiores del proveedor accesibles al cliente, aun cuando se hayan declarado privados. Esto tiene el mismo efecto que la estructura friend de C++.

Notación

Una dependencia de permiso se muestra como una flecha discontinua del cliente (el elemento que gana permiso) al proveedor (el elemento que concede el permiso) con el «**permit**» de la palabra clave junto a la flecha.

Ejemplo

En Figura 14.223, el **GestorDeImpresión** puede anular cualquier trabajo y puede reiniciar la impresora, incluso aunque esas funciones son privadas.

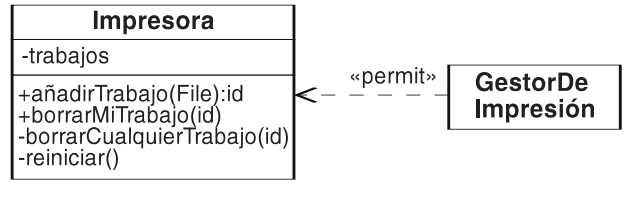


Figura 14.223 Dependencia de permiso

peso

Número de tokens consumidos por una transición de actividad particular por la ejecución de un nodo de actividad.

Semántica

En el modelo de actividad básico, un token se consume desde una transición entrante por cada ejecución de un nodo de actividad. En el modelo avanzado, un peso se puede colocar en una transición para permitir que varios tokens sean consumidos por una sola ejecución.

El peso es un entero positivo adjunto a una transición que especifica el mínimo número de tokens a consumir por la transición. También puede tener el valor especial **all**, que indica que todos los tokens disponibles en la transición se consumirán siempre que el nodo comience su ejecución.

Notación

El peso se representa colocando una cadena en una transición de actividad con el formato:

{ weight = **`valor`** }

donde **`valor`** es un entero positivo o la cadena **all**, que indica que se consumen todos los tokens disponibles.

pin

Un punto de conexión en una acción para valores de entrada y de salida.

Semántica

Es un nodo de objeto que representa un punto de conexión para valores de entrada y salida de una acción. Hay un pin de entrada para cada parámetro de entrada y un pin de salida para cada parámetro de salida. Los pins de una acción se ordenan para que puedan ser emparejados con los parámetros. Los pins tienen nombres y tipos.

Normalmente no serán implementados como mecanismos distintos. Son sólo parte de los formalismos de modelado de acciones.

Pueden unirse varias declaraciones de propósito-especial a un pin, tales como la habilidad para aceptar cadenas de datos y la habilidad para la salida de objetos de excepción.

Notación

Un pin se muestra como un rectángulo pequeño unido al exterior de la frontera de un símbolo de acción (una caja redondeada). Un nombre puede ponerse cerca del símbolo del pin o puede usarse la anotación del parámetro llena (Figura 14.224).

Si un pin acepta cadenas de caracteres, la palabra clave {**stream**} se pone cerca del símbolo.

Figura 14.224 Notación pin

Si un pin produce objetos de excepción, un triángulo pequeño se pone cerca del símbolo.

Si una acción se muestra desconectada de otras acciones, las flechas pequeñas pueden ser puestas dentro del símbolo del pin para distinguir los pins de entrada y salida.

Pueden ponerse números (empezando con 1) cerca del símbolo del pin para mostrar la clasificación de pins. Normalmente los nombres son suficientes para establecer la clasificación implícita, así que raramente se usan números.

Están disponibles varias opciones de representación, pero parecen innecesarias y menos intuitivas que la notación normal, así que no animamos a su uso.

Ejemplo

La Figura 14.225 muestra varios nodos de actividad conectados a través de los pines. Las entradas a los nodos de actividad son nodos del objeto que representan los datos.

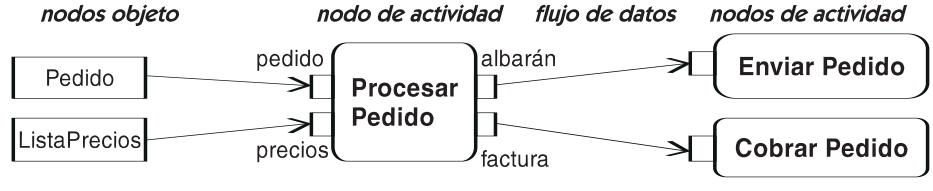


Figura 14.225 Nodos conectados

plantilla

Un elemento parametrizado del modelo. Para usarlo, se deben ligar los parámetros a los valores actuales (en tiempo de modelado). Sinónimo: elemento parametrizado.

Véase también ligadura, elemento ligado.

Semántica

Una plantilla es la descripción de un elemento con uno o más parámetros formales. Define una familia de elementos potenciales, cada uno especificado mediante la ligadura de parámetros a valores actuales. Típicamente, los parámetros son clasificadores que representan tipos de atributos, pero también pueden ser enteros o incluso operaciones. Los elementos subordinados dentro de la definición de una plantilla, se definen en términos de los paráme-

tros formales, de tal forma que pasan a estar ligados cuando la propia plantilla se asocia a los valores reales.

La parametrización de plantillas puede aplicarse a clasificadores (tales como clases y colaboraciones), paquetes, y operaciones. Casi cualquier tipo de elemento puede ser un parámetro de la plantilla.

Clasificadores. Una plantilla de clasificador es un clasificador parametrizado. El cuerpo de una plantilla puede contener usos de los parámetros formales, normalmente como tipos de propiedades, aunque también en otras posiciones. Un clasificador real se genera enlazando valores a los parámetros. Los atributos y operaciones dentro de la plantilla clasificador, pueden usar los parámetros formales, normalmente como tipos. El clasificador de la plantilla puede ser un subtipo de un clasificador regular. Cuando la plantilla se enlaza, el clasificador ligado es un subtipo del clasificador regular. Las asociaciones se pueden modelar usando partes dentro de un clasificador estructurado.

Una clase plantilla no es una clase usable directamente, porque tiene parámetros sin ligar. Sus parámetros deben ser enlazados a valores reales, para crear una clase real. Una clase plantilla puede ser una subclase de una clase ordinaria, lo que implica que todas las clases formadas por la ligadura con la plantilla son subclases de la clase dada.

Otro tipo de clasificadores, tales como casos de uso y señales, también pueden parametrizarse.

Un clasificador enlazado puede añadir elementos a los generados por la ligadura. Esto es equivalente a generar clasificadores anónimos, mediante la ligadura, haciendo que la ligadura con nombre sea un hijo.

Un clasificador puede enlazar múltiples plantillas al mismo tiempo, esto es equivalente a generar clasificadores ligados anónimos, para cada ligadura, y a continuación hacer que el clasificador con nombre sea un subtipo de cada uno de ellos.

Un parámetro de una plantilla se puede restringir para ser una clase que sea descendiente de una clase específica. Una clase que sustituya un parámetro debe ser descendiente de la clase dada. Como alternativa, el parámetro puede ser restringido para que sea un sustituto válido de la clase dada, sin necesidad de ser un descendiente. Esto permite que los argumentos reales satisfagan las interfaces de la clase dada, sin tener la misma estructura.

Las plantillas pueden ser redefinidas. *Véase* redefinición (plantilla).

Colaboraciones. Una colaboración es un clasificador y por lo tanto puede ser una plantilla. Típicamente, los tipos de los roles, son parámetros de la plantilla . Una plantilla de colaboración es un patrón estructurado. Observe que el uso de una colaboración en un modelo de clases no es lo mismo que la ligadura de una colaboración; el último es una colaboración (un descriptor), no el uso de una colaboración (una referencia a una colaboración).

Paquetes. Un paquete puede ser una plantilla. Los elementos del paquete son a menudo parámetros de la plantilla. La misma plantilla paquete se puede asociar múltiples veces con el mismo elemento. Esto significa que el resultado de cada ligadura individual se combina (utilizando la unión de los paquetes) en un único paquete resultante. Esta es la forma de aplicar el mismo patrón múltiples veces a diferentes conjuntos de clases.

Operaciones. Una operación puede ser una plantilla. Típicamente, los tipos de sus parámetros son parámetros de la plantilla. Las restricciones de la operación también pueden contener parámetros de la plantilla.

Elementos bien formados. Los contenidos de una plantilla no están directamente sujetos a las reglas de correcta formación de modelos. Esto es porque incluyen parámetros que no tienen semántica completa hasta que están ligados. Una plantilla es un tipo de elemento de segundo nivel —no uno que modela los sistemas directamente, sino uno que genera elementos del modelo. Los contenidos de una plantilla están fuera de la semántica del sistema. Los resultados de ligar una plantilla son, normalmente, elementos del modelo que están sujetos a las reglas de formación correcta, y son elementos normales del sistema de destino. Es posible derivar algunas reglas de buena formación para plantillas, considerando que los elementos resultantes de la ligadura deben estar bien formados, pero no vamos a enumerarlas. En este sentido, cuando una plantilla se enlaza, sus contenidos se duplican, y los parámetros se fijan a los argumentos. El resultado se vuelve parte del modelo efectivo, tal y como si se hubiera incluido directamente.

Notación

Véase la Figura 14.226.

Parámetros. Los parámetros tienen la sintaxis

`nombre`:**`tipo`** = **`valor-por-defecto`**

donde **nombre** es un identificador para el parámetro, con ámbito dentro de la plantilla;

`tipo` es una cadena que define una expresión de tipo para el parámetro;

`valor-por-defecto` es una expresión opcional para usar un valor por defecto, si no se proporciona ningún valor para el parámetro durante la ligadura.

Si el nombre de tipo se emite, se asume que es una clase. Otros tipos de parámetro (tales como una expresión entera), se deben mostrar explícitamente y se deben evaluar para validar la expresión de tipos.

Un parámetro se puede restringir para indicar que su argumento debe ser un descendiente de una clase específica. Esto se muestra usando la sintaxis

`nombre`:**`tipo`** < **`nombre-de-clase`**

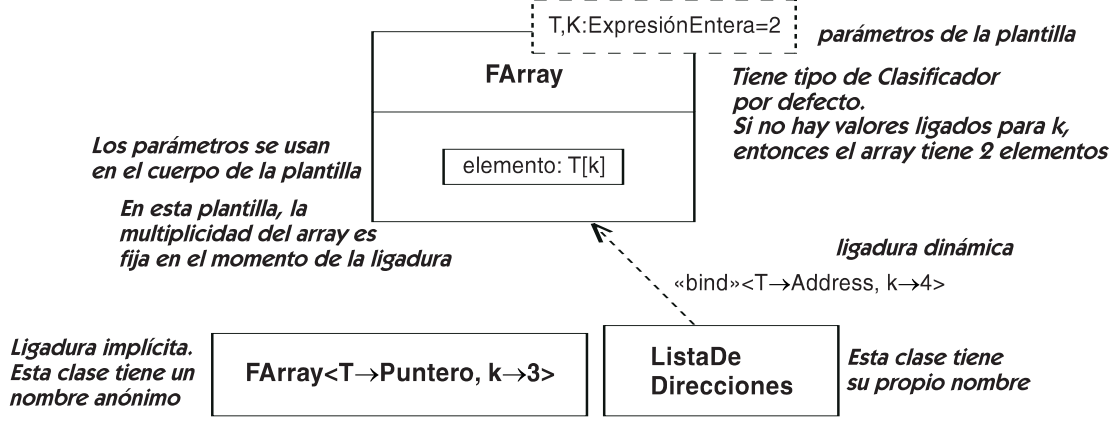


Figura 14.226 Notación de la plantilla de clase

Clasificadores. En una definición de una plantilla, se superpone un pequeño rectángulo con línea discontinua en la esquina superior derecha del rectángulo del clasificador. El rectángulo con la línea discontinua contiene una lista con los parámetros formales de la plantilla. Cada parámetro tiene un nombre y un tipo, y puede incluir opcionalmente un valor por defecto, el cual se utilizará si no se proporciona ningún valor en el momento de la ligadura. La lista no debe estar vacía (de lo contrario, no es una plantilla), aunque puede suprimirse para su presentación. El nombre, atributos, y operaciones de la clase parametrizada, aparecen de la manera habitual en el rectángulo de la clase, pero se pueden utilizar los nombres de los parámetros formales. Si la plantilla es un clasificador estructural, sus partes pueden mostrarse dentro de un compartimento gráfico. A menudo, los tipos de alguna de sus partes son los parámetros de la plantilla.

Si se añaden características al clasificador ligado, además de las generadas por la propia ligadura, se mostrarán en el compartimento de las características dentro del rectángulo que representa al clasificador ligado.

La Figura 14.226 muestra una plantilla de clase, con un parámetro entero y un parámetro de clase. La plantilla tiene una asociación con uno de sus parámetros.

Una plantilla puede ser hija de otro elemento. Esto significa que cada elemento generado mediante la ligadura, a partir de él, es un hijo del elemento dado. Por ejemplo, en la Figura 14.227, cualquier clase generada a partir de **Poli**, es un **Polígono**. Por lo tanto, **Hexágono** es un hijo de **Forma**.

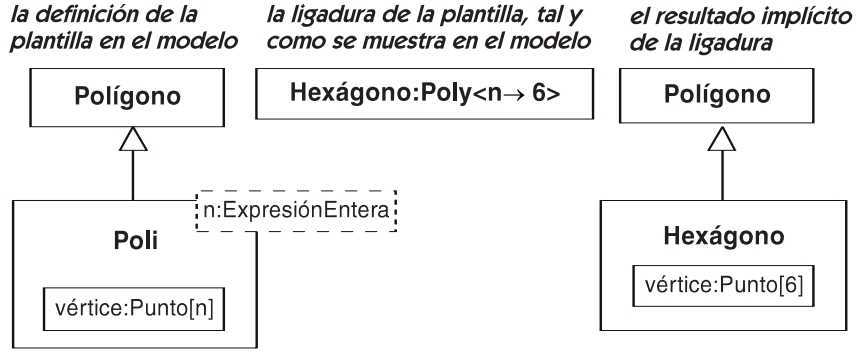


Figura 14.227 Plantilla de subclase

La Figura 14.228, muestra la adición de características a una clase ligada, declara una ListaDiezPrincipales, una lista de chistes, que cuenta el anfitrión de un show. El anfitrión y la fecha del show se añaden a los elementos generados por la plantilla. Las características se muestran en su compartimento usual, dentro de la clase ligada. Para una clase estructurada, se pueden añadir partes en un compartimento gráfico.

La Figura 14.229, muestra la ligadura múltiple de una clase ligada. ListaDePedidos es tanto un **VArray** y un **Bufer**. Ambos parámetros de la platilla están ligados a la clase **Pedido**. Observe que los nombres de los parámetros, no tienen nada que ver los unos con los otros, incluso aunque los dos se llamen **T**.

Se puede restringir un parámetro, de tal forma que el valor real deba ser un descendiente de cierta clase. La Figura 14.230, muestra una declaración de la plantilla de la clase **Polilínea**. Una

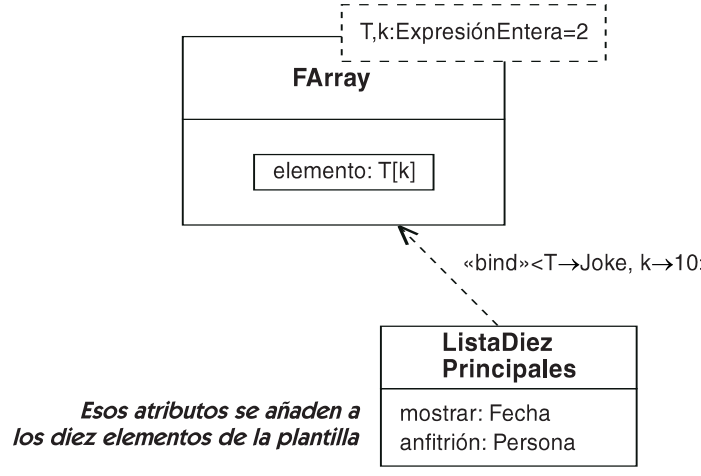


Figura 14.228 Añadiendo elementos a la clase ligada

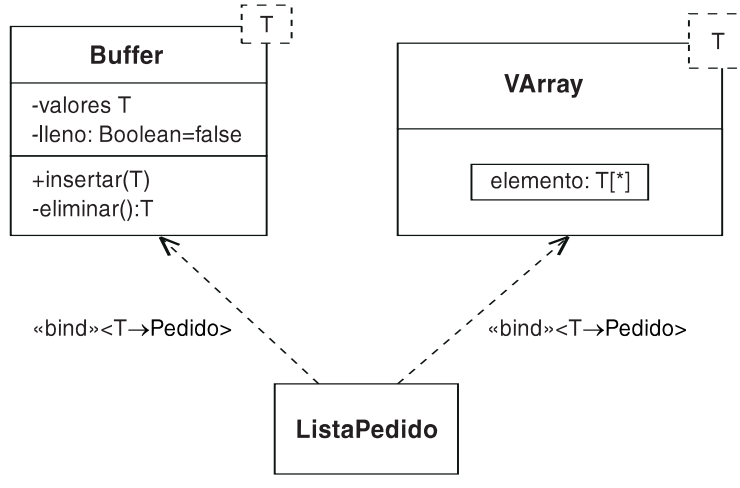


Figura 14.229 Múltiples ligaduras de una clase

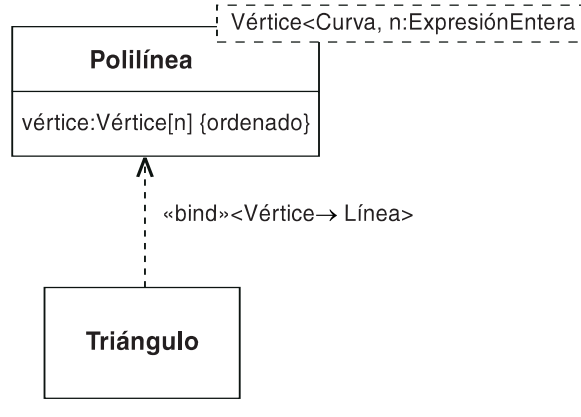


Figura 14.230 Parámetros de clase con restricción

PoliLinea es una lista ordenada de vértices. Algunas polilíneas tienen vértices rectos y algunas curvas. Para cualquier enlace de **Polilínea**, el tipo vértice debe ser una subclase de la clase abstracta **curva**. Para un **triángulo**, habrá tres vértices rectos.

Una colaboración es un clasificador que usa la misma notación que una plantilla. La Figura 14.231 muestra la colaboración **Venta** como una plantilla que tiene tres roles. Los tipos **Propiedad** y **Agente** son parámetros de la plantilla. El **EstadoRealDeVenta** es una colaboración ligada, donde propiedad debe ser un tipo EstadoReal y los agentes deben ser de tipo Realtor.

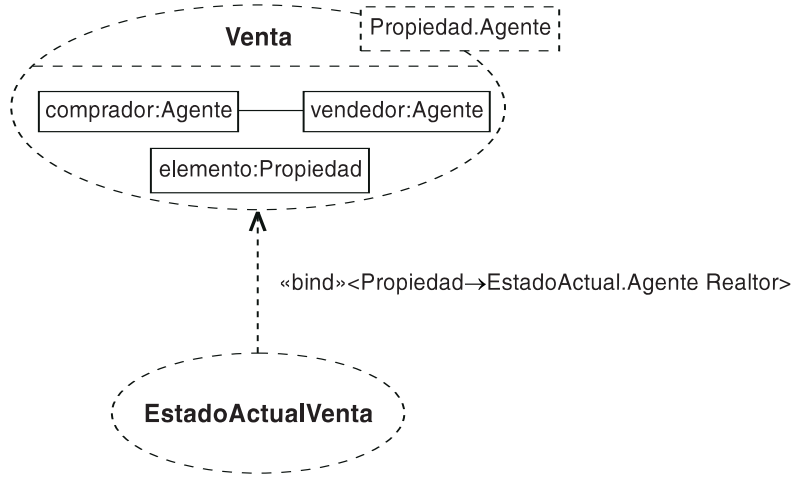


Figura 14.231 Plantilla de colaboración

Paquetes. Una plantilla de un paquete se muestra con el símbolo del paquete (un rectángulo con una sola tapa), con un pequeño rectángulo punteado superpuesto en la esquina superior derecha. La lista de los parámetros se coloca en el rectángulo punteado.

La Figura 14.232, muestra una plantilla que define el paradigma sujeto-vista. Las clases **Sujeto** y **Vista** no son parámetros, por lo tanto, no se copian en la plantilla. Las clases **S** y **V** son parámetros. Las clases enlazadas a ellos se convierten en subclases de **Sujeto** y **Vista**. El paquete enlazado se enlaza dos veces a la misma plantilla. Cada ligadura añade un par de clases al paquete. Las clases **Sujeto** y **Vista** son lo mismo en ambas ligaduras, así que no aparecen dos veces. La Figura 14.233 muestra el resultado efectivo de la ligadura de paquetes.

La Figura 14.234 muestra el uso de la sustitución de cadenas para modelar el ejemplo anterior, de una forma significativamente distinta. Una expresión de cadena puede colocarse dentro de signos de dólar ($ $) para indicar que tiene que ser interpretado como un nombre. Los fragmentos de las expresiones de cadena pueden ser parámetros de la plantilla, encerrados entre ángulos (< >). Esto permite que las cadenas se pasen como argumentos de enlace y se concatenen con cadenas literales, para generar nombres para los elementos de la plantilla. En este ejemplo, los nombres de los elementos Vista son generados a partir de los nombres de los elementos Sujeto, añadiendo la palabra **Display**. Otra diferencia con la Figura 14.232 es que los argumentos son cadenas, en lugar de clases existentes, lo que se requería en el modelo previo. Las clases son generadas como parte de la expansión de la plantilla.

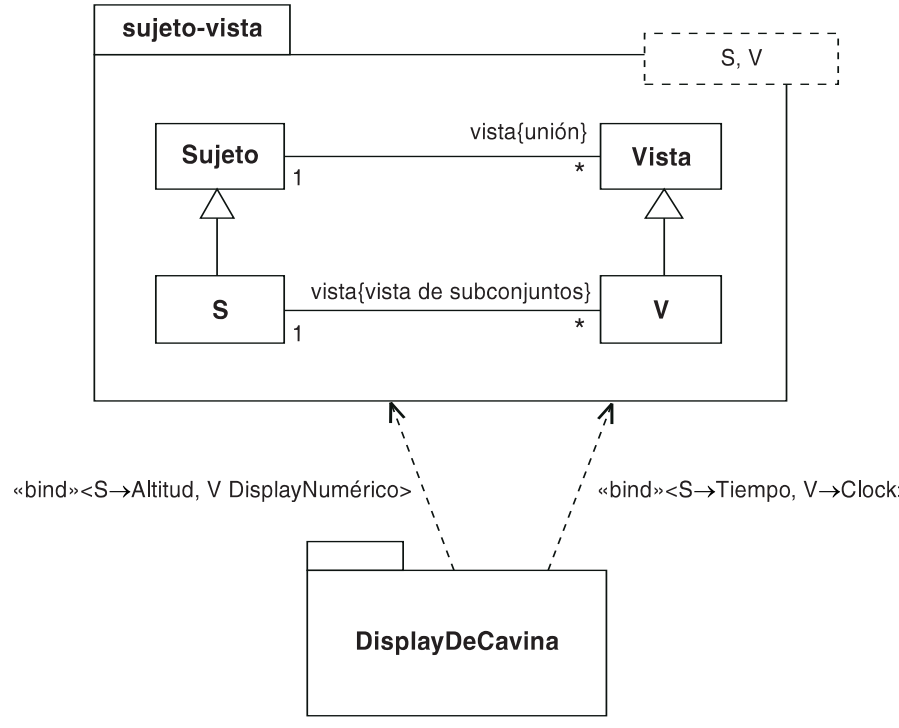


Figura 14.232 Plantilla de paquete

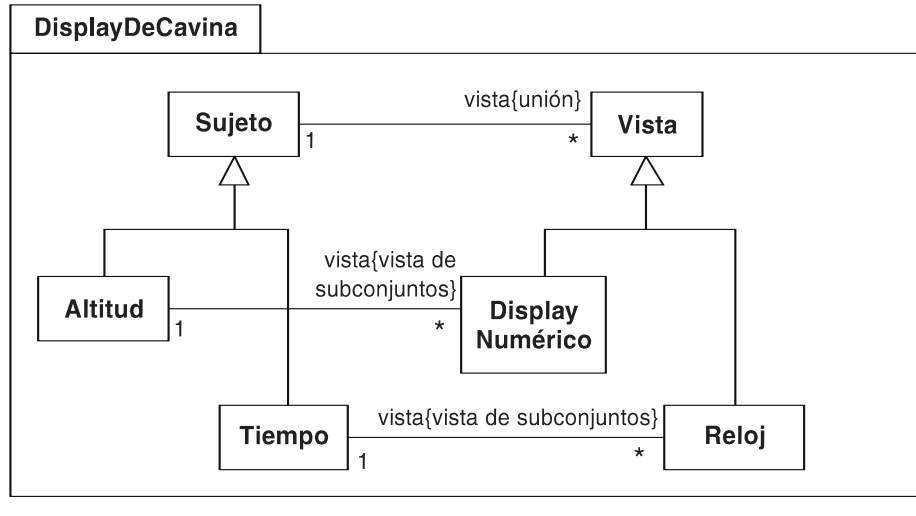


Figura 14.233 Resultado de ligadura de paquetes

Operaciones. No está claro si hay una notación para operaciones parametrizadas. No se da ningún ejemplo en la especificación de UML. Probablemente, en cualquier caso, es preferible evitar las plantillas de operaciones, ya que están demasiado cerca del nivel de implementación, y su uso en la mayoría de los lenguajes de programación está desaconsejado.

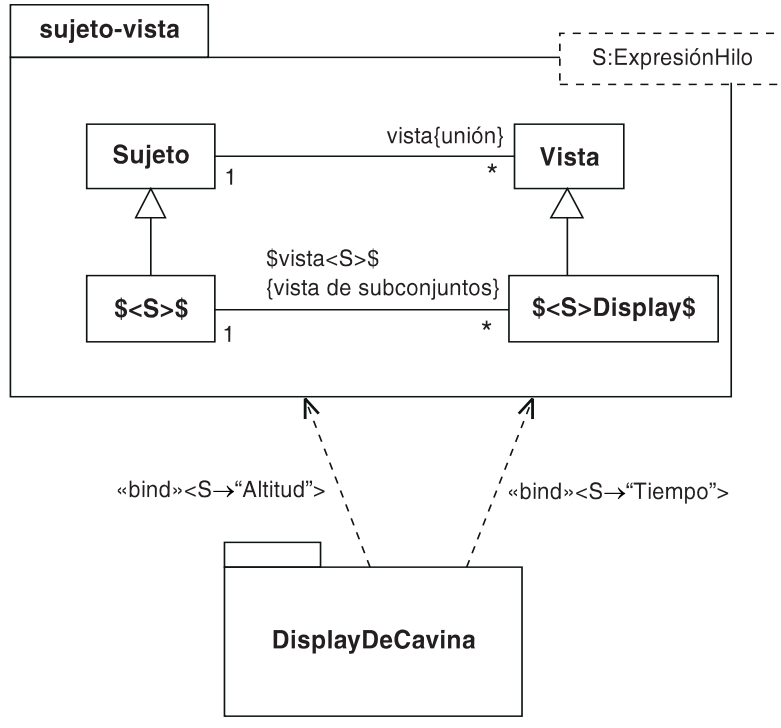


Figura 14.235 Subsección de cadenas dentro de la plantilla de paquete

Discusión

El modelo efectivo es el modelo implícito resultante de enlazar todas las plantillas, es el modelo implícito que describe un sistema. Los parámetros de la plantilla no tienen significado dentro del modelo efectivo, en sí mismo, porque han sido ligados. Sólo pueden ser utilizados dentro del ámbito del propio cuerpo de la plantilla. Esto es adecuado para manipular los elementos constitutivos contenidos dentro de los elementos parametrizados, por ejemplo para atributos u operaciones dentro de una clase parametrizada.

La definición de una plantilla es un fragmento de un modelo que no es parte del modelo efectivo. Cuando la plantilla se enlaza, el cuerpo implícitamente se copia y los parámetros se reemplazan por los argumentos, y la copia pasa a ser parte del modelo efectivo. Cada instanciación de la plantilla produce una añadidura al modelo efectivo.

Las relaciones entre plantillas y sus parámetros son más difíciles. Por ejemplo, una clase plantilla puede tener asociaciones o generalizaciones a otras clases. Si esas clases son parámetros de la plantilla, no pueden ser parte de la propia definición de la clase, porque no son características, ya que no son parte del modelo fuera de la plantilla. Esto puede (posiblemente) ser modelado, pero no hay notación para ello, porque la notación asume que todos los parámetros aparecen como contenidos de la propia clase plantilla. Los parámetros de la plantilla no pueden ser pintados como una clase separada fuera de la clase plantilla, así que no parece que haya una forma de denotarlo correctamente.

En principio, una definición de una plantilla debería incluir un cuerpo que pueda incluir elementos externos. Esos elementos pueden incluir asociaciones y generalizaciones a los parámetros de la plantilla. Sin embargo, la especificación no define explícitamente un cuerpo, aunque una lectura detallada sugiere que algunos parámetros de la plantilla podrían ser parte del cuerpo, pero no parte de los propios elementos de la plantilla.

Una plantilla, normalmente, no puede ser padre de otro elemento. Esto significaría que cada elemento generado por la ligadura de la plantilla es padre de otro elemento. Aunque alguien pueda encontrar sentido a tal situación, parece poco plausible.

Dos plantillas no tienen asociaciones entre ellas sólo por el hecho de compartir un parámetro. (El intentar hacer esto, significa que cualquier instanciación de la primera plantilla estaría relacionada con cualquier instanciación de la segunda, lo que normalmente no es deseable. Este punto ha sido mal entendido con frecuencia por los autores en el pasado.) Un parámetro tiene ámbito solamente dentro de la propia plantilla. Usar el mismo nombre para un parámetro en dos plantillas, no lo convierte en el mismo parámetro. Generalmente, si dos plantillas tienen elementos parametrizados, que deben estar relacionados, una de las plantillas debe instanciarse dentro del cuerpo de la otra . (Hay que destacar que una plantilla se instancia implícitamente dentro de su propio cuerpo. Por lo tanto, ambas plantillas son instanciadas efectivamente dentro del cuerpo y sus relaciones existen por lo tanto, entre los elementos instanciados.) La Figura 14.235, muestra un intento correcto e incorrecto para definir tal relación —en este caso, con una clase puntero parametrizada que apunta a un array parametrizado del mismo tipo.

Se puede utilizar una aproximación similar para declarar clases parametrizadas, que son hijas de otra clase plantilla, ligadas con el mismo parámetro. Otra aproximación es instanciar ambas

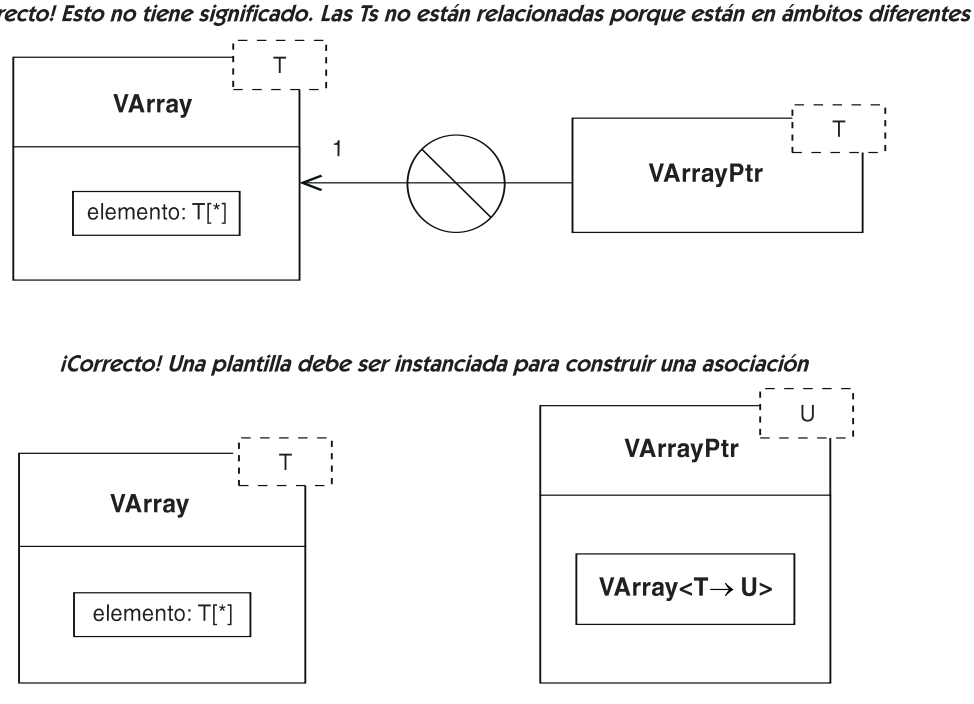


Figura 14.235 Asociación entre plantillas

plantillas dentro de una tercera plantilla, que tiene un tercer parámetro. El parámetro se utiliza para ligar una copia de cada una de las otras plantillas. Por lo tanto, se puede construir una asociación entre las copias instanciadas de la plantilla. En un caso más práctico, no es necesario, porque la relación se puede declarar en el cuerpo de una de las plantillas.

Las plantillas deberían usarse de forma separada. En muchos casos (como en el lenguaje C++), ellos realizan tareas que son preferibles para el polimorfismo y la generalización, en los casos en los que la máxima eficiencia no es necesaria. Debido a que son generadores, sus resultados no son siempre aparentes.

polimórfico/a

Indica una operación cuya aplicación puede ser proporcionada por una clase descendiente (un método o una máquina de estado disparada por un evento de llamada). Una operación que es no polimórfica es una operación concreta.

Véase también operación abstracta, generalización, herencia, método.

Semántica

Si una operación es polimórfica, entonces puede proporcionarse para ella un método en una clase descendiente (ya sea que un método haya sido o no proporcionado en la clase original). Por otra parte, un método debe estar disponible para la operación en la clase que declara la operación, y el método no puede ser eliminado en una clase descendiente. Un método está disponible si es declarado por una clase o heredado de un ancestro. Una operación abstracta debe ser polimórfica (porque no tiene ninguna implementación directa).

Una operación es no polimórfica si se declara para ser una operación concreta.

Si una operación se declara polimórfica en una clase —esto es, si no se declara como una operación concreta puede declararse para ser una operación concreta en una clase descendiente. Esto impide que sea sobrescrita en un descendiente. Una operación concreta no puede declararse polimórfica en una clase descendiente. No puede sobreescribirse a ninguna profundidad.

UML no define las reglas para combinación de métodos, si un método se declara en una clase y es sobreescrito en una clase descendiente. Mecanismos como declarar antes de, después de y alrededor de los métodos, pueden manejarse en un lenguaje-específico usando valores etiquetados. Acciones tales como llamar explícitamente a métodos heredados son, en cualquier caso, dependientes del lenguaje.

Notación

Una operación no polimórfica se declara usando la palabra clave {**leaf**}. De lo contrario, se asume como polimórfica.

Discusión

Una operación abstracta es necesariamente polimórfica. Por otra parte, no podría implementarse en absoluto. Bertrand Meyer llama a esto una operación diferida, porque su especificación se define en una clase pero su aplicación es diferida a subclases. Esto es esencial, probablemente lo

más esencial, en el uso de herencia en diseño y programación. Usando herencia, pueden aplicarse operaciones a los conjuntos de objetos de clases mezcladas. El llamador no necesita conocer la identidad del objeto. Es sólo requisito que todos los objetos conforman una clase del antepasado que define las operaciones deseadas. La clase ancestro no necesita implementar las operaciones. Debe simplemente definir sus firmas. El llamador desconoce la lista uniforme de posibles subclases. Esto significa que esas nuevas subclases pueden agregarse después, sin romper las operaciones polimórficas en ellos. El código fuente que invoca las operaciones no necesita cambios cuando se agregan nuevas subclases. La habilidad de agregar las nuevas clases después de que el código original es escrito es uno de los pilares claves de la tecnología orientada-objetos.

Un uso más problemático del polimorfismo es el reemplazo (eliminación) del método definido en una clase por un método diferente definido en una subclase. Esto es a menudo citado como una forma de compartir, pero es peligroso. La sobreescritura no es incremental, es necesario que todo el método original se reproduzca en el método del hijo, incluso para hacer un cambio pequeño. Este tipo de repetición es propensa a errores. En particular, si el método original se cambia después, no hay ninguna garantía que el método del hijo también se cambie. Hay casos en que una subclase puede usar una aplicación completamente diferente de una operación, pero muchos expertos desecharían tal sobreescritura debido al peligro inherente. En general, los métodos deben ser heredados completamente sin destruir lo definido; en el último caso válido es cuando no hay implementación en la superclase, así que no hay ningún peligro de redundancia o inconsistencia.

Para permitir a una subclase extender la implementación de una operación sin perder el método heredado, la mayoría de los lenguajes de programación proporcionan alguna forma de combinación del método, que usa el método heredado pero permite que el código adicional se agregue a él. En C++, un método heredado debe ser invocado explícitamente por clase y nombre de la operación que empotra la jerarquía de la clase en el código, aunque no es un acercamiento muy robusto. En Smalltalk, un método puede invocar una operación **super**, los cuales causan que la operación sea manejada por el método heredado. Si la jerarquía de la clase cambia, entonces la herencia todavía trabaja, posiblemente con un método de una clase diferente. Sin embargo, el método destruido debe proporcionar una llamada explícitamente a **super**. Los errores pueden ocurrir y ocurren, porque los programadores se olvidan de insertar las llamadas cuando ocurre un cambio. Finalmente, CLOS proporciona un método automático muy general y complicadas reglas de combinación de métodos que pueden invocar varios métodos durante la ejecución de una sola llamada de la operación. La operación total se lleva a cabo en varios fragmentos en lugar de obligándose a ser un solo método. Esto es muy general pero más difícil de manejar para el usuario.

UML no fuerza ningún acercamiento de combinación de métodos. La especificación de UML2 asume que esa resolución de operaciones para comportamiento son especificadas de algún modo, pero no restrinje o proporciona una manera uniforme de especificarlo. La combinación de métodos es un punto de la variación semántica. Cualquiera de estos acercamientos puede usarse. Si el lenguaje de programación es débil en combinación con el método, entonces una herramienta de modelado puede ser capaz de proporcionar ayuda generando el código del lenguaje-programación apropiado o advirtiendo sobre posibles alertas si se usa el método para sobreescribir.

poscondición

Una restricción que debe ser verdad a la realización de una operación.

Semántica

Poscondición es una expresión Booleana que debe ser verdad después de la ejecución de la operación completa. Es una aserción, no una declaración ejecutable. Dependiendo de la forma exacta de la expresión, podría ser posible verificarlo automáticamente por anticipado. Puede ser útil para probar la poscondición después de la operación, pero esto está en la naturaleza de depurar un programa. Se supone que la condición es verdad y algo es un error de la programación. Una poscondición es una restricción en el impulsor de un funcionamiento. Si no está satisfecho, entonces la operación ha sido llevada a cabo incorrectamente.

Véase también invariable, precondición.

Estructura

Una poscondición es una restricción que puede atarse a una acción, una actividad, una operación, un comportamiento o una transición en una máquina de estado de protocolo.

Notación

Acción

Una poscondición se muestra como texto en una nota con la palabra clave «**localPostcondition**».

La nota se ata al símbolo de acción por una línea discontinua (Figura 14.236).

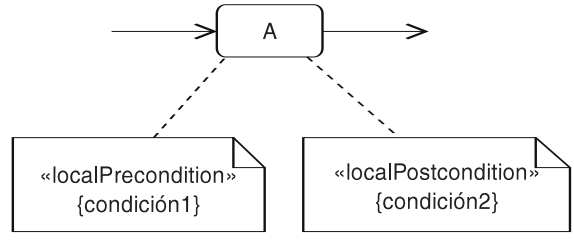


Figura 14.236 Condiciones en acción

Actividad

Una poscondición se muestra como una cadena de texto dentro del límite del nodo de actividad en el lado derecho superior, precedido por la palabra clave «**Postcondition**» (Figura 14.237).

Operación

Una poscondición puede mostrarse como texto en una nota con la palabra clave «**postcondition**».

La nota se ata al cordón del texto de la operación por una línea discontinua (Figura 14.238).

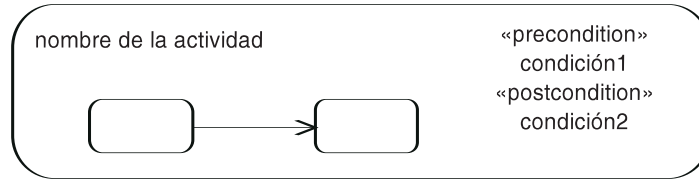


Figura 14.237 Condiciones en actividad

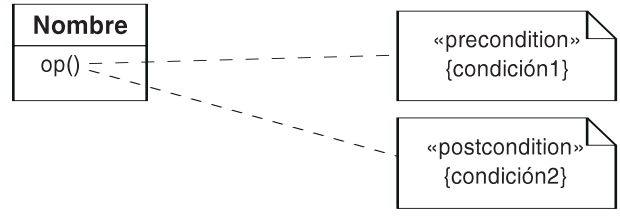


Figura 14.238 Condiciones en operación

Transición en una máquina de estado de protocolo

La flecha de la transición tiene una cadena de texto con la sintaxis:

`[precondición] evento / [la postcondición]`

Esta sintaxis se aplica a las máquinas de estado protocolares, no a las máquinas de estado conductuales (Figura14.239).

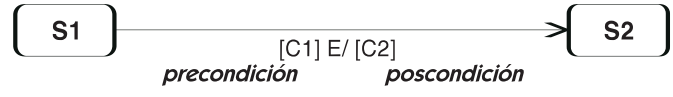


Figura 14.239 Condiciones en el protocolo de transición de una máquina de estados

Ejemplo

La Figura 14.240 muestra una poscondición en una operación de clasificación de arreglos. El nuevo valor del arreglo (a') se relaciona al valor original (a). Este ejemplo se expresa en lenguaje natural estructurado. La especificación en un lenguaje más formal también es posible.

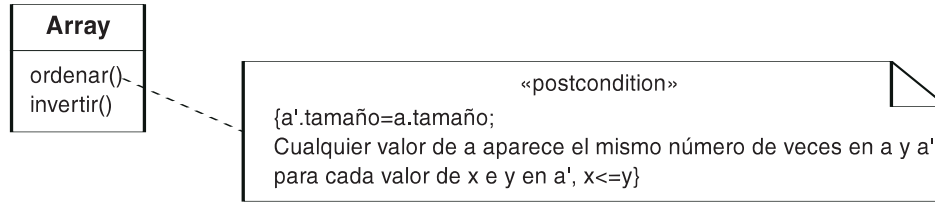


Figura 14.240 Poscondición

precondición

Una restricción que debe ser verdad cuando se invoca una operación.

Véase también poscondición.

Semántica

Una precondición es una expresión Booleana que debe ser verdad cuando se llama a una función. Es responsabilidad del llamador satisfacer la condición. El receptor no debe tener que verificar la condición. Una precondición no es una condición de guarda. Es una condición que debe ser verdad, no una manera de ejecutar un funcionamiento opcionalmente. Como una práctica de implementación, puede ser útil probar la precondición al principio de la operación, pero esto tiene naturaleza de depuración del programa. La condición se supone que es verdad y cualquier otra cosa es un error de programación. Si la condición no está satisfecha, no se puede realizar ninguna afirmación sobre la integridad del funcionamiento o el sistema. Es responsable de prever posibles fallos. En la práctica, que el receptor verifique explícitamente las precondiciones puede descubrir muchos errores.

Véase también invariante, poscondición.

Estructura

Una precondición es una restricción que puede asociarse a una acción, una actividad, una operación, un comportamiento o una transición en el protocolo de una máquina de estados.

Notación

Una precondición tiene la misma notación que una poscondición con la sustitución apropiada de palabra clave. *Véase* poscondición para los diagramas.

Acción

Una precondición se muestra como texto en una nota con la palabra clave «**localPrecondition**».

La nota se une al símbolo de acción por una línea discontinua (Figura 14.236).

Actividad

Una precondición se muestra como una cadena de texto precedida por la palabra clave «**Precondition**».

La cadena de texto se pone dentro del límite de símbolo de la actividad en la esquina superior derecha (Figura 14.237).

Funcionamiento

Una precondición se puede mostrar como texto en una nota con la palabra clave «**precondition**».

La nota se une a la cadena de texto de la operación por una línea discontinua (Figura 14.238).

Transición en una máquina estatal protocolar

La flecha de la transición tiene una cadena de texto con la sintaxis:

[**precondición**] **evento** / [**postcondición**]

Esta sintaxis se aplica a las máquinas de estado protocolares, no a las máquinas de estado conductuales (Figura 14.239).

Ejemplo

La Figura 14.241 muestra una precondición en un operador de producto de matrices.

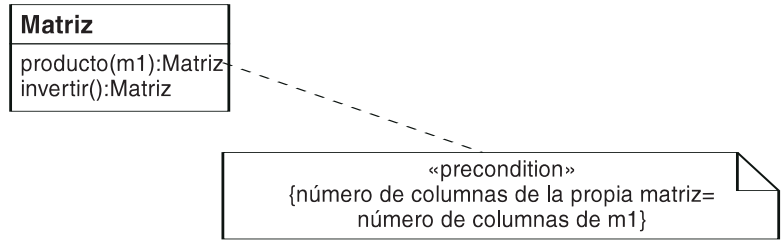


Figura 14.241 Precondición

principio de capacidad de sustitución

El principio mediante el cual, a partir de una declaración de una variable o parámetro, cuyo tipo se declara como X, cualquier instancia de una clase que sea descendiente de X puede ser usada como valor real, sin violar la semántica de la declaración y su uso. En otras palabras, una instancia de un elemento descendiente puede ser sustituida por una instancia de un elemento antepasado.

(Atribuido a la profesora de MIT Bárbara Liscov.)

Véase también generalización, implementación de la herencia, herencia, polimorfismo, procedimiento.

Discusión

El propósito es asegurar que las operaciones polimórficas funcionen libremente. La relación de generalización soporta sustitución, mediante la sobrecarga, y la redefinición de características concretas se evita.

La consecuencia del principio de sustitución es que un hijo no puede eliminar ni renunciar a las propiedades de su padre. De lo contrario, el hijo no sería sustituible en una situación en la que se declara el uso de su padre.

Este no es un principio de lógica, sino más bien una regla práctica de programación, que ofrece un mayor grado de encapsulamiento. Lenguajes no tradicionales, basados en reglas de ejecución tales como la clasificación dinámica, pueden encontrar este principio menos útil.

principio de sustitución de Liskov

Véase principio de capacidad de sustitución.

privado

Un valor de visibilidad que indica que el elemento dado no es visible fuera de su propio espacio de nombres, incluso a los descendientes del espacio de nombres.

procedimiento

Un algoritmo expresado en una forma que puede ejecutarse, normalmente con parámetros que pueden ser proporcionados en invocación.

Semántica

El procedimiento es un término informal para un algoritmo parametrizado ejecutable, normalmente expresado como una lista de pasos que se ejecutan según las reglas de un lenguaje dado.

UML no restringe la forma que los procedimientos pueden tomar. Una máquina de estados podría ser considerada un procedimiento y cualquier procedimiento podría ser diseñado por una máquina de estados, UML ya trata con los problemas semánticos que tienen los procedimientos.

UML tiene la metaclase **Behaviour**, que es algo más general que un procedimiento en el que se permiten especificaciones imprecisas así como especificaciones totalmente ejecutables.

proceso

1. Una unidad pesada de concurrencia y ejecución en un sistema operativo. *Véase* hilo que incluye unidades pesadas y unidades ligeras de proceso. Si es necesario, puede hacerse una distinción de implementación basada en estereotipos.

2. Un proceso de desarrollo software —el camino y pautas por los cuales se desarrolla un sistema.

3. Para ejecutar un algoritmo o para gestionar algo dinámicamente.

proceso de desarrollo

Conjunto de guías y de actividades de trabajo parcialmente ordenadas proporcionadas con la intención de producir software de una forma controlada y reproducible. El propósito de un proceso de desarrollo de software es asegurar el éxito y la calidad de un sistema terminado.

Véase también etapas de modelado.

Discusión

UML es un lenguaje de modelado, no un proceso, y su propósito es describir modelos que se pueden generar desde varios procesos de desarrollo. Por estandarización, es más importante describir los artefactos resultantes de un desarrollo que el proceso que se sigue para su producción. Esto es debido a que hay muchas formas correctas de construir un modelo, y un modelo terminado se puede utilizar sin saber cómo se hizo. Sin embargo, la intención de UML es dar soporte a un amplio rango de procesos.

Para obtener detalles de un proceso de desarrollo iterativo, incremental, guiado por casos de uso y centrado en la arquitectura, respaldado por los autores de este libro, *véase* [Jacobson-99].

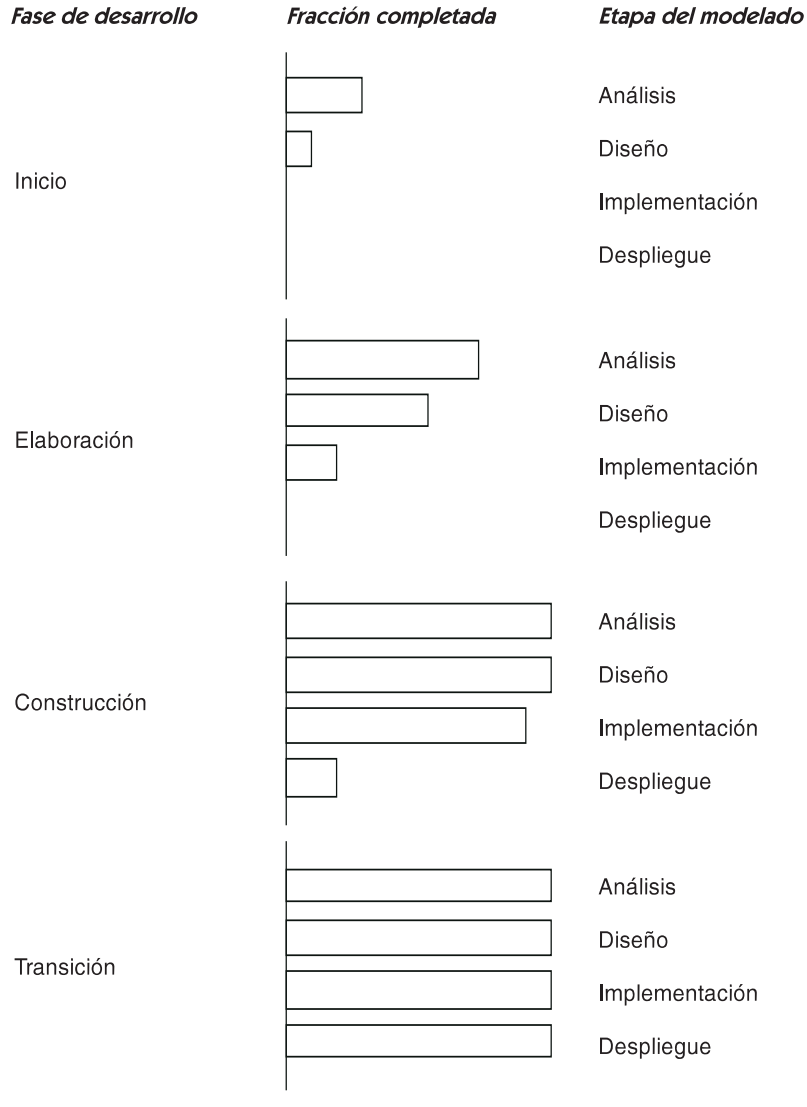


Figura 14.242 Progreso después de cada fase de desarrollo

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. Las fases son secuenciales dentro de una versión de la 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 hasta el despliegue, cada uno a su propio ritmo. Aunque las fases de desarrollo y las etapas de modelado no están sincronizadas, hay una correlación entre ellas. En las fases de desarrollo tempranas y en las primeras iteraciones de una fase, hay más énfasis en las etapas del modelo iniciales.

La Figura 14.242 muestra el equilibrio de esfuerzo entre las sucesivas fases e iteraciones. Durante el inicio la atención se centra principalmente en el análisis, con un esqueleto de elementos progresando hacia el diseño y la implementación durante la elaboración. Durante la construcción y la transición todos los elementos deben ser llevados a su finalización.

process (estereotipo de Componente)

Un componente basado en transacciones.

Véase clase activa, proceso, hilo.

propiedad

Un rasgo estructural de un clasificador. En particular, los atributos y extremos de la asociación son propiedades. Una parte estructurada en un clasificador estructurado también es una propiedad.

Vease también asociación, extremo de la asociación, atribuir, redefinición (propiedad).

Semántica

El constituyente estructural principal de asociaciones y clasificadores es una descripción de valores que ocurren en instancias de los elementos. En clasificadores, las propiedades estructurales se llaman atributos; representan valores que ocurren en instancias de clasificadores.

En asociaciones, se llaman propiedades estructurales los extremos de la asociación; representan valores que son conectados por asociaciones. Una asociación es una relación entre clasificadores, en la que ninguno es el clasificador dominante. Un atributo es una relación entre los clasificadores, en la cual el clasificador es el dueño y el otro aparece en el atributo. Es posible describir la misma relación como un atributo, una asociación, o ambos simultáneamente.

Estructura

Una propiedad estructural es poseída por un clasificador o una asociación. Si se posee por un clasificador, es un atributo del clasificador y es navegable del clasificador al valor contenido en el atributo. Un atributo puede planearse adicionalmente como un extremo de la asociación que no es poseído por la asociación. Si un extremo es poseído por una asociación puede o no puede ser navegable. La diferencia semántica entre un atributo y un extremo de asociación es pequeña y una relación dada puede diseñarse como un atributo, una asociación o ambos simultáneamente. La distinción habilita diferentes estrategias empaquetando: Los atributos deben ser manejados

con sus propios clasificadores, considerando que pueden agregarse fines de la asociación libremente sin modificar los clasificadores que participan.

Una propiedad estructural tiene las escenas siguientes. *Véanse* las entradas individuales para más información.

calificador	Una lista de atributos usada como seleccionadores para escoger los objetos dentro de un conjunto de valores. Si un calificador está presente, entonces la multiplicidad reprime la cardinalidad del conjunto de valores e incluso los valores del calificador; la multiplicidad inhábil se asume que es mucho menos específica, por otra parte. *Véase* calificador.
derivado	Una bandera que especifica si la propiedad es calculable de otros valores y por consiguiente no representa un grado de libertad. El propio cómputo debe especificarse como una restricción, si se desea. Una propiedad derivada es a menudo de sólo lectura; si es modificable, la semántica de actualización de su valor es específica de la implementación (es mejor evitarlo en la mayoría de los modelos).
estática	Una escena Booleana. Si es verdadero, el valor de la propiedad es compartido por todas las instancias del clasificador; alternadamente, puede que sea considerada una propiedad del propio clasificador. Si es falso cada caso del clasificador tiene su propio valor de propiedad. El valor predeterminado es falso (no estático).
multiplicidad	El posible número de valores del atributo o fin de asociación que puede existir simultáneamente en un objeto que tenga la propiedad. Se especifica como un rango entero. El valor más común para atributos es “exactamente uno” denotando un atributo escalar. El valor “cero o uno” denota una propiedad con un valor optativo. Un valor perdido es discernible de cualquier valor en el dominio de la propiedad tipo. (En otros términos, la ausencia de un valor es diferente del valor cero o nulo.) Si el límite superior es mayor que uno, la clasificación y escenas de singularidad son pertinentes. *Véase* multiplicidad.
mutabilidad	Si el valor de la propiedad puede cambiar después de la inicialización. La opción es **modificable** o **solo lectura**. El valor predeterminado es **modificable**. La definición de inicialización es vaga y probablemente sea implementación-dependiente. Sólo una propiedad navegable puede ser **solo lectura**.
navegabilidad	Una bandera que indica si es posible cruzar una asociación para obtener el objeto o conjunto de objetos para un fin de asociación con un valor o tupla de valores de todos los otros fines. El valor predeterminado es verdadero (navegable). Un

atributo es siempre navegable. Una asociación puede ser navegable aún cuando su fin no sea poseído por clases.

nombre — El nombre de la propiedad, es decir, el nombre del atributo o fin de asociación, una cadena identificadora. Este nombre identifica un atributo dentro de un clasificador y un fin de asociación dentro de una asociación (posible ambos para una propiedad dada). El nombre debe ser único dentro de una asociación y también entre directo y heredado y elementos de propiedad (atributos y fines de asociación visibles para la clase) de la clase de fuente. El mismo nombre puede usarse en redefiniciones dentro de los clasificadores descendientes, pero la especificación de UML debe ser consultada para las restricciones en uso.

nombre de rol — El nombre del fin de asociación, una cadena del identificador. El término *rolename* es informal.

ordenando — Si (y potencialmente cómo) el conjunto de objetos relacionados es pedido, una enumeración con los valores {**unordered**, **ordered**}. Parte de la especificación de multiplicidad.

redefinición — Un fin de asociación puede redefinir una propiedad definida en un clasificador del antepasado o asociación. La propiedad redefinida puede o no puede tener el mismo nombre y visibilidad. *Véase* redefinición.

singularidad — Si la colección de objetos relacionada con un objeto dado puede contener duplicados. Si es verdadero, ningún duplicado se permite y la colección es un conjunto (si no ordenado) o un conjunto pedido (si ordenado); si es falso, se permiten duplicados y la colección es una bolsa (si no ordenado) o una lista (si ordenado). Parte de la especificación de multiplicidad.

subconjunto — Una propiedad puede marcarse como un subconjunto de otra propiedad especificada. El conjunto de valores asociado con un objeto por la propiedad debe ser igual que un subconjunto del conjunto de valores asociados con el objeto por la propiedad especificada. Por ejemplo, esto permite definir una asociación y, a continuación, dividida en las asociaciones más especializadas, tal como una asociación del empleado en asociaciones de obrero y gerente. Una propiedad del subconjunto puede tener un tipo más restrictivo y multiplicidad que la propiedad original. *Véase* subconjunto.

tipo — El tipo de valor indicado por la propiedad. Como un atributo, éste es el tipo de valor del atributo; como fin de una asociación, éste es el tipo del objeto enlazado.

tipo de agregación — Si el juego de valores es un agregado o compuesto; una enumeración con los valores {**none**, **shared**, **composite**}. Si el valor no es ninguno, la asociación se llama agregación.

	Una agregación compartida permite más de "un todo" para una "parte" dada y no implica gestión de "la parte" por el "todo". Una agregación compuesta (composición) permite solamente un solo entero para una parte dada e implica dirección de la parte por el todo. El valor predeterminado no es **ninguno**. Sólo una asociación binaria puede ser una agregación o composición; sólo un fin puede ser un agregado de cualquier tipo y las otras marcas del fin (valor de **ninguno**) las partes. Si ambos fines tienen el valor **ninguno**, la asociación no es una agregación. *Véase* agregación.
unión derivada	Una bandera que especifica si la propiedad (y, en una asociación, la propia asociación) está definida como la unión de todas las propiedades que son especificadas como subconjuntos. Si es verdadero, entonces cualquier instancia de una asociación necesariamente es una instancia de las asociaciones del subconjunto. El ajuste derivado es verdadero para una unión derivada.
valor predefinido	Una expresión para el valor inicial de un atributo evaluado y asignado al atributo cuando el propio clasificador es instanciado. Esta escena es optativa. Si está ausente, ningún valor inicial se especifica (pero alguna otra parte del plan, como una operación de creación, puede especificar un valor).
visibilidad	Si la propiedad es accesible a otros clasificadores. En una asociación, la visibilidad se pone en el fin designado, es decir, el fin opuesto del clasificador que posee la propiedad. Cada dirección de transversal a través de una asociación tiene su propio valor de visibilidad.

Notación

El extremo de un camino de la asociación se conecta al borde del rectángulo del símbolo de clase correspondiente. Se muestran propiedades de extremo de asociación como adornos delante o cerca del final del camino al que une a un símbolo del clasificador (Figura 14.147). *Véase* asociación.

El ajuste de atributos se muestra como subcadenas dentro de la cadena del atributo entero. *Véase* atributo para la sintaxis de cadena de atributo.

Se muestran propiedades de texto dentro de las abrazaderas al final de una cadena del atributo o casi al fin de la asociación. Las etiquetas de texto en el fin de una asociación se ponen cerca del fin de la asociación para que no puedan confundirse con etiquetas en la asociación entera, pero no hay ningún posicionamiento relativo especificado de etiquetas múltiples del texto. Pueden moverse alrededor del diagrama para hacer una limpieza.

La lista siguiente es un resumen breve de adornos para cada escena modelada.

Consulte los artículos individuales para más detalles.

calificador	Un rectángulo pequeño entre el fin del camino y la clase de la fuente en un transversal. El rectángulo contiene uno o más atributos de los calificadores de asociación.

derivado	Para un atributo derivado, mostrado como un slash (/) delante del nombre del atributo. Para una asociación derivada, mostrado como un slash delante del nombre de la asociación (un slash por sí mismo sólo está permitido si no hay ningún nombre de asociación).
multiplicidad	Para un atributo, el intervalo de multiplicidad en anaqueles del cuadrado después del nombre del tipo, por ejemplo, **vértice**: **Punto [3..*]**. En ausencia de una cadena, el valor predeterminado es exactamente uno. Para un fin de asociación, una etiqueta de texto cerca del fin del camino, en la forma **`min..max`** (sin cualquier abrazadera). En ausencia de una etiqueta, la multiplicidad es normalmente no especificada, pero otras convenciones a veces son adoptadas.
mutabilidad	La propiedad del texto {**addOnly**} cerca del fin designado, normalmente omitido para {**modificable**}, pero permitido para énfasis.
navegabilidad	Para un atributo, la navegabilidad siempre es verdadera y no necesita que se muestre. Para un fin de asociación, una punta de flecha en el fin de un camino, muestra la navegabilidad en esa dirección. Una pequeña X al final de un camino muestra la no navegabilidad en esa dirección. A menudo la convención siguiente es adoptada: Una flecha en un fin y ninguna flecha en los otros fines indica no navegabilidad en los fines sin marca. Si ningún fin tiene una punta de flecha, la asociación es navegable en todas las direcciones (porque hay poca necesidad de asociaciones que sean no navegables en cualquier dirección). Una convención diferente es la que desmarcó que los fines tienen la navegabilidad no especificada. En cualquier paquete, la misma convención debe usarse para un diagrama entero y, preferentemente, para todos los diagramas en un modelo.
nombre	Para un atributo, el nombre se muestra como una cadena de texto precediendo el tipo (if any). Si se muestra un solo nombre, es el nombre del atributo y el tipo es no especificado. Para el fin de una asociación, el nombre se muestra como una cadena de texto cerca del fin del camino de la asociación unido al símbolo del clasificador designado.
nombre de rol	Un nivel de texto cerca del fin de la etiqueta designada.
ordenar	La cadena de propiedad {**ordered**} en la cadena del atributo o cerca del fin designado indica un conjunto pedido; la propiedad {**seq**}, {**sequence**}, {**list**} u {**ordered bag**} indica una bolsa pedida, es decir, una lista ordinaria con posibles duplicados. La ausencia de una marca indica un juego no ordenado.
redefinición	La cadena de propiedad {redefined **`original`**} indica una redefinición de propiedad **`original`**.

singularidad	La cadena de propiedad {**bag**} en la cadena de atributo o casi al fin de la etiqueta designada indica una colección (bolsa). La cadena de propiedad {**list**}, {**ordered bag**}, {**seq**} o {**sequence**}, indica una colección no única de órdenes (sucesión). La ausencia de una marca indica un conjunto.
subconjunto de parámetros	La cadena de propiedad {subsets **`original`**} indica un subconjunto de parámetros de propiedad **`original`**. Una lista coma-separada de nombres es permitida.
tipo	Para un atributo, el tipo se muestra como una cadena de texto después de dos puntos (:). Para una asociación, la línea de camino de asociación está unida al rectángulo que simboliza el tipo.
tipo de agregación	Un diamante pequeño al final de la asociación que representa el entero, un diamante hueco para un agregado compartido, un diamante lleno para un agregado de la composición. Ningún adorno para una parte o una no agregación. La cadena {**shared**} o {**composite**} también puede ponerse cerca de un fin o en una cadena de atributo. Allí no parece ser una manera de marcar un atributo que representa una parte sin incluir una asociación.
unión derivada	La cadena de propiedad {**union**} se pone en la cadena de atributo o cera del fin de la asociación.
valor predefinido	Para un atributo, mostrado como una expresión de cadena de texto seguida de un signo igual. Un valor predefinido para una asociación raramente se muestra.
visibilidad	El símbolo (+ # – ~) de visibilidad prefijado para el nombre de rol.

Discusión

Observe que el término *propiedad* también se usa en un sentido general para significar cualquier valor ligado a un elemento modelo.

protegido

Un valor de visibilidad que indica que el elemento dado es visible fuera de su propio espacio de nombres sólo a los descendientes del espacio de nombres.

proveedor

Un elemento que proporciona servicios que pueden ser invocados por otros. Contraste: cliente. En notación, el proveedor aparece junto a la punta de una flecha de dependencia punteada.

Véase dependencia.

proyección

Una correspondencia de un conjunto para un subconjunto de él. La mayoría de los modelos y diagramas son proyecciones del conjunto completo de información que está potencialmente disponible.

pseudoestado

Un vértice en una máquina de estado que tiene la forma de un estado pero tiene comportamiento especial.

Semántica

Un pseudoestado es un estado transitorio que estructura los detalles de una transición. Cuando el pseudoestado está activo, una máquina de estado no ha completado su transición y no procesará eventos. Los pseudoestados se usan para encadenar segmentos de transición, y una transición no implica una transición automática a otro estado, sin requerir un evento.

Pseudoestados incluyen las opciones de punto de la entrada, punto de salida, horquilla, estado de historia, estado inicial, ensamblador, unión y terminación.

Un estado final no es un pseudoestado. Puede permanecer activo cuando una máquina de estado tiene completado finalización, pero tiene las restricciones en las transiciones que pueden partir de él.

Notación

Cada clase de pseudoestado tiene su propia notación.

público

Un valor de visibilidad que indica que el elemento dado es visible fuera de su propio espacio de nombres.

puerta

Punto de conexión en una interacción o en un fragmento de interacción para un mensaje que viene o va fuera de la interacción o del fragmento.

Semántica

Una puerta es un parámetro que representa un mensaje que cruza el límite de una interacción o de un fragmento de interacción. Los mensajes dentro de la interacción se pueden conectar a la puerta. Si la interacción se referencia dentro de otra interacción, los mensajes se pueden conectar a sus puertas, Cuando se ejecuta la interacción, los mensajes conectados a través de puertas se distribuirán de manera adecuada.

Notación

En un diagrama de secuencia, una puerta es simplemente un punto del límite del diagrama de secuencia o del fragmento de interacción que se conecta a la cabeza o a la cola de una flecha de mensaje. El nombre de la flecha de mensaje conectada es el nombre de la puerta (Figura 14.243).

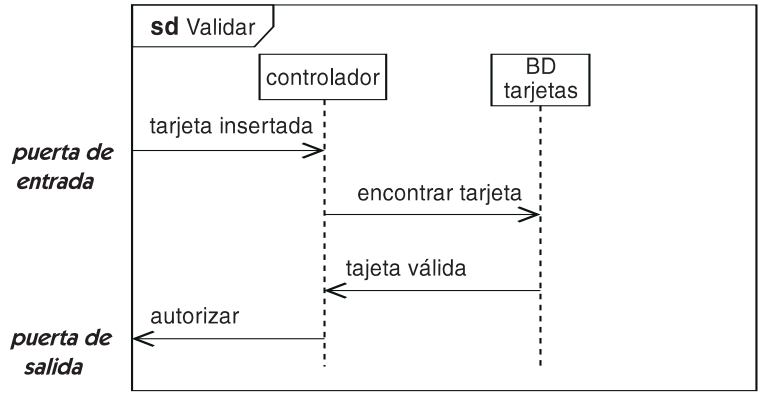


Figura 14.243 Puertas

puerto

Un rasgo estructural de un clasificador que encapsula la interacción entre los contenidos del clasificador y su entorno.

Véase interfaz, clasificador estructurado, parte estructurada.

Semántica

Un puerto es un punto de conexión entre un clasificador y su entorno. Las conexiones del mundo externo se hacen a los puertos según sean requeridos por las interfaces declaradas en un puerto. Los clientes externos del clasificador no tienen la visibilidad o interacción directa con los contenidos o la implementación del clasificador. Cuando el clasificador se implementa, se conectan puertos en partes interiores a sus puertos externos de acuerdo con las interfaces declaradas. El uso de permisos de los puertos en el interior de la estructura de un clasificador es modificado sin afectar a los clientes externos, con tal de que se apoyen las interfaces en los puertos correctamente. Los puertos permiten partes encapsuladas para ser “clasificados juntos”, para formar la implementación de un clasificador.

El comportamiento de un puerto se especifica por sus interfaces proporcionadas e interfaces requeridas.

Se crean puertos como parte de la instanciación de su clasificador y se destruyen con él. No pueden ser creados ni pueden ser destruidos durante la vida de un objeto.

Un puerto puede especificarse con multiplicidad. Si la multiplicidad no es un entero las cardinalidades deben especificarse cuando el clasificador es instanciado. Se crea una instancia de

puerto separada por cada valor en el conjunto de multiplicidad. Si la multiplicidad está ordenada, los puertos pueden ser identificados por sus posiciones del índice.

Una llamada a un objeto puede hacerse usando un puerto específico, si la interfaz es proporcionada para soportar el puerto de la operación. El comportamiento para el clasificador puede determinar qué puerto recibió una llamada y puede usar esta información en el método o seleccionar un método para llevar a cabo la llamada. Si un puerto es un puerto de comportamiento, se llevan a cabo las llamadas por el comportamiento del clasificador. Si un puerto no es un puerto de comportamiento, se remiten llamadas a la parte interior que posee el puerto para la implementación.

Estructura

Un clasificador puede tener cualquier número de puertos. Cada puerto tiene un conjunto de interfaces proporcionadas y un juego de interfaces requeridas. Una interfaz proporcionada declara un servicio que el clasificador prepara para proporcionar a un cliente anónimo fuera. No es necesario para el cliente tener una relación predefinida con el clasificador. Una interfaz requerida declara un servicio del que el clasificador requiere un proveedor anónimo fuera. Otro clasificador conectado al puerto puede requerir servicios del dueño del puerto, pero también debe prepararse para proporcionar los servicios pedidos al dueño.

Pueden declararse dos propiedades de aplicación en puertos:

puerto de servicio	Si falso, el puerto sólo se usa en la aplicación interior del clasificador y no es requerido por su ambiente, por consiguiente puede anularse o puede alterarse sin afectar el uso del clasificador. El valor predeterminado es verdad (el puerto se necesita).
puerto de comportamiento	Si verdadero, las demandas en el puerto se llevan a cabo directamente por la aplicación de comportamiento declarada (como una máquina de estado o procedimiento) del clasificador. Si falso, el puerto es un puerto de la comisión en el que debe conectarse a un puerto una parte interior y una demanda externa se transmitirá a la parte interior para la aplicación. Puertos de la comisión permiten clasificadores más grandes para ser “alambrados juntos” de partes más pequeñas sin la especificación de comportamiento adicional, pero en el futuro al más profundo comportamiento nivelado anidado debe ser llevado a cabo directamente.
visibilidad	Un puerto puede ser público, privado o protegido.

No hay ninguna asunción sobre cómo un puerto se lleva a cabo. Podría llevarse a cabo como un objeto explícito o podría ser meramente un concepto virtual que no hace explícitamente aparecer en la aplicación.

Si se atan varios conectores a un puerto del interior, es un punto de variación semántica si se remitirán demandas en todos los conectores o si el conector se seleccionará de algún modo para cada demanda.

Notación

Puertos declarados en clasificadores estructurados

Un puerto se muestra como un cuadrado pequeño que monta el límite de un rectángulo del clasificador.

El nombre del puerto se pone cerca del cuadrado. El cuadrado puede ponerse en el interior del rectángulo del clasificador para indicar un puerto con visibilidad restringida, como un puerto de servicio; éstos deben ser raros, porque el propósito principal de los puertos es encapsular comunicación con el ambiente.

El tipo de un puerto puede mostrarse siguiendo una coma, usando la sintaxis:

nombre: **Tipo** [**multiplicidad**]

Las varias partes de esta sintaxis son optativas. La multiplicidad puede usarse para declarar un conjunto de puertos del mismo tipo. Todos los puertos de un tipo dado tienen las mismas interfaces.

En lugar de declarar el puerto por tipos, pueden mostrarse las interfaces para el puerto usando símbolos de la interfaz. Una interfaz proporcionada es mostrada por un círculo pequeño conectado al cuadrado del puerto por una línea. Una interfaz requerida es mostrada por un semicírculo pequeño conectado al cuadrado del puerto por una línea. El nombre de una interfaz se pone cerca del círculo, semicírculo o línea. Las interfaces múltiples de cualquier tipo pueden atarse al mismo puerto.

Ejemplo

La Figura 14.244 muestra una declaración de la cámara de vídeo con sus puertos. La cámara de vídeo tiene dos puertos de entrada: una entrada de TV y una entrada del micrófono. Éstos se muestran como interfaces proporcionadas, porque la cámara de vídeo acepta sus entradas y actos en ellos. La cámara de vídeo tiene un tipo de puerto de salida, una salida audio, pero hay dos casos de este puerto en cada caso de la cámara de vídeo. Esto se muestra como una interfaz requerida porque la cámara de vídeo requiere los servicios de otro dispositivo (como un amplificador o un auricular) para procesar las salidas.

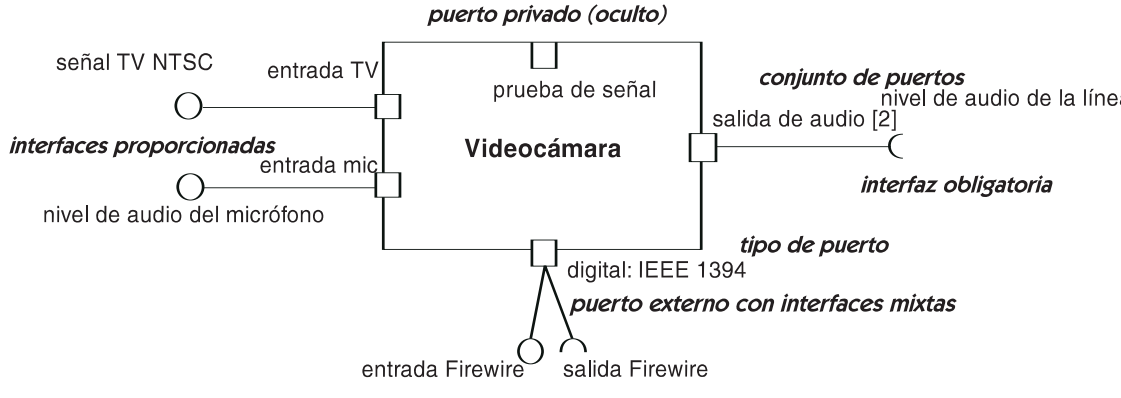


Figura 14.244 Clasificador estructural con puertos

La cámara de vídeo también se muestra con un puerto mezclado, el puerto IEEE 1394 (Firewire). Este puerto maneja al mismo tiempo la entrada y la salida. Esto se muestra conectando ambos a la interfaz provista y la requerida para ello. (Este acercamiento realmente no es satisfactorio para los protocolos complejos. *Véase* la discusión para las recomendaciones.)

Usando puertos dentro de los clasificadores estructurados

Un clasificador con puertos puede usarse como parte de otro clasificador estructurado.

Cada uso del clasificador como una parte de un clasificador estructurado es mostrado por un rectángulo dentro la declaración del clasificador estructurado. El rectángulo interior tiene un cuadrado pequeño para cada puerto en la declaración de su tipo. Si uno de los puertos se declarara con multiplicidad, el cuadrado se muestra para cada caso del puerto en la parte particular. Un número del índice puede usarse en anaqueles para distinguir los casos múltiples de la misma declaración del puerto:

`nombre` [**`entero`**]

Pueden conectarse los puertos en las partes interiores a los puertos en otras partes interiores o a los puertos de la comisión (puertos no de comportamiento) en el límite del clasificador estructurado.

Los conectores son mostrados por caminos de línea continua. Un puerto de la comisión y un puerto interior conectado a él deben ser del mismo tipo, porque una demanda recibida en uno es también recibida en el otro. Dos puertos interiores conectados juntos deben ser de tipos complementarios, porque una demanda enviada a través de uno se realiza por el otro. Dos tipos son complementarios si los servicios requeridos de cada uno son un subconjunto de los servicios del otro. En el caso más simple, las interfaces de uno serán lo mismo que la interfaz requerida del otro.

Un puerto de comportamiento en el clasificador estructurado (uno por el que se lleva a cabo directamente el clasificador estructurado) es mostrado por una línea de puerto a un símbolo de estado pequeño (un rectángulo con esquinas redondeadas). Esto significa sugerir a una máquina de estado, aunque también se permiten otras formas de aplicación de comportamiento.

La Figura 14.245 muestra la anotación para los puertos interiores y conectores.

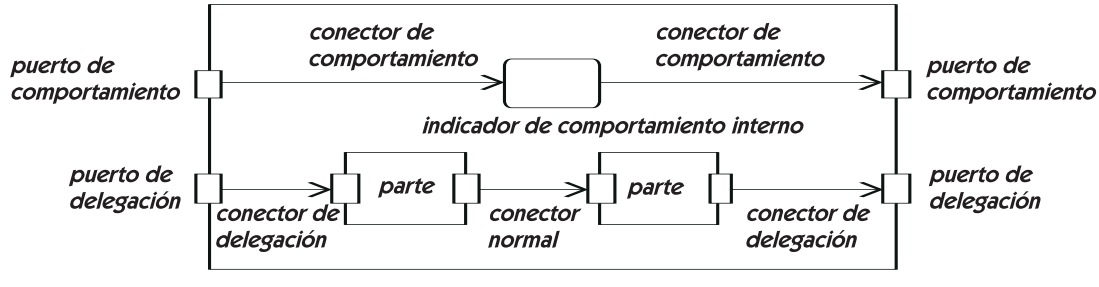


Figura 14.245 Puertos internos en un clasificador estructural

Un puerto puede ser conectado por conectores múltiples a partes u otros puertos. El significando de conexiones múltiples es un punto de la variación semántica: o un mensaje es copiado a cada conector o un mensaje va exactamente a un conector. Esta capacidad sólo es útil dentro de un ambiente de aplicación al que la especificación provee el apoyo.

Ejemplo

La Figura 14.246 muestra un programa formato de libro (grandemente simplificado) construido de módulos más pequeños. Acepta un texto crudo sin una mesa de contenidos o pone en un índice y produce un libro estructurado con una mesa de contenidos delante y un índice atrás. La entrada original se duplica en 3 copias. Uno se pasa a un componente que extrae las entradas para la mesa de contenidos, uno se pasa a un componente que extrae el índice entradas y el tercero se pasa inalterado a la última combinación. Las listas de entradas se estructuran y, a continuación, pasan al módulo del combinador que los fusiona en un solo texto de salida.

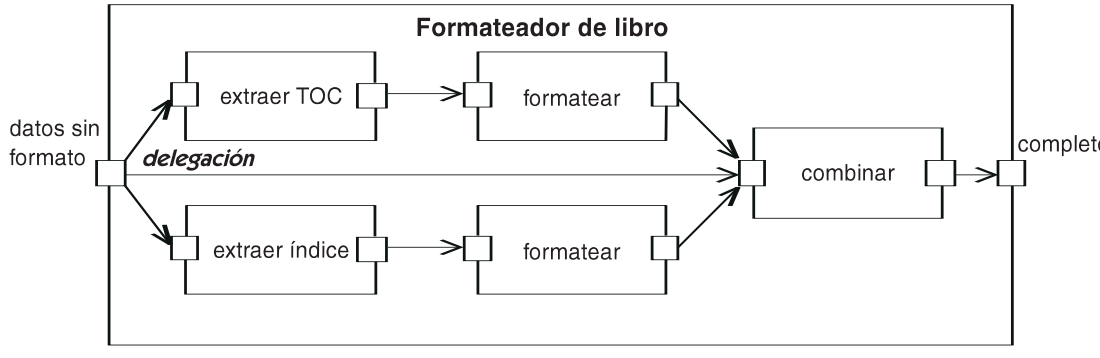


Figura 14.246 Puertos y conectores dentro de una clase estructurada

Historia

Los puertos y los clasificadores estructurados en UML2 son una mejora mayor que incrementa enormemente la habilidad de construir modelos estructurados.

Discusión

La especificación de interfaces, tanto proporcionadas como requeridas, es un concepto restrictivo basado en el pensamiento de la programación tradicional de interfaces como conjuntos de operaciones llamadas independientemente y respondidas. Mientras esto es adecuado en casos simples, se construyen a menudo sistemas grandes formados por partes que actúan recíprocamente según sucesiones bien definidas de mensajes. Una sucesión de interacción bien-definida se llama protocolo. Los protocolos pueden ser especificados por gramáticas, interacciones o máquinas de estado de protocolo. UML2 no apoya la declaración de protocolos explícitamente, pero el tipo de un puerto puede identificarse con un protocolo. Para mostrar un protocolo como un símbolo de la interfaz, la distinción entre interfaces provistas y requeridas puede simplemente ignorarse (porque no tiene sentido en sistemas complejos) y los símbolos de rótula pueden usarse para representar las mitades complementarias de un protocolo complejo, sin asunción de que el protocolo puede reducirse a un conjunto de operaciones simples.

puerto complejo

Puerto que tiene múltiples interfaces.

Semántica

Un puerto puede tener múltiples interfaces, tanto interfaces proporcionadas como interfaces requeridas.

Notación

Un puerto complejo se representa conectando varios símbolos de interfaz al símbolo de puerto. Si se desea, el símbolo de puerto se puede dibujar como un rectángulo en vez de como un cuadrado, de forma que los símbolos de interfaz se puedan conectar más fácilmente. *Véase* la Figura 14.247.

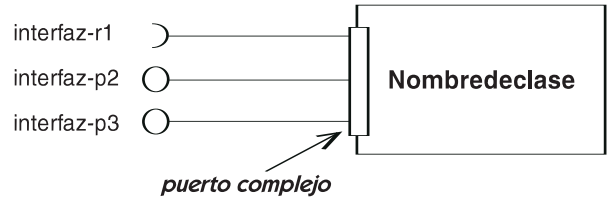


Figura 247 Puerto complejo

punto de conexión

Referencia a un punto de entrada o de salida de una máquina de estados, utilizada con una referencia a la máquina de estados por una submáquina de estados dentro de otra máquina de estados.

Semántica

Una máquina de estados se puede definir con la intención de que sea referenciada desde otras máquinas de estados como un tipo de subrutina de máquina de estados. Tal referencia es una submáquina de estados. Si siempre se entra a la submáquina a través de su estado inicial y siempre se sale de ella a través de su estado final, no se necesitan definir interfaces adicionales. Sin embargo, a veces una submáquina tiene varios puntos de entrada o de salida que deben ser visibles para las máquinas de estados desde las que es referenciada. Un punto de entrada es un lugar adicional en el que la máquina de estados puede empezar la ejecución. Un punto de salida es un lugar adicional en el que una máquina de estados puede terminar su ejecución. Los puntos de entrada y de salida son pseudoestados en la máquina de estados.

Un punto de conexión es una referencia a un punto de entrada o a un punto de salida de una máquina de estados cuando se utiliza dentro de otra máquina de estados. Cuando la máquina de estados es referenciada por una submáquina de estados en otra máquina de estados, las transiciones pueden conectar a puntos de conexión con nombre en la submáquina de estados que

corresponden a los puntos de conexión definidos en la máquina de estados referenciada. Aunque los puntos de conexión representan flujo de control, realizan el mismo tipo de parametrización que los parámetros y argumentos en los procedimientos.

Véase submáquina de estados.

Notación

Un punto de entrada se representa como un pequeño círculo vacío. Un punto de salida se representa como un pequeño círculo con una 'X' dentro. El nombre del punto de entrada o de salida se coloca cerca del círculo. El punto de entrada o de salida se coloca dentro del borde exterior de la máquina de estados a que se aplica. Un punto de entrada puede aparecer en cualquier lugar en el que pueda aparecer un estado inicial, y un punto de salida puede aparecer en los mismos lugares en que pueda aparecer un estado final.

Un punto de conexión se representa como un pequeño círculo vacío o como un pequeño círculo con una 'X' dentro; se dibuja en el borde de la caja redondeada que representa una submáquina de estados (Figura 14.248). Dentro de la caja redondeada está el nombre de la submáquina de estados y dos puntos seguidos por el nombre de la máquina de estados referenciada. Las transiciones de la máquina de estados interna pueden conectar a los puntos de conexión (con un punto de entrada como destino y un punto de salida como origen).

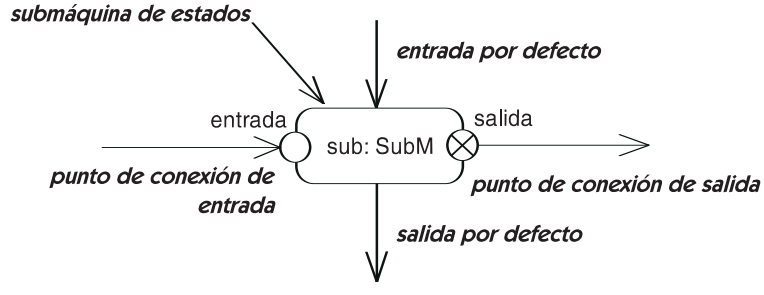


Figura 14.248 Puntos de conexión

Una notación alternativa elimina los círculos y representa el punto de conexión colocando en la flecha de transición una etiqueta de texto de la forma:

(via **`nombre-de-conexión`**)

Como la notación de los círculos es clara e intuitiva, mejor se evita la notación alternativa.

Historia

UML1 tiene un concepto de estados matriz para representar entradas y salidas a las submáquinas. La nueva notación de punto de conexión es más clara y es paralela a la notación similar para los parámetros.

punto de entrada

Dentro de un estado, un pseudoestado visible externamente que puede ser el destino de una transición externa. Especifica un estado interno dentro del estado que se convierte en destino efectivo de cualquier transición.

Semántica

Un punto de entrada es un mecanismo de encapsulación. Permite la definición de estados iniciales alternativos dentro de un estado para su uso por transiciones externas. Se utilizan cuando hay más de una forma de entrar en un estado y el subestado inicial por defecto no fuera suficiente. Cada punto de entrada designa un estado interno dentro del estado propietario. Los puntos de entrada tienen nombres que son visibles externamente. Una transición externa puede tener un punto de entrada como su destino. Una transición conectada a un punto de entrada es efectivamente conectada al estado designado como su destino, pero la transición externa no necesita conocer los detalles internos del estado para hacer la conexión. Este mecanismo es particularmente útil con submáquinas, en las que la transición referencia a la submáquina y no sería posible una conexión directa a un estado interior sin hacer una copia recursiva de la submáquina.

Observe que el subestado inicial de un estado se puede considerar equivalente a un punto de entrada sin nombre.

Independientemente de cómo se entra a un estado, si a través de un punto de entrada, de un estado inicial o de una transición explícita a un subestado interno, la actividad entrar del estado se ejecuta antes de que se produzca la transferencia al estado interno.

Notación

Un punto de entrada se representa como un pequeño círculo en el límite del símbolo del estado. El nombre del punto de entrada se coloca cerca del círculo. Se dibuja una flecha continua desde el círculo al estado interno designado dentro del símbolo de estado.

La Figura 14.249 muestra un estado que representa jugar a un juego computerizado de ajedrez. Inicialmente el jugador puede elegir jugar con piezas blancas o negras, cada una de las cuales están modeladas como un punto de entrada. Debido a la simetría no hay estado inicial por defecto. El estado también tiene tres puntos de salida, que corresponden a que ganen las blancas, las negras o a que se produzca un empate.

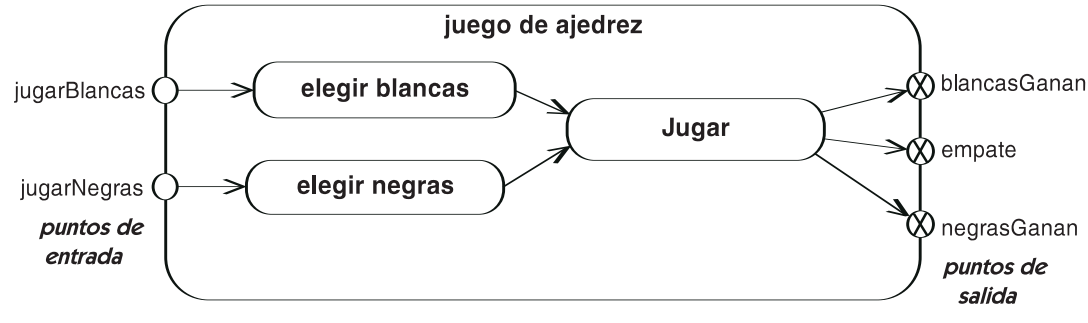


Figura 14.249 Punto de entrada y salida

Historia

Los puntos de entrada y de salida reemplazan a los estados abreviados externos en una formulación UML2 mucho más limpia.

punto de extensión

Marcador con nombre que identifica una ubicación o conjunto de ubicaciones dentro de la secuencia de comportamiento de un caso de uso, en las que se puede insertar comportamiento adicional. Una declaración de punto de extensión abre las puertas a la extensión del 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 mediante una relación de extensión). La relación de extensión contiene una lista de nombres de puntos de extensión que indica dónde inserta su comportamiento el segmento de inserción del caso de uso de extensión.

Véase también extender, caso de uso.

Semántica

Un punto de extensión tiene un nombre. Referencia a un conjunto de una o más ubicaciones dentro de una secuencia de comportamiento de un caso de uso. El concepto de ubicación no está definido dentro de UML, y por tanto es un punto de variación semántica dependiente de la implementación.

Una ubicación puede corresponder a un estado dentro de una descripción de máquina de estados de un caso de uso, o su equivalente en una descripción diferente —entre dos instrucciones en una lista de instrucciones o entre dos mensajes en una interacción.

Una relación de extensión contiene una condición opcional y una lista de referencias a puntos de extensión igual en número al número de segmentos de inserción en el caso de uso de extensión. Un segmento de inserción puede realizarse si se satisface la condición mientras una instancia de un caso de uso está ejecutando el caso de uso base en cualquier ubicación en el punto de extensión correspondiente al segmento de inserción.

La ubicación de un punto de extensión se puede cambiar sin afectar a su identidad. La utilización de puntos de extensión con nombre separa la especificación de las secuencias de comportamiento de extensión de los detalles internos del caso de uso base. Se puede modificar o recolocar el caso de uso base sin afectar a las extensiones. Más aún, se puede mover un punto de extensión dentro de la base sin afectar a la relación o al caso de uso de extensión. Como con todos los tipos de modularidad, esta independencia requiere una buena elección de los puntos de extensión, y no está garantizada para todas las circunstancias.

Notación

Los puntos de extensión de un caso de uso se pueden listar como cadenas en un comportamiento llamado **extension points** (Figura 14.250).

Observe que no hay ningún lenguaje de texto estándar para las secuencias de comportamiento, y por tanto la sintaxis para describir puntos de extensión y ubicaciones no está definida de manera precisa. La notación de la Figura 14.250 es simplemente una sugerencia.

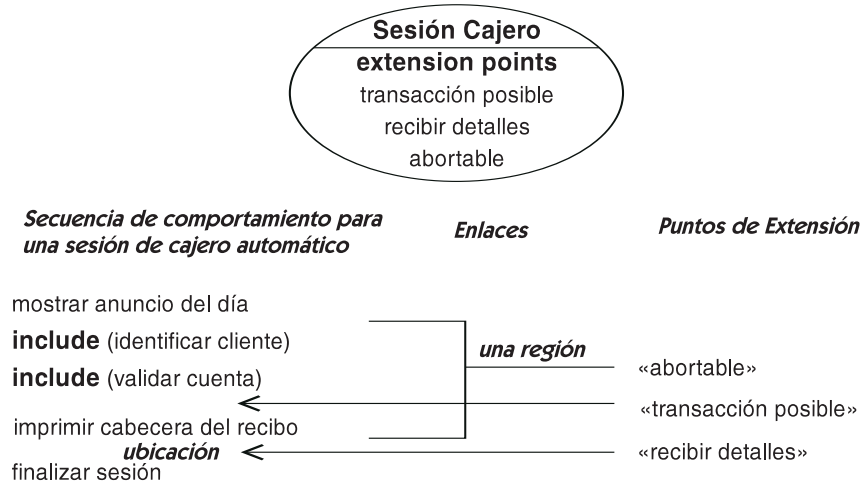


Figura 14.250 Declaraciones de puntos de extensión

punto de salida

Dentro de un estado, un pseudoestado visible externamente que puede ser el origen de una transición externa. Representa un estado final dentro del estado que puede estar conectado a una transición externa.

Véase también punto de entrada.

Semántica

Un punto de salida es un mecanismo de encapsulación. Permite la definición de estados finales alternativos dentro de un estado para su uso por transiciones externas. Se utilizan cuando hay más de una forma de completar un estado y el estado final por defecto no es suficiente. Cada punto de salida representa un estado final con nombre dentro del estado propietario. Los puntos de salida tienen nombres que son visibles externamente. Una transición externa puede tener un punto de salida como su origen. Una transición conectada a un punto de salida es efectivamente conectada al estado final con nombre interno designado como su origen, pero la transición externa no necesita conocer los detalles internos del estado para hacer la conexión. Este mecanismo es particularmente útil con submáquinas, en las que la transición referencia a la submáquina y no sería posible una conexión directa a un estado interno sin hacer una copia recursiva de la submáquina.

Observe que el estado final por defecto de un estado puede ser considerado equivalente a un punto de salida sin nombre.

Independientemente de cómo se salga de un estado, si a través de un punto de salida, un estado final por defecto o una transición explícita a un estado externo, la actividad salir del estado se ejecuta antes de que se produzca la transferencia al estado externo.

Notación

Un punto de salida se representa con un pequeño círculo con una X dentro en el límite del símbolo de estado. El nombre del punto de salida se coloca cerca del círculo. Las transiciones desde los estados internos se pueden conectar al punto de salida como destino.

La Figura 14.249 muestra un estado que representa jugar a un juego computerizado de ajedrez. Inicialmente el jugador puede elegir jugar con piezas blancas o negras, cada una de las cuales están modeladas como un punto de entrada. Debido a la simetría no hay estado inicial por defecto. El estado también tiene tres puntos de salida, que corresponden a que ganen las blancas, las negras o a que se produzca un empate.

punto de variación semántica

Un punto de variación en la semántica de un metamodelo. Proporciona un grado de libertad intencional para la interpretación de la semántica del metamodelo.

Discusión

No es conveniente la misma semántica de la ejecución para todas las posibles implementaciones. Diferentes lenguajes de programación y los propósitos diferentes requieren variaciones en la semántica, alguna sutil, alguna fuerte. Un punto de variación semántica es un problema en que varios diseñadores o los distintos ambientes de la ejecución discrepan sobre la semántica específica, a menudo por buenas razones. Simplemente identificando y nombrando los puntos de variación semántica, los problemas sobre la semántica "correcta" de un sistema se pueden evitar.

Por ejemplo, la opción de si permitir la clasificación múltiple o la clasificación dinámica es un punto de variación semántica. Cada opción es una variación semántica.

Otros ejemplos de puntos de variación semántica incluyen si una llamada puede devolver más de un valor y si las clases existen al tiempo de ejecución como objetos reales.

punto de vista

Perspectiva desde la que se ve una vista.

Discusión

UML incorpora un cierto número de puntos de vista, incluyendo modelado del mundo real, análisis de aplicación, diseño de alto nivel, modelado de implementación y programación visual de los lenguajes existentes. (La última es una utilización desafortunada de UML por muchos usuarios que no aprecian el poder de la abstracción y del modelado.) Los múltiples puntos de vista a veces causan confusión puesto que los mismos conceptos se pueden utilizar de diferentes formas para conseguir diferentes propósitos. Es desafortunado que el propio UML no contenga buenas formas de declarar el punto de vista que expresa un modelo, sino que los modeladores deben tener cuidado de separar los diferentes puntos de vista en diferentes modelos y etiquetar cada modelo con el punto de vista que expresa.

ranura

Un lugar en un objeto u otra instancia para un valor.

realización

La relación entre una especificación y su implementación; una indicación de la herencia de comportamiento sin la herencia de estructura.

Véase también interfaz.

Semántica

Una especificación describe el comportamiento o estructura de algo sin determinar cómo se llevará a cabo el comportamiento. Una implementación proporciona los detalles sobre cómo implementar el comportamiento de una manera eficazmente calculable. La relación entre un elemento que especifica el comportamiento y uno que proporciona la implementación se llama realización. Hay muchas maneras de comprender, en general, una especificación. Similarmente, un elemento puede comprender más de una especificación. La realización es por consiguiente, una relación muchos-a-muchos entre los elementos.

El significado de realización es que el elemento del cliente debe apoyar todo el comportamiento del elemento del proveedor pero no necesita que coincida su estructura o aplicación. Por ejemplo, un cliente clasificador debe apoyar las operaciones del clasificador del proveedor y debe apoyar todas las máquinas de estado que especifican comportamiento externo del proveedor.

Pero cualquier atributo, asociaciones, métodos o máquinas de estado del proveedor que especifican la aplicación no son pertinentes al cliente. Observe que el cliente no hace realmente inherentes las operaciones del proveedor. Debe declararlos él o heredarlos de un antepasado para que se cubran todas las operaciones del proveedor.

En otras palabras, el proveedor en una realización indica cuáles operaciones deben estar presentes en el cliente, pero el cliente es responsable para proporcionarlas. En el sentido más general de realización, los nombres de las operaciones no necesitan coincidir, sólo su comportamiento total.

Ciertos tipos de elementos, tales como interfaces y casos de uso, se intentan para especificar el comportamiento y no contienen la información de aplicación. Otros piensan los tipos de elementos, como clases, para llevar a cabo el comportamiento. Ello contiene la información de aplicación, pero también pueden usarse en una forma más abstracta como especificadores. Normalmente, la realización relaciona un elemento de la especificación, como un caso de uso o una interfaz, a un elemento de aplicación, como una colaboración o una clase. Es posible usar un elemento de aplicación, como una clase, para la especificación.

Puede ponerse al lado de la especificación de una relación de realización. En este caso, solamente parte de la especificación del efecto de clase del proveedor afecta al cliente. Las partes de la aplicación son irrelevantes para la relación de la realización. Más precisamente entonces, la realización es una relación entre dos elementos en que las partes de especificación del comportamiento externo, parte de uno reprima la aplicación del otro. Podría pensarse como herencia de especificación de comportamiento sin herencia de estructura o aplicación (y con la necesidad de declarar las operaciones realmente por el cliente).

Un componente puede ser comprendido por un conjunto de clases que juntos llevan a cabo el comportamiento especificado por el proveedor y requieren interfaces del componente.

Si el elemento de la especificación es una clase abstracta sin los atributos, ninguna asociación y sólo las operaciones abstractas, cualquier especialización de la clase abstracta implícitamente comprende la clase abstracta, como allí nada es heredar pero sí especificado.

El elemento de implementación debe apoyar todo el comportamiento incluido en el elemento especificado. Por ejemplo, una clase debe apoyar todos los funcionamientos de las interfaces que comprende, con semántica que es consistente con todas las especificaciones requeridas por las interfaces.

La clase puede llevar a cabo las operaciones adicionales y la aplicación de las operaciones puede hacer las cosas adicionales, con tal de que la operación no viole las características técnicas de las interfaces.

Notación

La relación de la realización es mostrada por un camino discontinuo con una punta de flecha triangular cerrada en el fin del elemento que proporciona la especificación y con su cola en el elemento que proporciona la aplicación (Figura 14.251).

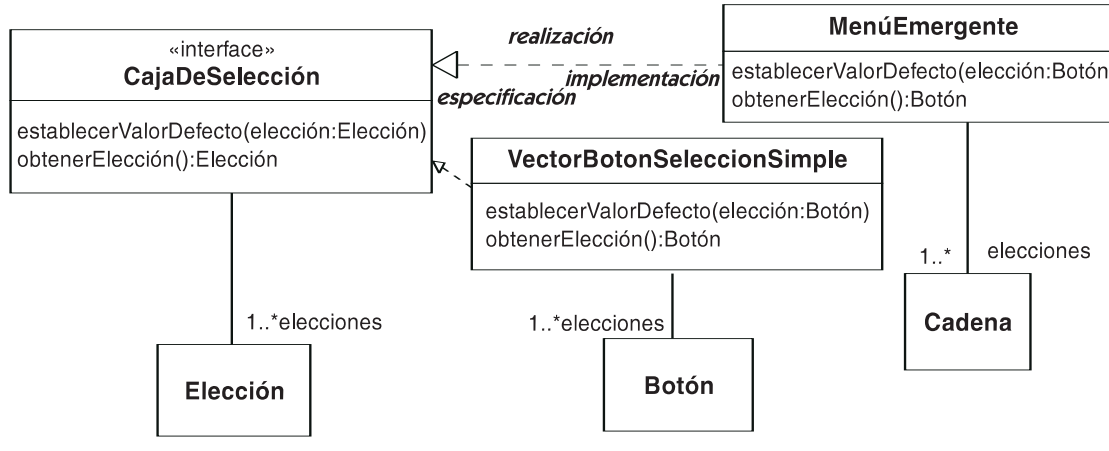


Figura 14.251 Relación de la realización

Discusión

Otro caso importante es la realización de un caso de uso por una colaboración (Figura 14.252). Un caso de uso especifica la funcionalidad visible externamente y las secuencias de comportamiento, pero no proporciona una aplicación. Una colaboración describe los objetos que llevan a cabo el comportamiento de casos de uso y la manera en que actúan recíprocamente para hacerlo. Normalmente, una colaboración lleva a cabo un caso de uso, pero una colaboración puede ser llevada a cabo usando colaboraciones subordinadas de cada uno de los que parte el trabajo. Los objetos y clases usados para implementar una colaboración usualmente aparecen en otras cola-

boraciones también. Cada clase en la colaboración consagra la parte de su funcionalidad al caso de uso a llevarse a cabo. Por consiguiente, un caso de uso es eventualmente implementado por rodajas a través de varias clases.

Figura 14.252 Realización del caso de uso por colaboración

realizar

Para proveer la implementación a un elemento de la especificación.

Véase realización.

realization (estereotipo de Clasificador)

Un clasificador que especifica la aplicación física de un dominio. Contraste con especificación.

Véase realización.

recepción

Una declaración de que un clasificador está preparado para reaccionar al recibo de una señal. Una recepción es un rasgo de un clasificador.

Semántica

Una recepción es una declaración de que una instancia de un clasificador se prepara para manejar y reacciona al recibo de una instancia de una señal. Una recepción es similar (en espíritu) a una operación. Declara el tipo de señal que el clasificador acepta y especifica el efecto del recibo de una instancia de una señal.

El recibo de una señal puede activar una transición de la máquina de estado, puede causar la ejecución de un procedimiento o puede tener otro efecto especificado por el mecanismo de resolución. Si un recibo de una señal causa la ejecución de un procedimiento síncrono cualquier esfuerzo por volver se ignora.

Notación

Una recepción puede mostrarse en la lista de operación de una clase o interfaz que usa la sintaxis para un funcionamiento con el «**signal**» de la palabra clave delante del nombre señalado. Los parámetros corresponden a los atributos de la señal; todos deben ser solamente parámetros.

Alternadamente, una lista de señales puede ponerse en su propio compartimiento; el compartimiento tiene las **Signals** del nombre. Se muestran ambas maneras en la Figura 14.253.

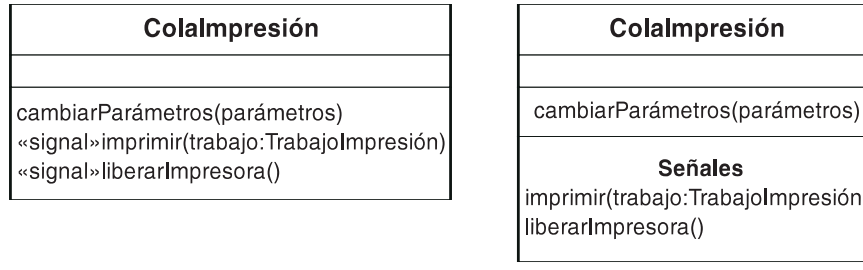


Figura 14.253 Dos formas de representar recepción de señales

receptor

El objeto que se ocupa de una instancia del mensaje pasado de un objeto del remitente. La recepción de un mensaje normalmente causa un efecto en el receptor.

recibir

Para ocuparse de una instancia de mensaje pasado de un objeto del remitente.

Véase receptor, remitente, substitución.

Semántica

La recepción de un mensaje por un objeto es un evento. Si el mensaje representa una llamada es un evento de llamada; si el mensaje representa una señal de envío, es un evento de señal. Si una operación (en el caso de una llamada) o una recepción (en el caso de un signo) se declara por el clasificador del propio objeto, el recibo del mensaje por un objeto puede resolverse en efecto, como la ejecución de un procedimiento o la activación de una transición.

(La declaración de la recepción puede ser implícita si la recepción se resuelve a una activación.)

En lugar de declarar un efecto para un tipo dado de recepción, un objeto puede ejecutar una acción de aceptación, en cuyo caso el objeto espera hasta que un evento de un tipo especificado sea ocupado por el objeto.

red de Petri

Un formalismo para computación que enfatiza la concurrencia, distribución, la computación asíncrona con puntos de sincronización explícitos, en lugar del modelo secuencial y monolítico de las máquinas de Turing y la mayoría de los autómatas tradicionales, característico de las máquinas de Turing y la teoría de autómatas más tradicional. Fueron inventados por el científico alemán de computadoras Carl Adam Petri en su tesis de Ph.D. en 1962 y han engendrado un campo entero de investigación. El modelo de actividad UML2 está ligeramente basado en fundamentos de Petri.

redefines

Palabra clave que indica la redefinición de un rasgo en un clasificador.

Véase redefinición (de funcionamiento), redefinición (de propiedad).

redefinición

Una modificación de la especificación de un elemento definido dentro del contexto de una especificación de clasificador, a una nueva especificación dentro del contexto de un descendiente del clasificador.

Semántica

Un elemento redefinible definido en un clasificador puede redefinirse en un clasificador descendiente.

La redefinición hace referencia a la definición original dentro del clasificador donde fue definido. La redefinición puede aumentar, puede reprimir o puede sobreescribir la especificación original. El principio de sustitución puede ser violado por redefinición, porque la redefinición puede invalidar una restricción de un antepasado.

Notación

La anotación depende del tipo de elemento que se redefine.

Véase redefinición (de comportamiento), redefinición (de clasificador), redefinición (de operación), redefinición (de propiedad), redefinición (de máquina de estado), redefinición (de plantilla).

Historia

La redefinición es un nuevo concepto de UML2.

Discusión

La redefinición debe usarse poco y con cuidado, porque puede producir modelos poco claros.

redefinición (de clasificador)

Semántica

Un clasificador declarado dentro de otro clasificador puede redefinirse dentro de una especialización del clasificador que lo contiene. La especificación no parece indicar lo que los cambios permiten en el clasificador redefinido, sin embargo. Probablemente los atributos y operaciones pueden agregarse; no está claro si los atributos y operaciones existentes pueden modificarse y cómo hacerlo. Posiblemente estarán permitidos los mismos tipos de cambios permitidos en una subclase, en un clasificador redefinido.

redefinición (de comportamiento)

Semántica

Un comportamiento puede redefinir otro comportamiento. El tipo de comportamiento puede cambiarse.

Por ejemplo, una máquina de estado podría ser reemplazada por un procedimiento.

Si implementa una característica conductual, la redefinición reemplaza el comportamiento original en el clasificador especializado.

Si lleva a cabo un clasificador (como un comportamiento de clasificador), la redefinición extiende la definición original pero no lo reemplaza.

Una interacción puede sustituirse por otra interacción en la especialización de un clasificador propio.

Notación

Véase redefinición (de operación), redefinición (de máquina de estado).

redefinición (de máquina de estados)

Especializando un clasificador, el comportamiento de las máquinas de estados que especifican el clasificador y sus métodos pueden ser redefinidas. (El documento usa la palabra *extended* refiriéndose a la redefinición de máquinas de estados.) Cuando una máquina de estados se redefine, sus estados, regiones y transiciones pueden redefinirse.

Pueden redefinirse estados. Un estado simple puede convertirse en un estado compuesto agregando una región. Un estado compuesto puede añadir regiones, vértices, transiciones, actividades de entrada/salida/hacer y transiciones a regiones heredadas de la máquina de estados original. Si el estado es parte de una región, una redefinición del estado debe ser parte de una redefinición de la región. Una redefinición de un estado de la submáquina puede reemplazar las referencias a la submáquina con tal de que tenga la misma entrada y puntos de la salida que la submáquina original. La nueva referencia de la submáquina puede tener entradas y puntos de salida adicionales.

Una región puede redefinirse. La redefinición puede agregar vértices y transiciones y los estados y transiciones pueden ser redefinidos.

Una transición puede redefinirse. Una transición redefinida puede redefinir su contenido y el estado destino, pero no su estado de origen ni su disparador. Una transición redefinida debe ser identificada de forma única por una tupla (origen, destino, disparador). Esto excluye las transiciones que sólo difieren en la condición de guarda de redefinirse.

Si un clasificador tiene padres múltiples, la redefinición de la máquina de estados tiene una región ortogonal que corresponde a la máquina de estados de cada clasificador padre.

Notación

Una máquina de estados redefinida tiene la palabra clave {**extended**} como parte de su etiqueta de nombre.

En una máquina de estados redefinida, se muestran añadidos y cambios usando líneas continuas. Se asume que estados, regiones y transiciones que no se muestran son heredados (inalterado de la máquina de estados original). Los estados heredados, regiones y transiciones pueden dibujarse usando las líneas discontinuas si necesita establecer el contexto para los aditamentos, por ejemplo, si están implicados en transiciones.

Ejemplo

La Figura 14.254 muestra una máquina de estados para un expendedor automático. Esto sirve como base para una redefinición de la máquina de estados. La Figura 14.255 muestra una redefinición de la máquina de estados para el expendedor automático.

Las líneas discontinuas muestran estados heredados de la definición base y las líneas sólidas muestran los estados y transiciones incluidas en la redefinición.

Observe que algunos estados, como el estado inicial y **Ocioso**, no se muestran porque no han cambiado. Las transiciones de **Ocioso** a **Efectivo Presente** y desde **Efectivo Presente** a **Artículo Seleccionado** no se muestran porque no han cambiado.

El estado simple **Artículo Seleccionado** se ha extendido para ser un estado compuesto con dos regiones. Tiene una transición adicional para **Efectivo Presente** activado por fuera de existencias. Observe que la transición activada por **dispensado** todavía está presente, pero se sobreescribe por una transición con el mismo disparador dentro del estado compuesto.

redefinición (de operación)

Semántica

Una operación puede redefinirse en una especialización de una clase. Esta redefinición puede especializar los tipos de los parámetros formales o resultados del retorno, agregar las nuevas pre-

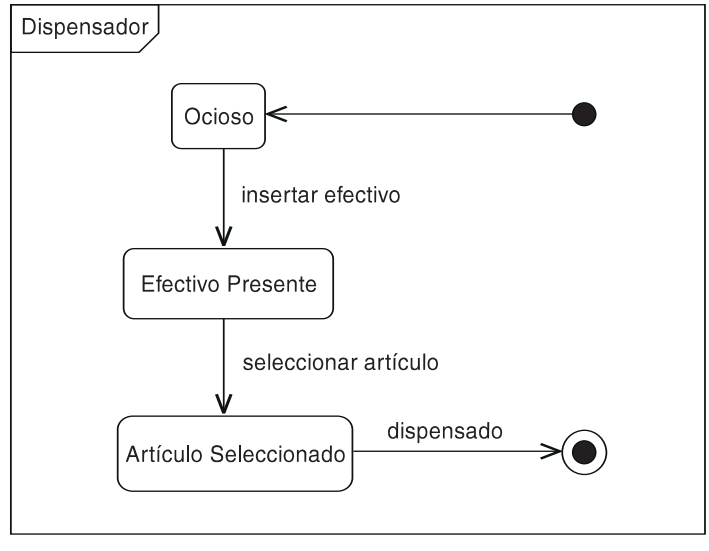


Figura 14.254 Máquina de estados original para la redefinición

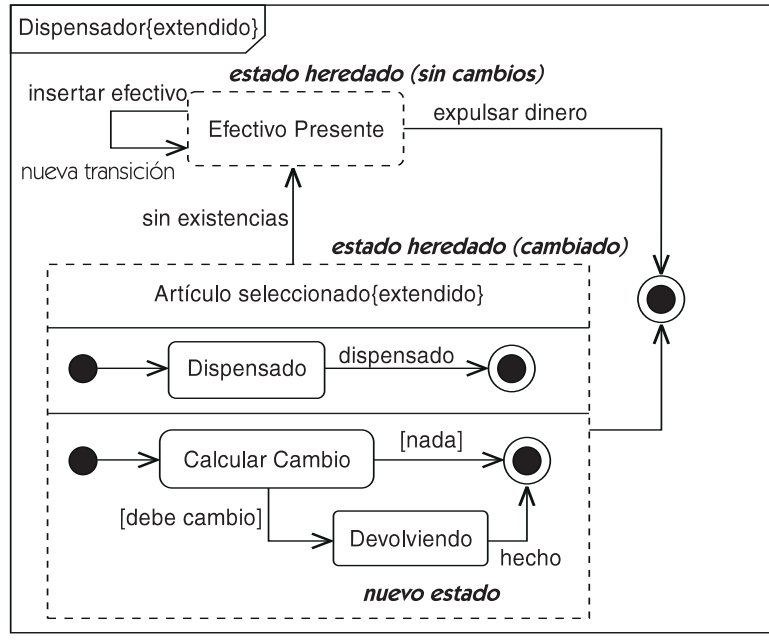


Figura 14.255 Redefinición de la máquina de estados

condiciones o poscondiciones, agregar las nuevas excepciones levantadas o por otra parte redefine la especificación de la operación.

La BodyCondition (una restricción en el valor de retorno) puede destruirse en una redefinición. Las poscondiciones sólo pueden fortalecerse en una redefinición.

Notación

La nueva operación se incluye en la subclase añadiendo la siguiente cadena:

{redefine
`nombre-de-operación`}

redefinición (de plantilla)

Semántica

Una firma de la plantilla redefinida puede agregar parámetros de la plantilla adicionales.

Probablemente el elemento en el que la plantilla está basada (como clasificador) puede ser especializado del modo usual, con otras redefiniciones legales.

Notación

Ninguna.

Discusión

Probablemente es una idea mala redefinir una plantilla realmente, porque estarían ocurriendo demasiadas cosas simultáneamente. La experiencia de C++ muestra el peligro de mezclar plantillas, sobrecarga, y herencia.

redefinición (de propiedad)

Semántica

Una propiedad (atributo o fin de la asociación) puede redefinirse en un clasificador descendiente.

Entre las características de una propiedad que puede redefinirse están nombre, tipo (puede especializarse), valor predefinido, estado de la derivación, la visibilidad, multiplicidad, y constreñimientos en valores.

La interacción de especialización de la asociación con redefinición del extremo de asociación y el subconjunto de parámetros no está definida.

Notación

Una redefinición tiene la sintaxis

{**redefine**
el `nombre-de-propiedad`}

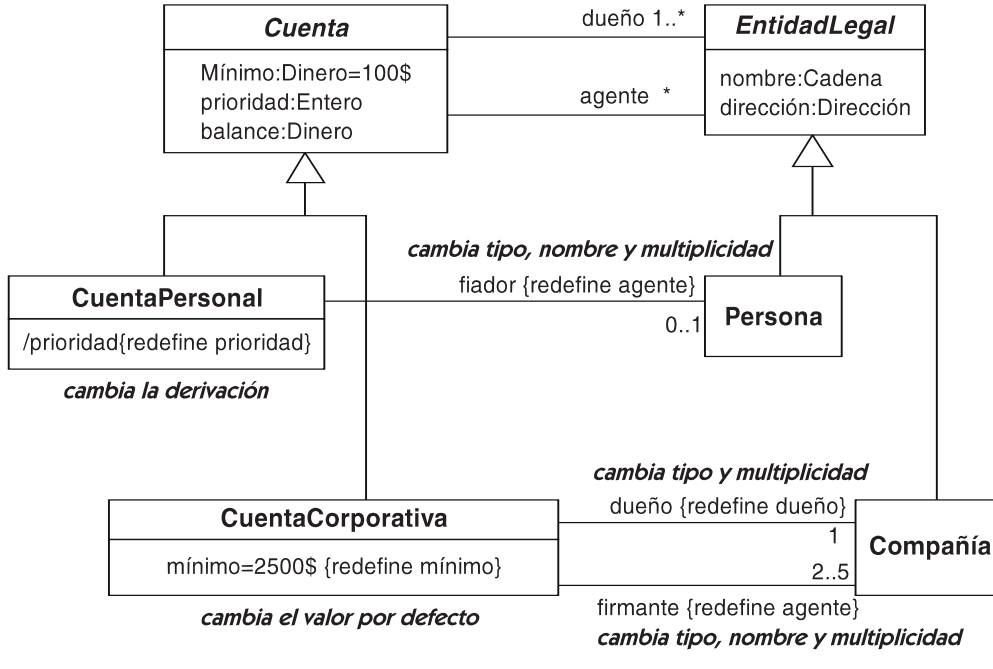


Figura 14.256 Redefinición de propiedades

donde **`nombre-de-propiedad`** es el nombre del atributo o fin de la asociación que está siendo redefinido. El cordón se pone después del cordón del atributo o cerca del fin de una asociación línea con la propiedad definiendo. El nombre de la nueva propiedad puede ser diferente de la propiedad original.

Ejemplo

La Figura 14.256 muestra varias redefiniciones de atributos y fines de la asociación.

Discusión

Una propiedad con el mismo nombre que una propiedad que se habría heredado se supone que ha sido redefinida sin la necesidad de usar la palabra clave **redefine**. Esto es una regla peligrosa, ya que puede llevar fácilmente a errores graves. No lo use. Sea siempre explícito sobre la redefinición, ya es bastante confusa por sí misma.

referencia

Una designación de un elemento del modelo; a menudo un llamado apuntador, pero no puede asumirse ninguna implementación en general.

Semántica

Los elementos modelo son conectados por dos metarelaciones: propiedad y referencia.

La propiedad es la relación entre un elemento y sus partes constitutivas, las partes que se definen dentro de él y poseídas por él. Las formas de relación de propiedad son un árbol estricto. Los elementos contenidos son subordinados a los elementos del contenedor. La propiedad, control de la configuración y almacenamiento de modelos son basados en la jerarquía de contenedores.

La referencia es una relación entre elementos en el mismo nivel de detalle o entre elementos en contenedores diferentes. Por ejemplo, la referencia es la relación entre una asociación y sus clases participantes, entre un atributo y la clase o tipo de datos que son su propiedad tipo o entre una plantilla limitada y sus valores argumento. Para que sea posible una referencia, el elemento que realiza la referencia debe tener visibilidad sobre el elemento que está siendo referenciado. Esto requiere que el elemento al que se hace referencia tenga visibilidad que le permite ser visto fuera de su paquete, a menos que el origen de la referencia esté en el mismo paquete.

Observe que la referencia es una relación del metamodelo interior, no una relación visible por el usuario; se usa para construir las otras relaciones.

refinamiento

Una relación que representa una especificación más completa de algo que ya estaba especificado a un cierto nivel de detalle o a un nivel semántico diferente.

Véase abstracción.

Semántica

Un refinamiento es una conexión histórica o calculable entre dos elementos con una correspondencia (no necesariamente completa) entre ellos. A menudo, los dos elementos están en mode-

los 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 venir de una biblioteca de clases específica. Un elemento puede refinar un elemento en el mismo modelo, sin embargo. Por ejemplo, una versión perfeccionada de una clase es un refinamiento de la simple pero versión ineficaz de la clase. La relación de refinamiento puede contener una descripción de la correspondencia que puede escribirse con un lenguaje formal (como OCL o un lenguaje de programación o un lenguaje lógico). O puede ser un texto informal (que, obviamente, evita cualquier cómputo automático, pero puede ser útil en fases tempranas de desarrollo). El refinamiento puede usarse para modelar el desarrollo por etapas, optimización, transformación, y elaboración del esqueleto del sistema.

Estructura

El refinamiento es un tipo de dependencia de abstracción. Relaciona a un cliente (el elemento que se desarrolla más) con un proveedor (el elemento que es la base para el refinamiento).

Notación

Un refinamiento se indica por una flecha de dependencia (una flecha discontinua con su cabeza en el elemento más general y va detrás del elemento más específico) con la palabra clave «**refine**». La correspondencia puede unirse a la ruta de la dependencia por una línea discontinua conectada a una nota. Se han propuesto varios tipos de refinamiento y pueden ser indicados en detalle mediante el uso de estereotipos. En muchos casos, el refinamiento conecta los elementos en modelos diferentes y no será por consiguiente gráficamente visible.

Ejemplo

La optimización es una forma típica de refinamiento. La Figura 14.257 muestra un tablero de ajedrez que tiene una representación simple en el modelo de análisis, pero tiene una representación más detallada y oscura en el modelo de diseño. La clase de diseño no es una especialización

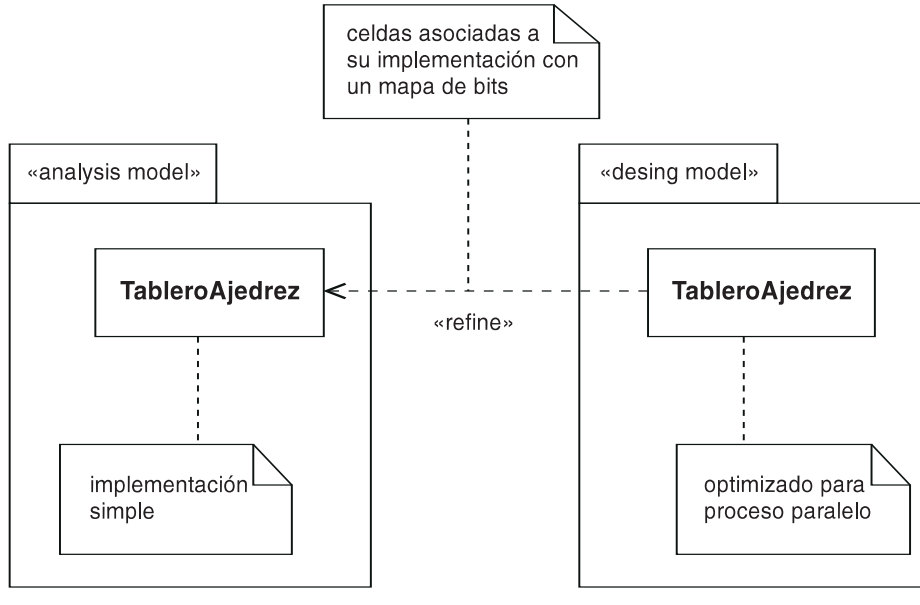


Figura 14.257 Refinamiento

de la clase de análisis, porque tiene una forma totalmente diferente. La clase en el modelo de análisis y en el modelo de diseño tiene el mismo nombre, porque representan el mismo concepto a dos niveles semánticos diferentes.

Discusión

La distinción entre refinamiento y realización no está clara en la especificación de UML y puede ser que el refinamiento sea una redundancia innecesaria sin semánticas precisas.

refine (estereotipo de Dependencia de abstracción)

Un estereotipo en dependencia que denota una relación de refinamiento.

Véase refinamiento.

región

Un constituyente directo de un estado compuesto o una máquina de estados, que contiene vértices (subestados y pseudoestados) y transiciones que forman un fragmento anidado de una máquina de estado.

Véase máquina de estado, estado compuesto, transición, vértice.

Semántica

Una región es un fragmento anidado de una máquina de estado que contiene un conjunto conectado de estados, pseudoestados y transiciones. También se permiten transiciones a y de otras regiones. Las regiones son los elementos constituyentes directos de los estados compuestos y las máquinas de estados son la estructura que permite la composición anidada de máquinas de estado.

Los subestados directos de una región son mutuamente excluyentes. Si una región está activa, exactamente un subestado directo está activo. Si una región contiene (directa o indirectamente) estados compuestos, una región activa puede tener múltiples subestados indirectos activos a niveles anidados.

Si una región es parte de un estado no ortogonal, representa una descomposición secuencial del estado. Sólo un subestado directo puede estar activo a la vez.

Si una región es parte de un estado ortogonal, representa una descomposición coexistente del estado. Las otras regiones de la misma región ortogonal son sus regiones ortogonales de par. Un subestado directo de cada región ortogonal está activo cuando el estado ortogonal está activo.

Notación

Los contenidos de una región se representan como un gráfico conectado de símbolos de estado y flechas de transición. Para un estado no ortogonal, los contenidos de su región son simple-

mente anidados, sin su símbolo de estado. Para un estado ortogonal, su símbolo de estado está dividido en secciones por líneas discontinuas y cada sección muestra los contenidos de una región diferente. El nombre de la región puede ser incluido en su sección. *Véase* estado compuesto para los ejemplos de notación.

Discusión

Comparado al autómata de estados tradicional de computación, las regiones proporcionan la habilidad de definir los estados anidados y las transiciones necesarias, reduciendo así enormemente el número de transiciones necesarias en sistemas complejos. Esta característica fue agregada por David Harel en sus cartas de estado, que proporcionan la fundación de la mayoría de los conceptos de máquina de estado en UML.

región crítica

Fragmento combinado en una interacción cuyos eventos no se pueden intercalar con eventos de regiones concurrentes.

Semántica

Una región crítica tiene un subfragmento. Una secuencia de eventos en una única línea de vida en la región crítica no debe estar intercalada con ningún otro evento en otras regiones.

No hay ninguna restricción sobre los eventos en otras líneas de vida, de forma que esto no descarta actividad concurrente que no afecte a la región crítica. Esta construcción anula a cualquier construcción paralela que de otra forma pudiera permitir el intercalado.

Notación

Una región crítica se representa como un rectángulo con la etiqueta **critical** dentro de un pentágono en la esquina superior izquierda. Las líneas de vida que atraviesa el rectángulo quedan cubiertas por la región crítica. Todos los eventos dentro del rectángulo forman parte de la secuencia no interrumpible definida por la región. Los eventos sobre líneas de vida que estén fuera del rectángulo no quedan cubiertos por la región, y por tanto su orden no está restringido por la región crítica.

Ejemplo

La Figura 14.258 muestra parte del diagrama de secuencia de un reproductor de DVD (con mucho detalle omitido). En el bucle principal, el reproductor muestra fotogramas de forma repetida. En cualquier momento (puesto que está dentro de una construcción paralela), el usuario puede enviar un mensaje de pausa al reproductor. Después de que envíe el mensaje de pausa, el usuario envía un mensaje de reanudación. Puesto que la secuencia pausa-reanudación está dentro de una región crítica, no puede intervenir ningún mensaje mostrar fotograma. Por tanto, el reproductor deja de mostrar fotogramas hasta que se produce el mensaje de reanudación.

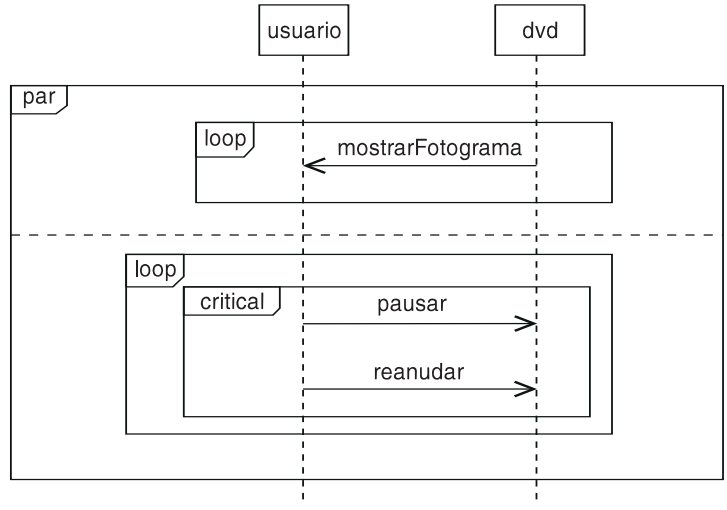


Figura 14.258 Región crítica

región de actividad interrumpible

Conjunto de nodos de actividad y transiciones en el que se termina la actividad si se produce un evento designado. *Véase* interrupción.

región de expansión

Nodo de actividad estructurado que se ejecuta una vez por cada elemento dentro de una colección de entrada.

Semántica

Una región de expansión es un mecanismo para aplicar un cálculo de forma repetida a cada uno de los elementos de una colección de valores. Es una construcción “para todos”. Fuera de la región, las entradas y las salidas se modelan como tokens cuyos valores son colecciones como conjuntos, bolsas y listas. Dentro de la región, las colecciones se expanden en elementos individuales, y la región se ejecuta una vez por cada grupo de elementos elegidos, una desde cada colección de entrada y salida. Todas las colecciones de entrada deben tener el mismo tamaño y deben ser del mismo tipo (conjunto, bolsa, lista, etc.). Las colecciones de salida se construyen como parte de la ejecución de la región de expansión; su tamaño será el mismo que el tamaño de las colecciones de entrada. Las colecciones no necesitan tener el mismo tipo de elemento, aunque cada colección tendrá elementos de un tipo dado (aunque pueden variar dentro de una colección mediante polimorfismo).

El propósito de una región de expansión es expandir cada colección en una colección de grupos individuales de elementos, conteniendo cada grupo un elemento de cada colección. El cálculo dentro de la región de expansión se escribe en términos de entradas y salidas que representan elementos individuales, no colecciones. Cada grupo de elementos es una “porción”

del grupo total de colecciones. La región de expansión se ejecuta una vez por cada porción de valores, mapeando elementos de entrada a elementos de salida. En otras palabras, la región de expansión separa las colecciones en porciones de valores individuales, y vuelve a reunir los valores de salida en colecciones.

Las colecciones de entrada y de salida de una región de expansión se identifican explícitamente como nodos de expansión. La actividad dentro de una región de expansión también puede leer valores de su contexto exterior. Las entradas que no pasen a través de nodos de expansión tienen el mismo valor fijo para cada ejecución de la región. No es posible conectar salidas de acciones internas fuera de la región, puesto que podría desajustar una colección de valores a un solo valor. Una colección de valores debe ser reunida a través de un nodo de expansión de salida.

Normalmente una región de expansión opera en paralelo para todos los elementos. No debería haber conflictos entre las distintas ejecuciones de forma que el resultado no depende del orden de ejecución, que no está especificado y puede ser concurrente. Opcionalmente, una región de expansión puede especificarse para que se ejecute de forma iterativa. Si la colección de entrada está ordenada, los elementos son entregados para su ejecución en el mismo orden. Una ejecución iterativa hace posible la acumulación de resultados sobre la colección entera de valores. Puesto que las ejecuciones son secuenciales, se pueden hacer accesos a recursos o valores compartidos. También es posible especificar que la ejecución usara flujos de valores internos.

Notación

Una región de expansión se representa mediante una caja discontinua redondeada. Se puede representar una de las palabras **«parallel»**, **«iterative»** o **«stream»** para indicar el estilo de ejecución. Las colecciones de salida y de entrada (nodos de expansión) se representan en el límite de la caja con pequeños rectángulos divididos en una secuencia de compartimentos más pequeños (es decir, algo parecido a una secuencia de ranuras). Dentro de la caja se puede representar un fragmento de un gráfico de actividad. El fragmento puede obtener entradas de los símbolos de nodos de expansión de entrada y proporcionar salidas a los símbolos de nodos de expansión de salida. Dentro de la región de expansión, los tipos de los flujos de objetos corresponden a elementos individuales de las colecciones. Fuera de la región de expansión, se pueden conectar flechas de flujo de datos a los nodos de expansión. Los tipos de dichos flujos de datos corresponden a colecciones de valores.

Como atajo, los símbolos de nodo de expansión se pueden colocar directamente en un símbolo de acción para indicar la ejecución de una sola acción dentro de una región de expansión.

La Figura 14.259 muestra una región de expansión con dos colecciones de entrada y una colección de salida. Contiene dos acciones internas que toman dos valores de entrada y producen un valor de salida, que es reunido en una colección de salida.

Ejemplo

La Figura 14.260 muestra el algoritmo de la FFT (Transformada Rápida de Fourier – Fast Fourier Transformation) como una región de expansión. Representa una pasada a través del bucle principal del algoritmo, que se ejecuta *log(n)* veces durante el algoritmo completo. La entrada que recibe el algoritmo es un array de valores complejos *S*. El array de entrada es dividido en dos subarrays por la operación *cortar*, que opera sobre arrays. Fuera de la región de expansión, es

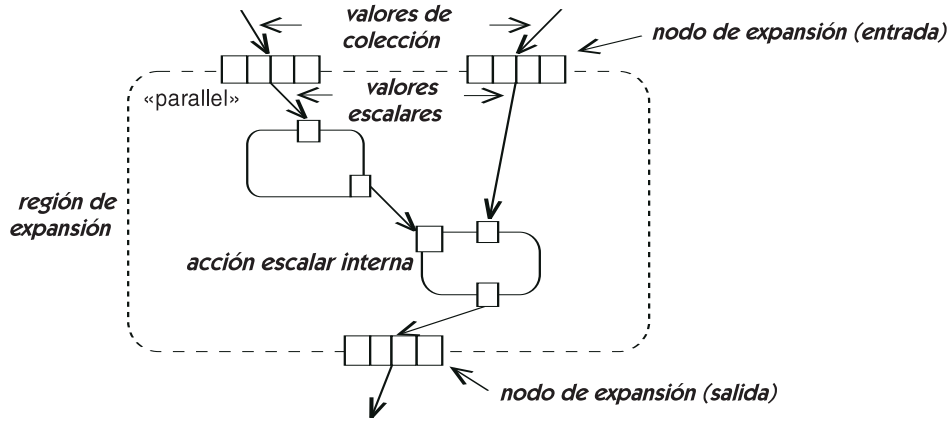


Figura 14.259 Región de expansión

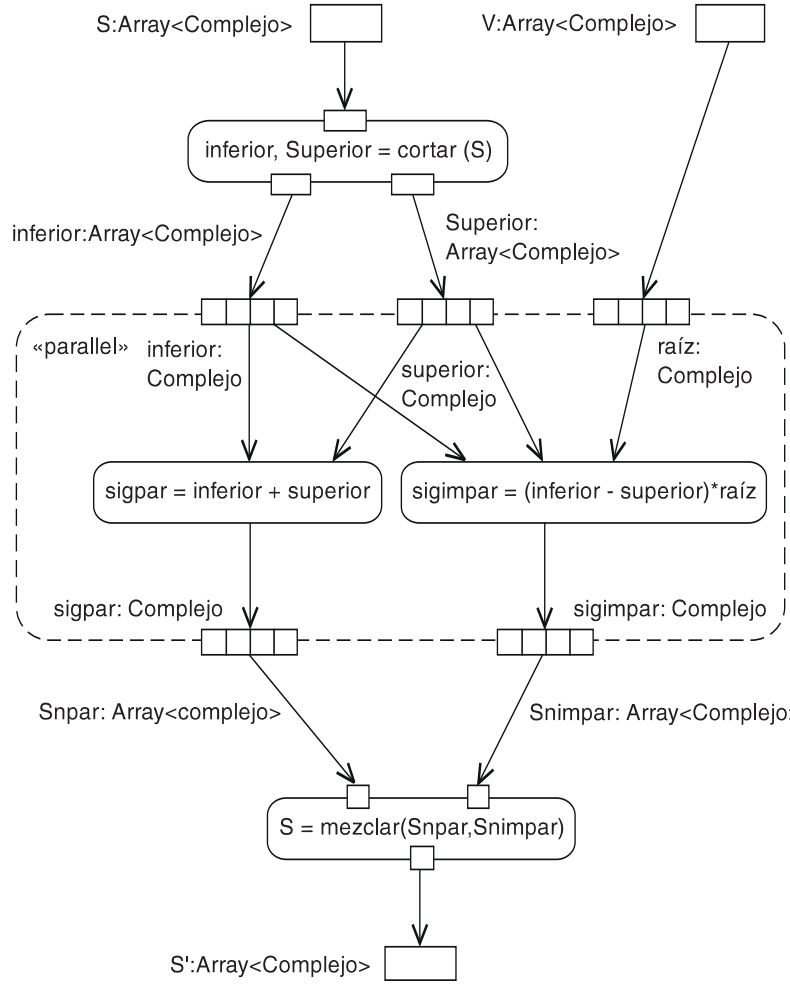


Figura 14.260 FFT como una región de expansión

conveniente expresar el algoritmo en términos de operaciones sobre arrays enteros. Los dos subarrays y otro array de raíces complejas son las colecciones de entrada de la región de expansión. El interior de la región de expansión representa la operación conocida como mariposa. Toma un par de valores, uno de cada subarray, y un valor de raíz compleja, y calcula dos valores nuevos de subarray para la siguiente pasada. La región de expansión se ejecuta una vez para cada porción de valores de entrada. Los dos valores de salida se reúnen en dos subarrays de salida, que son combinados en un solo array utilizando la operación de arrays *mezclar*. (Intercala elementos de las dos entradas.)

La región de expansión permite representar juntas en el mismo diagrama operaciones sobre arrays completos y operaciones sobre elementos individuales de arrays. Permite que dos niveles de detalle sean utilizados juntos.

Historia

Las regiones de expansión reemplazan a un mecanismo mucho más difícil de manejar de UML1 para expresar acciones sobre conjuntos de valores.

región ortogonal

Una región de un estado ortogonal. Si un estado ortogonal está activo, cada una de sus regiones es concurrentemente activa.

Véase transición compleja, estado compuesto, región ortogonal, región.

relación

Una materialización de la conexión semántica entre los elementos del modelo. Esto incluye la asociación y varios tipos de relaciones binarias dirigidas.

Semántica

La Tabla 14.3 muestra los distintos tipos de relaciones de UML. La primera columna (tipo) muestra las agrupaciones bajo las que se colocan en el metamodelo. La segunda columna (variedad) muestra los tipos diferentes de relaciones. La tercera columna (notación) muestra la notación 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 clave) muestra las palabras claves del texto y la sintaxis adicional para las relaciones que lo requieran.

repositorio (depósito)

Un lugar de almacenamiento para modelos, interfaces y aplicaciones; parte de un entorno para la manipulación de los artefactos de desarrollo.

Tabla 14.3 Relaciones en UML

Tipo	*Variedad*	*Notación*	*Palabra clave o símbolo*
abstracción	derivación	dependencia	«derive»
	manifestación	dependencia	«manifest»
	realización	realización	
	refinamiento	dependencia	«refine»
	dependencia de traza	dependencia	«trace»
asociación		asociación	
ligadura		dependencia	«bind» (**parámetro**$_{\texttt{lista}}$)
(despliegue)		dependencia	«deploy» o anidamiento físico
(extender)		dependencia	«extend» (**punto de extensión**$_{\texttt{lista}}$)
(extensión)		extensión	
(generalización)		generalización	
(importe)	privado	dependencia	«access»
	público	dependencia	«import»
incluir		dependencia	«include»
flujo de información		dependencia	«flow»
fusión de paquetes		dependencia	«merge»
permiso		dependencia	«permit»
conformidad con el protocolo			no especificado
sustitución		dependencia	«substitute»
uso	llamada	dependencia	«call»
	creación	dependencia	«create»
	instanciación	dependencia	«instantiate»
	responsabilidad	dependencia	«responsibility»
	envío	dependencia	«send»

requisito

Una característica deseada, propiedad o comportamiento de un sistema.

Semántica

Un requisito de texto puede modelarse como un comentario.

Discusión

El término *requisito* es una palabra del lenguaje natural que corresponde a una variedad de estructuras de UML que se supone que especifica las características deseadas de un sistema.

Más comúnmente, los requisitos que corresponden a las transacciones visibles por el usuario serán capturados como casos de uso. Requisitos no funcionales, como actualización y métricas de calidad, pueden capturarse como declaraciones de texto que eventualmente se trazarán a los elementos del diseño final. Los comentarios de UML y las restricciones pueden usarse para representar los requisitos no funcionales.

resolución

El mecanismo de determinación de un efecto conductual para una llamada basada en una operación y un objeto designado.

Semántica

UML permite una gama amplia de procesos para determinar el efecto de un evento. Los mecanismos de herencia orientados a objetos tradicionales de Smalltalk, C++ y Java son muy familiares, pero la definición de UML es bastante amplia para apoyar los mecanismos alternos, tales como la delegación basada en objetos o el polimorfismo multivariable de CLOS. Esta generalidad se logra asumiendo un mecanismo que convierte la llamada de una operación en un objeto designado en la determinación del comportamiento que va a ser ejecutado. Este mecanismo se llama resolución. Se asume que cualquier modelo de UML específico proporcionará un mecanismo de la resolución aunque no se proporciona ningún medio formal para definir el mecanismo dentro del propio UML.

El mecanismo de resolución específico es un punto de variación semántica del modelo y del entorno de ejecución.

La especificación de UML proporciona las reglas siguientes: El recibo de una llamada o una señal por un objeto es un evento. La ocurrencia de un evento puede habilitar una transición de la máquina de estados del objeto que lo posee, si un disparador para el evento aparece en una transición se deja un estado en la configuración de estado activa. Además de, o en cambio de, activar una transición, un evento puede causar la ejecución de un comportamiento, como un procedimiento. El resultado del proceso de la resolución puede producir un comportamiento que debe ser ejecutado con los parámetros del evento.

La especificación de UML define las reglas del proceso de la resolución orientado a objeto que es el proceso de resolución predefinido, y la que se corresponde a lenguajes orientados a objetos tradicionales para las llamadas de operaciones: La clase del objeto designado es examinada para

encontrar una declaración del método unida a la operación llamada. Un método es un comportamiento. Si un determinado método se encuentra en la clase, el comportamiento se ejecuta con los argumentos de la llamada. Si no se encuentra ningún método que corresponda a la operación, se examina la clase del padre para encontrar un método que corresponda a la operación, y continúa de manera ascendente en la jerarquía de la generalización hasta que no se encuentre ningún método en la clase raíz, en cuyo caso hay un error a menos que una acción se adapte perfectamente a la operación. Si una clase tiene múltiples padres, todos ellos se examinan para buscar métodos concordantes, y si se encuentra más de un método, hay un error. Sin embargo, si se encuentra el mismo método buscando a lo largo de múltiples caminos debido a los múltiples padres, no es un error.

Si una llamada dispara una transición y resuelve un método, el método se ejecuta permitiendo devolver los valores al llamador y, a continuación, la transición se puede disparar asincrónicamente.

resolver

Experimentar el proceso de resolución cuando un evento se maneja por un objeto para producir un efecto de comportamiento.

Véase resolución.

responsabilidad

Un contrato u obligación de una clase u otro elemento.

Semántica

La responsabilidad es un concepto informal. Una responsabilidad puede representarse como un estereotipo en un comentario. El comentario está enlazado a una clase u otro elemento que tiene la responsabilidad. La responsabilidad se expresa como una cadena de texto.

Notación

Pueden mostrarse responsabilidades en un compartimiento nombrado dentro de un símbolo del rectángulo del clasificador (Figura 14.261).

responsibility (estereotipo de Uso)

Una dependencia que expresa un contrato u obligación de un elemento.

Véase responsabilidad.

restricción

Condición semántica o restricción representada como texto en lenguaje natural o en un lenguaje formal especificado.

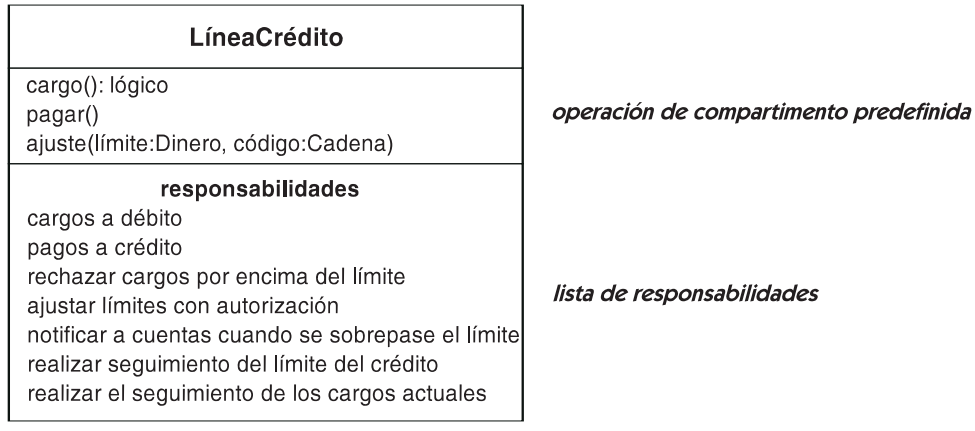


Figura 14.261 Compartimento para responsabilidades

Semántica

Una restricción es una condición semántica o una restricción expresada como una sentencia lingüística en algún lenguaje textual.

En general, una restricción se puede adjuntar a cualquier elemento del modelo o lista de elementos del modelo. Representa información semántica adjunta a un elemento del modelo, no sólo a una vista de él. Cada restricción tiene un cuerpo y un lenguaje de interpretación. El cuerpo es una cadena codificando una expresión lógica 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 un lenguaje natural. En el último caso, la restricción será informal y no sujeta a aplicación automática (lo que no significa que la aplicación automática sea siempre práctica para todos los lenguajes formales). UML proporciona el lenguaje de restricciones OCL [Warmer-99], pero también se pueden utilizar otros lenguajes.

Algunas restricciones comunes tienen nombres para evitar escribir una sentencia completa cada vez que se necesitan. Por ejemplo, la restricción **xor** entre dos asociaciones que comparten una clase común significa que un único objeto de la clase compartida debe pertenecer a una sola de las asociaciones en un momento dado.

Una restricción es una aserción, no un mecanismo ejecutable. Indica una restricción que debe ser impuesta por el correcto diseño del sistema. Cómo garantizar una restricción es una decisión de diseño.

Las restricciones en tiempo de ejecución están para ser evaluadas en ciertos momentos cuando un sistema instanciado es “estable” —es decir, entre la ejecución de operaciones y no en la mitad de una transacción atómica. Durante la ejecución de una operación, puede haber momentos en que las restricciones son violadas temporalmente. Puesto que los modelos pueden tener varios niveles de granularidad, las invariantes —es decir, restricciones que siempre son verdaderas— siempre tienen alguna cantidad de informalidad. En contraste, las precondiciones y las postcondiciones son restricciones que son evaluadas en tiempos bien determinados, a la invocación de una operación y a su finalización, de forma que no están sujetas a preocupaciones sobre cambios completados parcialmente.

Una restricción no se puede aplicar a sí misma.

Una restricción heredada —una restricción en un elemento del modelo antepasado en un estereotipo— debe ser observada aunque estén definidas restricciones adicionales en los descendientes. Una restricción heredada no puede ser separada o suplantada. Un modelo así está construido de forma pobre y debe ser reformulado. Sin embargo, una restricción heredada puede ser intensificada añadiendo restricciones adicionales. Si las restricciones heredadas por un elemento entran en conflicto, el modelo está mal formado.

Notación

Una restricción se representa como una cadena de texto encerrada entre paréntesis ({}). La cadena de texto es el cuerpo codificado escrito en un lenguaje de restricciones. Si una restricción tiene un nombre, el nombre se muestra como una cadena seguida por dos puntos, todo precediendo al texto de la restricción.

Se espera que las herramientas proporcionen uno o más lenguajes en cuyo formato se puedan escribir las restricciones. Un lenguaje predefinido para escribir restricciones es OCL. Dependiendo del modelo, un lenguaje informático como C++ puede ser útil para algunas restricciones. Por otro lado, la restricción se puede escribir en lenguaje natural, dejando bajo la responsabilidad humana la interpretación y aplicación. El lenguaje de cada restricción es parte de la propia restricción, aunque el lenguaje normalmente no se muestra en el diagrama (la herramienta mantendrá constancia de él).

Para una lista de elementos representada por cadenas de texto en un compartimento (como los atributos de una clase): Puede aparecer una cadena de restricción como una entrada en la lista (Figura 14.262). La entrada no representa un elemento del modelo. Es una *restricción en cascada* que se aplica a todos los siguientes elementos de la lista hasta que aparezca otra restricción en la lista o hasta el final de la misma. La restricción en cascada puede ser reemplazada más tarde en la lista por otra restricción en cascada. Para quitar la restricción en cascada, reemplácela por una restricción vacía. Una restricción adjunta a un elemento individual de la lista no reemplaza a la restricción en cascada pero puede aumentarla con restricciones adicionales.

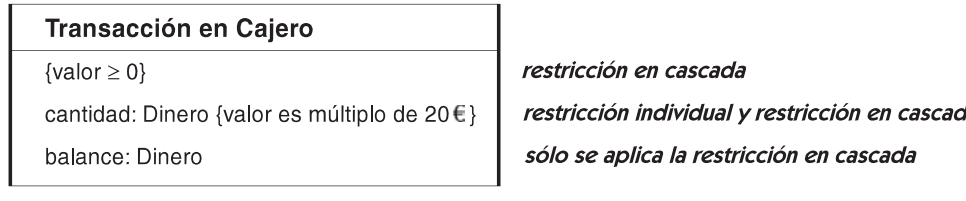


Figura 14.262 Restricciones dentro de listas

Para un solo símbolo gráfico (como una clase o un camino de la asociación): La cadena de restricción puede colocarse cerca del símbolo, preferiblemente cerca del nombre del símbolo, si existe.

Para dos símbolos gráficos (como dos clases o dos asociaciones): La restricción se representa como una línea discontinua desde un elemento al otro elemento, etiquetada con la cadena de restricción (entre paréntesis). La dirección de la flecha es información relevante dentro de la restricción (Figura 14.263).

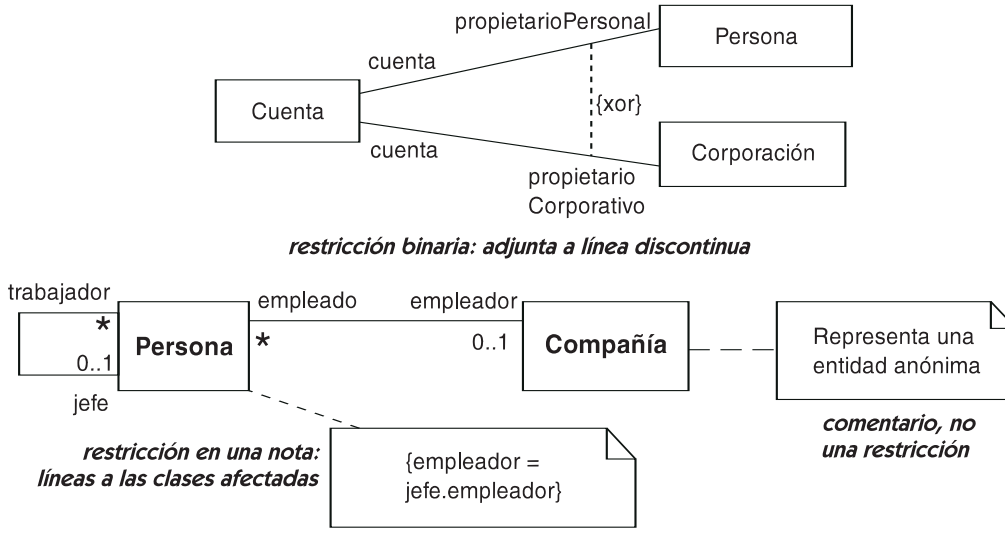


Figura 14.263 Notación de restricciones

Para tres o más símbolos gráficos: La cadena de restricción se coloca en un símbolo de nota y se adjunta a cada símbolo mediante una línea discontinua (Figura 14.263). Esta notación también se puede utilizar para los otros casos. Para tres o más caminos del mismo tipo (como caminos de generalización o caminos de asociación), la restricción se puede adjuntar a una línea discontinua cruzando todos los caminos. En caso de ambigüedad, las distintas líneas pueden ser numeradas o etiquetadas para establecer su correspondencia con la restricción.

El lenguaje de una restricción normalmente no se muestra. Se puede mostrar colocando el nombre del lenguaje entre paréntesis, todo dentro de los paréntesis de la propia restricción y precediendo al cuerpo de la restricción.

Discusión

Una restricción crea una sentencia semántica sobre el propio modelo, mientras que un comentario es una sentencia de texto sin obligación semántica y puede estar adjunto bien a un elemento del modelo o bien a un elemento de presentación. Tanto las restricciones como los comentarios pueden mostrarse utilizando notas. En principio, las restricciones son ejecutables por las herramientas. En la práctica, algunas pueden ser difíciles de plantear formalmente y puede requerir aplicación humana. En el sentido amplio de la palabra, muchos elementos en un modelo son restricciones, pero la palabra se utiliza para indicar sentencias semánticas que son difíciles de expresar utilizando los elementos predefinidos del modelo y que deben ser expresadas lingüísticamente.

Las restricciones se pueden expresar en cualquier lenguaje apropiado o incluso en lenguaje humano, aunque una restricción en lenguaje humano no se puede verificar mediante una herramienta. El lenguaje OCL [Warmer-99] está diseñado para especificar restricciones UML, pero bajo ciertas circunstancias un lenguaje de programación puede ser más apropiado.

Puesto que las restricciones se expresan como cadenas de texto, una herramienta genérica de modelado puede introducirlas y mantenerlas sin entender su significado. Por supuesto, una

herramienta o extensión que verifique o aplique la restricción debe entender la sintaxis y semántica del lenguaje destino.

Se puede adjuntar una lista de restricciones a la definición de un estereotipo. Esto indica que todos los elementos a los que se les aplique el estereotipo están sujetos a las restricciones.

Aplicación. Cuando el modelo contiene restricciones, no es necesario decir qué hacer si son violadas. Un modelo es una declaración de lo que se supone que ocurre. Es el trabajo de la implementación hacer que ocurra. Un programa podría contener aserciones y otros mecanismos de validación, pero el fallo de una restricción debe considerarse como un fallo de programación. Por supuesto, si un modelo puede ayudar a producir un programa que es correcto por construcción o puede ser verificado como correcto, entonces ha servido a su propósito.

restricción de duración

Restricción sobre una duración.

Notación

Una restricción de duración en un diagrama de secuencia se puede representar dibujando una línea vertical con una flecha abierta en ambos extremos entre la posición vertical de dos eventos. Se coloca una expresión para la duración entre paréntesis en el centro de la flecha. La restricción puede representarse también como una expresión de texto entre paréntesis. La Figura 14.6 muestra un ejemplo.

restricción de tiempo

Una restricción en el tiempo de una especificación de una ocurrencia expresado como un intervalo de tiempo.

Semántica

Una restricción de tiempo puede hacer referencia a una especificación de una ocurrencia simple o a un intervalo de tiempo entre dos ocurrencias. Se expresa como un intervalo de tiempo, pero la longitud del intervalo puede ser cero para restringirlo a un único punto en el tiempo. El tiempo puede ser tiempo absoluto o relativo a un tiempo inicial.

Notación

Una restricción de ocurrencia única se muestra en un diagrama de secuencia, dibujando una marca horizontal y colocando una expresión de tiempo entre llaves junto a él. Una restricción en un intervalo de tiempo se muestra dibujando una flecha con dos cabezas, entre las dos marcas de tiempo y colocando una expresión de intervalo de tiempo junto a ellos.

Ejemplo

La Figura 14.264, muestra varias restricciones de tiempo. El receptor debe descolgarse entre las 7 a.m. y las 7 p.m. El tono de marcado se debe recibir en menos de un segundo después de levantar el receptor. El primer dígito se debe marcar dentro de los siguientes diez segundos.

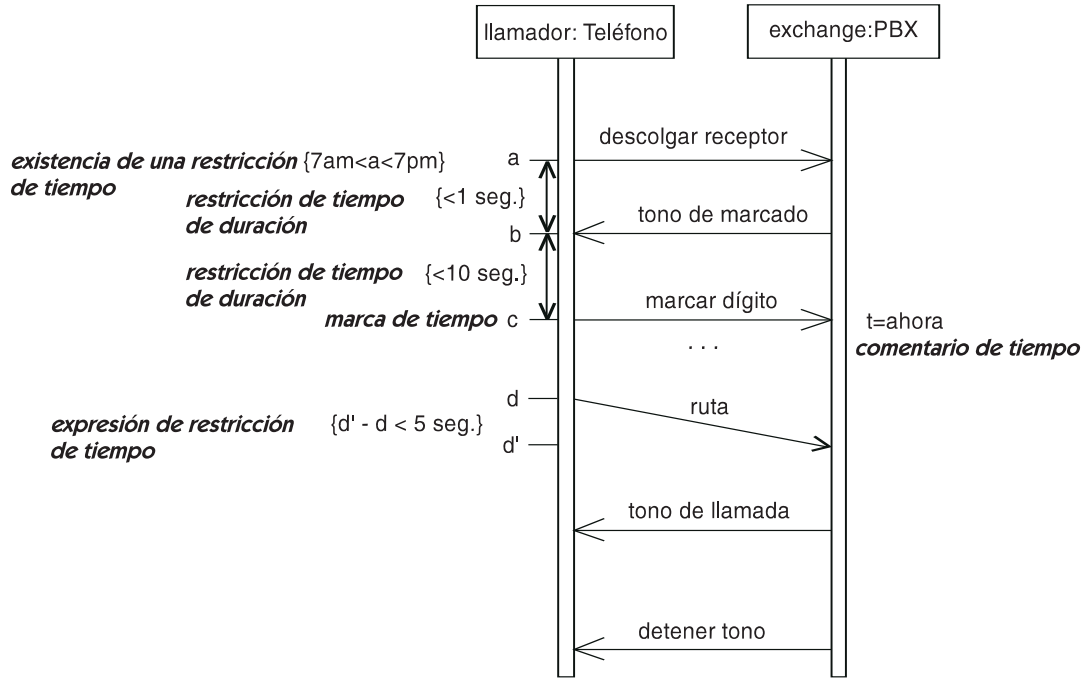


Figura 14.264 Restricción de tiempo

Una restricción de tiempo, también se puede representar como una cadena, tal y como se hace en la última restricción, que dice que el mensaje enrutado debe ser entregado en menos de cinco segundos.

resumen

Filtrar, combinar y abstraer las propiedades de un conjunto de elementos, en su contenedor, para llegar a un nivel más alto, una vista más abstracta de un sistema.

Véase paquete.

Semántica

Contenedores tales como paquetes y clases pueden tener propiedades y relaciones derivadas, que resumen las propiedades y relaciones de su contenido. Esto permite al modelador obtener una mejor comprensión de un sistema, a un nivel más alto, menos detallado y más fácil de entender. Por ejemplo, una dependencia entre dos paquetes indica que la dependencia existe entre por lo menos una pareja de elementos de esos dos paquetes. El resumen tiene menos detalle que la información original. Puede haber una o más parejas de dependencias individuales, representadas a nivel de paquete. En cualquier caso, el modelador sabe que un cambio a un paquete puede afectar al otro. Si se necesita más detalle, el modelador siempre puede examinar el contenido en detalle, una vez que el resumen de alto nivel le ha permitido detectarlo.

De manera similar, una dependencia de uso entre dos clases, normalmente indica una dependencia entre sus operaciones, tales como un método o una clase, que llama a una operación (no un método) en otra clase. Muchas dependencias a nivel de clase derivan de dependencias entre operaciones o atributos.

En general, las relaciones que se resumen en un contenedor indican la existencia de al menos un uso de la relación entre contenidos. Normalmente no indican que los elementos participan en la relación.

retorno

No hay ninguna acción de retorno explícita. La acción de retorno es implícita. A la terminación de ejecución del comportamiento invocado por una llamada síncrona, los valores de salida de la ejecución del comportamiento se ensamblan en un mensaje que se lleva a la ejecución de la acción de la llamada, restaurando el control a él y proporcionando los valores de retorno.

Véase acción.

reutilización

El uso de un artefacto preexistente.

Discusión

La reutilización se exige a menudo en la tecnología orientada a objetos, pero esta demanda es exagerada. Conceptos como operaciones de polimorfismo y encapsulación de la estructura interna son actualmente mucho más beneficiosos. Tenga presente que la reutilización puede ocurrir a través de otros medios además de la herencia, incluyendo la copia de código.

Uno de los grandes errores de diseño es forzar la generalización inapropiada en un intento de lograr reutilización, que en cambio a menudo es causa de confusión.

rol

Un elemento constitutivo de un clasificador estructurado del que representa la apariencia de una instancia (o, posiblemente, conjunto de instancias) dentro del contexto definido por el clasificador estructurado.

Véase colaboración, conector, clasificador estructurado.

Semántica

Un rol es la apariencia de un individuo dentro del contexto definido por un clasificador estructurado. Una instancia de una colaboración no posee los objetos en los que aparece su contexto; hace referencia a ellos meramente y pueden aparecer en contextos de colaboración múltiple. Una instancia de una clase estructurada posee sus partes; existen dentro de su contexto y no puede ser la parte de otra clase estructurada.

Un rol difiere de una asociación en que un rol se enlaza a un clasificador particular y todos los roles de un solo clasificador están implícitamente relacionados. La relación implícita, entre los papeles o roles dentro de un clasificador estructurado dado, puede modelarse usando explícitamente conectores.

rol de clasificador

Este concepto de UML1 ha sido reemplazado por rol y conector dentro de un clasificador estructurado. La sintaxis y la semántica ha cambiado considerablemente.

rol en la 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 conector, colaboración.

ruta

Una serie conectada de segmentos gráficos que conectan un símbolo a otro, normalmente representa una relación.

Notación

Una ruta es una cadena de símbolos gráficos conectados que representan a un concepto semántico único. Se usan rutas en la notación de varios tipos de relaciones, como asociaciones, generalizaciones y dependencias. Los puntos finales de dos segmentos conectados coinciden.

Un segmento puede ser un segmento de línea recto, un arco, o alguna otra forma (como una curva), aunque muchas herramientas sólo soportan líneas y arcos (Figura 14.265). Pueden dibujarse líneas con cualquier ángulo, aunque algunas herramientas de modelado prefieren restringir las líneas a los ángulos de ortogonales y posiblemente forzarlos hacia una recta regular para mejorar la apariencia y facilitar el diseño.

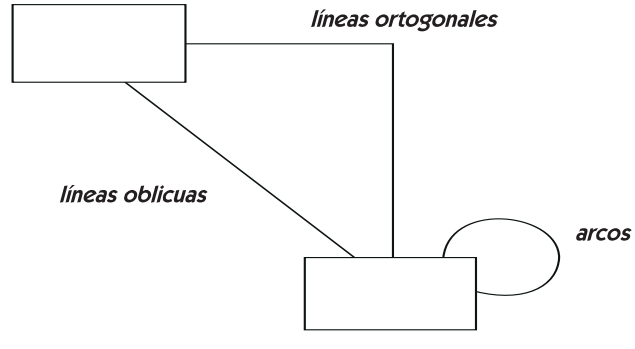


Figura 14.265 Ruta

Generalmente, la posición de una ruta no tiene importancia, aunque los caminos deben evitar regiones cerradas y cruces, porque cruzar el límite de una región gráfica puede tener significado. (Por ejemplo, una asociación entre dos clases en colaboración debe dibujarse dentro de la región de la colaboración para indicar una asociación entre los objetos de la misma instancia de la colaboración, considerando que una ruta hacia fuera de la región indicaría una asociación entre los objetos de instancias de colaboraciones diferentes.) Para ser más precisos, una ruta es topológica. Su posición exacta no tiene ninguna semántica, pero su conexión a, e intersección con otros, los símbolos puede tener importancia. El diseño exacto de las rutas afecta de forma importante al entendimiento y la estética, claro, y puede connotar sutilmente la importancia de relaciones y afinidades al real-mundo. Pero tales consideraciones son para los humanos y no las computadoras. Se espera que las herramientas apoyen el desarrollo fácil y permitan cambiar la posición de las rutas.

En la mayoría de los diagramas, el cruce de líneas no tiene importancia. Para evitar la ambigüedad sobre el significado del cruce de líneas, se puede dibujar un semicírculo pequeño o un hueco junto al punto de cruce (Figura 14.266). Normalmente, las herramientas de modelado tratan un cruce como dos líneas independientes. Las herramientas de modelado deben evitar asignaciones de ruta que podrían confundirse con cruces.

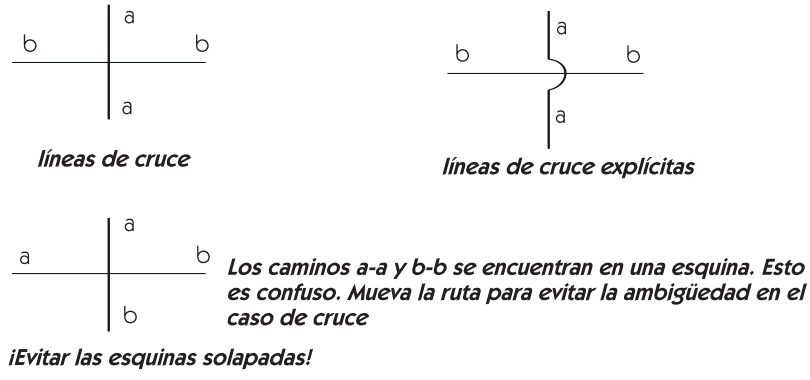


Figura 14.266 Cruce de rutas

En algunas relaciones (como agregación y generalización), varios caminos del mismo tipo pueden conectar a un solo símbolo. Si las propiedades de varios de los elementos se emparejan, entonces los segmentos de línea conectados al símbolo pueden ser combinados en un solo símbolo con una base y varias ramas. La base que se conecta al símbolo compartido y las ramas son caminos que se conectan a cada símbolo (Figura 14.267). Ésta es puramente una opción de la presentación gráfica. Conceptualmente, los caminos individuales son distintos. Esta opción de la presentación no se puede usar cuando la información modelada sobre varios segmentos no es idéntica.

script (estereotipo de Artefacto)

Un fichero que contiene texto que puede ser interpretado por un sistema de computadora.

Véase artefacto.

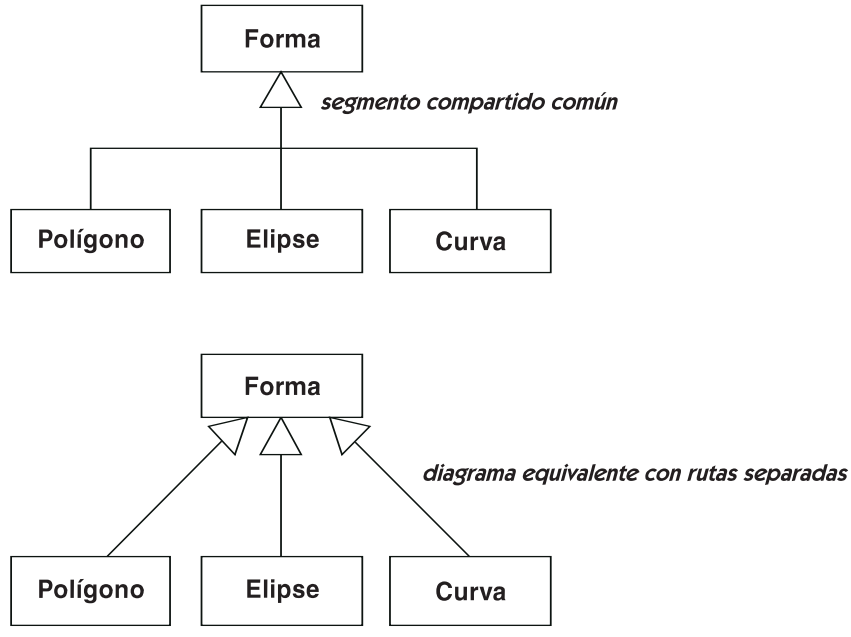


Figura 14.267 Rutas con un segmento compartido

sd

Etiqueta en un marco del diagrama que indica un diagrama de secuencia.

secuencia de acciones

Término obsoleto de UML1. La secuencia de acciones ahora se puede conectar mediante las transiciones de actividad.

secuenciación débil

Operador en un fragmento combinado de una interacción que indica secuenciación débil en los subfragmentos. *Véase también* secuenciación estricta.

Semántica

Este operador indica que las especificaciones de suceso en las mismas líneas de vida en diferentes operandos anidados están ordenadas, pero que las especificaciones de suceso en diferentes líneas de vida se pueden intercalar.

La distinción con la secuenciación estricta es sutil y es poco probable que sea utilizada por la mayoría de modeladores. *Véase* la especificación de UML para más detalles.

secuenciación estricta

Un operador en un fragmento combinado de una interacción que indica la secuencia estricta de los operadores. *Véase también* secuencia débil.

Semántica

Este operador indica que los operandos de más alto nivel del fragmento combinado son ejecutados estrictamente en orden, en lugar de posiblemente entrelazados.

La diferencia es sutil e improbable para la mayoría de los modeladores. *Véase* la especificación de UML para más detalles.

segmento

Un fragmento de la transición que puede incluir un pseudoestado.

Semántica

Un segmento es un fragmento de la transición que puede incluir los pseudoestados. Una transición compuesta es una transición entre dos estados. Puede componerse de una cadena de segmentos.

Durante la ejecución de un sistema, un cambio de estado debe envolver una transición compuesta completa. No es posible para un pseudoestado permanecer activo. Los segmentos son, por consiguiente, las unidades sintácticas para construir las transiciones compuestas y son transiciones en nombre solamente.

semántica

La especificación formal del significado y comportamiento de algo.

send (estereotipo de dependencia de uso)

Una dependencia, cuyo origen y destino es una operación o un clasificador, especificando que el cliente envía una señal a algún destino no especificado.

Vea envío, señal.

señal

La especificación de un bloque de información para ser comunicado asincrónicamente entre objetos. Las señales tienen los parámetros expresados como atributos.

Véase también evento, mensaje, envío.

Semántica

Una señal es un clasificador con nombre pensado para comunicación explícita entre objetos. Tiene atributos que constituyen su información. Se envía explícitamente por un objeto a otro objeto mediante una acción de envío. Una acción de transmisión envía una señal al conjunto de todos los objetos, aunque, en la práctica, se llevaría a cabo de modo diferente por eficacia. El remitente especifica los atributos de la señal en el momento que se envía o como una lista del argumento o proporcionando un objeto señalado cuyos atributos ya han sido inicializados. Una vez que una señal se ha enviado, el remitente reasume la ejecución concurrentemente con la transmisión del signo al destino. El recibo de una señal por su objeto designado es un evento considerado como una transición disparadora en la máquina de estados del receptor o invocación de métodos asíncronos. Una señal enviada a un conjunto de objetos puede activar cero o una transición en cada objeto independientemente. Las señales son medios explícitos por los que los objetos pueden comunicarse entre sí asincrónicamente. Para realizar la comunicación síncrona, deben usarse dos señales, una en cada dirección.

Las señales son los clasificadores. Una señal puede tener un supertipo. Hereda los atributos del supertipo y puede agregar los atributos adicionales propios. Un signo activa cualquier transición declarada para usar uno de sus signos de antepasados.

Una declaración señalada tiene el alcance dentro del paquete en el que se declara. No está restringido a una sola clase. Se declaran señales independientemente de clases.

Una clase o interfaz puede declarar las señales que se prepara para manejar. Una declaración tal es una recepción. Una recepción puede especificar un método a ser ejecutado al recibo de una señal. Una señal es un clasificador y puede tener operaciones que puedan acceder y modificar sus atributos. Estas operaciones pueden ser usadas por el remitente para construir el objeto designado y por el receptor para acceder a su valor.

Notación

La palabra clave del estereotipo «**signal**» dentro de un compartimiento de una operación indica la declaración de una recepción. El nombre del signo puede ser seguido de una lista de sus parámetros (atributos) dentro de paréntesis. La declaración no puede tener un tipo de retorno.

La declaración de un tipo de señal puede expresarse como un estereotipo en un símbolo de clase.

La palabra clave «**signal**» aparece en un rectángulo sobre el nombre de la señal. Los parámetros de la señal aparecen como atributos dentro del compartimiento del atributo. El compartimiento de las operaciones puede contener las operaciones de acceso.

Un parámetro de señal se declara como un atributo que puede tener un valor inicial que puede ser eliminado durante la inicialización o el envío. El valor inicial se usa si se crea una instancia de una señal, se inicializa y, a continuación, se envía como un objeto. Si una señal se envía usando la sintaxis de operación-llamada, los valores iniciales de los atributos de la señal son los valores por defecto de los parámetros en la acción de llamada.

La Figura 14.268 muestra el uso de notación de generalización para relacionar una señal hija con su padre. El hijo hereda los parámetros de sus antepasados y puede agregar parámetros adicionales a los suyos propios. Por ejemplo, la señal de **BotonRatónAbajo** tiene los atributos **tiempo**, **dispositivo** y **ubicación**.

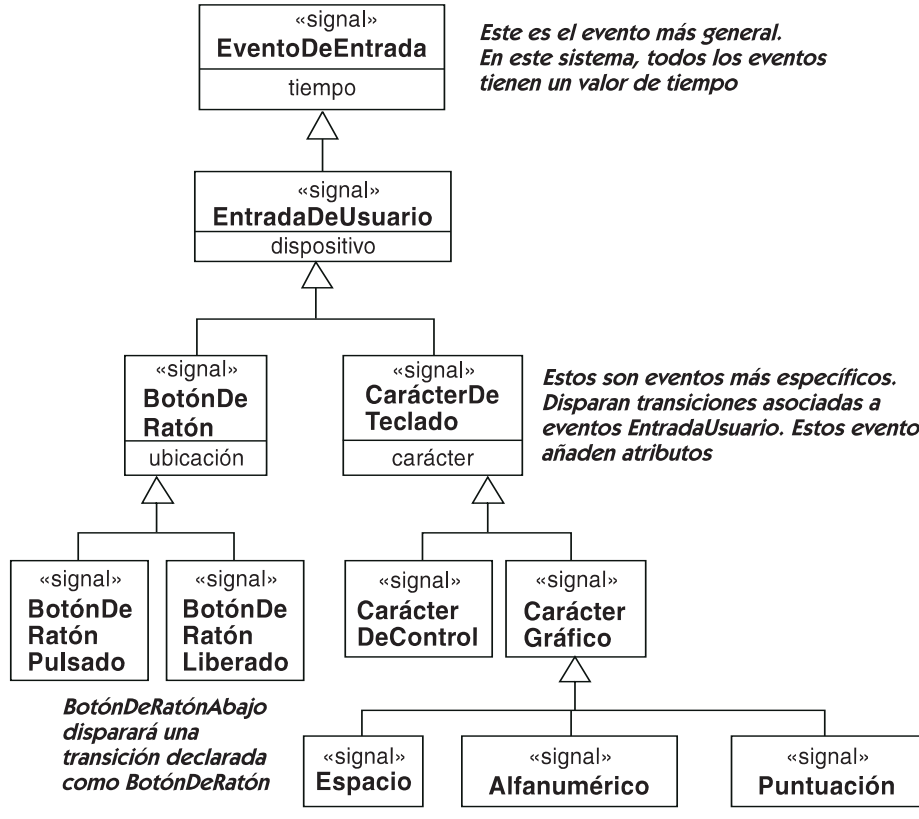


Figura 14.268 Declaración de señales

Para usar una señal como disparador de una transición, use la sintaxis:

`nombre-de-señal`(**`parámetro`**$_{\texttt{lista}}$)

Un parámetro tiene la sintaxis:

`nombre-de-parámetro:tipo-expresión`

Discusión

Una señal es la comunicación fundamental entre los objetos, teniendo una semántica más simple y más clara que las llamadas a procedimientos. Una señal es inherentemente una comunicación asíncrona en sentido único de un objeto a otro, en el cual toda la información es pasada por valor. Es un modelo conveniente para comunicación en sistemas concurrentes distribuidos.

Para construir comunicaciones síncronas, use pares de señales, uno en cada dirección.

Una llamada puede considerarse como una señal con un parámetro de indicador de retorno implícito, aunque incluye un mecanismo bastante complicado.

seq

1. Etiqueta para un fragmento combinado dentro de un diagrama de secuencia que representa secuencia débil.

2. Palabra clave en una especificación de multiplicidad que indica que los elementos del conjunto forman una secuencia ordenada.

service (estereotipo de Componente)

Un componente sin estado, componente funcional que devuelve un valor en lugar de realizar un efecto colateral.

Véase componente.

servicio de metaobjetos

Véase MOF.

signatura

El nombre y propiedades del parámetro de un rasgo conductual, como una operación o una señal. Una signatura puede incluir los tipos de retorno optativos (para las operaciones, no para las señales).

Semántica

La signatura de una operación es parte de su declaración. Algo (pero no todo) de la signatura se usa para las operaciones de máquina y métodos para verificar conflicto o eliminación. Los detalles de lo que se incluye para emparejar y lo que se excluye puede ser específico del lenguaje. Si dos firmas emparejan, pero las propiedades restantes son incoherentes (por ejemplo, un parámetro **in** corresponde a un parámetro **out**), entonces hay conflicto de declaraciones y el modelo está malformado.

singleton

Una clase que tiene (por declaración) exactamente una instancia. Un singleton es una manera de representar conocimiento global en una aplicación, todavía dentro del marco de la orientación a objetos.

Semántica

Cada aplicación debe tener una clase singleton (a menudo implícitamente) al menos para establecer el contexto para la aplicación. A menudo, la clase singleton iguala a la aplicación misma y es implementado por la pila de control y espacio de direcciones en una computadora.

Un singleton debe existir dentro del contexto que declara su alcance, como una clase estructurada o componente. A menudo el contexto es una clase estructurada que representa el sistema completo.

Notación

Un singleton se muestra como un símbolo de clase con un pequeño "1" en la esquina superior derecha (Figura 14.269). Este valor representa la multiplicidad de la clase dentro del contexto.

Un singleton debe existir dentro de un contexto. Esta notación puede usarse dentro de un diagrama de la clase entendiendo que hay un contexto de ejecución implícito.

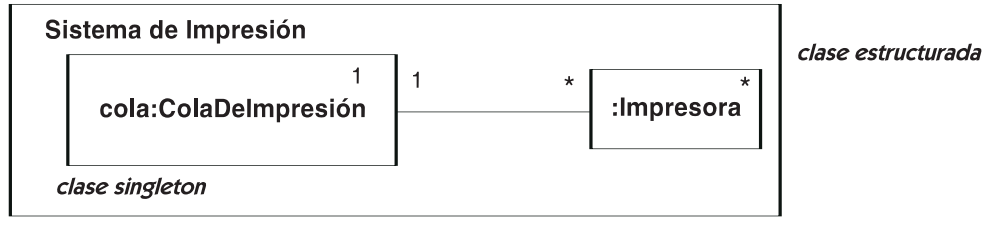


Figura 14.269 Clase singleton

sistema

Una colección de unidades conectadas, organizadas para lograr un propósito. Un sistema puede describirse por uno o más modelos, posiblemente desde diferentes puntos de vista. El "modelo completo" describe a todo el sistema.

Semántica

El sistema se modela mediante un subsistema de nivel superior, que indirectamente contiene el conjunto completo de elementos del modelo, que trabajan juntos para lograr un propósito completo, perteneciente al mundo real.

solicitud

Un objeto de datos enviado a un objeto designado como una señal.

Véase enviar, señal.

source (estereotipo de Artefacto)

Un fichero que contiene el texto de un programa que puede compilarse en código ejecutable.

Véase fichero.

specification (estereotipo de Clasificador)

Un clasificador que describe un dominio de objetos sin proporcionar una implementación física.

Típicamente describe interfaces o restricciones en los objetos sin enumerar sus características. *Véase* realización, especificación.

strict

La palabra clave que indica una secuencia estricta en un fragmento combinado en un diagrama de interacción.

Véase secuencia estricta.

subclase

El hijo de otra clase en una relación de generalización. Esto es, una descripción más específica. La clase hija se llama subclase. La clase padre se llama superclase. *Véase* generalización, herencia.

Semántica

Un subclase hereda la estructura, relaciones y comportamiento de su superclase y puede agregar elementos a ella.

subconjunto

Palabra clave en el extremo de una asociación o atributo que indica que la colección de objetos que forman un valor es un subconjunto de la colección de objetos que forman el valor de otro extremo de una asociación o atributo.

Véase también unión.

Semántica

Algunas veces, la misma propiedad (una asociación o un atributo), se puede modelar a más de un nivel de granularidad, un nivel general que cubre varios casos y uno más específico dividido en casos específicos. Si una propiedad se declara como un subconjunto de otra propiedad, el conjunto de valores para la primera propiedad es un subconjunto de los valores para la segunda propiedad. Un valor para una única propiedad puede encontrarse entre los valores de otras propiedades. Típicamente, la propiedad más general se encuentra dividida en múltiples subconjuntos que cubren su conjunto completo de posibles valores, en cuyo caso, se declarará como una unión. Esto no es, sin embargo, siempre verdadero.

Notación

Una declaración de subconjuntos tiene la forma:

{subsets **`nombre-de-otra-propiedad`**`}`

La declaración se coloca después de la declaración de la propiedad que forma el subconjunto y referencia a la propiedad que incluye el subconjunto.

La Figura 14.270 muestra un ejemplo de declaraciones de subconjuntos y uniones en asociaciones. El extremo de la asociación **dirección** se declara a alto nivel entre las clases **vehículo** y **control de dirección**. Cada una de esas clases, se especializa en varias subclases. Dirección no es totalmente general entre las subclases, sin embargo. Un coche sólo se dirige mediante un volante, un barco sólo se dirige mediante un timón, y un caballo sólo se dirige mediante las riendas. (O.K., hay otras formas de dirigir estos elementos. Esto es sólo un ejemplo.) La declaración de los subconjuntos en cada una de las asociaciones especializadas, declara que son casos especiales de la asociación general **dirección**. La declaración de uniones en asociaciones generales, declara que no son casos únicos, cada instancia de la asociación dirección debe también ser una instancia de las asociaciones que son subconjuntos de ella. Por lo tanto, sabemos que no podemos dirigir un caballo con un volante, aunque una asociación general lo permitiría.

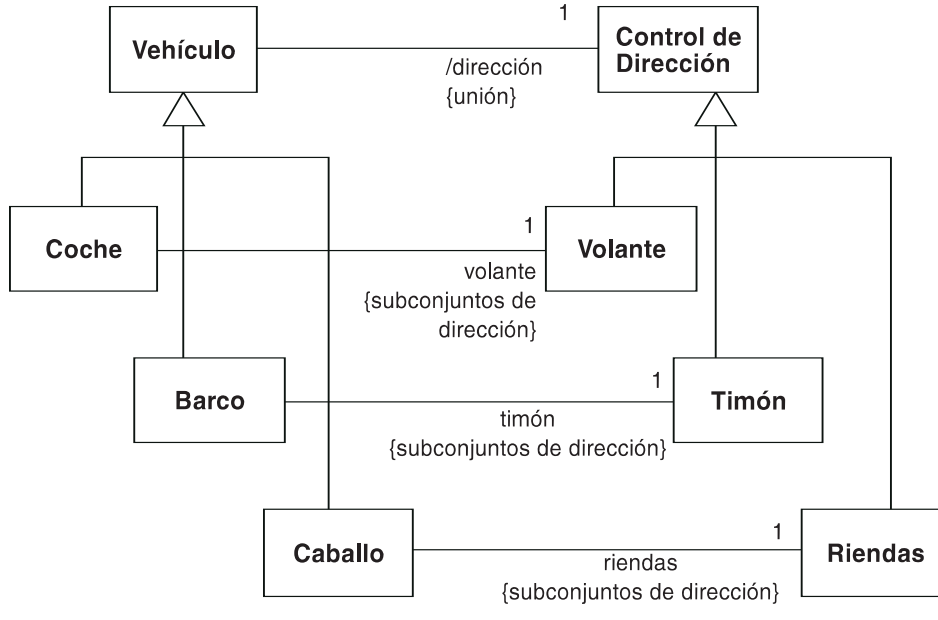


Figura 14.270 Subconjuntos y uniones

Este patrón ocurre con frecuencia, cuando una asociación entre clases generales necesita aplicarse a subclases.

Si la declaración de la unión en una asociación general no existe, entonces pueden existir otras combinaciones de valores, además de los subconjuntos explícitos.

subconjunto de par

Definición de una propiedad que cubre un subconjunto de instancias de otra propiedad.

Véase subconjuntos.

subestado

Un estado que es parte de un estado compuesto.

Véase estado compuesto, subestado directo, subestado indirecto, región.

Semántica

Un estado compuesto contiene una o más regiones. Cada región, contiene uno o más estados. Cada estado es un subestado directo de la región a la que pertenece y del estado compuesto que posee a la región. Es un subestado indirecto de estados a alto nivel.

Los subestados directos dentro de una región simple son mutuamente excluyentes. Si la región está activa, uno y sólo uno de sus subestados directos estará activo. Si un estado compuesto está activo, cada región está activa, así que uno y sólo uno de los subestados directos de cada región estará activo. Los subestados forman un árbol "and-or" de actividades —la descomposición de estados compuestos en múltiples regiones, es un "and" de la actividad, mientras que la descomposición de regiones en múltiples subestados, es un "or" de la actividad.

El anidamiento de subestados dentro de estados, es de alguna manera similar (pero no lo mismo) que la generalización de clasificadores. Un subestado anidado, responde a todas las transiciones que salen de sus estados contenidos, si no están redefinidas por transiciones de más bajo nivel.

Notación

Véase estado compuesto para ejemplos de notación.

subestado concurrente

Este término de UML1 no aparece en UML2. La concurrencia ahora se modela utilizando regiones ortogonales. *Véase* región ortogonal, estado ortogonal.

subestado directo

Con respecto a un estado compuesto, estado contenido por él sin estar contenido en un estado intermedio; un estado de nivel superior dentro de una región de un estado compuesto.

Véase estado compuesto, subestado indirecto, región, estado.

Semántica

Un estado compuesto contiene una o más regiones. Cada región directamente contiene uno o más estados. Un estado de nivel superior de una región se conoce como *subestado directo* del estado compuesto que contiene a la región. La expresión también se puede utilizar con respecto a una región. Si está activo un estado compuesto, está activo exactamente un subestado directo de cada región.

Los subestados directos por sí mismos pueden incluir estados compuestos. Sus subestados se conocen como subestados indirectos del estado compuesto original. Puesto que algunos de los

subestados directos o indirectos pueden tener varias regiones, un estado puede tener varios subestados indirectos activos por región.

subestado disjunto

Este término UML1 no aparece en UML2. *Véase* subestado directo.

subestado indirecto

Respecto a un estado compuesto, estado contenido por el que también está contenido por otro subestado del estado compuesto, y que por tanto no es un subestado directo del estado compuesto; estado anidado en otro nivel distinto al nivel superior.

Véase estado compuesto, subestado directo.

submáquina

Una máquina de estados que puede invocarse como parte de otra máquina de estados. No se asocia a una clase pero en cambio es un tipo de subprograma de la máquina de estados. Tiene la semántica de ser equivalente a que su contenido fuera reproducido e insertado al estado que le hace referencia.

Véase máquina de estados, estados, estado de la submáquina.

subsistema (estereotipo de Componente)

Una unidad de descomposición grande para un sistema. Se modela en UML como un estereotipo de componente.

Semántica

Un subsistema es una pieza coherente del diseño de un sistema. Puede tener su propia especificación y porciones de realización. Normalmente, un subsistema se instancia indirectamente, eso es, su comportamiento se realiza por otras clases.

El sistema en sí mismo constituye un subsistema de nivel superior. La realización de un subsistema puede ser escrita como una colaboración de subsistemas de nivel inferior. De esta forma, el sistema completo puede expandirse, como una jerarquía de subsistemas, hasta que los subsistemas de nivel más bajo se definen en términos de clases ordinarias.

Un subsistema se modela como un componente (grande).

Las connotaciones de un subsistema varían enormemente, dependiendo del punto de vista del modelado.

Notación

Un susbsistema se nombra como un símbolo de componente, con la palabra clave «**subsystem**» (Figura 14.271).

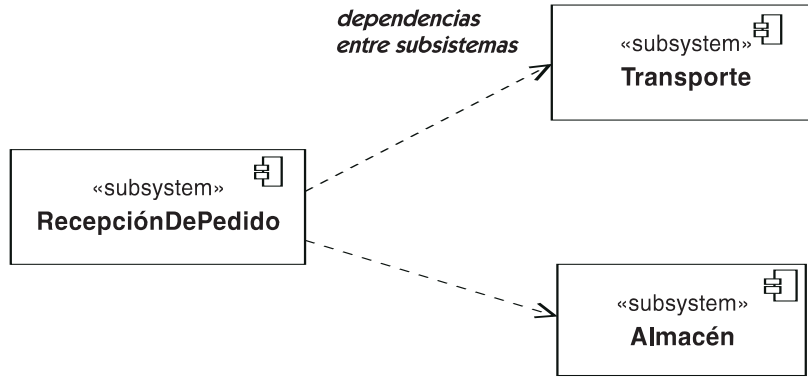


Figura 14.271 Subsistemas

subtipo

Un tipo que es hijo de otro tipo.

Se puede utilizar el término hijo, más neutro, para cualquier elemento generalizable.

Véase generalización.

suceso

Algo notable que pasa durante la ejecución de un sistema; una instancia de un evento. A veces llamada *ocurrencia de un evento*, pero la palabra *evento* es redundante. Ocurrencia.

Semántica

Una ocurrencia es algo que pasa en espacio y tiempo, así como un objeto es algo que existe con un límite bien-definido. Por ejemplo, “El Rey Luis XIV murió en Versalles el 1 de septiembre de 1715” es una ocurrencia. Un evento es el descriptor de un grupo de ocurrencias potenciales que comparten rasgos comunes, así como una clase es el descriptor de un grupo de objetos potenciales que comparten los rasgos comunes. Por ejemplo “muerte del rey” es un evento al que siguen muchas consecuencias y por lo tanto es interesante. Objetos y ocurrencias son las únicas entidades en tiempo de ejecución, considerando que las clases y eventos son los elementos ejemplares que describen las entidades tiempo de ejecución. Un objeto es una instancia de una clase y una ocurrencia es una instancia de un evento. (Precisamente en estos términos, la palabra instancia sólo se aplica a los clasificadores, eso es, la

estructura estática, pero la relación realmente es la misma.) En cualquier caso, tenemos la relación:

objeto es a clase como ocurrencia es a evento

Los objetos y ocurrencias normalmente no aparecen en los modelos. Historias (trazas en terminología de UML) describe objetos y ocurrencias únicas, pero modelos tienen más que ver con patrones repetibles que se aplican a muchos objetos y ocurrencias. Por ejemplo, una ocurrencia del evento "muerte del rey" puede activar las consecuencias como un funeral de estado, lucha de potenciales sucesores por el puesto y la coronación de un nuevo rey. En UML, una máquina de estados podría modelar estas consecuencias; sus transiciones serían activadas por eventos.

Un evento es un tipo, es decir, una categoría de ocurrencias con características similares. Un modelo necesita a menudo modelar la ocurrencia de un evento dado en un contexto particular, como una sucesión de eventos de tipos específicos. Por ejemplo, podríamos tener la secuencia "Cuando un evento 'el rey muere' ocurre, entonces el evento 'funeral de estado' ocurre, entonces 'la coronación del nuevo rey' ocurre." El modelo no está refiriéndose a una ocurrencia particular como "Luis XIV muere", pero sí a cualquier ocurrencia del evento "muere el rey". Este tipo de elemento ejemplar es una especificación de la ocurrencia. Describe muchas ocurrencias potenciales de un evento dado dentro de un contexto específico.

Notación

Las ocurrencias individuales raramente aparecen en los modelos, no hay ninguna notación definida por consiguiente. Normalmente las características técnicas de la ocurrencia aparecen en modelos para mostrar las secuencias de eventos.

sujeto

El clasificador cuyo comportamiento se describe por un caso de uso.

Semántica

El sujeto es un clasificador que comprende el comportamiento definido por un caso de uso. Un caso de uso no necesita tener sujeto (pero entonces sería una descripción vaga del comportamiento del sistema). El sujeto a menudo, pero no siempre, posee el caso de uso. El caso de uso tiene acceso a las características de su sujeto.

Un clasificador puede tener muchos casos de uso. Cada caso de uso describe una faceta del comportamiento total del clasificador.

Notación

Un sujeto se representa por un rectángulo con su caso de uso dibujado dentro. El nombre del sujeto se muestra como una cadena de texto. Los actores que se comunican con los casos de uso se dibujan fuera del rectángulo. *Véase* la Figura 14.41 para un ejemplo.

superclase

El padre de otra clase, en una relación de generalización —eso es la especificación del elemento más general. La clase hija es clasificada como subclase. La clase padre es llamada superclase.

Véase generalización.

Semántica

Una subclase hereda la estructura, relaciones y comportamiento de su superclase, a los que puede añadir otros nuevos.

supertipo

Sinónimo de superclase. Se puede usar el término más neutro padre, para cualquier elemento generalizable.

Véase generalización.

supratipo

Una metaclase cuyas instancias son subclases de una clase dada.

Véase también generalización, conjunto de generalización, metaclase.

Semántica

Las subclases de una clase dada pueden ser consideradas instancias de una metaclase en sí mismas. Tal metaclase es llamada supertipo. Por ejemplo, la clase **Árbol** puede tener subclases **Abeto**, **Olmo** y **Encina**. Considerados como objetos, esas subclases son instancias de la metaclase **EspeciesDeÁrbol**. **EspeciesDeÁrbol** es el supertipo que se encuentra sobre **Árbol**.

Las instancias de un supertipo son todas subclases de una única superclase, así que el conjunto de relaciones de generalización constituyen un conjunto de generalización. Una superclase puede tener múltiples conjuntos de generalización y por lo tanto múltiples supertipos. Supertipos y conjuntos de generalización son formas alternativas de buscar la misma cualidad. Un supertipo añade la capacidad de declarar propiedades que se aplican a las clases individuales en los conjuntos de generalización. Esas propiedades se aplican a las propias clases, no a los objetos que son instancias de las clases. Por ejemplo **EspeciesDeÁrbol** puede tener propiedades como distribución geográfica y tiempo medio de vida.

Notación

Un supertipo se muestra como una clase con la palabra clave «**powertype**» (Figura 14.272). El nombre de un supertipo se puede usar como etiqueta para el conjunto de generalización en una flecha de generalización (*véase* conjunto de generalización).

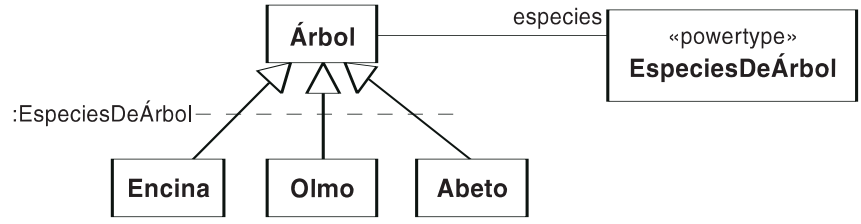


Figura 14.272 Supertipo

sustitución

Una dependencia entre clasificadores, que declara que el clasificador origen puede ser sustituido en un lugar donde el clasificador destino fue declarado como tipo.

Semántica

La generalización, habitualmente, implica sustitución (*véase* principio de sustitución), pero algunas veces, un clasificador especifica un contrato para interacciones —interfaces y puertos— sin especificar sus características. Cualquier clasificador que implemente las mismas interfaces y puertos, puede ser sustituido por el clasificador original, incluso si no comparte ninguna de las estructuras internas o implementaciones. Esta relación se expresa mediante una dependencia de sustitución, desde el clasificador de implementación hasta el clasificador que especifica el contrato. Se puede usar en situaciones donde la generalización es inapropiada.

Notación

La sustitución se representa como una dependencia, dibujando una flecha punteada con la palabra clave «**substitute**».

sytemModel (estereotipo de Modelo)

Un modelo de alto nivel que contiene una colección de modelos, que describen un sistema físico.

Véase modelo, subsistema, sistema.

terminación

Una terminación cuya ejecución termina la ejecución de una máquina de estados por el propio objeto.

Semántica

La ejecución de un pseudoestado de ejecución termina la ejecución de una máquina de estados. Normalmente, esto significa que el objeto que lo posee se destruye.

Bajo las circunstancias normales, una máquina de estados termina cuando alcanza su estado final de más alto nivel, por lo tanto, el subestado de terminación rara vez se necesita.

Notación

Un subestado de terminación se muestra con una X grande.

tiempo de análisis

Periodo durante el cual se realiza una actividad de análisis del proceso de desarrollo de un sistema. No se asume que todo el análisis de un sistema se realiza al mismo tiempo o que precede a otras actividades, como el diseño o la implementación. Las distintas actividades son secuenciales para un único elemento, pero distintas actividades se pueden entremezclar para el sistema completo.

Véase tiempo de diseño, tiempo de modelado.

tiempo de compilación

Se refiere a algo que sucede durante la compilación de un módulo de software.

Véase tiempo de modelado, tiempo de ejecución.

tiempo de diseño

Se refiere a lo que ocurre durante una actividad de diseño del proceso de desarrollo de software. Contrastar con: tiempo de análisis.

Véase tiempo de modelado, etapas de modelado.

tiempo de ejecución

El período de tiempo durante el que un programa de computadora se ejecuta. Contraste: tiempo de modelado.

tiempo de modelado

Referencia a algo que ocurre durante la actividad de modelado del proceso de desarrollo de software. Incluye análisis y diseño. Nota de utilización: Cuando se discuten los objetos del sistema, a menudo es importante distinguir entre tiempo de modelado y asuntos de ejecución.

Véase también proceso de desarrollo, etapas de modelado.

tipo

Declaración de un clasificador que restringe el valor de un atributo, parámetro, resultado de una operación o variable. El valor real debe ser una instancia del tipo o de uno de sus descendientes.

tipo de concurrencia

Especificación de la capacidad de una operación para aceptar llamadas simultáneas.

Semántica

La ejecución de cada llamada a una operación pasiva es independiente. Sin embargo, puesto que las operaciones pueden acceder a recursos compartidos en un objeto, a veces se necesita impedir la ejecución concurrente de operaciones múltiples. La propiedad de concurrencia de una operación puede tener los siguientes valores:

concurrente	No hay restricciones en la invocación simultánea de operaciones sobre el mismo objeto.
secuencial	La invocación simultánea de operaciones sobre el mismo objeto puede causar problemas. El modelador es responsable de asegurar que las invocaciones simultáneas no se producen. Si se producen, el sistema está mal formado.
con guardas	Puede producirse la invocación simultánea de operaciones sobre el mismo objeto, pero dichas invocaciones son capturadas y gestionadas dinámicamente por el entorno de ejecución de forma que sólo se puede ejecutar una invocación en un momento dado. Cuando se completa una ejecución, se permite proceder a una invocación bloqueada, si existe. No hay ninguna garantía de que las invocaciones se ejecutarán en cualquier orden particular (como el orden en que fueron llamadas). El modelador es responsable de asegurar que los interbloqueos sobre recursos compartidos no causen que una invocación tenga que esperar indefinidamente, impidiendo proceder a otras invocaciones sobre el mismo objeto (y potencialmente despejar el bloqueo).

Discusión

Esta propiedad parece ser un vestigio de Ada que mezcla conceptos lógicos con asuntos de implementación, de una forma no totalmente satisfactoria. La especificación es ambigua sobre si se aplica a múltiples invocaciones de la misma operación o a invocaciones de diferentes operaciones en el mismo objeto. Las elecciones no cubren todas las posibilidades de un sistema transaccional. Por ejemplo, no se proporciona la vuelta atrás de operaciones (rollback). En cualquier caso, incluir un procesamiento completo de transacciones estaría más allá del ámbito de un modelo universal como UML. Probablemente, para los modeladores sería mejor evitar por completo este concepto y llevar a cabo la serialización utilizando objetos activos, en vez de esconder la sincronización bajo una propiedad aparentemente complicada.

tipo de datos

Descriptor de un conjunto de valores que carecen de identidad (existencia independiente y la posibilidad de efectos laterales). Los tipos de datos incluyen tipos primitivos predefinidos y tipos definibles por el usuario. Los tipos primitivos incluyen números, cadenas y valores lógi-

cos. Los tipos definibles por los usuarios son enumeraciones. Los tipos de datos anónimos pensados para su implementación en un lenguaje de programación se pueden definir utilizando tipos del lenguajes dentro de perfiles.

Véase también clasificador, identidad.

Semántica

Los tipos de datos son las primitivas predefinidas que se necesitan como base para los tipos definibles por los usuarios. Su semántica está definida matemáticamente fuera del mecanismo de construcción de tipos en un lenguaje. Los números están predefinidos. Incluyen a los enteros y a los reales. Las cadenas también están predefinidas. Estos tipos de datos no son definibles por el usuario.

Los tipos enumerados son conjuntos finitos de elementos con nombre, definibles por el usuario, que tienen definida una ordenación entre ellos, pero no tienen ninguna otra propiedad computacional. Un tipo enumerado tiene un nombre y una lista de constantes enumeradas. El tipo enumerado **Boolean** está predefinido con los literales enumerados **false** y **true.**

Se pueden definir operaciones sobre los tipos de datos, y las operaciones pueden tener tipos de datos como parámetros. Puesto que un tipo de datos no tiene identidad y es simplemente un valor puro, las operaciones sobre los tipos de datos no los modifica; en vez de eso, simplemente devuelven valores. Eso hace que no tenga sentido hablar de crear un nuevo valor de tipo de datos, puesto que no tienen identidad. Todos los valores de tipo de datos son (conceptualmente) predefinidos. Una operación sobre un tipo de datos es una consulta que no puede cambiar el estado del sistema pero puede devolver un valor.

Un tipo de datos también puede ser descrito por un tipo de lenguaje —una expresión de tipo de datos en un lenguaje de programación. Dicha expresión designa un tipo de datos anónimo en un lenguaje de programación de destino. Por ejemplo, la expresión **Persona*(*)(String)** denota una expresión de tipo en C++ que no corresponde a un tipo de datos simple con un nombre. Los tipos de lenguaje no están definidos en la especificación de UML, pero podrían ser añadidos por un perfil para un lenguaje particular.

tipo del lenguaje

Tipo de datos anónimo definido en la sintaxis de un lenguaje de programación.

Véase también tipo de datos.

Semántica

Un tipo del lenguaje es una construcción que sigue las reglas de sintaxis de un lenguaje de programación particular, normalmente expresado como una cadena. En UML2, sería una clase de tipo de datos. Cuando modelamos estrechamente las construcciones de un lenguaje de programación, podría usarse como el tipo de un atributo, variable o parámetro.

Por ejemplo, el tipo de datos de C++ «**Persona*(*) (Contrato*,int)**» podría definirse como un tipo del lenguaje de C++.

UML no contiene un tipo de datos genérico variante ajustable para declarar tipos de datos del lenguaje de programación. Si se necesitan, se deben añadir en perfiles para un lenguaje de programación particular.

tipo primitivo

Un dato básico predefinido, como un entero, lógico o cadena.

Véase también enumeración.

Semántica

Las instancias de los tipos primitivos no tienen identidad. Si dos instancias tienen la misma representación, entonces son indistinguibles y pueden ser pasados a por valor sin pérdida de información.

Los tipos primitivos incluyen números, cadenas, y posiblemente otro tipo de datos dependientes del sistema, por ejemplo fechas y dinero, cuya semántica se define fuera UML. Los tipos primitivos adicionales pueden ser agregados por perfiles. Los tipos primitivos normalmente corresponden estrechamente a aquellos encontrados en el lenguaje de programación designado.

Véase también enumeraciones, que son tipos de datos definidos por el usuario que no son tipos primitivos.

tipo registro

Un tipo del registro tradicional puede representarse en UML como un tipo de datos con atributos.

token

Presencia de un punto de control, incluyendo posibles valores de datos, durante la ejecución de una actividad.

Semántica

La semántica de ejecución de las actividades se basa en el concepto de tokens como marcadores de actividades. El concepto originalmente viene de la teoría de redes de Petri. Un token es un marcador en tiempo de ejecución de la presencia de un punto de control. Un token también puede incluir valores de datos que estén implicados por el punto de control. Un gráfico de actividad puede tener varios tokens, indicando ejecución concurrente.

Las reglas de ejecución de acciones y de nodos de actividad se basan en tokens. En general, una acción puede comenzar su ejecución cuando los tokens están presentes en todos sus pines de entrada. Cuando comienza la ejecución, elimina los tokens desde los pines de entrada e internaliza sus valores de datos. Cuando una acción finaliza su ejecución, produce tokens en cada uno de sus pines de salida. Los tokens contienen valores de datos apropiados para las distintas salidas. Esos tokens pueden activar la ejecución de otras acciones.

Las actividades son construcciones de control que incluyen acciones. Cada tipo de nodo de actividad tiene sus propias reglas de activación. Algunas actividades requieren tokens en todos sus pines de entrada, mientras que algunas (como las fusiones) pueden activarse mediante tokens

en un subconjunto de sus pines de entrada. En cualquier caso, los tokens activadores se consumen antes de que la actividad comience su ejecución, y los nuevos tokens se producen en algunos o en todos los pines de salida cuando la actividad finaliza la ejecución.

La clave para los tokens es que se consumen cuando activan la ejecución de una acción o actividad. Si dos nodos de actividad compiten por los mismos tokens, sólo un nodo se ejecutará, aunque la selección puede ser no determinista. Puesto que los tokens combinan control y datos, evitan los problemas "acción-a-una-distancia" inherentes en la separación de control y datos. Modelan la concurrencia de una forma natural pero efectiva.

trace (estereotipo de Dependencia de abstracción)

Dependencia de abstracción que indica un proceso de desarrollo histórico u otras relaciones extras del modelo entre dos elementos que representan el mismo concepto sin especificar las reglas para derivar uno desde el otro. Este es el tipo de dependencia menos específica, y tiene semántica mínima. Es más para utilizar como recordatorio para el pensamiento humano durante el desarrollo. Está pensada para permitir la trazabilidad de los requisitos a lo largo del desarrollo.

Véase abstracción, dependencia, modelo.

Semántica

Una traza es una variedad de dependencia que indica una conexión entre dos elementos que representan el mismo concepto a diferentes niveles de significado. No representa semántica dentro de un modelo. A cambio, representa conexiones entre elementos con diferente semántica —es decir, entre elementos desde diferentes modelos en diferentes planos de significado. No hay un mapeo explícito entre los elementos. A menudo, representa una conexión entre dos formas de capturar un concepto en diferentes etapas de desarrollo. Por ejemplo, dos elementos que son variaciones del mismo tema podrían estar relacionados por una traza. Una traza no representa una relación entre instancias de tiempo de ejecución. En vez de ello, es una dependencia entre los propios elementos del modelo.

Una utilización considerable de traza es para hacer un seguimiento de los requisitos que se han cambiado a lo largo del desarrollo de un sistema. Las dependencias de traza pueden relacionar elementos en dos tipos de modelo (como en un modelo de casos de uso y un modelo de diseño) o en dos versiones del mismo tipo de modelo.

Notación

Una traza se indica mediante una flecha de dependencia (una flecha discontinua con su cola sobre el elemento más nuevo y su cabeza sobre el elemento más viejo) con la palabra clave «**trace**». Sin embargo, normalmente los elementos están de diferentes modelos que no se muestran simultáneamente, de forma que en la práctica la relación a menudo se implementaría en una herramienta como un hipervínculo.

transición

Relación dentro de una máquina de estados entre dos estados que indica que un objeto en el primer estado, cuando se produce un evento especificado y se satisface una condición de guarda

especificada, realizará unos efectos especificados (acción o actividad) y entra en el segundo estado. En dicho cambio de estado, la transición se dice que se dispara. Una transición simple tiene un único estado origen y un solo estado destino. Una transición compleja tiene más de un estado origen y/o más de un estado destino. Representa un cambio en el número de estados activos concurrentemente, o una división o unión de control. Una transición interna tiene un estado origen pero no estado destino. Representa una respuesta a un evento sin un cambio de estado. Los estados y transiciones son los vértices y nodos de máquinas de estados.

Véase también máquina de estados de protocolo, máquina de estados.

Semántica

Las transiciones representan las rutas potenciales entre los estados en la historia de vida de un objeto, así como las acciones realizadas en el cambio de estado. Una transición indica la forma en que un objeto en un estado responde a la ocurrencia de un evento. Los estados y las transiciones son los vértices y arcos de una máquina de estados que describe las posibles historias de vida de las instancias de un clasificador.

Estructura

Una transición tiene un estado origen, un desencadenador de evento, una condición de guarda, una acción y un estado destino. Algunos de ellos pueden estar ausentes en una transición particular.

Estado origen. El estado origen es el estado que se ve afectado por la transición. Si un objeto está en el estado origen, se puede disparar una transición saliente del estado si el objeto recibe el evento desencadenador de la transición y la condición de guarda (si existe) se satisface. El estado origen se vuelve inactivo después de que la transición se dispare.

Estado destino. El estado destino es el estado que se vuelve activo después de la finalización de la transición. Es el estado para el que cambia el objeto maestro. El estado destino no se usa en una transición interna, que no realiza un cambio de estado.

Desencadenador de evento. El desencadenador de evento es el evento cuya recepción por el objeto en el estado origen hace a la transición seleccionable para su disparo, siempre que su condición de guarda se satisfaga. Si el evento tiene parámetros, sus valores están disponibles para la transición y se pueden usar en expresiones para la condición de guarda y sus efectos. El evento que desencadena una transición se convierte en el evento actual y puede ser accedido por las acciones siguientes que son parte del paso de ejecución hasta finalización iniciado por el evento.

Una transición sin un evento desencadenador explícito se conoce como una transición de finalización (o una transición sin desencadenador) y se desencadena implícitamente en la finalización de cualquier actividad hacer interna en el estado. Un estado compuesto indica su finalización alcanzando el estado final para cada una de sus regiones. Si un estado no tiene actividad interna o estados anidados, entonces se desencadena una transición de finalización inmediatamente cuando se entra en el estado después de que cualesquiera actividades entrar se ejecuten. Observe que una transición de finalización debe satisfacer su condición de guarda para su disparo. Si la condición de guarda es falsa cuando se produce la finalización, entonces se consume el evento de finalización implícito y la transición no se disparará más tarde incluso si la condición de guarda se hace verdadera. (En cambio este tipo de comportamiento se puede modelar con un evento de cambio.)

Observe que todas las apariciones de un evento dentro de una máquina de estados deben tener la misma signatura.

Condición de guarda. La condición de guarda es una expresión lógica que se evalúa cuando se desencadena una transición por el manejador de un evento, incluyendo un evento implícito de finalización sobre una transición de finalización. Si la máquina de estados está realizando un paso de ejecutar hasta finalizar cuando se produce un evento, el evento se coloca en el depósito de eventos para el objeto propietario de la máquina de estados hasta que el paso se complete y la máquina de estados está inactiva; de otro modo el evento es manejado inmediatamente. Si la expresión de guarda se evalúa a verdadero, la transición es elegible para su disparo. Si la expresión se evalúa a falso, la transición no se dispara. Si un evento es manejado pero ninguna transición se torna elegible para su disparo el evento se descarta. Esto no es un error. Las transiciones múltiples con diferentes condiciones de guarda se pueden desencadenar por el mismo evento. Si el evento se produce, todas las condiciones de guarda se prueban. Si más de una condición de guarda es verdadera, sólo se disparará una transición. La elección de la transición que se disparará no es determinista si no se da una regla de prioridad.

Observe que la condición de guarda se evalúa sólo una vez, en el momento en que el evento es manejado. Si la condición se evalúa a falso y más tarde se vuelve verdadera, la transición no se disparará a no ser que se produzca otro evento y la condición sea verdadera en ese momento. Observe que una condición de guarda no es la forma apropiada de monitorizar un valor de forma continua. Para dicha situación debería usarse un evento de cambio.

Si una transición no tiene condición de guarda, entones la condición de guarda se trata como verdadera y la transición está activa si se produce su evento desencadenador. Si están activas varias transiciones, sólo una se disparará. La elección puede ser no determinista.

Por comodidad una condición de guarda se puede dividir en una serie de condiciones de guarda más sencillas. De hecho, varias condiciones de guarda pueden bifurcarse desde un solo evento desencadenador o condición de guarda. Cada ruta a través del árbol representa una sola transición desencadenada por el (único) evento desencadenador con una condición de guarda efectiva diferente que es la conjunción (“y”) de las condiciones de guarda a lo largo de su ruta. Todas las expresiones a lo largo de dicha ruta se evalúan antes de que se elija una transición para su disparo. Una transición no puede dispararse parcialmente. En efecto, un conjunto de transiciones independientes puede compartir parte de sus descripciones. La Figura 14.273 muestra un ejemplo.

Observe que los árboles de condiciones de guarda y la capacidad de ordenar transiciones para su elegibilidad son meramente comodidades, puesto que se puede conseguir el mismo efecto mediante un conjunto de transiciones independientes, cada una por su propia condición de guarda disjunta (pero *véase* vértice de elección debajo).

Efecto. Una transición puede contener un efecto, es decir, una acción o actividad que se ejecuta cuando se dispara la transición. Este comportamiento puede acceder y modificar el objeto que posee la máquina de estados (e indirectamente, otros objetos que pueda alcanzar). El efecto puede utilizar parámetros del evento desencadenador, así como los atributos y asociaciones del objeto poseedor. El evento desencadenador está disponible como el evento actual durante el paso entero de ejecutar hasta finalizar iniciado por el evento, incluyendo segmentos sin desencadenador posteriores y acciones entrar y salir.

Un efecto se hace con la intención de ser finito y debería normalmente ser bastante corto, puesto que la máquina de estados no puede procesar eventos adicionales hasta que la ejecución ha fina-

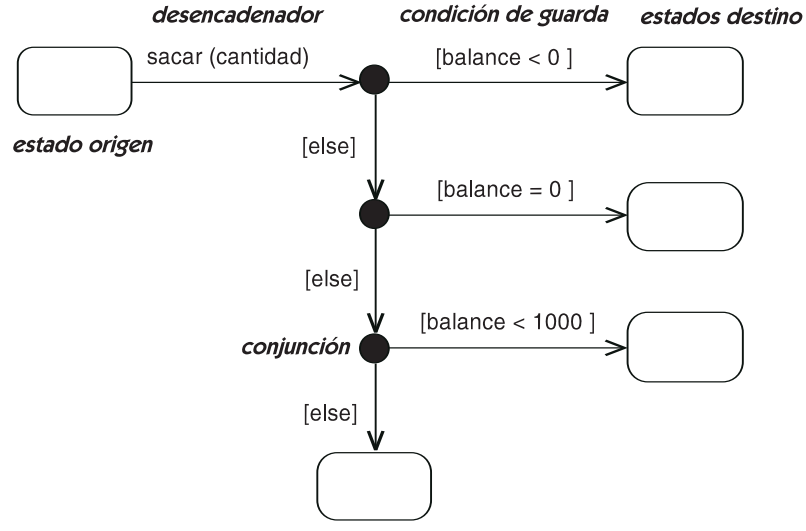


Figura 14.273 Árbol de condiciones de guarda

lizado. Cualquier comportamiento intencionado para continuar durante un tiempo extendido debería ser modelado como una actividad hacer adjunta a un estado más que a una transición.

Bifurcaciones. Por comodidad, varias transiciones que comparten el mismo evento desencadenador pero tienen diferentes condiciones de guarda se pueden agrupar juntas en el modelo y notación para evitar la duplicación del desencadenador o de la parte común de la condición de guarda. Esto es simplemente una comodidad de representación y no afecta a la semántica de las transiciones.

Véase bifurcación para detalles de representación y notación.

Vértice de elección. Un vértice de elección permite que una bifurcación dinámica dependa de acciones ejecutadas antes en la misma transición. Una transición puede ser activada para su disparo si se satisfacen todas las condiciones de guarda hasta el vértice de elección. Las condiciones de guarda en los segmentos de transición después del vértice de elección no se evalúan para determinar el disparo. Si la transición se dispara, sus efectos se ejecutan en los segmentos hasta el vértice de elección. Después se evalúan las condiciones de guarda de los segmentos que abandonan el vértice de elección, incluyendo los resultados de los efectos que se acaban de ejecutar. Eso permite que una acción afecte a los resultados de la bifurcación. Sin embargo, el modelador debe asegurar que al menos un segmento de transición saliente del vértice de elección se activará, o la maquina de estados fallará. Un vértice de elección es un pseudoestado y no puede permanecer activo. Una forma sencilla de asegurar el éxito es incluir una condición else entre los segmentos de salida. La Figura 14.274 muestra un ejemplo en el que la bifurcación depende del resultado de la operación de selección anterior en la transición.

Transición de finalización. Una transición que no tenga un evento se conoce como una transición de finalización. Se desencadena mediante la finalización de actividad en su estado origen. Si el estado origen es un estado sencillo, está finalizado cuando la actividad entrar y la actividad hacer del estado han finalizado. Si es un estado compuesto, finaliza cuando cada una de sus regiones ha alcanzado su estado final. Aunque la finalización se conoce como evento de fina-

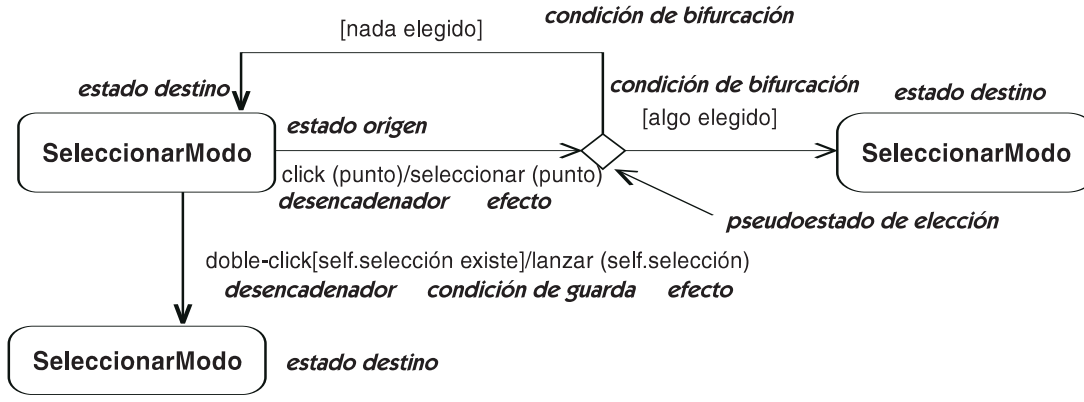


Figura 14.274 Transiciones

lización no va al depósito de eventos, pero se procesa inmediatamente como parte del paso de ejecutar hasta finalizar siempre que se pueda aplicar.

Véase también transición compleja, división, unión para transiciones con varios estados origen o destino.

Notación

Una transición se representa como una flecha continua desde un estado (el estado *origen*) hasta otro estado (estado *destino*), etiquetado con una cadena de transición. La Figura 14.284 muestra una transición entre dos estados y una división en segmentos.

Una transición tiene una etiqueta de texto de la forma:

⌊**nombre** :⌋$_{\text{opc}}$ **nombre-de-evento**$_{\text{opc}}$ ⌊(**lista-de-parámetros**)⌋$_{\text{opc}}$
⌊**[condición-de-guarda]**⌋$_{\text{opc}}$ ⌊**/ lista-de-efectos**⌋$_{\text{opc}}$

El *nombre* se puede utilizar como referencia a la transición en expresiones, particularmente para formar marcas de tiempo. Va seguido de dos puntos.

El *nombre-de-evento* da nombre a un evento y va seguido de sus parámetros. La lista de parámetros se puede omitir si no hay parámetros. El nombre de evento y la lista de parámetros se omiten para las transiciones de finalización. La **lista-de-parámetros** tiene el formato:

⌊**nombre** : **tipo**⌋$_{\text{lista,}}$

La *condición-de-guarda* es una expresión lógica escrita en términos de parámetros del evento desencadenador y atributos y enlaces del objeto descrito por la máquina de estados. La condición de guarda también puede implicar pruebas del estado de los estados concurrentes de la máquina actual o estados explícitamente designados de algún objeto alcanzable —[**in Estado1**] y [**not in Estado2**] son ejemplos. Los nombres de estados puede estar totalmente calificados por los estados anidados que los contienen, produciendo nombres de ruta de la forma **Estado1::Estado2::Estado3**. Esto se puede utilizar si el mismo nombre de estado aparece en diferentes regiones de estados compuestos de la máquina global.

La *lista-de-efectos* es una expresión procedimental que se ejecuta si se dispara la transición y cuando lo haga. Se puede escribir en términos de operaciones, atributos y enlaces de los objetos propietarios y de los parámetros del evento desencadenador. Las acciones pueden incluir llamadas, envíos y otros tipos de acciones. La **`lista-de-efectos`** puede contener más de una acción. La sintaxis de la expresión es dependiente de la implementación.

Transición interna. Una transición interna se representa como una cadena de transición dentro de un símbolo de estado.

Bifurcaciones. Una transición puede incluir un segmento con un evento desencadenador seguido por un árbol de conjunciones, dibujadas como pequeños círculos. Esto es equivalente a un conjunto de transiciones individuales, una por cada ruta a través del árbol, cuya condición de guarda es la "y" de todas las condiciones a lo largo de la ruta. Sólo el segmento final de cualquier ruta puede tener una acción.

Elecciones. Una transición puede incluir un segmento con un evento desencadenador seguido por un árbol de elecciones, dibujadas como pequeños diamantes. Cualquier segmento puede tener una acción. Las condiciones de guarda sobre los segmentos que siguen a la elección se evalúan después de que se hayan ejecutado cualesquiera efectos precedentes.

Transiciones complejas. *Véase* transición compleja para ver la notación para las transiciones con varios estados origen o destino.

Transición de protocolo. *Véase* máquina de estados de protocolo para la notación.

Discusión

Una transición representa un cambio atómico desde un estado a otro, posiblemente acompañada por una acción atómica. Una transición no es interrumpible. Las acciones de una transición deben ser cortas, normalmente triviales, cálculos como sentencias de asignación y cálculos aritméticos sencillos.

transición compleja

Transición con más de un estado origen y/o más de un estado destino. Representa una respuesta a un evento que origina un cambio en la cantidad de concurrencia. Es una sincronización de control, una división de control, o ambas, dependiendo del número de orígenes y destinos. (Este término se presenta aquí por conveniencia, puesto que en la especificación de UML2 no se define ningún término equivalente.)

Véase también bifurcación, estado compuesto, división, unión, fusión, región ortogonal.

Semántica

A alto nivel, un sistema pasa por una serie de estados, pero la vista monolítica de que un sistema tiene un solo estado es demasiado restrictiva para grandes sistemas con distribución y concurrencia. Un sistema puede tener múltiples estados ortogonales a la vez. El conjunto de estados activos se conoce como configuración del estado activo. Si un estado anidado está activo, entonces todos los estados que lo contienen estarán activos. Si el objeto permite concurrencia, entonces puede estar activa más de una región ortogonal.

En muchos casos, se puede modelar la actividad de un sistema como un conjunto de hilos de control que evolucionan independientemente del resto o que interactúan de ciertas formas limitadas. Cada transición afecta, como mucho, a unos pocos estados de la configuración del estado activo. Cuando se dispara una transición, los estados activos que no se ven afectados permanecen activos. El progreso de los hilos en un momento dado se puede capturar como un subconjunto de estados dentro de la configuración del estado activo, un subconjunto por cada hilo. Cada conjunto de estados evoluciona de forma independiente en respuesta a los eventos. Si el número de estados activos es constante, el modelo de estados no es más que 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. Un estado puede transformarse en dos o más estados concurrentes (una división de control), y dos o más estados concurrentes pueden transformarse en un estado (una unión de control). La máquina de estados controla el número de estados concurrentes y su evolución.

Una transición compleja es una transición a o desde un estado ortogonal. Los vértices en uno de los extremos de la transición deben incluir un estado de cada región ortogonal en el estado ortogonal. Una transición compleja tiene más de un estado origen y/o estado destino. Si tiene varios estados origen, representa una unión de control. Si tiene varios estados destino, representa una división de control. Si tiene varios estados origen y destino, representa una sincronización de hilos paralelos.

Si una transición compleja tiene varios estados origen, todos ellos deben estar activos antes de que la transición sea candidata a ser disparada. El orden en que se vayan activando es irrelevante. Si todos los estados origen están activos y se produce el evento de disparo, se permite la transición y se puede disparar siempre que su condición de guarda sea verdadera. Cada transición compleja tiene un solo disparador, incluso si hay varios estados origen. UML no admite el concepto de ocurrencia simultánea de eventos; cada evento debe disparar una transición separada y después los estados resultantes pueden ir seguidos por una unión.

Si una transición compleja con varios estados origen no tiene un evento disparador (es decir, si es una transición de finalización) se disparará cuando todos sus estados origen explícitos se vuelvan activos. Si en ese momento se satisface su condición de guarda, se dispara.

Cuando se dispara una transición compleja, todos los estados origen y todos sus pares dentro del mismo estado compuesto dejan de estar activos, y todos los estados destino y sus pares se vuelven activos.

En situaciones más complicadas, se puede expandir la condición de guarda para permitir el disparo cuando algún subconjunto de los estados está activo; esta habilidad puede llevar fácilmente a modelos mal formados, y debería evitarse o utilizarse con sumo cuidado.

El metamodelo de UML no incluye la transición compleja como una metaclase. La transición compleja se modela como un conjunto de transiciones con un solo origen y un solo destino, conectados por pseudoestados de división o unión, pero el efecto es el mismo de una transición compleja con varios orígenes o destinos.

Ejemplo

La Figura 14.275 muestra un estado compuesto concurrente típico, con transiciones complejas entrando y saliendo de él. La Figura 14.276 muestra una historia de ejecución típica de esta

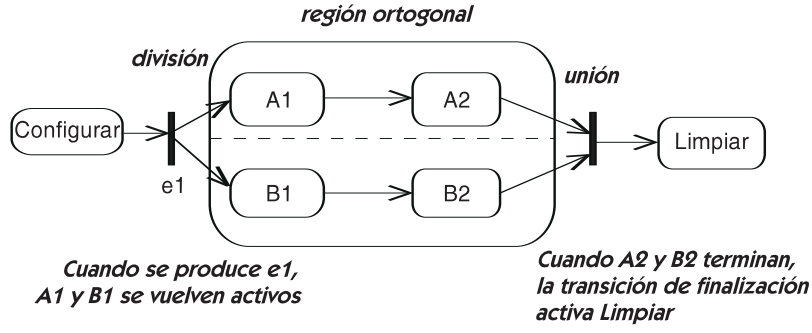


Figura 14.275 División y unión

máquina (los estados activos se muestran en azul). La historia muestra la variación en el tiempo del número de estados activos.

Estados ortogonales

A no ser que se estructure con cuidado la máquina de estados, un conjunto de transiciones complejas puede llevar a inconsistencias, incluyendo interbloqueos, ocupación múltiple de un estado y otros problemas. Se ha estudiado minuciosamente el problema en la teoría de redes de Petri, y la solución habitual es imponer reglas de formación correcta a la máquina de estados, con el fin de evitar el peligro de inconsistencias. Son las reglas de "programación estructurada" para máquinas de estados. Hay distintos enfoques, cada uno con sus ventajas e inconvenientes. Las reglas adoptadas por UML requieren que una máquina de estados se descomponga en estados más pequeños utilizando un árbol *y/o*. La ventaja es que una estructura bien anidada es fácil de establecer, mantener y comprender. La desventaja es que ciertas configuraciones significativas están prohibidas. En la balanza, es similar a las soluciones de compromiso adoptadas para renunciar a las cláusulas goto en la programación estructurada.

Se puede descomponer un estado compuesto en un conjunto de subestados mutuamente excluyentes (si tiene exactamente una región, una descomposición "o") o en un conjunto de regiones ortogonales, cada una de las cuales contiene subestados mutuamente excluyentes (si tiene más de una región, una descomposición "y" seguida de una descomposición "o"). La estructura es recursiva. En general, se alternan capas "y" con capas "o". Una capa "y" representa descomposición ortogonal —todos los subestados están activos de forma concurrente. Un estado "o" representa una descomposición secuencial— un subestado activo en cada momento. Se puede obtener un conjunto lícito de estados ortogonales expandiendo recursivamente los nodos del árbol, empezando por la raíz, reemplazando un estado "y" por todos sus hijos y un estado "o" por uno de sus hijos. Esto corresponde a la estructura anidada de los diagramas de estados.

Ejemplo

La Figura 14.277 muestra un árbol y/o de estados que corresponde a la máquina de estados de la Figura 14.275. Se ha coloreado en negrita un conjunto típico de estados activos concurrentemente, concretamente los correspondientes al tercer paso de la Figura 14.276.

Si una transición entra en un estado ortogonal (uno con más de una región), entonces entra en todas las subregiones ortogonales. Si la transición entra en un estado no ortogonal (uno con una

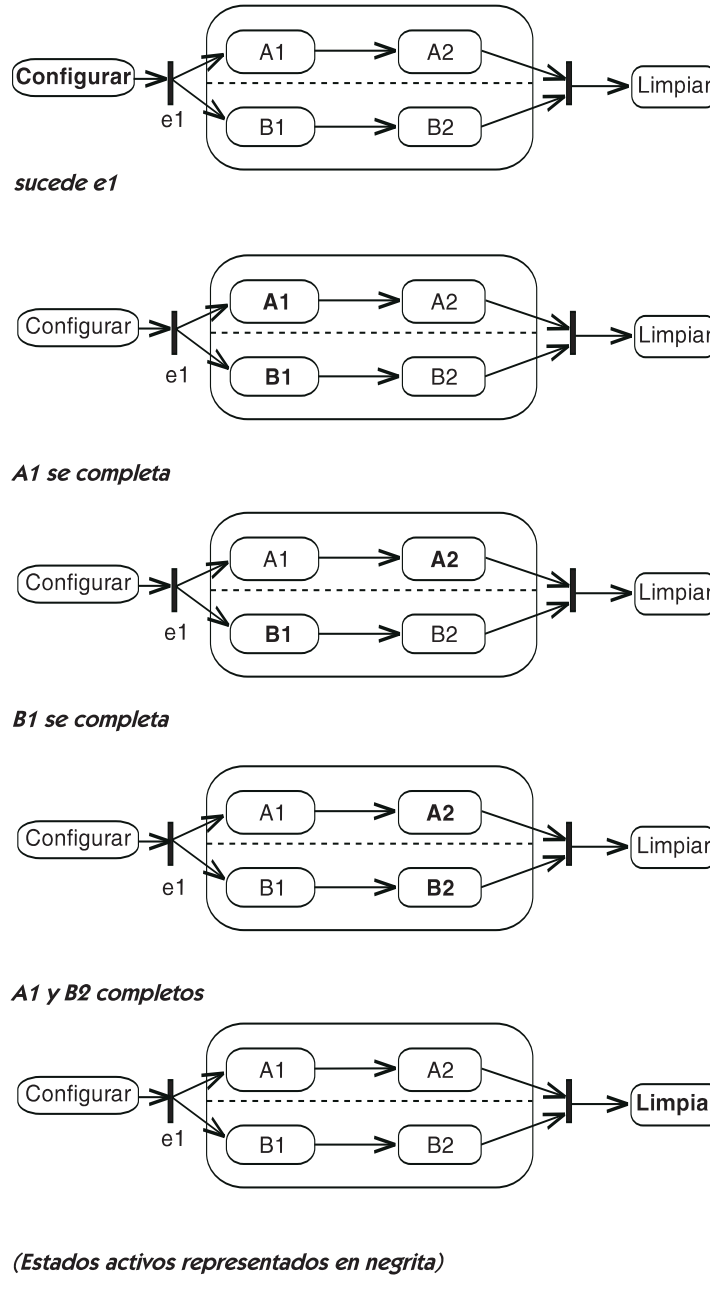
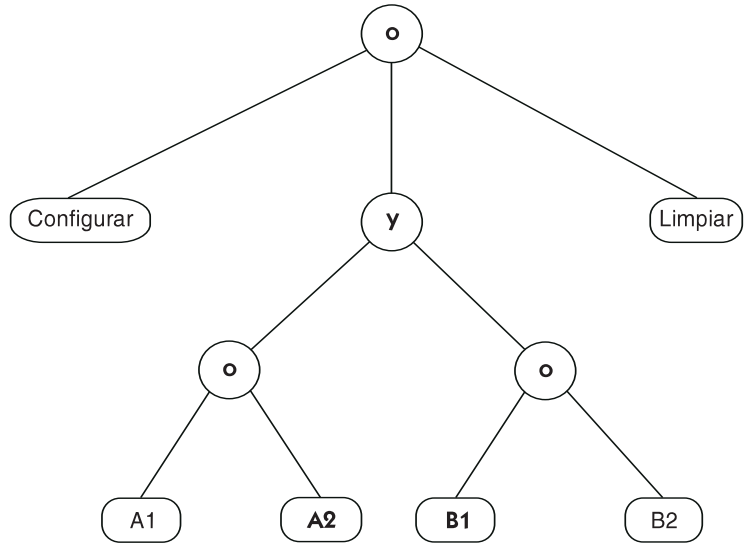


Figura 14.276 Historia de estados activos en una máquina de estados con regiones ortogonales

sola región), entonces entra exactamente en un subestado. El estado activo dentro de un estado no ortogonal o de una región puede cambiar. Con un estado ortogonal, un estado permanece activo en cada región ortogonal mientras el estado ortogonal permanezca activo. Un estado en una región ortogonal puede ser descompuesto más aún.

Por tanto, una transición simple (una que tiene una entrada y una salida) debe conectar dos estados en la misma región no ortogonal o dos estados separados sólo por niveles “o”. Una tran-



(Conjunto típico de estados activos mostrado en negrita)

Figura 14.277 Árbol y/o de estados anidados

sición compleja debe conectar todas las subregiones dentro de un estado ortogonal con un estado de fuera de la región ortogonal (omitimos casos más complicados, pero deben seguir los principios mencionados anteriormente). En otras palabras, una transición que entra en un estado ortogonal debe entrar en un estado de cada subregión; una transición que abandona un estado ortogonal debe abandonar cada subregión.

Hay disponible una representación más corta: Si una transición compleja entra en un estado ortogonal pero omite transiciones explícitas a una o más de las subregiones, entonces hay implícitamente una transición al estado inicial de cada subregión no enumerada. Si una subregión no tiene estado inicial, el modelo está mal formado. Si una transición compleja abandona un estado ortogonal, hay una transición implícita desde cada subregión no enumerada. Si se dispara una transición compleja, se termina cualquier actividad dentro de una subregión —es decir, representa una salida forzada. Se puede conectar una transición al propio estado ortogonal que la contiene. Ello implica una transición al estado inicial de cada subregión —una situación común de modelado. Asimismo, una transición desde un estado ortogonal implica la salida forzada de cada subregión (si tiene un evento disparador) o esperar a que cada subregión se complete (si no tiene disparador, es decir, si es una transición de finalización).

Las reglas en las transiciones complejas aseguran que no puedan estar activas de forma concurrente combinaciones de estados sin sentido. Un conjunto de subregiones ortogonales es una partición del estado compuesto que las engloba. Todas están activas o ninguna de ellas lo está.

Notación

Una transición compleja se representa como una barra corta y gruesa (una barra de sincronización, que puede representar sincronización, división o ambas). La barra puede tener una o más flechas

continuas de transición desde los estados a la barra (los estados son los estados origen); la barra puede tener una o más flechas continuas desde la barra a los estados (los estados son los estados destino). Se puede mostrar una etiqueta de transición cerca de la barra, que describe el evento disparador, condición de guarda, y las acciones, como se describe bajo la transición. Las flechas individuales no tienen sus propias cadenas de transición; simplemente forman parte de la transición global.

Ejemplo

La Figura 14.278 muestra la máquina de estados de la Figura 14.275 con una transición de salida adicional. También muestra una división implícita desde el estado **Configurar** a los estados iniciales en cada subregión y una unión implícita desde los estados finales en cada subregión al estado **Limpiar**.

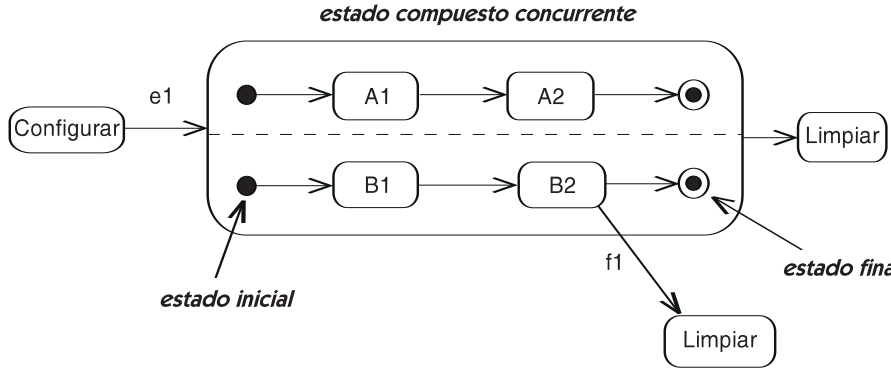


Figura 14.278 Transiciones complejas (división, unión)

Si se produce el evento **f1** cuando está activo el estado **B2**, entonces se produce la transición al estado **Limpiar**. Esta transición es una unión implícita; termina tanto el estado **A2** como el estado **B2**.

transición compuesta

Transición representada por varios segmentos conectados por pseudoestados.

Semántica

Una transición compuesta es una transición implícita compuesta de varios segmentos conectados por pseudoestados de unión, conjunción, división o elección. La transición se debe ejecutar en su totalidad. No es posible ejecutar algunos segmentos y dejar uno de los pseudoestados activo.

Una transición compuesta puede implicar bifurcaciones por medio de conjunciones o elecciones. Si ninguno de los pseudoestados son vértices de elección, entonces todas las condiciones de guarda en todos los posibles caminos a través de los segmentos se evalúan antes de que se dispare la transición, y la transición sólo se dispara si se evalúan a verdadero todas las condiciones de al menos un camino. Cualesquiera acciones realizadas durante la ejecución de la transición no afectan a las condiciones de guarda ni altera el camino elegido. Si no existe ningún camino válido en el momento de la evaluación, la transición no se dispara.

Si alguno de los caminos implica vértices de elección, la evaluación de las condiciones de guarda sigue sólo hasta cualesquiera vértices de elección (o hasta estados destino en caminos que no tienen vértices de elección). Si al menos uno de dichos caminos se evalúa a verdadero, se elige un camino y se dispara la transición. Se realizan cualesquiera acciones de dichos caminos. Cuando se encuentra un vértice de elección, se vuelven a evaluar las condiciones en los caminos salientes usando los resultados de la ejecución hasta dicho punto. Se elige un camino saliente para el que se evalúen todas las condiciones a verdadero (hasta cualquier otro vértice de elección posterior) y la ejecución continúa en el camino elegido, y seguimos así hasta que encontremos un estado destino y la transición compuesta se complete.

Si se encuentra un vértice de elección y ningún camino saliente se evalúa a verdadero, el modelo está mal formado. Un vértice de elección no puede permanecer activo. Un vértice de elección es una garantía para el modelador de que al menos un camino saliente satisfará su condición de guarda. Si no se pueden examinar fácilmente las condiciones de guarda de forma que se asegure que cubren todas las posibilidades, la utilización de una condición **else** en un camino saliente garantizará que siempre exista una elección válida.

Historia

El vértice de elección se ha añadido en UML2.

transición condicional

Utilización de guardas para elegir disparar una entre varias transiciones de máquina de estados.

Semántica

Una transición de máquina de estados incluye una condición de guarda. Cuando se produce un evento de disparo para una transición, se evalúa la guarda. Si se evalúa a verdadero, se puede disparar la transición.

Las transiciones de máquina de estados se pueden utilizar para representar condicionales de las siguientes maneras:

- Un conjunto de transiciones que abandonan un estado origen tienen eventos de finalización como sus disparadores y condiciones de guarda, que juntos cubren todas las posibilidades. Cuando el estado origen se activa y se completan sus actividades entrar o hacer, se elegirá una de las transiciones. Esta situación es muy parecida a una construcción condicional tradicional en un lenguaje de programación.
- Un conjunto de transiciones que abandonan un estado origen y todas tienen el mismo evento disparador y condiciones de guarda que juntas cubren todas las posibilidades. Cuando se produce el evento disparador, se elegirá una de las transiciones. Esta situación también se puede modelar utilizando estados de conjunción para evitar la repetición del disparador. Un segmento desde el estado origen hasta el estado de conjunción tiene el evento de disparo, y un conjunto de transiciones que abandonan el estado de conjunción tienen condiciones de guarda pero no disparadores. En ambos casos, las condiciones de guarda se evalúan después de que se produzca el evento pero antes de que se elija una transición.

- Un segmento de transición va desde un estado origen a un pseudoestado de elección, y un conjunto de segmentos de transición abandona el pseudoestado de elección. Cada uno de ellos tiene una condición de guarda pero ningún disparador. En este caso, una acción en el segmento de transición inicial puede afectar a los valores que aparecen en las guardas. Las guardas se evalúan después de que se dispare el segmento inicial. Una de ellas debe evaluarse a verdadera o de lo contrario el modelo estará mal formado. El pseudoestado de elección es equivalente a un estado anónimo seguido de un conjunto de transiciones de finalización.

Notación

Véase notación para elección, condición de guarda, transición.

transición de actividad

Relación de secuencia entre dos nodos de actividad, que posiblemente incluye datos.

Semántica

Una transición de actividad es una relación de secuencia entre un nodo de actividad origen y un nodo de actividad destino. El nodo destino no se puede ejecutar hasta que el nodo origen no ha completado su ejecución y ha emitido un token en la transición de actividad. Si un nodo tiene múltiples edges en los que él es el destino, no se puede jecutar hasta que todos ellos tienen tokens, a menos que las reglas para el tipo particular de nodo de actividad especifiquen que un subconjunto de edges de entrada pueden habilitar su ejecución. Un edge puede representar un único flujo de control o puede representar el flujo de datos, incluyendo el flujo implícito de control que indica que se ha producido el valor del dato.

Estructura

Guarda. Una guarda es una expresión Booleana vinculada a un edge que determina si el edge puede habilitar un nodo de actividad destino. Si el valor de la expresión es verdadero, el nodo se puede habilitar; en otro caso, no se puede habilitar.

Peso. Un peso indica el número de tokens consumidos cuando un nodo destino se ejecuta. El valor por defecto es uno cuando no se especifica peso. *Véase* peso.

Notación

Véase flujo de control y flujo de datos para la notación.

transición de actividad interrumpible

Especificación de un evento cuya ocurrencia termina toda la actividad dentro de la región designada y transfiere el control a una actividad manejadora de interrupción designada.

Véase interrupción.

transición de alto nivel

Véase transición de grupo.

transición de finalización

Transición que carece de un evento disparador explícito, y que es disparada por la finalización de la actividad en el estado origen.

Véase también actividad hacer, transición, disparador.

Semántica

Una transición de finalización se representa como una transición que no tiene un evento disparador explícito. La transición se dispara de forma implícita cuando su estado origen (incluyendo estados anidados) ha completado todas las actividades. La finalización puede ser una de las siguientes:

- Finalización de una acción de entrada y una actividad hacer contenida en ella, si existe.
- Alcanzar un estado final en un estado compuesto no ortogonal.
- Alcanzar los estados finales de todas las regiones de un estado ortogonal.
- Alcanzar un punto de salida o estado final no etiquetados dentro de una submáquina referenciada.

Se ejecuta una actividad de salida en un estado cuando se dispara la transición de finalización, antes de cualesquiera acciones de la propia transición.

Una transición de finalización puede tener una condición de guarda y un efecto. Normalmente es indeseable tener una transición de finalización con guarda aislada, puesto que si la condición de guarda es falsa, la transición nunca se disparará (ya que el disparador implícito sucede sólo una vez). Ocasionalmente, puede ser de utilidad para representar algún tipo de fallo, siempre que una transición disparada saque finalmente al objeto del estado muerto. De manera más común, un conjunto de transiciones de finalización con guarda tienen condiciones que cubren todas las posibilidades, de forma que una de ellas será disparada inmediatamente cuando el estado termina.

Las transiciones de finalización también se utilizan para conectar estados iniciales y estados de historia a sus estados sucesores, puesto que dichos pseudoestados no pueden permanecer activos después de la finalización de la actividad.

Un evento de finalización tiene prioridad sobre los eventos normales. Si se produce un evento de finalización, se dispara inmediatamente una transición de finalización activa, sin que se coloque en el conjunto de eventos normales. Por tanto, se puede considerar un evento de finalización más como un tipo especial de construcción de control que como un evento normal.

Notación

Una transición de finalización se representa por medio de una flecha de transición sin evento disparador. La flecha puede tener una condición de guarda entre corchetes, y puede tener una expresión de efecto después de una barra inclinada (/). A menudo no tiene ninguna etiqueta.

Ejemplo

La Figura 14.279 muestra un fragmento de una máquina de estados para una aplicación de solicitud de entradas. El estado **Seleccionando** permanece activo en tanto en cuanto el cliente permanezca eligiendo fechas. Cuando el cliente presiona el botón "terminar", el estado **Seleccionando** alcanza su estado final. Esto dispara la transición de finalización, que lleva al estado **Seleccionado**.

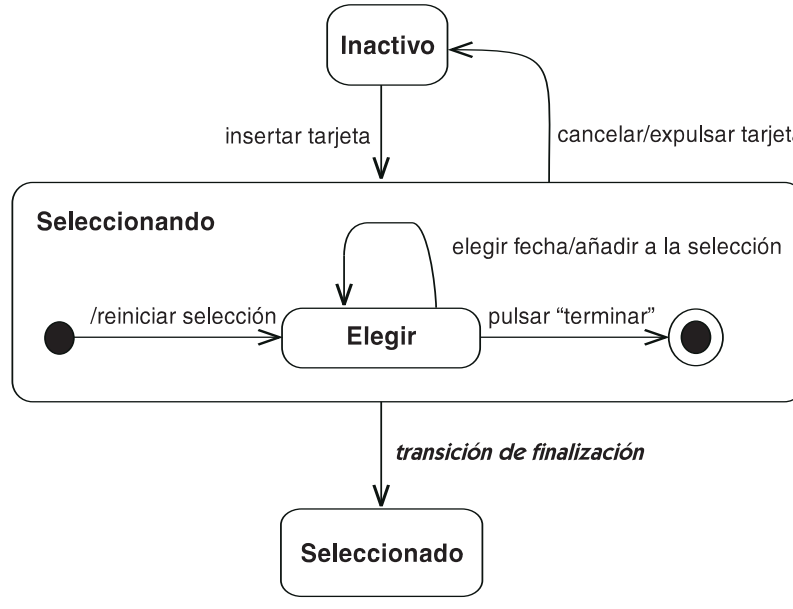


Figura 14.279 Transición de finalización

transición de grupo

Transición cuyo origen es un estado compuesto.

Semántica

Se conoce como una *transición de grupo* (o una *transición de alto nivel*) a una transición cuyo origen es un estado compuesto. Se aplica siempre que la configuración del estado activo incluya uno o más subestados contenidos (directa o indirectamente) por el estado compuesto. Si el desencadenador de la transición de grupo se satisface y no se satisface ninguna transición en un estado anidado dentro del estado compuesto, el disparo de la transición de grupo fuerza la terminación de la actividad dentro del estado compuesto. Todos los hilos de actividad terminados ejecutan sus actividades salir hasta que todos los subestados directos del estado compuesto alcanzan la finalización. Si el estado destino es externo al estado compuesto ejecuta la actividad salir del estado compuesto. Entonces se ejecutan las demás actividades encontradas en el camino del estado o estados destino, y la configuración del estado activo se actualiza reemplazando los estados terminados con el nuevo o nuevos estados destino.

transición de protocolo

Una transición en una máquina de estado protocolar.

Semántica

Las transiciones protocolares especifican secuencias legales de eventos. No especifican los efectos.

Notación

Véase máquina de estados de protocolo.

transición interna

Transición adjunta a un estado que tiene una acción pero no implica un cambio de estado.

Véase también máquina de estados.

Semántica

Una transición interna permite a un evento desencadenar una actividad sin que se produzca un cambio de estado. Una transición interna tiene un estado origen pero no tiene estado destino. Si el desencadenador se satisface mientras el estado está activo, la transición se dispara y se ejecuta su actividad, pero el estado no cambia incluso aunque la transición interna esté adjunta y heredada de un estado que englobe al estado actual. Por tanto, no se ejecuta ninguna actividad entrar o salir. En este sentido, difiere de una autotransición, que origina la salida de los estados anidados y salir y volver a entrar en el estado origen, con la ejecución tanto de las actividades salir como de las actividades entrar.

Notación

Una transición interna se muestra como una entrada de texto dentro de un compartimento de un símbolo de estado. La entrada tiene la misma sintaxis que la etiqueta de texto de una transición externa. Puesto que no hay estado destino, no hay necesidad de adjuntarla a una flecha.

`nombre-de-evento` / **`expresión-de-actividad`**

Los nombres de evento **entry**, **exit** y **do** son palabras reservadas y no se pueden usar. Esas palabras reservadas se utilizan respectivamente para declarar una actividad entrar, una actividad salir y la ejecución de una actividad hacer. Para mantener la uniformidad, esas actividades especiales utilizan sintaxis de transición interna para especificar la acción. Sin embargo, no son transiciones internas, y por tanto las palabras reservadas no son nombres de evento.

La Figura 14.11 muestra la notación. Si se genera la señal ayuda mientras el estado está activo, se realiza la actividad **mostrar ayuda**. El estado **Escribiendo Contraseña** permanece activo.

Discusión

Se debe pensar en una transición interna como si fuera una “interrupción” que origina una acción pero que no afecta al estado actual, y por tanto no invoca acciones salir o entrar. Adjuntar una tran-

sición interna a un estado compuesto es una buena forma de modelar una acción que debe ocurrir sobre un cierto número de estados pero que no debe cambiar el estado activo —por ejemplo, mostrar un mensaje de ayuda o contar el número de ocurrencias de un evento. No es la forma adecuada de modelar un aborto o una excepción, que deberían modelarse mediante transiciones a un nuevo estado, ya que su ocurrencia invalida al estado actual.

transición simple

Una transición con un estado origen y un estado destino. Representa una respuesta a un evento con un cambio de estado dentro de una región de estados mutuamente excluyentes. La cantidad de concurrencia no produce cambios como resultado de ejecutarlo.

Contraste: transición compleja, transición compuesta.

transición sin desencadenador

Transición sin un desencadenador de eventos explícito. Una transición sin desencadenador que se marcha de un estado normal representa una transición de finalización, es decir, una transición que es desencadenada por la finalización de actividad en el estado más que por un evento explícito. Cuando abandona un pseudoestado, representa un segmento de transición que es atravesado automáticamente cuando el segmento precedente ha finalizado su acción. Las transiciones sin desencadenador se utilizan para conectar estados iniciales y estados de historia con sus estados destino.

transmisión

Transferencia de control e información entre objetos mediante mensajes.

Semántica

Un mensaje se crea como resultado de una acción de llamada o de una acción de envío. Un mensaje tiene un tipo (una operación o una señal) y una lista de valores de argumentos. Cada mensaje se dirige a un objeto particular. La codificación de un mensaje y la forma de su envío al objeto destino no son relevantes para la semántica de UML y no es visible para los objetos o sus comportamientos. El modelo de ejecución de UML cubre un amplio rango de posibilidades de implementación, incluyendo ejecución síncrona dentro de un solo procesador, transmisión síncrona y transmisión asíncrona. En el modelo de ejecución básico de UML, el tiempo de transmisión y de secuencia relativo a otros mensajes es desconocido. Si se desea, los perfiles dirigidos a entornos particulares de ejecución pueden añadir este tipo de información.

Los mensajes asíncronos (desde acciones de envío de señal o llamadas asíncronas) no requieren ninguna información acerca del emisor. Su transmisión y posterior recepción es concurrente con la ejecución actual del objeto que creó el mensaje. Los mensajes síncronos (de llamadas síncronas) causan el bloqueo de ejecución del llamante. Incluyen suficiente información para identificar la ejecución de llamada y para permitir despertarlo en el futuro y proporcionarle con valores de retorno. La codificación de esta información de retorno es opaca de forma explícita en el modelo UML; ningún objeto o acción puede acceder a ella, excepto en el acto de devolver

el control y los valores al llamante. Esto permite un amplio rango de implementaciones sin condicionar la forma de la tecnología de implementación. De nuevo, los perfiles intencionados para diversos entornos de implementación pueden elegir especificar la forma de esta información y dejarla disponible para las acciones de usuario.

traza

Secuencia de ocurrencias de eventos dentro de la ejecución de una interacción.

Véase también trace para una utilización no relacionada de la palabra *traza*.

Semántica

Una traza es una secuencia de ocurrencias de eventos como resultado de una ejecución particular de una interacción. La semántica de las interacciones se define en términos de trazas. Una interacción define un conjunto de trazas permitidas y un conjunto de trazas prohibidas. Todas las trazas en el conjunto permitido son definitivamente consistentes con la definición de la interacción. Cualesquiera otras trazas pueden o no ser consistentes con la definición. Puede ser necesario mirar a otras partes del modelo para determinar su validez.

Varias construcciones de interacción (como parallel) definen varias subtrazas independientes de sus partes. A no ser que esté restringido por otro lado, las ocurrencias de eventos para cada subtraza pueden intercalarse arbitrariamente para hacer una sola traza global. Este es el significado de concurrencia dentro de las interacciones.

Dos interacciones son equivalentes si definen los mismos conjuntos de trazas.

Notación

Las trazas son principalmente un concepto semántico para definir el significado de los modelos de interacción. Representan los resultados de una ejecución real. Un diagrama de secuencia, por contraste, es un modelo de una definición de interacción. Las secuencias de las especificaciones de eventos en el diagrama de secuencia no son las trazas reales; más bien son modelos desde los que las trazas se pueden derivar. La mejor forma de mostrar realmente una traza es mostrar una lista de ocurrencias de eventos de forma textual, pero a menudo no es útil excepto como un ejercicio teórico.

tupla

Lista ordenada de valores. Generalmente, el término implica que hay un conjunto de dichas listas de forma similar. (Este es un término estándar matemático.)

type (estereotipo de Clase)

Especificación de la estructura y comportamiento generales de un dominio de objetos sin proporcionar una implementación física.

Véase clase, implementation class (estereotipo de Clase), tipo.

Discusión

Han corrido ríos de tinta sobre la distinción entre tipo y clase, principalmente para justificar sentimientos intuitivos sobre la utilización de las palabras o las distinciones de los lenguajes de programación sin recurrir a bases matemáticas o necesidades prácticas.

unicidad

Indicador de si los valores de una colección con multiplicidad mayor que uno pueden contener valores duplicados. Una colección cuyos elementos no son únicos es una bolsa (bag) o una lista (list).

Semántica

La unicidad es una parte de la especificación de multiplicidad. Es un indicador lógico que expone si los valores de una colección deben ser únicos. Si son únicos, la colección es un conjunto (set). Si no son únicos, la colección es una bolsa (bag). Por defecto los valores deben ser únicos, y por tanto la colección es un conjunto (set).

Notación

Un indicador de multiplicidad puede contener las siguientes palabras clave:

set	Los valores deben ser únicos. Es el valor por defecto y la palabra clave se puede omitir (y normalmente se hace).
bag	Los valores no tienen por qué ser únicos. Se permiten valores duplicados.

La unicidad y la ordenación se pueden combinar de las siguientes formas:

	único	*no único*
no ordenado	**set** (valor por defecto)	**bag**
ordenado	**ordered** (set)	**sequence** o **seq** o **list**

unidad de distribución

Conjunto de objetos o componentes que están asignados a un proceso del sistema operativo o a un procesador como grupo. Una unidad de distribución se puede representar mediante un compuesto en tiempo de ejecución o mediante un agregado. Es un concepto de diseño en la vista de despliegue.

unión (join)

Lugar en una máquina de estados, diagrama de actividad, o diagrama de secuencia en que dos o más hilos o estados concurrentes se combinan para producir un hilo o estado; una unión "y" o deshacer una división. Antónimo: división. *Véase* transición compleja, estado compuesto.

Semántica

Una unión es un pseudoestado en una transición compleja con dos o más estados origen y un estado destino. Si todos los estados origen están activos y se produce el evento desencadenador, se ejecuta el efecto de la transición y el estado destino se vuelve activo. Los estados origen deben estar en regiones ortogonales diferentes de un estado compuesto.

Las transiciones de entrada individuales de la unión no pueden tener condiciones de guarda. La unión global puede tener una condición de guarda.

Un segmento de transición que abandona una unión puede tener un desencadenador siempre que ningún segmento previo o cualquier camino de entrada tengan un desencadenador. El desencadenador debe ser satisfecho antes de que se dispare la transición.

Notación

Una unión se representa como una barra gruesa con dos o más flechas de transición de entrada y una flecha de transición de salida (Figura 14.280). Puede tener una etiqueta de transición (condición de guarda, evento disparador y acción).

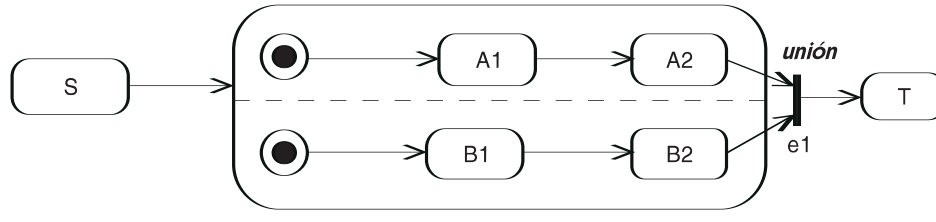


Figura 14.280 Unión

Discusión

Véase fusión para ver la discusión.

unión

Declaración de que una propiedad se define como la unión de otras propiedades especificadas explícitamente.

Véase también subsets.

Semántica

A veces la misma propiedad (un extremo de asociación o un atributo) se puede modelar con más de un nivel de granularidad, cuando un nivel general cubre varios casos y otro nivel más específico expande cada uno de esos casos en varios casos más específicos. Si la descripción de alto nivel es simplemente la unión de un conjunto de casos específicos, se conoce como *unión derivada*. La descripción de alto nivel proporciona una forma de hacer sentencias acerca todos los

casos específicos, pero cada instancia de la unión debe ser una instancia de uno de los casos específicos.

La propiedad de alto nivel se etiqueta con la propiedad de unión. Cada uno de sus casos específicos debe etiquetarse como subconjunto de la propiedad de alto nivel. La propiedad de unión se define por tanto como la unión de todas las propiedades que forman explícitamente un subconjunto de ella.

Notación

Una propiedad que sea unión derivada se etiqueta con la cadena {**union**}. Si la propiedad es un extremo de asociación, la etiqueta se coloca cerca del final de de la línea de asociación. Si la propiedad es un atributo, la etiqueta se coloca después de la cadena del atributo.

Véase la Figura 14.270 para ver un ejemplo.

unión derivada

Propiedad que se especifica como la unión de todas las demás propiedades declaradas como un subconjunto de ella.

Semántica

A menudo resulta de utilidad definir una propiedad abstracta, como un nombre del extremo de una asociación, como la unión de un conjunto de propiedades específicas. Por ejemplo, se podría definir a un familiar como un hermano, hijo o esposa. Dicha declaración se conoce como unión derivada. Quiere decir que la propiedad derivada es igual a la unión de todos sus subconjuntos explícitamente declarados y ninguno más.

Una unión derivada se especifica estableciendo el indicador de unión derivada en la propiedad abstracta. Cada propiedad que participa se especifica con la relación subconjunto a la propiedad que es una unión derivada.

Notación

Una propiedad de unión derivada se representa utilizando la cadena de propiedad {**union**} después de su nombre. Si la propiedad es un nombre de extremo de asociación, la cadena se puede colocar cerca del final del camino que representa la asociación. Cada propiedad subconjunto se representa utilizando una cadena de propiedad de la forma {**subsets nombre-de-unión**}, donde **nombre-de-unión** es el nombre de la propiedad unión.

Ejemplo

La Figura 14.281 muestra la derivación de la asociación relativa desde las relaciones subconjunto hijo-padre, marido-mujer y hermano. Puesto que hermano y familiar son simétricos, sólo mostramos un nombre de rol; no hay forma de declarar que una asociación es simétrica. Podríamos construir un modelo del mundo real más completo añadiendo tipos adicionales de familiares, como primos.

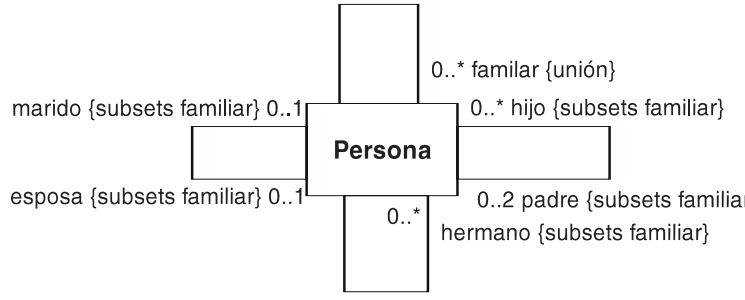


Figura 14.281 Unión derivada

use

Palabra clave utilizada en la notación para la dependencia de uso.

uso

Dependencia en la que un elemento (el cliente) requiere la presencia de otro elemento (el proveedor) para su correcto funcionamiento o implementación —normalmente por razones de implementación.

Véase también colaboración, dependencia, enlace transitorio.

Semántica

Una dependencia de uso es una situación en la que un elemento requiere la presencia de otro elemento para su correcta implementación o funcionamiento. Todos los elementos deben existir al mismo nivel de significado —es decir, no implican un cambio en el nivel de abstracción o realización (como un mapeo entre una clase a nivel de análisis y una clase a nivel de implementación). Frecuentemente una dependencia de uso implica elementos a nivel de implementación, como la inclusión de archivos de C++, que implica consecuencias de compilador. Un uso puede ser aún más estereotipado para indicar la naturaleza exacta de la dependencia, como la llamada a una operación de otra clase o la instanciación de un objeto de otra clase.

Notación

Un uso se indica mediante una flecha discontinua (dependencia) con la palabra clave «**use**». La cabeza de la flecha se coloca en el elemento proveedor (independiente), y la cola se coloca en el elemento cliente (dependiente).

Discusión

Un uso normalmente corresponde a un enlace transitorio —es decir, una conexión entre instancias de clases que no es significativa o no está presente todo el tiempo, sino sólo en cierto contexto, como la ejecución de un procedimiento de subrutina. La construcción de dependencia no

modela la información completa de esta situación, sólo el hecho de su existencia. La construcción de colaboración proporciona la capacidad de modelar dichas relaciones con todo detalle.

uso de la colaboración

Uso de una colaboración ligada a partes específicas dentro de un contexto.

Semántica

Una colaboración es la definición de un contexto que implica varios roles. Se puede utilizar una colaboración ligando los roles a clasificadores dentro de un contexto concreto, como la estructura interna de una clase o la definición de una colaboración mayor. Dicha colaboración ligada se conoce como uso de la colaboración. Se puede usar una colaboración muchas veces en diferentes usos de la colaboración.

Un clasificador ligado a un rol debe ser compatible con el tipo del rol, si lo tiene. Un clasificador es compatible si es del mismo tipo o un descendiente del tipo. También debe cumplir todas las restricciones del rol.

También puede darse el uso de una colaboración en un modelo de objetos. La clase de cada objeto debe ser compatible con el tipo del rol correspondiente. Tal uso de la colaboración representa un conjunto de objetos que interactúan para alcanzar los propósitos de la colaboración. Normalmente, dicho uso es dentro de algún contexto, como la especificación de un procedimiento, pero el contexto puede no estar siempre explicitado.

El uso de una colaboración puede aparecer dentro de la definición de una colaboración mayor. En este contexto, sus roles están ligados a roles de la colaboración mayor, más que a clasificadores. Los tipos de los roles de la colaboración mayor deben ser compatibles con los tipos de los roles de la colaboración utilizada en el uso. Cuando se liga la colaboración mayor, automáticamente se liga al rol correspondiente de la colaboración interna un objeto que a su vez está ligado a un rol de la colaboración mayor.

Un uso de una colaboración puede conectar un clasificador a una colaboración, para indicar que la colaboración explicita un aspecto del comportamiento de la colaboración. Puede haber muchos de los anteriores usos de una colaboración. Un clasificador puede tener un uso de una colaboración para mostrar la colaboración que representa el comportamiento principal del clasificador entero.

Notación

Un uso de la colaboración se representa como una elipse discontinua que contiene una cadena.

`nombreDelUso : Colaboración`

donde **`nombreDelUso`** identifica el uso particular y **`Colaboración`** es el nombre de la definición de la colaboración. Se puede omitir la cadena **`nombreDelUso`**. Para cada rol de la colaboración, se dibuja una línea discontinua desde la elipse hasta el símbolo rectangular del clasificador ligado al rol. El nombre del rol se sitúa al final de la línea, cerca del símbolo del clasificador.

Se puede utilizar un uso de la colaboración para vincular una colaboración a un clasificador para el cual muestra un aspecto de su comportamiento. Se representa mediante una flecha discontinua desde el uso de la colaboración hasta el símbolo del clasificador. Si la colaboración representa el comportamiento global del clasificador (en vez de representar sólo un aspecto del comportamiento), se coloca la palabra clave «**represents**» en la flecha (). *Véase* la Figura 14.282.

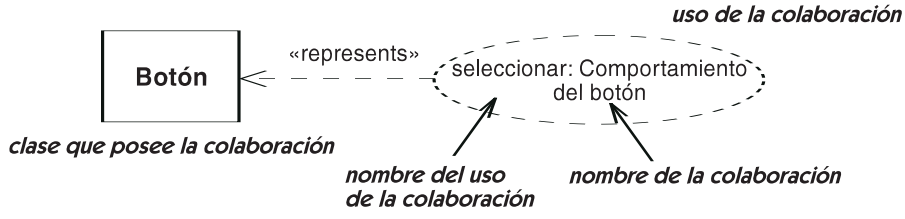


Figura 14.282 Comportamiento de una clase representado por una colaboración

Ejemplo

La Figura 14.283 muestra la definición de una colaboración muy general que representa cualquier venta. Tiene tres roles, un comprador, un vendedor y un elemento. Los compradores y vendedores deben ser agentes, lo que incluye personas y compañías. Se podría vincular a la colaboración una interacción que muestre el proceso de negociación, pago y demás.

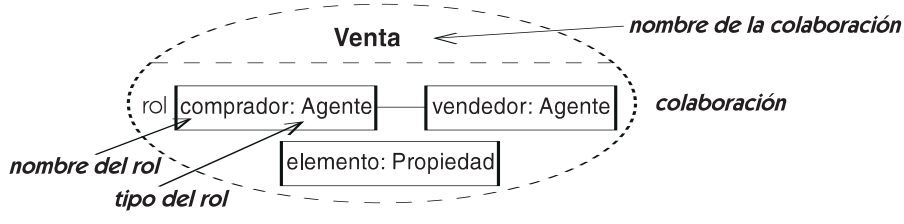


Figura 14.283 Definición de la colaboración

La Figura 14.284 muestra el uso de esta colaboración en un diagrama de objetos, representando una venta concreta, la compra de Luisiana. Napoleón y Thomas Jefferson son de tipo Ejecutivo, una clase de Agente. Luisiana es de tipo Tierra, una clase de propiedad. Las ligas son compatibles con los tipos de los roles.

La Figura 14.285 muestra la utilización de la colaboración Venta dentro de una colaboración mayor que representa la venta al por menor como una cadena de ventas. Esta colaboración tiene cuatro roles: un mayorista, un minorista, un cliente y un producto. La relación entre el mayorista y el minorista es la misma que la relación entre el minorista y el cliente —ambas relaciones son usos de la colaboración Venta. Téngase en cuenta que el rol producto está ligado al rol elemento en las dos colaboraciones anidadas. Esta colaboración muestra cómo un mismo producto pasa por la cadena de ventas, por lo que es importante que ambos roles elemento estén ligados al mismo rol producto.

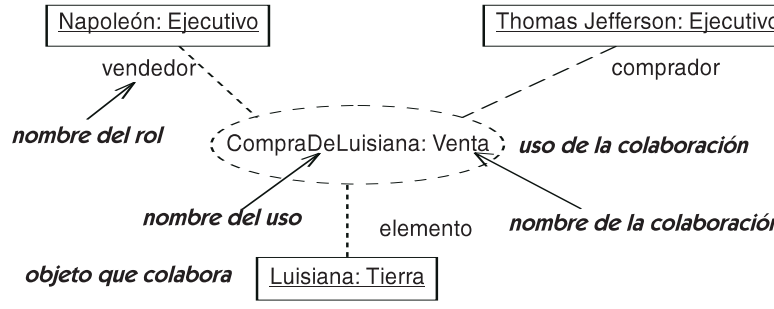


Figura 14.284 Uso de la colaboración en un diagrama de objetos

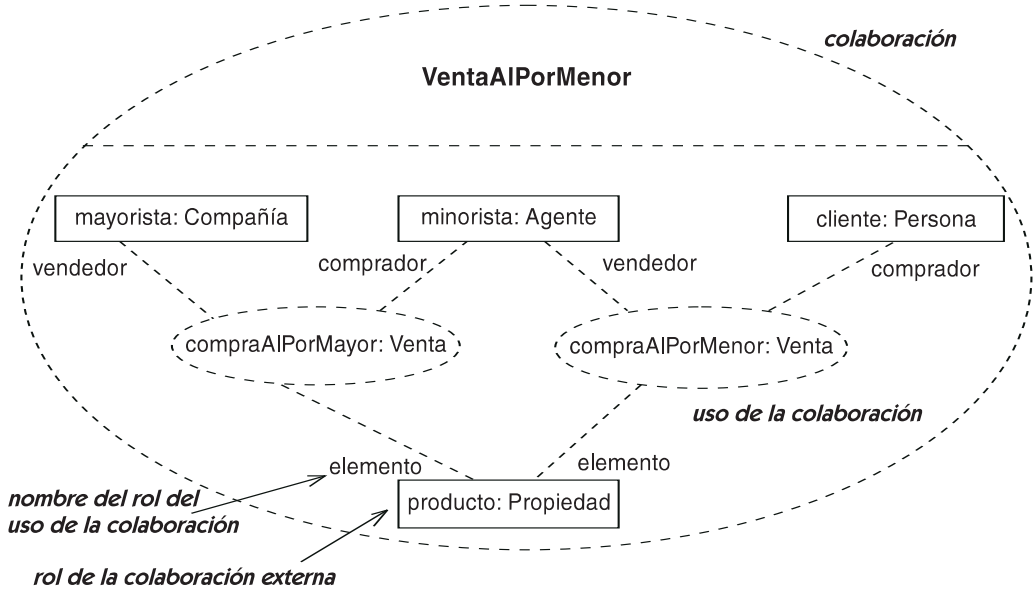


Figura 14.285 Colaboraciones anidadas

El mayorista es de tipo Compañía, que es una clase de Agente. El cliente es de tipo Persona, que también es una clase de Agente. Aunque la colaboración Venta sólo requiere que el comprador y el vendedor sean de tipo Agente, los roles en la colaboración VentaAlPorMenor son más específicos. El minorista es de tipo Agente; se ha mantenido general.

Este ejemplo muestra cómo poder construir grandes colaboraciones a partir de otras más pequeñas.

Discusión

Se usaba el término *ocurrencia de la colaboración*, pero una limpieza de terminología distingue ocurrencia como una instancia de un evento, es decir, algo en el dominio temporal, mientras que un uso de la colaboración está en el dominio estático.

uso de la interacción

Referencia a una interacción dentro de la definición de otra interacción.

Semántica

Un uso de una interacción es una referencia parametrizada a una interacción dentro del cuerpo de otra interacción. Al igual que con cualquier referencia modular, como los procedimientos, esto permite la reutilización de una definición en muchos contextos diferentes. Cuando se ejecuta un uso de una interacción, el efecto es el mismo que ejecutar la interacción referenciada con la sustitución de los argumentos proporcionados en el uso de la interacción.

Estructura

referent	Referencia a una interacción de destino que se ejecuta cuando lo haga el uso de la interacción.
argumentos	Lista de argumentos que son sustituidos con los parámetros de la interacción de destino.
puertas	Lista de puertas en la interacción global que se corresponden con las puertas parametrizadas de la interacción de destino.

El uso de la interacción debe abarcar todas las líneas de vida que aparecen dentro de la interacción referenciada y que suceden dentro de la interacción original. (La interacción referenciada podría añadir líneas de vida que suceden únicamente dentro de sí misma.)

Notación

Un uso de una interacción se representa en un diagrama de secuencia como un rectángulo con la etiqueta **ref** (de referencia). El rectángulo abarca las líneas de vida que están incluidas en la interacción referenciada. El nombre de la interacción referenciada se coloca en el rectángulo. Todos los parámetros de la referencia se colocan en una lista separada por comas entre paréntesis detrás del nombre. Los valores de retorno (si existen) se colocan después de los paréntesis con dos puntos (:) antes de la lista de valores.

La sintaxis completa de la cadena de nombre es:

⌊**nombre-del-atributo-de-retorno** = ⌋$_{opc}$ ⌊**uso-de-la-colaboración.**⌋$_{opc}$
nombre-de-la-interacción ⌊(**argumentoslista**)⌋$_{opc}$ ⌊**: valor-de-retorno** ⌋$_{opc}$

donde nombre-del-atributo-de-retorno es el nombre de un atributo que recibe el valor de retorno, uso-de-la-colaboración es el nombre del uso de la colaboración dentro de un clasificador estructurado (poco frecuente), nombre-de-la-interacción es el nombre de la interacción referenciada, argumentos es una lista separada por comas, y valor-de-retorno se asigna al atributo de retorno. La forma completa no es necesaria en ejemplos sencillos.

La Figura 14.286 muestra dos usos de la interacción, uno con un argumento y otro sin argumentos.

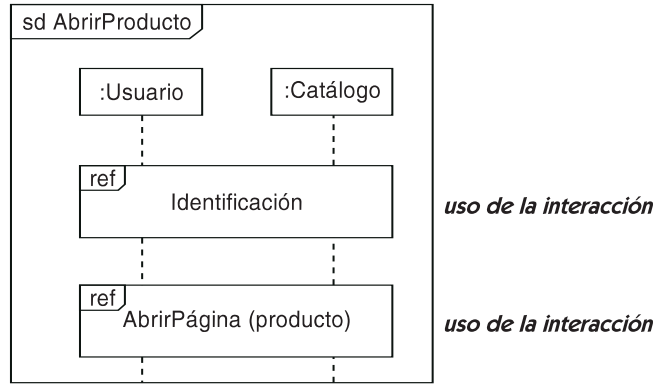


Figura 14.286 Usos de interacción

Discusión

Se conocía como *ocurrencia de la interacción*, pero entraba en conflicto con el significado de ocurrencia como una instancia de un evento.

uso de la tipografía

Se puede distinguir el texto a través de la utilización de diferentes tipografías y otros marcadores gráficos.

Véase también marcador gráfico.

Discusión

Las cursivas se utilizan para indicar una clase abstracta, atributo u operación. Se usan otras distinciones tipográficas para destacar o distinguir partes de la notación. Se recomienda que los nombres de los clasificadores y de las asociaciones se muestren en negrita y los elementos subsidiarios, como atributos, operaciones, nombres de rol y demás, se muestren en tipografía normal. Los nombres de los compartimentos deberían mostrarse en una tipografía distintiva, como en negrita pequeña, pero la elección se deja a las herramientas de edición. Una herramienta también es libre de utilizar distinciones tipográficas para resaltar elementos seleccionados, para distinguir palabras reservadas y palabras clave, y para codificar propiedades seleccionadas de un elemento, o puede permitir el uso de tales distinciones bajo el control del usuario. Se pueden aplicar consideraciones similares para el color, aunque su uso debería ser opcional puesto que muchas personas no distinguen los colores. Todos estos usos son extensiones por comodidad a la notación canónica descrita en este libro, que es suficiente para representar cualquier modelo.

utility (estereotipo de Clase)

Clase estereotipada que no tiene instancias. Describe una colección con nombre de atributos y operaciones no miembros, que son todas ellas de ámbito de clase.

Discusión

Más una técnica de programación que un concepto de modelado.

valor

Véase valor dato.

valor cadena

Un valor que es una cadena.

valor dato

Instancia de un tipo de datos, un valor sin identidad.

Véase también tipo de datos, objeto.

Semántica

Un valor de datos es un miembro de un dominio matemático —un valor puro. Dos valores de datos con la misma representación son indistinguibles; los valores de datos no tienen identidad. En un lenguaje de programación los valores de datos se pasan por valor. No tiene sentido pasarlos por referencia. Tampoco tiene sentido hablar de cambiar un valor de datos; su valor está fijo permanentemente. De hecho, ese es su valor. Normalmente, cuando uno habla de cambiar un valor de datos, quiere decir cambiar una variable que contiene un valor de datos para que contenga otro valor de datos nuevo. Pero los valores de datos por sí mismo son invariables.

valor etiquetado

Un par etiqueta valor, conectado a un elemento del modelo para contener información.

Véase también restricción, estereotipo.

Semántica

Un valor etiquetado es una pareja valor nombre, que puede conectarse a un elemento del modelo, que usa un estereotipo, que contiene una definición de etiqueta.

Los valores etiquetados representan información de modelado adicional, más allá de la definida en el metamodelo de UML. Se usa comúnmente para almacenar información de gestión del proyecto, tal como el autor de un elemento, el estado de las pruebas, o la importancia del elemento para el sistema final (las etiquetas pueden ser **autor**, **estado** e **importancia**).

Los valores etiquetados representan una modesta extensión a los metaatributos de las metaclases de UML. Esto no es un mecanismo de extensión general, pero puede usarse para añadir información a las metaclases existentes, para beneficio de las herramientas de soporte, tales como generadores de código, generadores de informes y simuladores. Para evitar la confusión,

las etiquetas deben diferir de los metaatributos existentes de los elementos del modelo a los que se aplican. Esta comprobación puede ser facilitada por una herramienta de modelado.

Algunas etiquetas está predefinidas en UML, otras pueden ser defenidas dentro de estereotipos. Los valores etiquetados, son mecanismos de extensión, que permiten asociar información arbitraria a los modelos.

Notación

Cada valor etiquetado se muestra en la forma

`etiqueta=valor`

donde **`etiqueta`** es el nombre de una etiqueta, y **`valor`** es un literal. Los valores etiquetados, pueden incluirse conjuntamente con otras palabras clave de propiedad, en un símbolo de comentario unido al rectángulo de un clasificador, mediante una línea discontinua. El clasificador debe tener un estereotipo declarando la definición de la etiqueta.

Se puede declarar una palabra clave para hacer referencia a una etiqueta con un valor concreto. En ese caso, la palabra clave se puede usar sola. La ausencia de la etiqueta, es equivalente a uno de los valores válidos para la etiqueta.

`etiqueta`

La Figura 14.287, muestra la aplicación del estereotipo autoría declarado en la Figura 14.92 a la clase cliente. Los valores de cada una de las etiquetas, se especifican para la clase cliente.

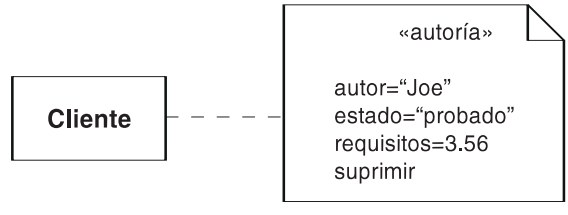


Figura 14.287 Valores etiquetados

Discusión

Los valores etiquetados son un medio de asociar información no semántica de gestión y seguimiento del proyecto a los modelos. Por ejemplo, la etiqueta **autor** puede contener el autor de un elemento y la etiqueta **estado** puede contener el estado de desarrollo, tal como **incompleto**, **probado**, **defectuoso**, **y completo**.

Los valores etiquetados, también son una forma de asociar controles dependientes del lenguaje de implementación, a un modelo UML sin construir los detalles del lenguaje en UML. Las marcas de los generadores de código, ayudas, y programas, pueden codificarse como valores etiquetados, sin que afecten al modelo subyacente. Múltiples conjuntos de etiquetas pueden coexistir, para lenguajes distintos, en el mismo modelo. Ni un editor de modelos, ni un analizador semántico, necesitan entender las etiquetas, pueden manipularlas como cadenas. Una herra-

mienta de soporte, tal como un generador de código, puede entender y procesar los valores etiquetados. Por ejemplo, un valor etiquetado puede dar nombre a la clase contenedora usada para sobrecargar la implementación por defecto de una asociación con multiplicidad muchos.

El uso de etiquetas, al igual que el uso de procedimientos en las bibliotecas de los lenguajes de programación, puede requerir un período de evolución, durante el cual puede haber conflictos entre los desarrolladores. A lo largo del tiempo, se desarrollarán usos estándar. UML no incluye un registro de etiquetas y no ofrece esperanzas de que se vayan a reservar etiquetas en el futuro.

valor inicial

Expresión que especifica el valor que tiene un atributo en un objeto justo después de que haya sido inicializado.

Véase también valor por defecto, inicialización.

Semántica

Un valor inicial es una expresión adjunta a un atributo. La expresión es una cadena de texto que también designa un leguaje para interpretar la expresión. Cuando se instancia un objeto que tiene el atributo, se evalúa la expresión de acuerdo con el lenguaje dado y el valor actual del sistema. El resultado de la evaluación se utiliza para inicializar el valor del atributo en el nuevo objeto.

El valor inicial es opcional. Si no está presente, entonces la declaración del atributo no especifica el valor que tiene un nuevo objeto (pero alguna otra parte del modelo global puede proporcionar dicha información).

Observe que un procedimiento explícito de inicialización para un objeto (como un constructor) puede sustituir una expresión de valor inicial y sobreescribir el valor del atributo.

El valor inicial es un atributo en el ámbito de una clase y se utiliza para inicializarlo una vez al comienzo de la ejecución. UML no especifica el orden relativo de inicialización de diferentes atributos en el ámbito de clase.

Notación

Un valor inicial para un atributo se representa como una expresión de prueba siguiendo a un signo igual (=). *Véase* atributo.

Discusión

En UML2, el valor inicial para un atributo ha sido renombrado confusamente como *valor por defecto*. Probablemente será más entendido si ignora este cambio, que también se utiliza (y con más sentido) para los parámetros con argumentos omitidos.

valor no especificado

Valor que no ha sido especificado todavía por el modelador.

Discusión

Un valor no especificado no es un valor en el sentido propio y no puede aparecer dentro de un modelo finalizado excepto para aquellas propiedades cuyo valor es innecesario o irrelevante. Por ejemplo, la multiplicidad no puede ser desconocida; debe tener algún valor. La falta de cualquier conocimiento es equivalente a una multiplicidad de muchos. La semántica de UML por tanto ni permite ni trata con la ausencia de valores o con valores no especificados.

Hay otra corriente para la que "no especificado" es una parte útil de un modelo sin terminar. Tiene el significado: "Todavía no he pensado en este valor, y he puesto una nota para recordar darle un valor más tarde". Es una sentencia explícita de que el modelo está incompleto. Es una capacidad útil que las herramientas pueden soportar. Por su propia naturaleza, dicho valor no puede aparecer en un modelo terminado, y no tiene sentido definir su semántica. Una herramienta puede proporcionar automáticamente un valor por defecto para un valor no especificado cuando se necesita un valor —por ejemplo, durante la generación de código— pero es simplemente una comodidad y no una parte de la semántica. Los valores no especificados están fuera de la semántica de UML.

De forma similar, no hay significado semántico para un valor por defecto de una propiedad. Una propiedad en el modelo simplemente tiene un valor. Una herramienta puede proporcionar automáticamente valores para las propiedades de los elementos recién creados. Pero de nuevo ésta es una comodidad operacional dentro de la herramienta, y no parte de la semántica de UML. Los modelos UML semánticamente completos no tienen valores por defecto o valores no especificados; simplemente tienen valores.

valor por defecto

Valor proporcionado automáticamente para un parámetro si no se proporciona ningún valor de argumento. *Véase* parámetro.

También se usa (un tanto engañoso) para referirnos al valor inicial de un atributo de un objeto recién creado. *Véase* valor inicial.

Véase también valor no especificado.

Notación

El valor por defecto para una operación o atributo se especifica colocando un signo igual (=) seguido por una expresión de cadena después de la especificación del elemento. Por ejemplo:

```
construirRectángulo (centro: Punto, ancho: Real, alto: Real, ángulo: Real=0)
```

valor tiempo

Valor que representa un momento del tiempo absoluto o relativo.

Véase expresión de tiempo.

variabilidad

Una propiedad que indica si el valor de un atributo o enlace puede cambiar.

Semántica

El indicador de sólo lectura es una restricción Booleana sobre una propiedad, es decir, un extremo de asociación o un atributo. Si es verdadero, no se puede modificar la propiedad después de su inicialización. Un extremo de asociación de sólo lectura debe ser navegable (de otra forma no tendría sentido). Si es falso, la propiedad se puede modificar (a menos que esté prohibido de otra forma).

Notación

Se sitúa la palabra clave **{readOnly}** en la propiedad que no se puede modificar. En aras de la claridad, se puede situar la palabra clave **{unrestricted}** en la propiedad que se puede modificar, aunque normalmente se asume que la ausencia de **{readOnly}** implica variabilidad.

Historia

En UML1 se podían especificar grados de variabilidad, como la capacidad de añadir elementos a un conjunto pero no modificarlos o eliminarlos.

Discusión

Este concepto se encuentra definido de una forma imprecisa. Evidentemente, una propiedad debe ser modificable inicialmente cuando se inicializa, pero el concepto de inicialización no se encuentra definido en UML. Probablemente sería mejor un rango de valores, como en UML1 pero con opciones adicionales, que un único valor Booleano.

variable

Ubicación que puede tener y cambiar valores durante la ejecución de un comportamiento. Las variables pueden aparecer en actividades y otras especificaciones procedimentales.

vértice

Origen o destino de una transición en una máquina de estados. Un vértice puede ser o bien un estado o un pseudoestado.

visibilidad

Enumeración cuyo valor (**public**, **protected**, **private o package**) denota si el elemento del modelo al que se refiere se puede ver desde fuera del espacio de nombres que lo engloba.

Véase también importar para una discusión de las reglas de visibilidad aplicadas a las referencias entre paquetes.

Semántica

La visibilidad declara la capacidad de un elemento de modelado de referenciar a un elemento que está en un espacio de nombres diferente del espacio de nombres del elemento que lo referencia. La visibilidad es parte de la relación entre un elemento y el contenedor que alberga. El contenedor puede ser un paquete, clase u algún otro espacio de nombres. Hay tres visibilidades predefinidas.

public	Cualquier elemento que puede ver el contenedor también puede ver el elemento indicado.
protected	Sólo un elemento dentro del contenedor o descendiente del contenedor puede ver al elemento indicado. Otros elementos no pueden ni referenciarlo ni utilizarlo.
private	Sólo un elemento dentro del contenedor puede ver el elemento. Otros elementos, incluyendo los elementos en los descendientes del contenedor, no pueden ni referenciarlo ni utilizarlo.
package	Se aplica a los elementos que no pueden existir libremente dentro de paquetes, como características o clasificadores. Sólo un elemento declarado en el mismo paquete puede ver al elemento.

Se podrían definir tipos de visibilidad adicionales para algunos lenguajes de programación, como la visibilidad de C++ *implementation*. (Realmente, todas las formas de visibilidad no pública son dependientes del lenguaje.) La utilización de elecciones adicionales debe ser por convención entre el usuario y cualesquiera herramientas de modelado y generadores de código.

Notación

La visibilidad se puede mostrar por una palabra clave de propiedad o por un signo de puntuación colocado delante del nombre de un elemento del modelo.

public	+
protected	#
private	–
package	~

El marcador de visibilidad se puede suprimir. La ausencia de un marcador de visibilidad indica que la visibilidad no se muestra, no que sea pública o que no esté definida. Una herramienta debería asignar visibilidades a los nuevos elementos incluso si la visibilidad no se muestra. El marcador de visibilidad es un atajo respecto a la cadena de especificación de la propiedad de visibilidad.

La visibilidad también se puede especificar mediante palabras clave (public, protected, private). Esta forma se utiliza a menudo como una lista de elementos que se aplican a un bloque entero de atributos u otros elementos de la lista.

Cualesquiera elecciones de visibilidad específicas de un lenguaje o definidas por el usuario se deben especificar mediante una cadena de propiedad o mediante una convención específica de las herramientas.

Clases. En una clase, el marcador de visibilidad se coloca en los elementos de la lista, como atributos y operaciones. Muestra si otra clase puede acceder a los elementos.

Asociaciones. En una asociación, el marcador de visibilidad se coloca en el nombre de rol de la clase destino (el extremo al que debería accederse utilizando la configuración de visibilidad). Muestra si la clase en el otro extremo puede cruzar la asociación hacia el extremo con el marcador de visibilidad.

Paquetes. En un paquete, el marcador de visibilidad se coloca en elementos contenidos directamente dentro del paquete, como clases, asociaciones y paquetes anidados. Muestra si otro paquete que accede o importa el primer paquete pueden ver los elementos.

visibilidad de paquete

Un valor de visibilidad que indica que el elemento dado (normalmente una característica) no es visible fuera del paquete que contiene el clasificador que posee el elemento.

vista

Proyección de un modelo, que es visto desde una perspectiva o posición de ventaja y omite entidades no relevantes para esa perspectiva. La palabra no se usa aquí para denotar un elemento de presentación. En cambio, incluye proyecciones tanto en modelo semántico como en la notación visual.

vista de actividad

Los aspectos del sistema que se encargan de la especificación del comportamiento mediante actividades conectadas con flujos de control. Esta vista contiene las especificaciones de las actividades que se muestran en diagramas de actividad. Se encuentra agrupada débilmente con otras vistas de comportamiento, como la vista dinámica.

Véase actividad.

vista de casos de uso

Aspecto del sistema relacionado con la especificación del comportamiento en términos de casos de uso. Un modelo de casos de uso es un modelo centrado en esta vista. La vista de casos de uso es parte del conjunto de conceptos de modelado agrupados juntos como la vista dinámica.

vista de comportamiento

Una vista de un modelo que hace énfasis en el comportamiento de las instancias en un sistema, incluyendo sus métodos, colaboraciones y estados de historia.

vista de despliegue

Vista que muestra los nodos en un sistema distribuido, los artefactos almacenados en cada nodo y los componentes y otros elementos que manifiestan los artefactos.

vista de diseño

Vista de un modelo que contiene una declaración estática de las clases, colaboraciones y componentes de un sistema, sus dependencias y su implementación interna un nivel por encima de un lenguaje de programación. La vista de diseño se ve afectada por temas como control de flujo, estructuras de datos, complejidad computacional, visibilidad y descomposición del modelo para su elaboración por varios equipos.

vista de gestión del modelo

Aspecto de un modelo que trata con la organización del propio modelo en partes estructuradas —a saber, paquetes y modelos. La vista de gestión del modelo a veces se considera una parte de la vista estática y a menudo se combina con la vista estática de los diagramas de clases.

vista de interacción

Vista de un modelo que muestra el intercambio de mensajes entre objetos para llevar a cabo algún propósito. Consiste en colaboraciones e interacciones y se representa utilizando diagramas de comunicación y de secuencia.

vista de máquina de estados

Aspecto del sistema que se ocupa de la especificación del comportamiento de elementos individuales a lo largo de la vida. Esta visión contiene las máquinas de estados. Se agrupa libremente con otras vistas del comportamiento en la vista dinámica.

vista dinámica

Aspecto de un modelo que trata con la especificación e implementación del comportamiento a lo largo del tiempo, como contraposición a la estructura estática encontrada en la vista estática. La vista dinámica es un término de agrupación que incluye la vista de caso de uso, vista de máquina de estado y vista de interacción.

vista estática

Una vista del modelo total que caracteriza los elementos en un sistema y las relaciones estáticas entre ellos. Incluye clasificadores y sus relaciones: asociación, generalización, dependencia y realización. A veces se llama vista de clases.

Semántica

La vista estática muestra la estructura estática de un sistema, en particular, los tipos de elementos que existen (como clases y tipos), su estructura interior, y sus relaciones con otros elementos. Las vistas estáticas no muestran la información temporal, aunque pueden contener instancias de elementos que tienen o describen comportamiento temporal, como especificaciones de funcionamientos o eventos.

Los elementos constitutivos de mayor nivel de una vista estática incluyen a los clasificadores (clase, interfaces, tipos), relaciones (asociación, generalización, dependencia, realización,) restricciones y comentarios. También contiene paquetes y subsistemas como unidades organizativas. Otros electores son subordinados a y están contenidos dentro de los elementos de más alto nivel.

Relacionada a la vista estática y a menudo combinada con ella en diagramas está la vista de diseño, vista del despliegue y la vista de gestión del modelo.

La vista estática puede contrastarse con la vista dinámica que la complementa y se construye a partir de ella.

vista estructural

Una vista de un modelo total que hace énfasis en la estructura de los objetos de un sistema incluyendo sus tipos, clases, relaciones, atributos y funciones.

vista funcional

Vista que trata con la descomposición de un sistema en funciones u operaciones que proporcionan su funcionalidad. Una vista funcional normalmente no se considera 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 da soporte directamente a la vista funcional, aunque una actividad tiene ciertas características funcionales.

vista general del diagrama de interacción

Variación de un diagrama de actividad que incorpora fragmentos de diagramas de secuencia junto con construcciones de flujo de control.

Notación

Una vista general del diagrama de interacción contiene notación de diagramas de secuencia, principalmente referencias y diagramas de secuencia anidados, con la notación de decisión y división de los diagramas de actividad. Los símbolos de control muestran el flujo de control de alto nivel entre los símbolos anidados, que puede contener secuencias, incluyendo mensajes.

La Figura 14.288 muestra un ejemplo de un estudiante que ha sido aceptado en una universidad. En primer lugar el estudiante debe aceptar o declinar la admisión. Después de aceptar, el estudiante debe tanto registrarse en las clases como solicitar alojamiento. Después de completar ambas tareas, el estudiante debe pagar las tasas. Si el pago no se recibe a tiempo, el estudiante es excluido por el secretario.

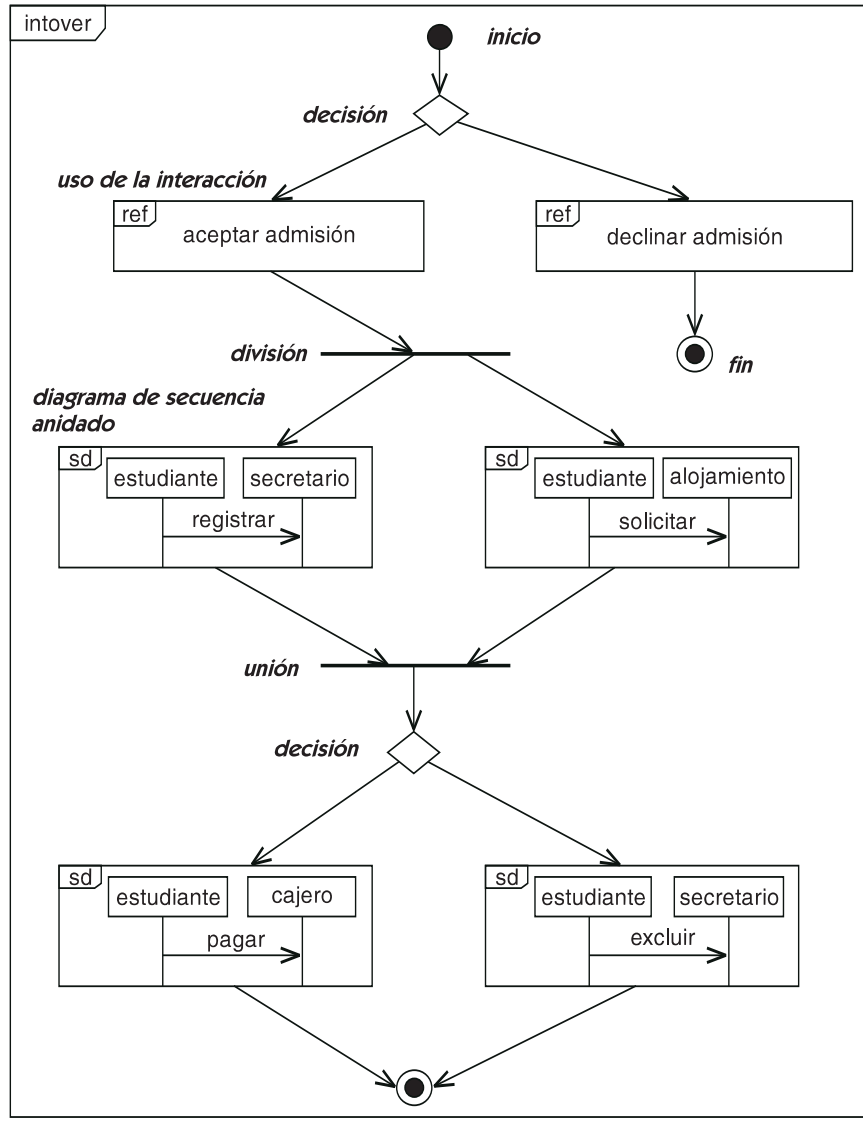


Figura 14.288 Vista general del diagrama de interacción

Discusión

Este diagrama es una extraña mezcla de conceptos de los diagramas de secuencia y de actividad. Intenta mezclar los mecanismos de flujo de control entre nodos de actividad de los diagramas de secuencia junto con la secuencia de mensajes entre líneas de vida de los diagramas de secuencia. Algunas críticas hacen pensar que añade poca novedad.

La especificación UML establece que se utilice la etiqueta **sd** en todos los diagramas que representen interacciones, pero eso parece confuso, de forma que hemos utilizado la etiqueta **intover** para indicar una vista general del diagrama de interacción.

weak

Palabra clave que indica un fragmento combinado de secuenciación débil en un diagrama de interacción.

Véase secuenciación débil.

XMI

Formato externo para la serialización e intercambio de modelos entre plataformas y sistemas. Es el formato estándar de intercambio de UML. Está basado en el lenguaje XML.

xor

Restricción aplicada a un conjunto de asociaciones que comparten una conexión a una clase, especificando que cualquier objeto de la clase compartida tendrá enlaces desde sólo una de las asociaciones. Es una restricción o exclusiva (no o inclusiva).

Véase asociación.

Parte 4: Apéndices

Apéndice A
Metamodelo UML

Documentos de definición de UML

UML está definido por un conjunto de documentos publicados por la OMG (Object Management Group) [UML-04]. Esos documentos se pueden encontrar en la web de la OMG (www.omg.org). Pueden ser actualizados de cuando en cuando por la OMG. Este capítulo explica la estructura del modelo semántico de UML descrito en los documentos.

UML se define formalmente utilizando un metamodelo —es decir, un modelo de las construcciones en UML. El propio metamodelo está expresado en UML. Este es un ejemplo de un intérprete metacircular —es decir, un lenguaje definido en términos de sí mismo. Las cosas no son completamente circulares. Sólo se utiliza un pequeño subconjunto de UML para definir el metamodelo. En principio, este punto fijo de la definición podía ser autosuficiente desde una definición más básica. En la práctica, entrar en tales detalles no es necesario.

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 sobre elementos expresados en lenguaje natural y en OCL; y una descripción textual de la semántica dinámica de las construcciones UML definidas en la sección. La semántica dinámica es por tanto informal, pero una descripción totalmente formal sería tanto nada práctica como ilegible para la mayoría.

La notación se describe en un capítulo separado que hace referencia al capítulo de semántica y mapea los símbolos a clases del metamodelo.

Estructura del documento de especificación

UML se define en dos especificaciones complementarias, *UML 2.0 Infraestructure* y *UML 2.0 Superstructure*. La infraestructura está intencionada para definir conceptos fundacionales que se pueden utilizar en parte o por entero por otras especificaciones, por ejemplo, por la Especificación de metaobjetos (MOF) y el Metamodelo de Almacén Común (CWM). Contiene sólo los conceptos estáticos básicos de UML y está orientado hacia la descripción de las estructuras de datos.

La superestructura de UML define UML por completo como es experimentado por los usuarios. Hay un subconjunto de la superestructura, conocido como el Núcleo, que incorpora en el documento de superestructura todas las partes relevantes de la infraestructura. La especificación

de superestructura es por tanto autocontenida, y los lectores normalmente no tendrán que leer la especificación de infraestructura a no ser que les importe la configuración de otras especificaciones en paralelo a UML.

El resto de este capítulo describe la organización de la especificación de superestructura de UML.

Estructura del metamodelo

El metamodelo se divide en dos paquetes principales, estructura y comportamiento, con dos paquetes de soporte, elementos auxiliares y perfiles (Figura A.1).

- El paquete de estructura define la estructura estática de UML. Dentro del paquete estructural, el paquete de clases forma los cimientos de todo lo demás en UML.
- El paquete de comportamiento define la estructura dinámica de UML. Dentro de este paquete, el paquete de comportamiento común no es utilizable directamente en los modelos, pero define las construcciones compartidas por los otros subpaquetes de comportamiento.
- El paquete de elementos auxiliares define conceptos como tipos de datos.
- El paquete de perfiles proporciona la capacidad de adaptar el modelo

Cada paquete se describe mediante un capítulo en el documento de especificación de superestructura. Las vistas que describimos en la perspectiva general de este libro corresponden aproximadamente a los capítulos de la especificación.

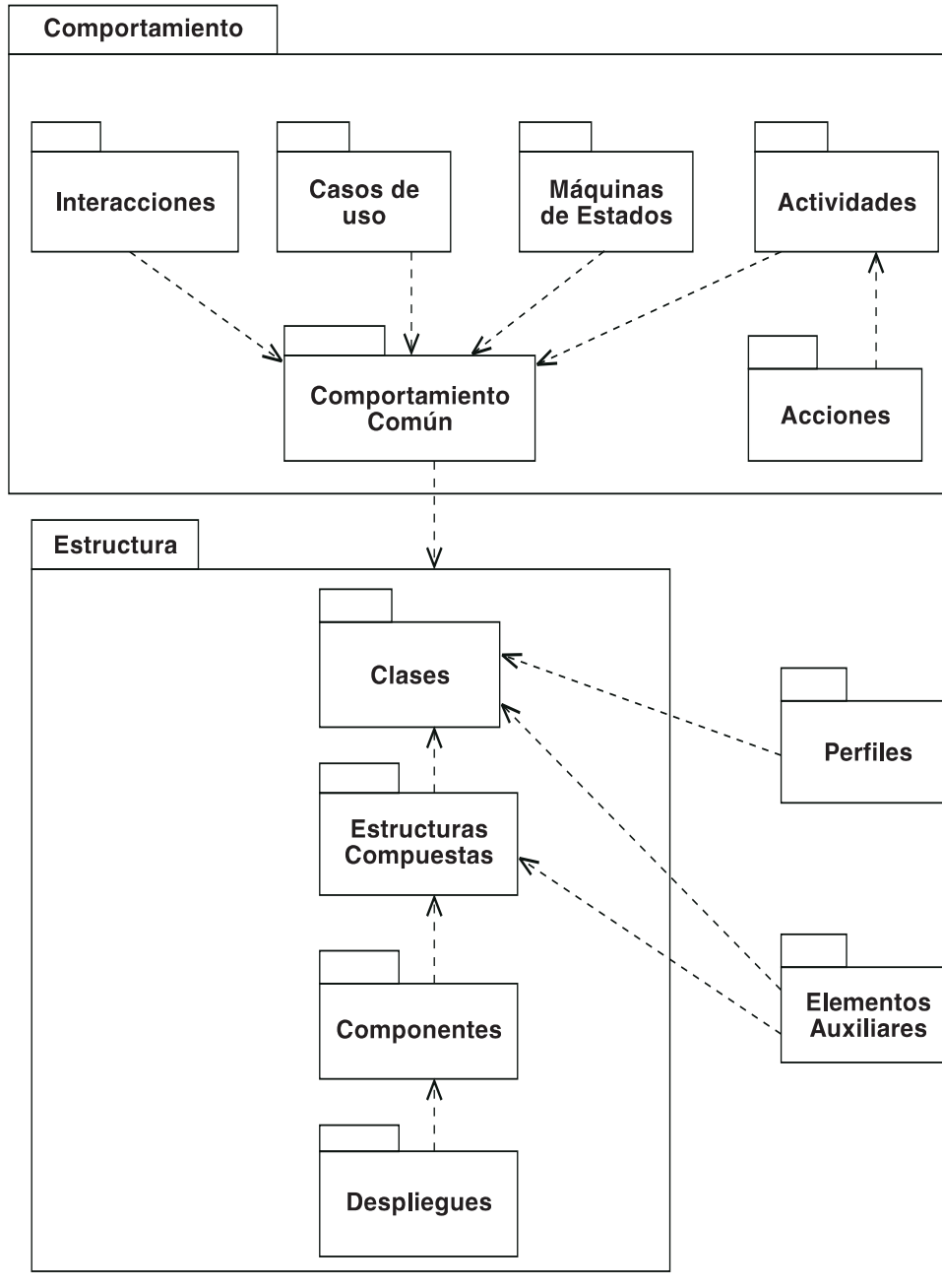


Figura A.1 Estructura de paquetes del metamodelo de UML

Apéndice B

Resumen de notación

Este capítulo contiene un breve resumen visual de notación. Están incluidos los principales elementos de notación, pero no se muestra cada variación u opción. Para ver los detalles completos, véase la entrada de diccionario para cada elemento.

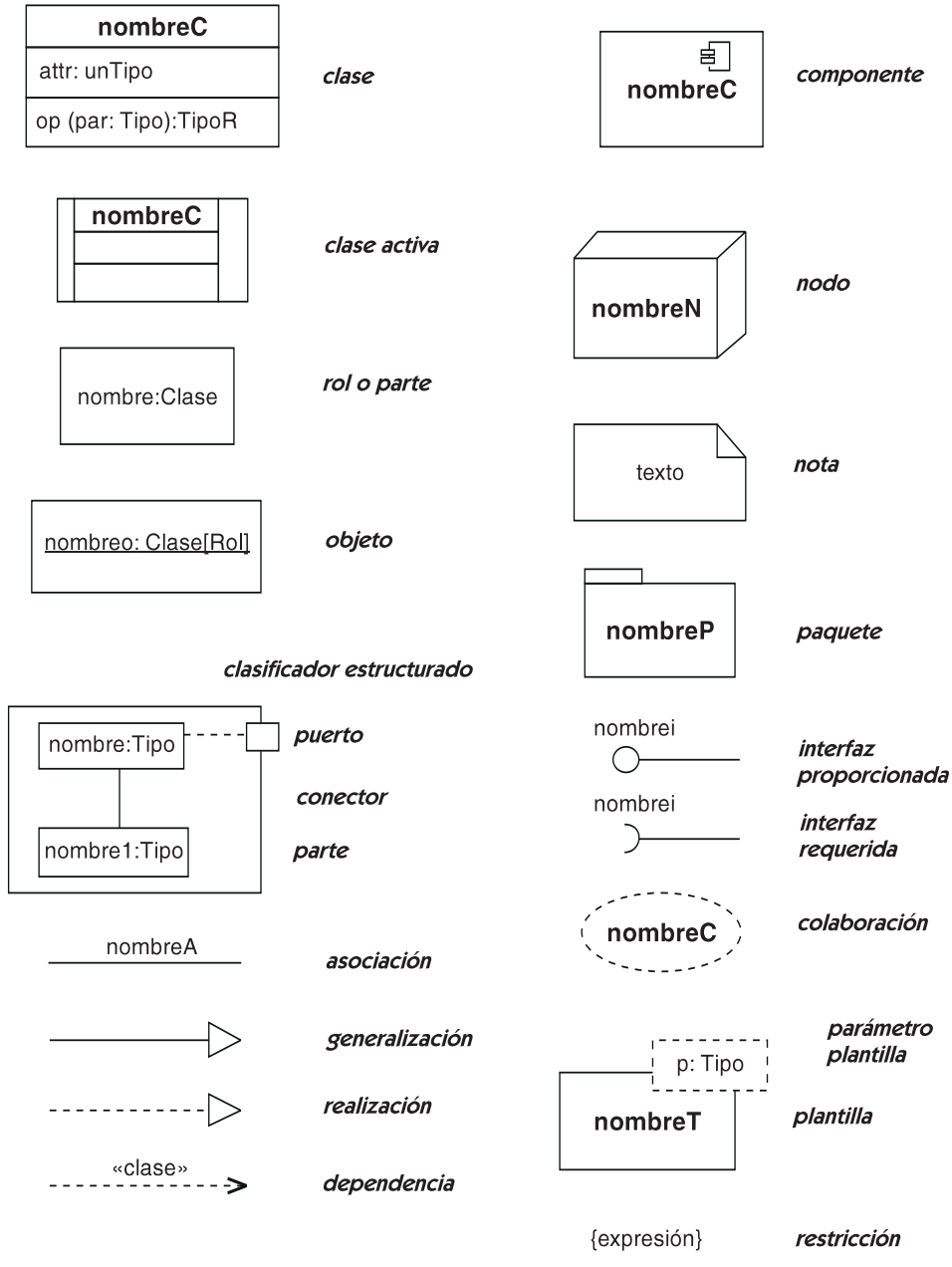


Figura B.1 Iconos de diagramas de clases, componentes, despliegue y colaboración

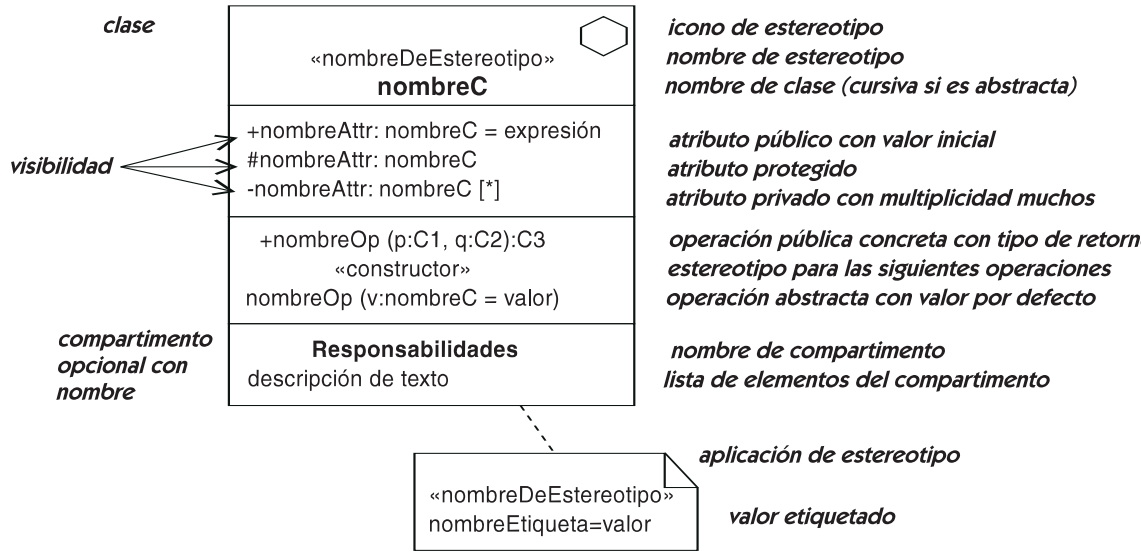


Figura B.2 Contenido de clase

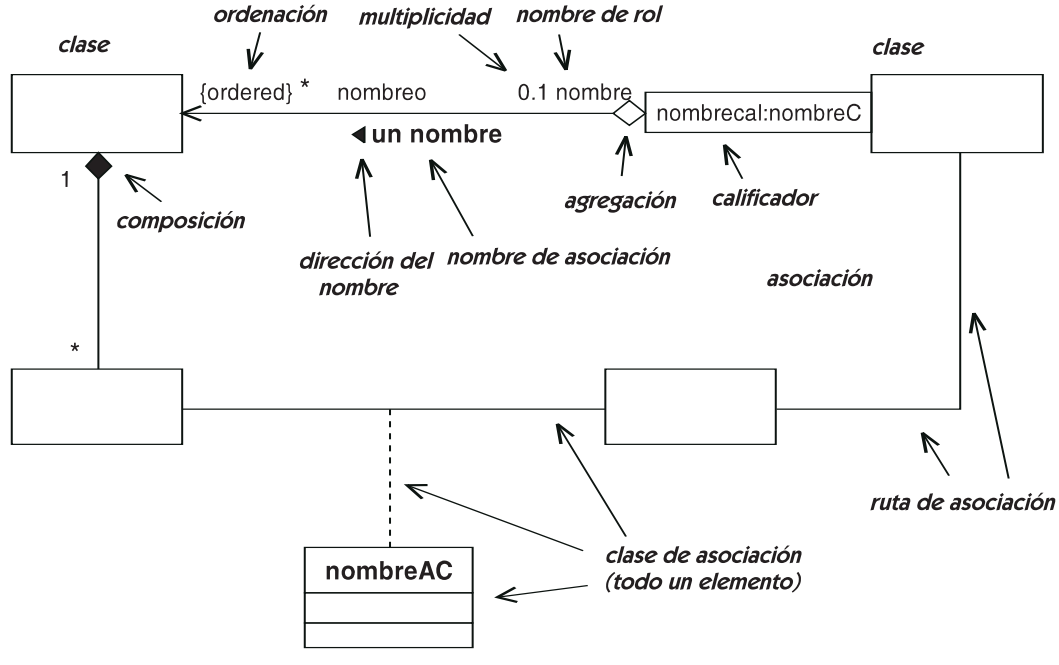


Figura B.3 Adornos de asociación dentro de un diagrama de clases

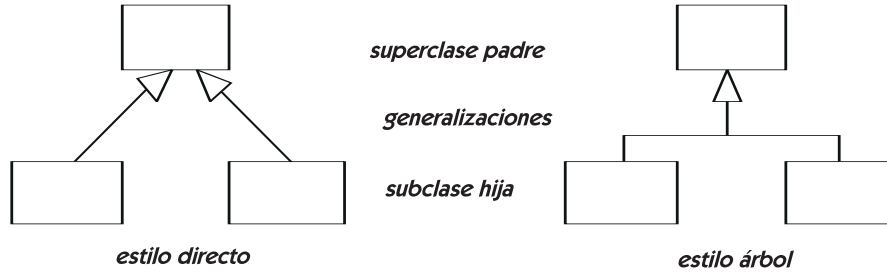


Figura B.4 Generalización

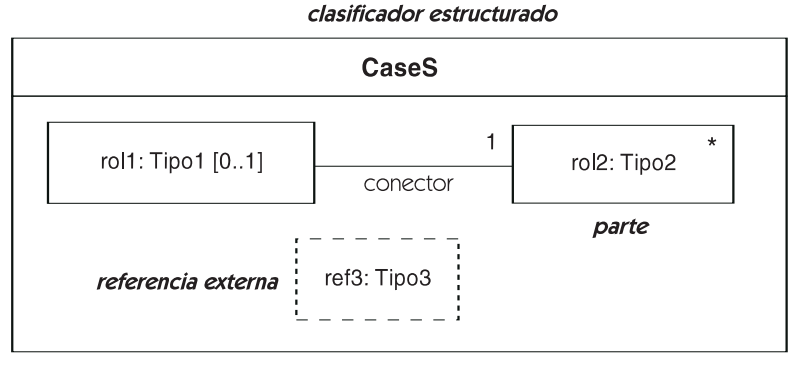


Figura B.5 Estructura interna: partes y conectores

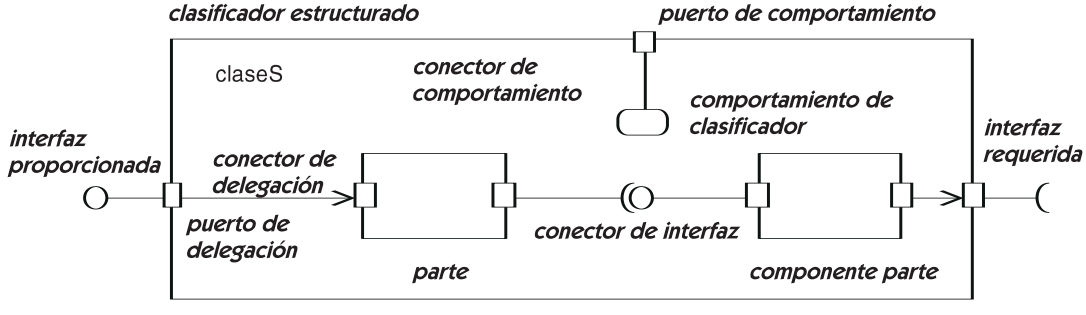


Figura B.6 Estructura interna: interfaces, puertos y conexiones internas

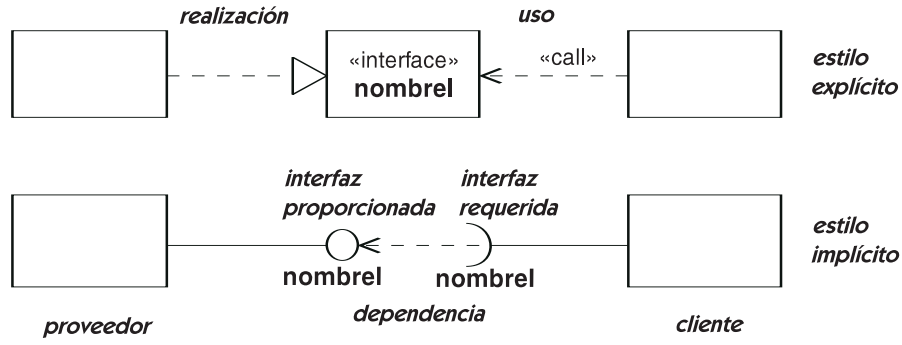


Figura B.7 Realización de una interfaz

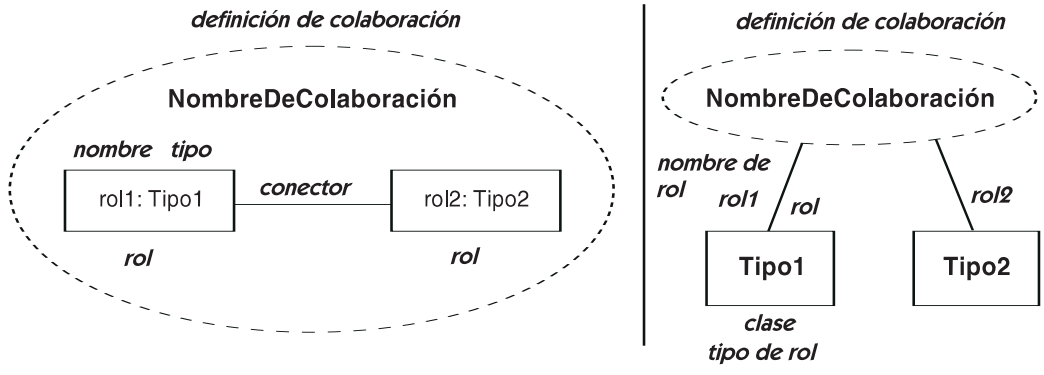


Figura B.8 Definición de colaboración —notaciones alternativas

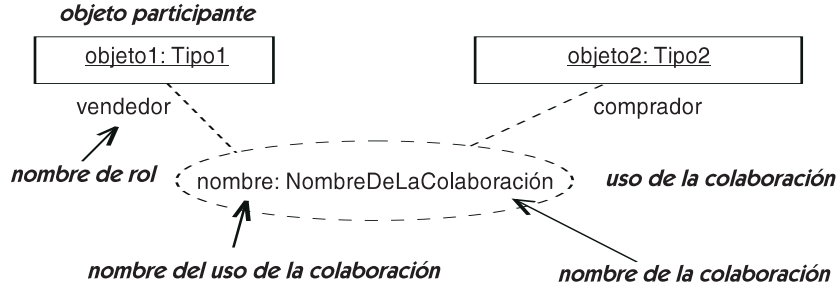


Figura B.9 Uso de la colaboración

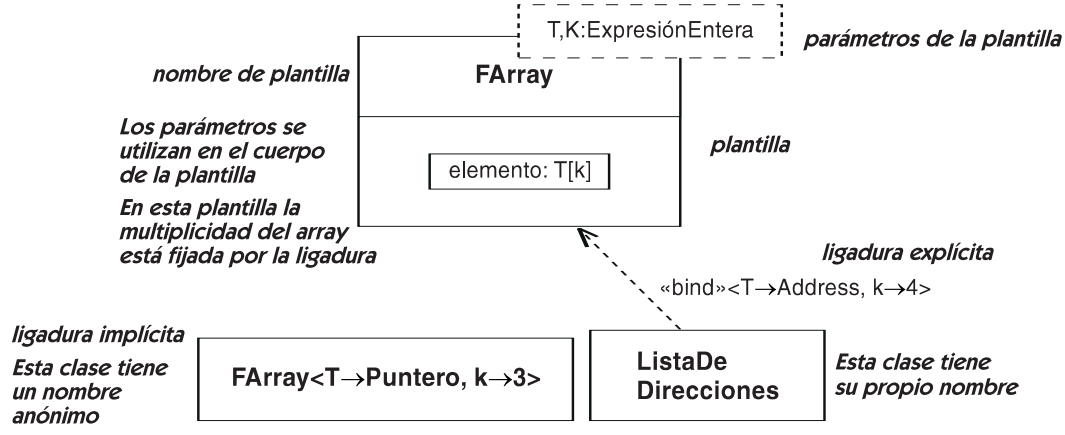


Figura B.10 Plantilla

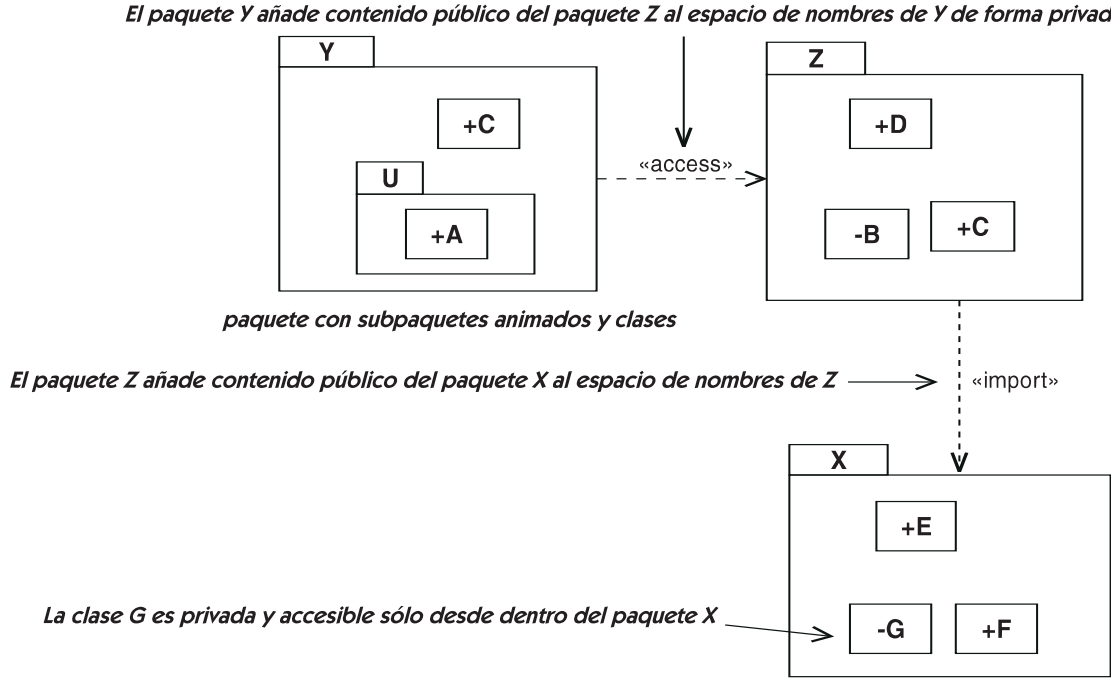


Figura B.11 Notación de paquetes

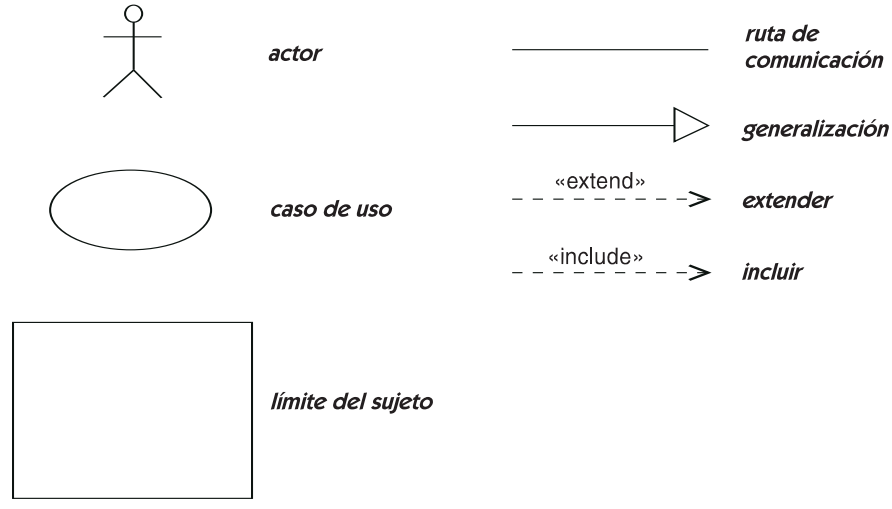


Figura B.12 Iconos de los diagramas de casos de uso

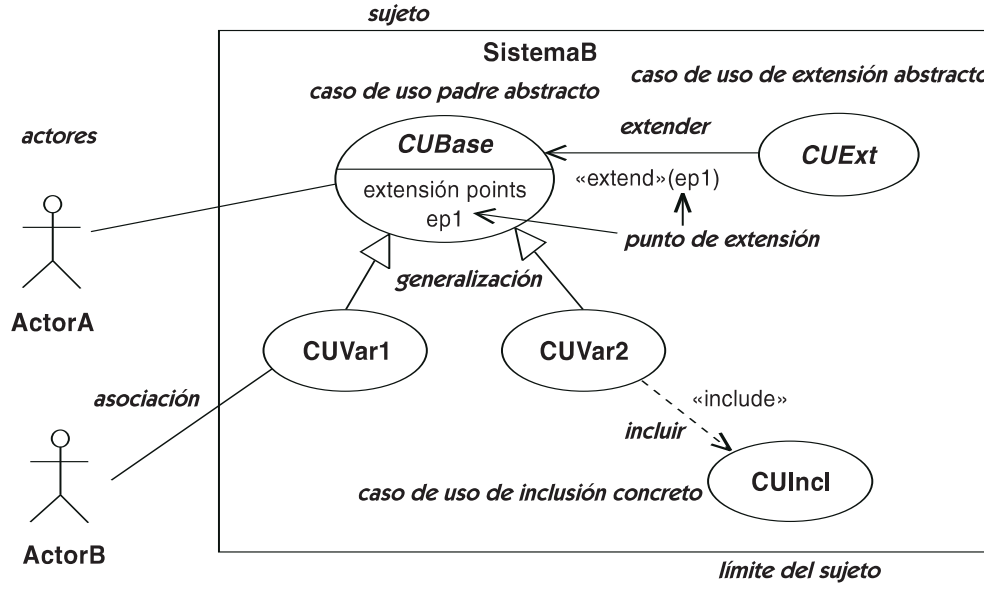


Figura B.13 Notación de diagramas de casos de uso

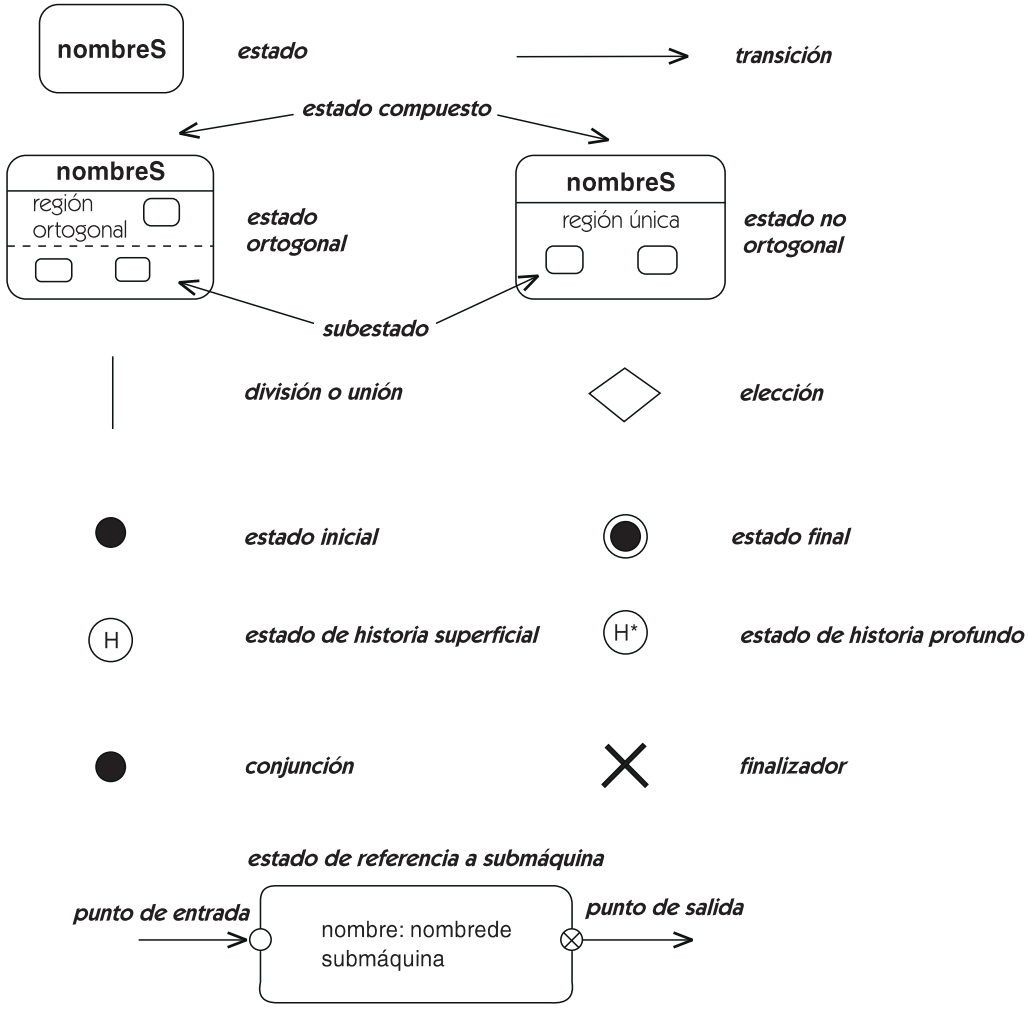


Figura B.14 Iconos de diagramas de máquinas de estado

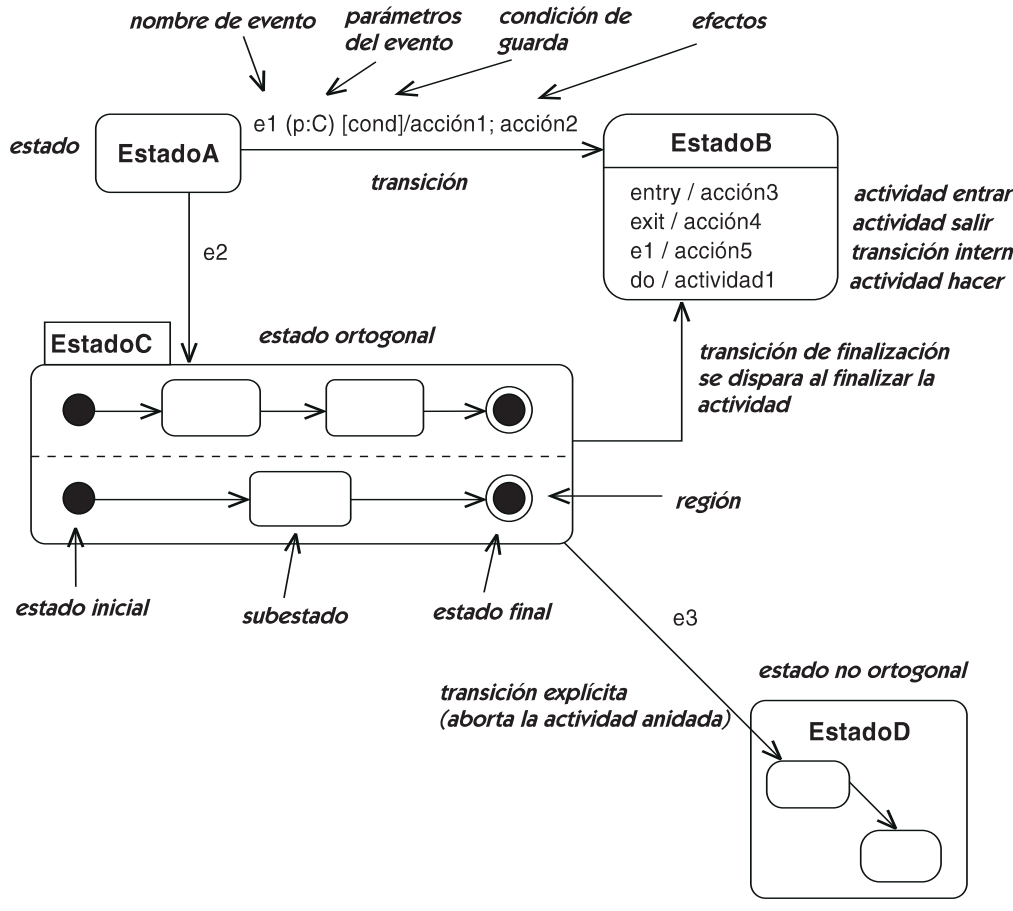


Figura B.15 Notación de máquina de estados

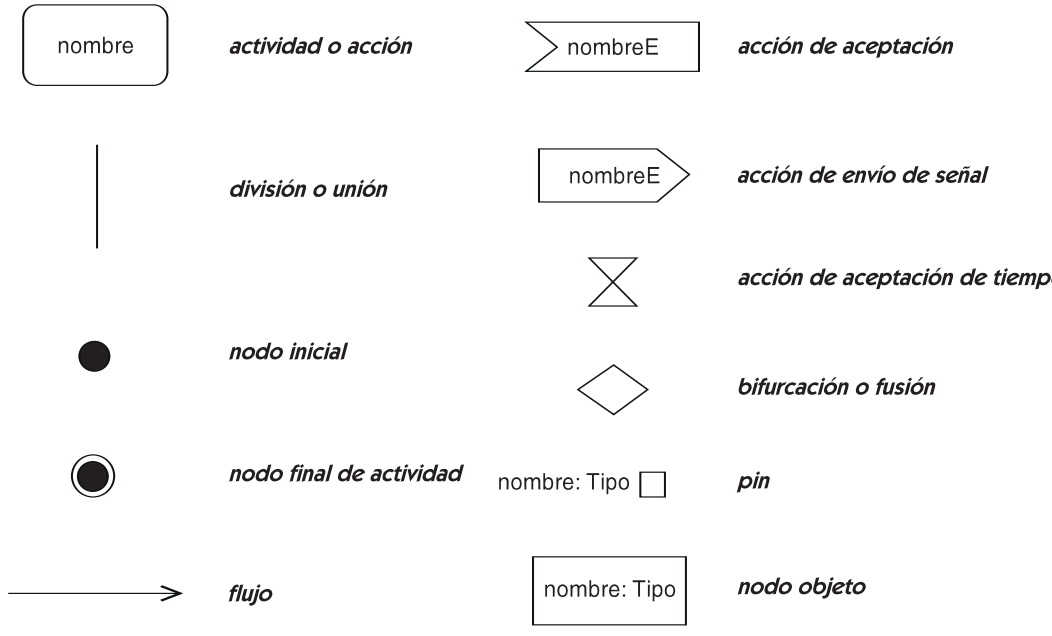


Figura B.16 Iconos de diagramas de actividad

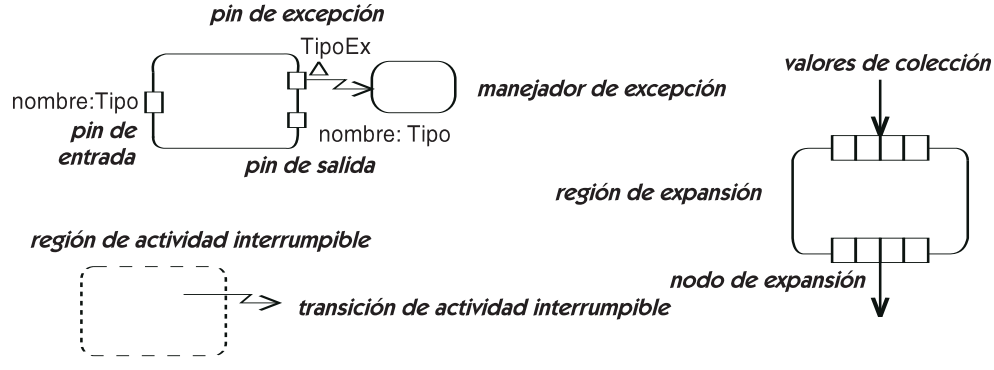


Figura B.17 Grupos de actividad e iconos

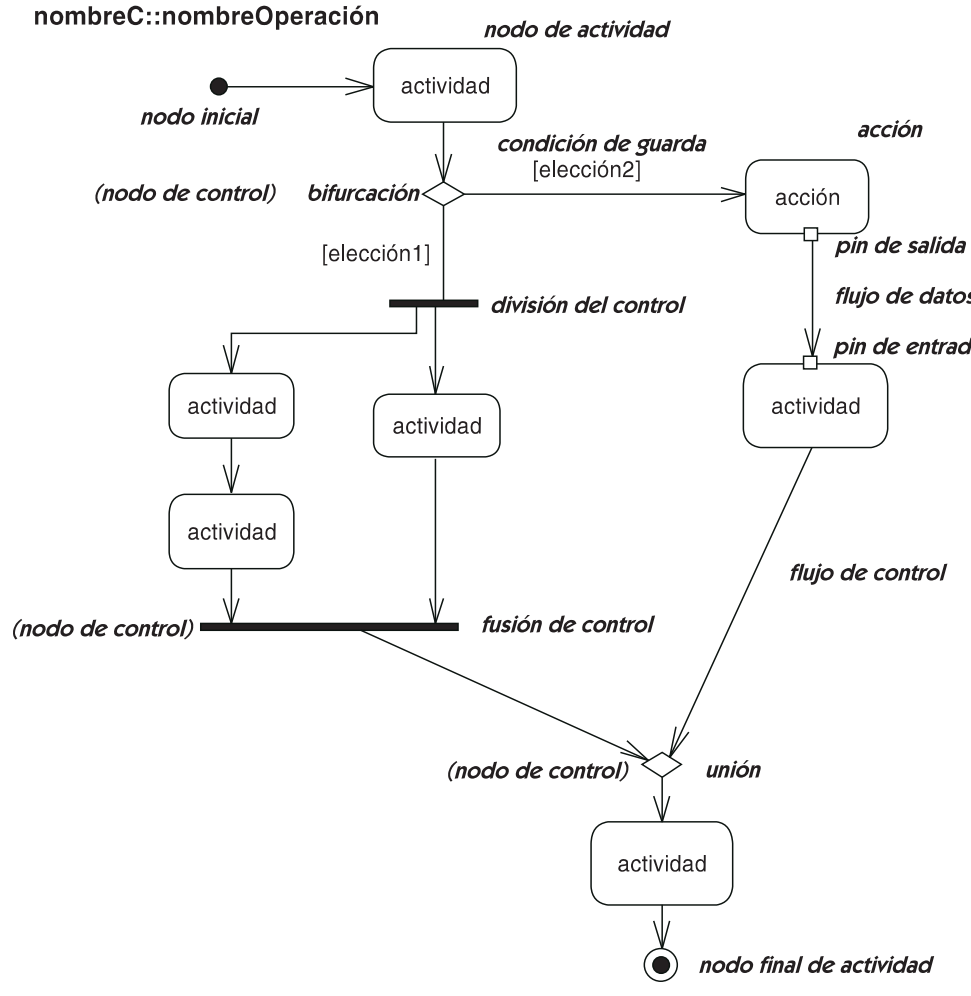


Figura B.18 Notación de diagrama de actividad

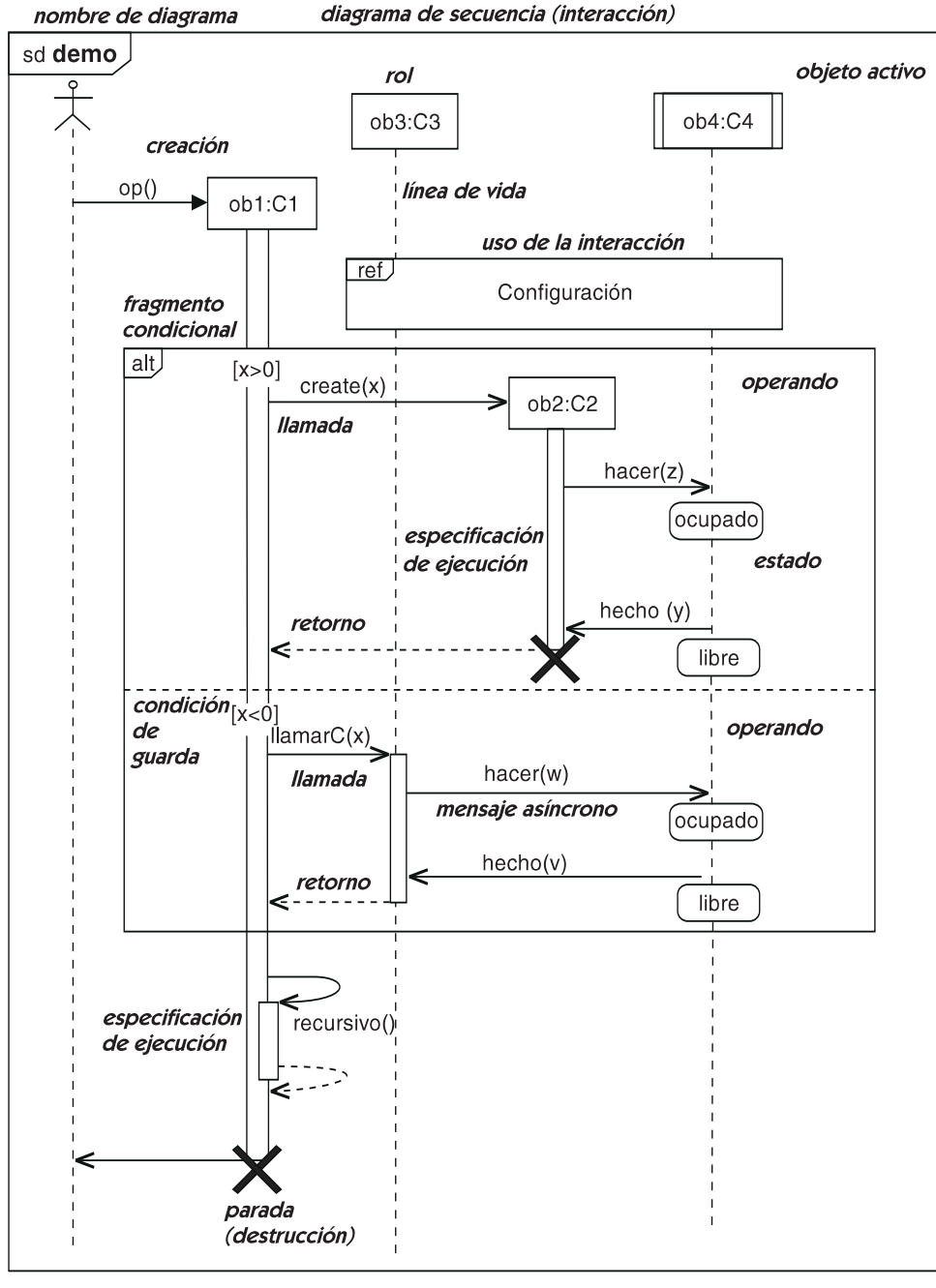


Figura B.19 Notación de diagrama de secuencia

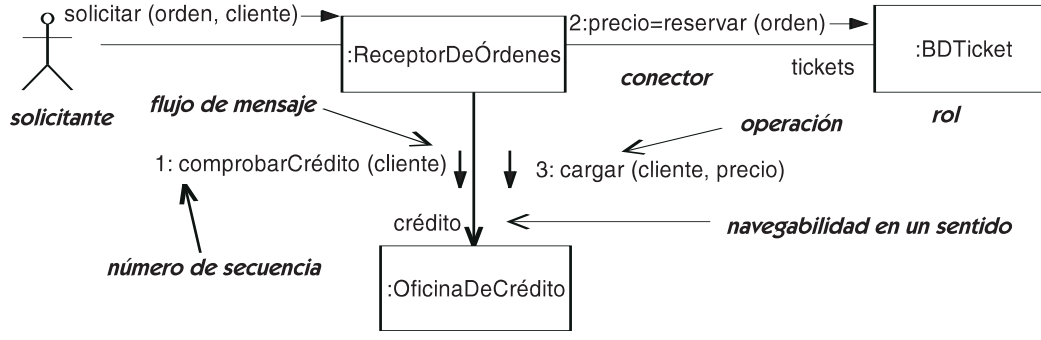


Figura B.20 Notación de diagrama de comunicación

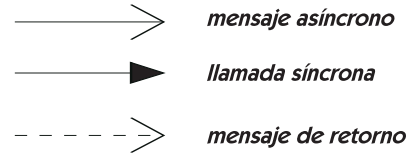


Figura B.21 Notación de mensajes

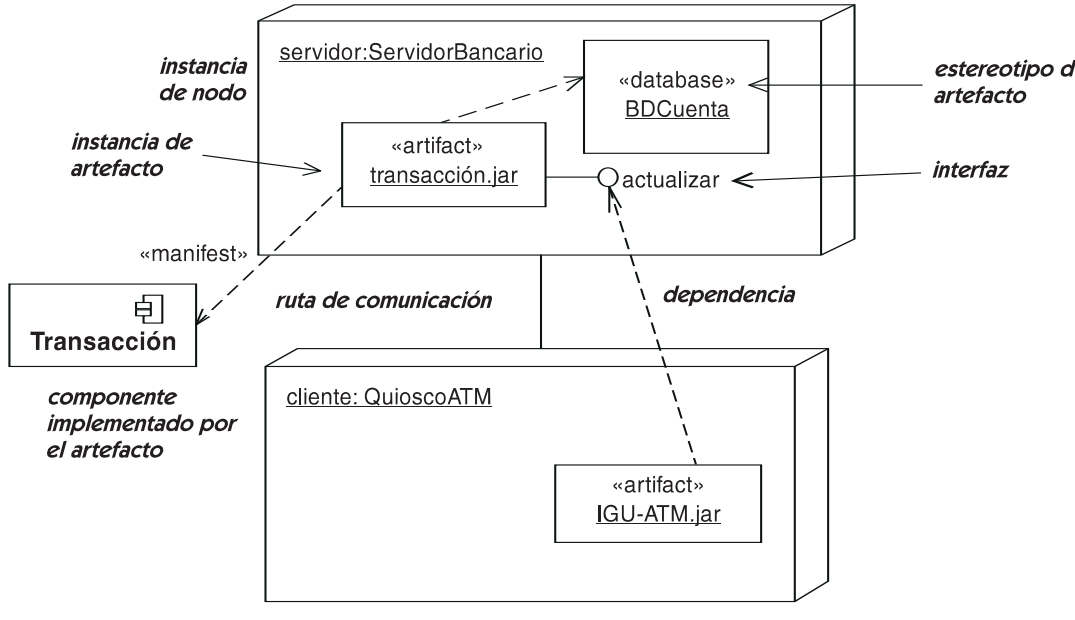


Figura B.22 Notación de nodo y artefacto

Bibliografía

[Birtwistle-75] G.M. Birtwistle, O-J. Dahl, B. Myhrhaug, K. Nygaard. Simula Begin. Petrocelli/Charter, New York, 1975.

[Blaha-98] Michael Blaha, William Premerlani. *Object-Oriented Modeling and Design for Database Applications*. Prentice Hall, Upper Saddle River, N.J., 1998.

[Blaha-05] Michael Blaha, James Rumbaugh. *Object-Oriented Modeling and Design with UML, 2nd edition.* Prentice Hall, Upper Saddle River, N.J., 2005.

[Booch-94] Grady Booch. *Object-Oriented Analysis and Design with Applications, 2nd ed*. Benjamin/Cummings, Redwood City, Calif., 1994.

[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 Patterns*. Wiley, Chichester, U.K., 1996.

[Coad-91] Peter Coad, Edward Yourdon. *Object-Oriented Analysis, 2nd ed*. 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.

[ECOOP] *ECOOP* yyyy, European *Conference on Object-Oriented Programming: Systems, Languages, and Applications* (where *yyy*=1987 and following). A series of conferences on object-oriented technology in Europe. The proceedings are published by Springer Verlag.

[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.

[Fowler-04] Martin Fowler. *UML Distilled, 3rd ed*. Addison-Wesley, Boston, 2004.

[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 Implementation*. Addison-Wesley, Reading, Mass., 1983.

[Harel-98] David Harel, Michal Politi. *Modeling Reactive Systems With Statecharts: The STATEMATE Approach*. McGraw-Hill, New York, N.Y., 1998.

[ITU-T Z.100] International Telecommunication Union. *Specification and Description Language (SDL)*. ITU-T Recommendation Z.100. Geneva, 1999.

[ITU-T Z.120] International Telecommunication Union. *Message Sequence Charts.* ITU-T Recommendation Z.120. Geneva, 1999.

[Jacobson-92] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Wokingham, England, 1992.

[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, New York, N.Y., 1988.

[OOPSLA] *OOPSLA* yyyy, *Conference on Object-Oriented Programming: Systems, Languages, and Applications* (where *yyy*=1986 and following). A series of ACM conferences on object-oriented technology. The proceedings are published as special issues of *ACM Sigplan Notices*.

[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, New York, N.Y., 1996.

[Rumbaugh-99] James Rumbaugh, Ivar Jacobson, Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, Mass., 1999. First edition of this book.

[Selic-94] Bran Selic, Garth Gullekson, Paul T. Ward. *Real-Time Object-Oriented Modeling*. Wiley, New 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, Version 1.1*. Object Management Group, Framingham, Mass., 1998. Internet: www.omg.org.

[UML-04] *Unified Modeling Language Specification, Version 2.0*. Object Management Group, Framingham, Mass., 2004. Internet: www.omg.org.

[UMLConf] *The* x *International Conference on the Unified Modeling Language, UML* yyyy (where *x*=an ordinal and *yyyy*=1998 and following). A series of conferences reporting research on UML. The proceedings are published by Springer-Verlag.

[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 Constraint 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 analítico

A

B

C

CH

D

E

F

G

H

I

J

K

L

LL

M

N

O

P

R

S

T

U

V

W

X

Y