



Tema 11 – Lenguaje Unificado de Modelado

Ingeniería del Software

Héctor Gómez Gauchía
Dep. Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense Madrid

Trabajando con Rubén Fuentes, Antonio Navarro, Juan Pavón y Pablo Gervás





Contenidos

- Introducción
 - Modelado
- Lenguaje Unificado de Modelado
 - Perspectiva general
 - Modelo de casos de uso
 - Modelo estructural
 - Modelo de comportamiento
 - Modelo de implementación





Modelar

- No se construye un edificio sin tener antes unos planos.
- Permite estructurar la solución a un problema.
 - abstrayendo para gestionar la complejidad
 - experimentando varias soluciones
 - reduciendo los costes
 - gestionando el riesgo de errores
- Ventajas de modelar:
 - Permite ver el sistema desde varias perspectivas haciendo más sencillo su entendimiento y desarrollo.
 - Mejora la comunicación.
 - con el cliente
 - del equipo de desarrollo
- El mero hecho de modelar no constituye ni análisis ni diseño, depende de la información que se incluye en los modelos.





Lenguajes de modelado

- Una buena notación libera al analista o diseñador de los aspectos repetitivos del modelado para que se concentre en el estudio de los aspectos específicos de su problema.
 - Indica cuáles son las primitivas que se utilizan para modelar en un dominio.
 - Aunque a cambio restringe los conceptos con los que se puede trabajar.
- Una notación estándar expresiva posibilita además:
 - Describir un universo o formular una arquitectura y comunicar estas decisiones de forma no ambigua.
 - Permite comprobar la consistencia y corrección de las decisiones adoptadas mediante herramientas automáticas.





Lenguaje Unificado de Modelado

- El Lenguaje Unificado de Modelado (*Unified Modelling Language*, UML) [OMG, 2011a; OMG, 2011b] es un lenguaje gráfico para el modelado de sistemas.
 - Especificar, visualizar, construir, documentar
- Especificado como un estándar abierto por el Grupo para la Gestión de Objetos (*Object Management Group*, OMG).
- Soportado por herramientas.
 - *Rational Software Architect* (RSA), *Together*, *Objectteering*, *Paradigm Plus*, *Eclipse*, ...





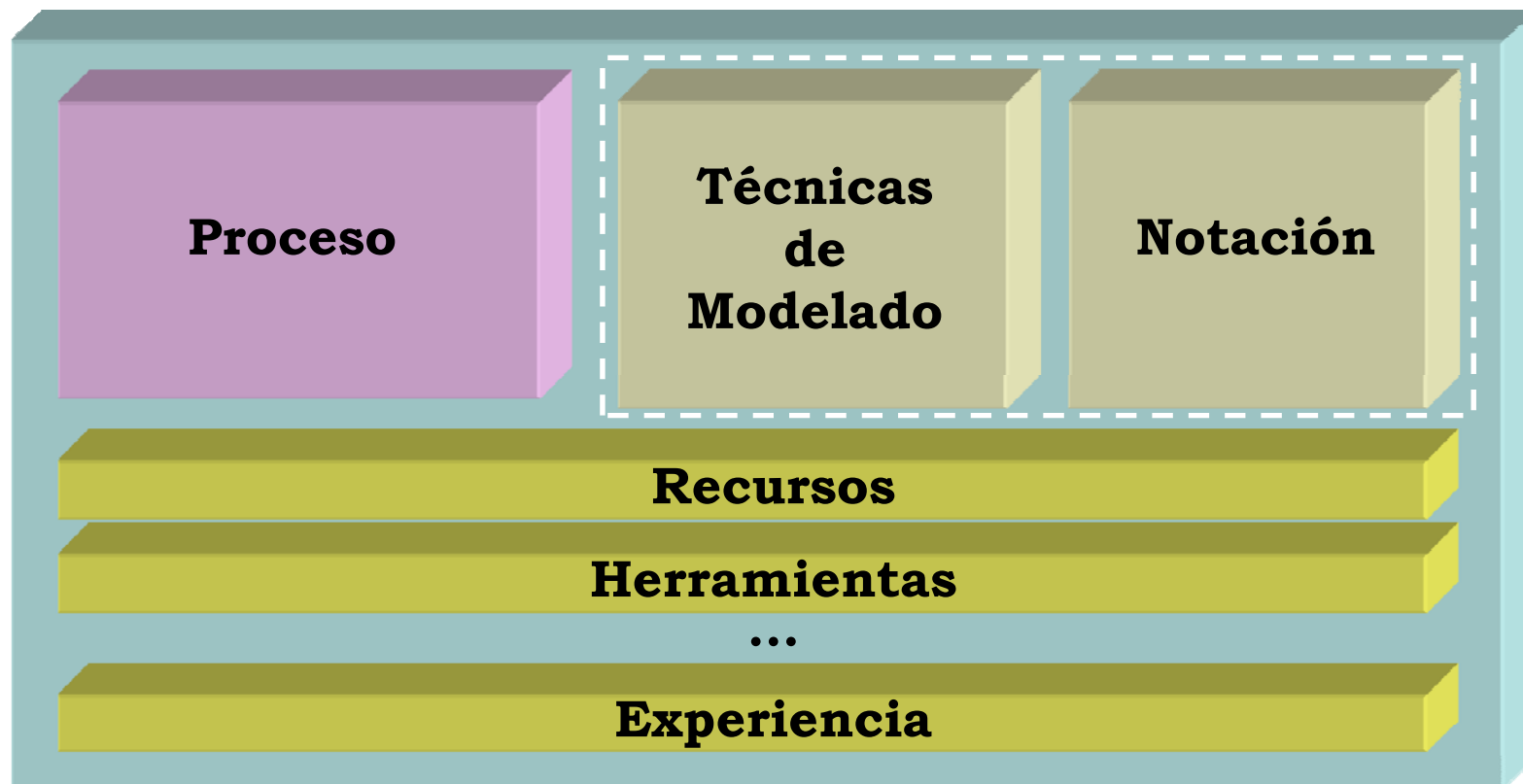
Ámbito de UML

- Se trata de un lenguaje de modelado de propósito general.
 - No vinculado a dominios, tareas o entornos de desarrollo específicos.
- Soporta todo el ciclo de vida de desarrollo del software.
 - Especificaciones de requisitos, análisis, arquitectura, diseño, implementación e implantación
- Soporta distintas áreas de aplicación.
 - Sistemas distribuidos, tiempo real, aplicaciones monoproceso...
 - Sistemas de Información Corporativos (*Management Information Systems*, MIS), Telecomunicaciones, Banca / Finanzas, Defensa / Espacio, Transporte, Distribución, Electro-medicina, Ciencia...



Ámbito de UML

- Metodologías de desarrollo de software Orientado a Objetos (OO)



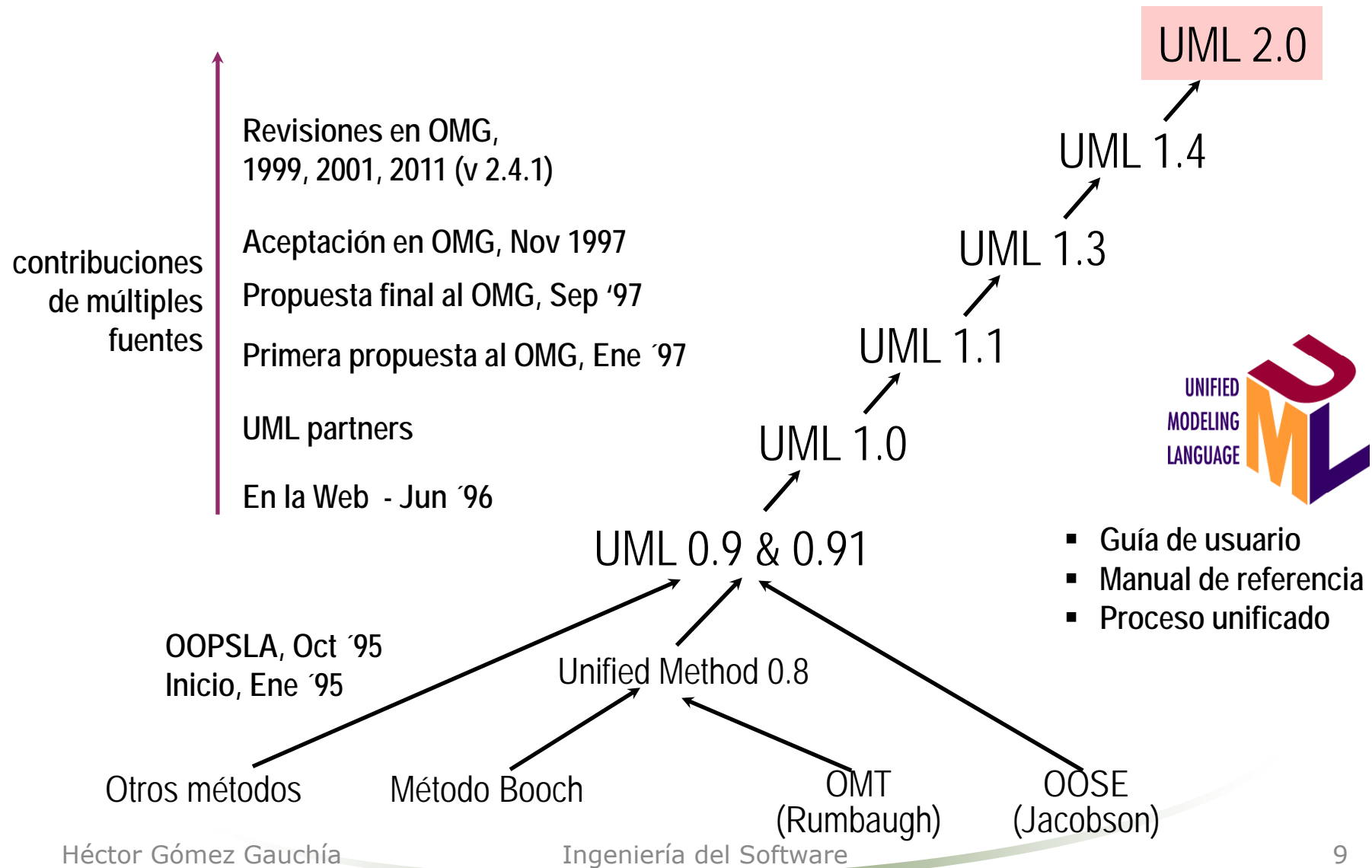


Objetivos de UML

- Definir un lenguaje de modelado visual fácil de aprender pero rico en significado.
- Estándar, estable y configurable.
 - Unificar las metodologías de análisis y diseño OO más conocidas.
 - Notación Booch de G. Booch
 - Técnica de Modelado de Objetos (*Object Modeling Technique*, OMT) de J. Rumbaugh et al.
 - Ingeniería del Software Orientada a Objetos (*Object-Oriented Software Engineering*, Objectory) de I. Jacobson
 - E incluir ideas de otros lenguajes de modelado.
- Ser independiente de lenguajes de programación o procesos particulares.
- Soportar conceptos de desarrollo de alto nivel.
 - Ej. colaboraciones, *frameworks*, patrones y componentes.
- Tratar aspectos del desarrollo de software actual.
 - Ej. escalabilidad, concurrencia, distribución, ejecutabilidad...



Evolución de UML





Contribuciones a la definición de UML (OMG)

Aonix
Colorado State University
Computer Associates
Concept Five
Data Access
EDS
Enea Data
Hewlett-Packard
IBM
I-Logix
InLine Software
Intellicorp
Kabira Technologies
Klasse Objecten
Lockheed Martin

Microsoft
ObjecTime
Oracle
Ptech
OAO Technology Solutions
Rational Software
Reich
SAP
Softeam
Sterling Software
Sun
Taskon
Telelogic
Unisys
...





PERSPECTIVA GENERAL DE LA NOTACIÓN



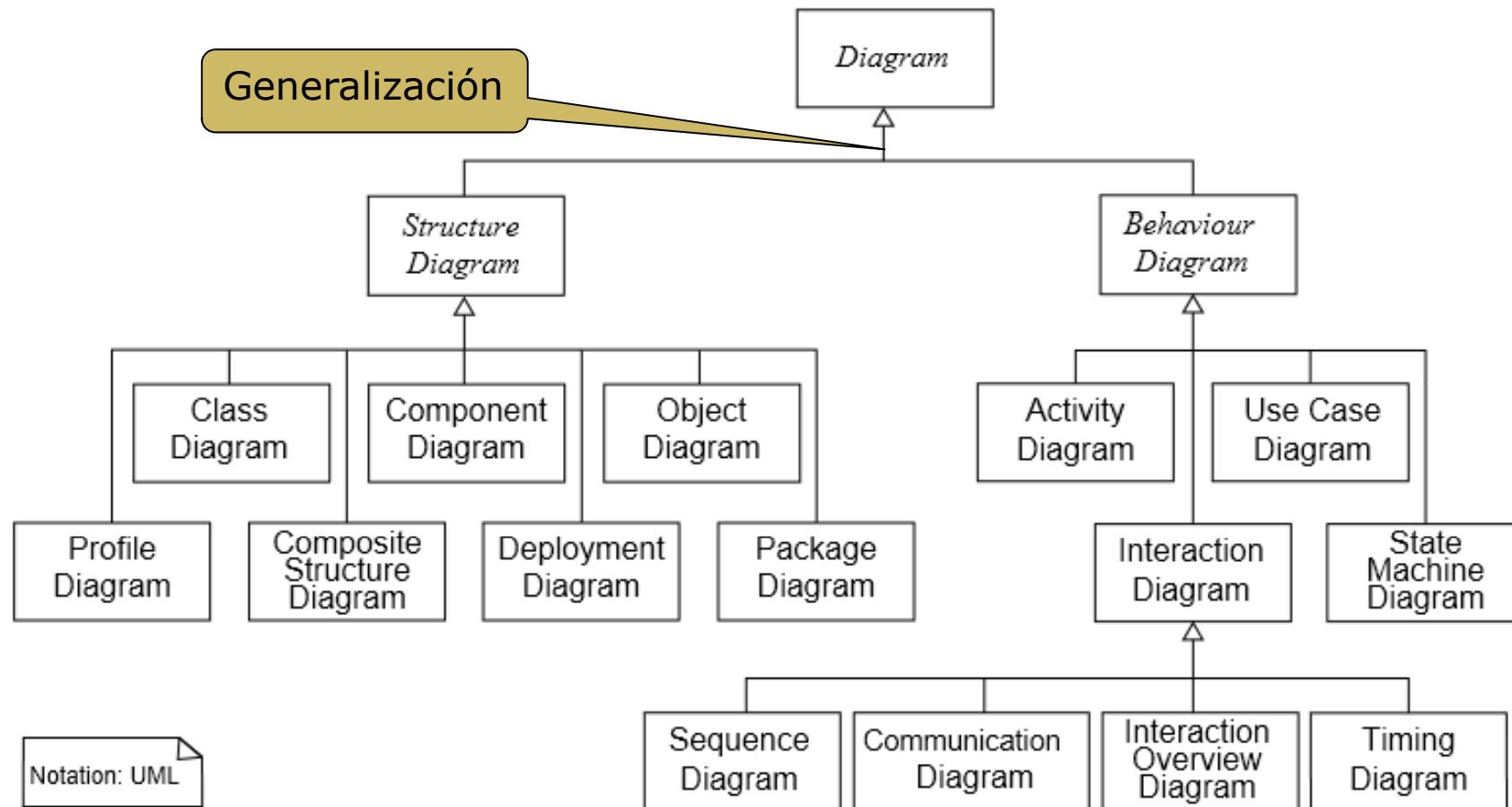


Lenguajes basados en grafos

- Los lenguajes de modelado *basados en grafos* incluyen los siguientes elementos básicos:
 - Nodos / entidades
 - Representan los conceptos del dominio
 - Arcos / relaciones
 - Representan las asociaciones de cualquier tipo entre nodos
 - Ej. relaciones todo-parte, semántica o requisito-implementación
- y opcionales:
 - Extremos de los arcos / roles
 - Caracterizan cada uno de los extremos de una relación
 - Propiedades
 - Atributos y valores de cualquiera de los elementos anteriores
 - Grafos / diagramas
 - Agrupaciones de los elementos anteriores
- UML es un lenguaje de modelado basado en grafos que incluye todos los elementos anteriores.



Diagramas de la notación



Fuente: http://en.wikipedia.org/wiki/Unified_Modeling_Language





Diagramas de la notación

Vistas	Estructural	Dinámica
Lógica	Diagrama de Casos de uso Diagrama de Clases Diagrama de Componentes Diagrama de Estructura compuesta Diagrama de Objetos	Diagrama de Actividades Diagrama de Secuencia Diagrama de Comunicación (UML-1.x → Diagramas de Colaboración) Diagrama de Máquina de Estados Diagrama de Tiempo
Física	Diagrama de Despliegue	

- UML se encuentra actualmente en la versión 2.4.1 (2011) [OMG, 2011a; OMG, 2011b].
 - Hay diferencias sustanciales entre las versiones 1.x y 2.x.



Modelos

- Casos de uso

- Clase

- Objetos

- Estructura compuesta

- Secuencia

- Comunicación

- Máquina de estados

- Actividades

- Tiempo

- Componentes

- Despliegue

Modelo de casos de uso

Modelo estructural

Modelo de comportamiento

Modelo de implementación





Elementos de UML

- Diagramas
- Entidades
 - Estructurales
 - Comportamiento
 - Agrupamiento
- Relaciones
- Propiedades
- Elementos comunes
 - Generalmente, entidades asociables a cualquier otro elemento.

Globalmente, los tipos de entidades de UML se denominan "clasificadores", y sus elementos concretos "instancias".

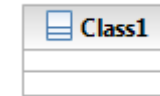


Ejemplo: entidades estructurales

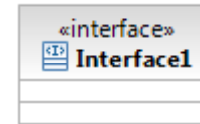
- Casos de uso



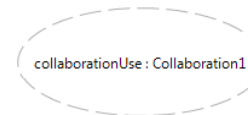
- Clases



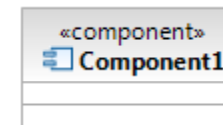
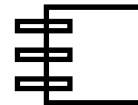
- Interfaces



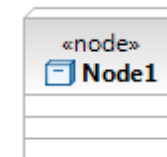
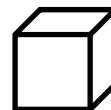
- Colaboraciones



- Componentes



- Nodos



Se pueden emplear representaciones icónicas o con estereotipos (nombres entre << y >>).

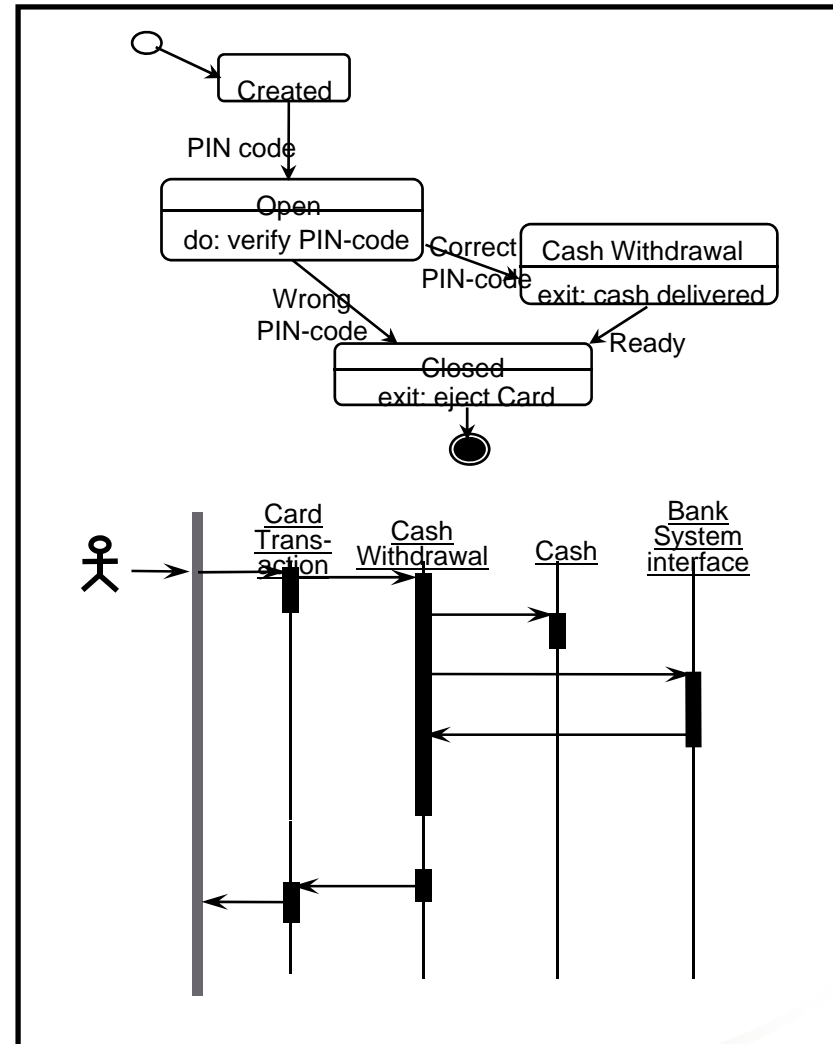
La columna izquierda muestra representaciones icónicas estándar, y la derecha con el icono al lado del nombre. El icono al lado del nombre se puede omitir siempre que figure el estereotipo.



Ejemplo: entidades de comportamiento

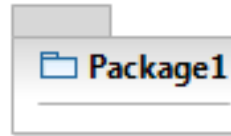
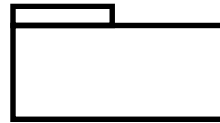
- Máquinas de estado
- Interacciones
 - Diagramas de secuencia o de comunicación

Los diagramas también se manipulan como parte de UML. Ej. agrupación en paquetes



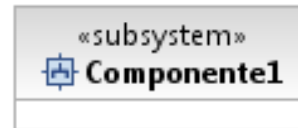
Ejemplo: entidades de agrupamiento

- Paquete






- Modelo

- Subsistema





Ejemplo: relaciones

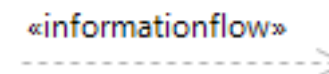
- Dependencia 
- Asociación 
- Generalización 





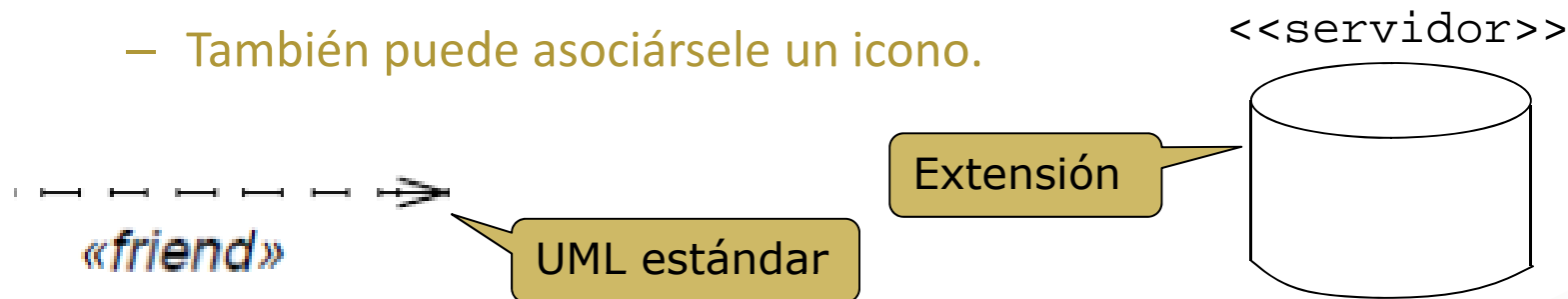
Elementos comunes

- Comentarios
 - Texto libre asociado a cualquier elemento.
- Restricciones
 - Representan condiciones que han de cumplir los elementos asociados.
 - Pueden representarse dentro de comentarios o asociados directamente al elemento.
- Flujo de información
 - Paso de datos o mensajes entre dos elementos.



Estereotipos

- Un *estereotipo* extiende el vocabulario de UML.
 - Se usa en la propia definición de UML.
- Los estereotipos permiten crear nuevas primitivas de modelado.
 - Las nuevas primitivas derivan de las existentes.
 - Pero son específicas para un problema.
- Se representan con el nombre del estereotipo encerrado entre comillas angulares « y » .
 - También puede asociársele un icono.



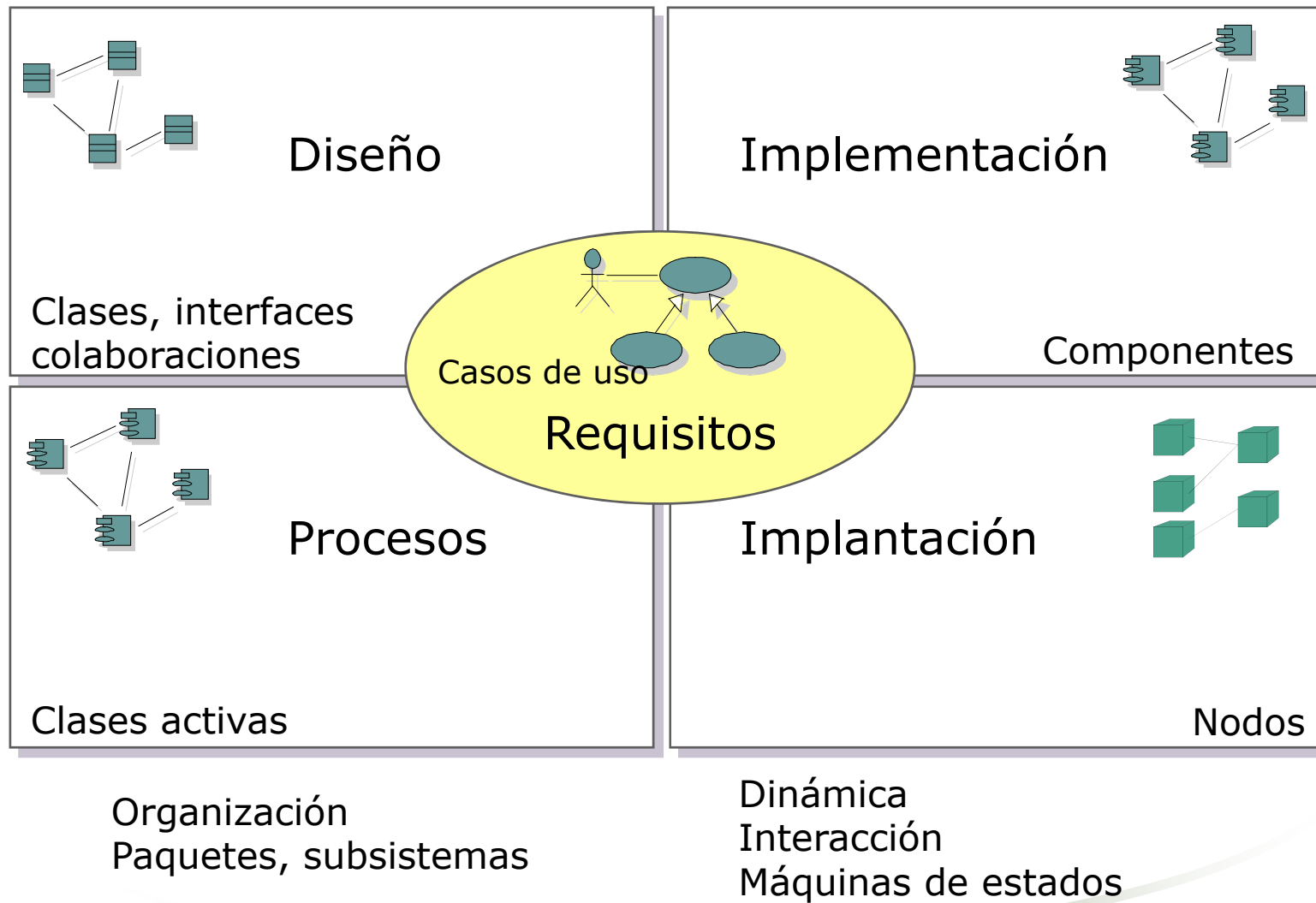


Perfiles

- Un *perfil* es una extensión del UML para un propósito específico.
 - Ej. dominio, plataforma o metodología.
- Define una serie de nuevos elementos del lenguaje:
 - Estereotipos para nuevas primitivas.
 - Nuevas entidades, relaciones, propiedades y restricciones de primitivas existentes.
 - Sintaxis para constructoras que no la tienen o cambios para otras que la tienen.
 - Ej. para acciones o cambiar el icono de los nodos de los diagramas de despliegue.



Arquitectura del sistema con UML





MODELO DE CASOS DE USO





Modelo de casos de uso

- Visión del sistema tal como se muestra a sus usuarios externos.
 - Lo desarrollan tanto los analistas como los expertos del dominio.
- Particiona la funcionalidad del sistema en:
 - transacciones (*casos de uso*)
 - que tienen significado para los usuarios (algunos de los *actores*)
- Se define mediante:
 - Diagramas de casos de uso
 - Descripción de los casos de uso con (según UML-2.4.1 [OMG, 2011b]):
 - Plantillas de texto → lenguaje natural estructurado
 - Pre y post-condiciones
 - Diagramas de actividades, interacción y máquinas de estado
 - *Colaboraciones*, con los actores y los casos de uso tipando sus partes





DIAGRAMA DE CASOS DE USO





Diagrama de casos de uso

- Un *diagrama de casos de uso* es un diagrama que muestra un conjunto de *casos de uso*, *actores* y sus relaciones.
- Está destinado a capturar los requisitos funcionales del sistema a alto nivel.
 - Qué funcionalidad se proporciona
 - Pero no cómo se proporciona
 - Sólo de forma limitada las relaciones entre las distintas funcionalidades



Diagrama de casos de uso: entidades

- Caso de uso
 - Conjunto de secuencias de acciones que ejecuta un sistema para producir un resultado observable de valor para un *actor*.
 - Incluye variantes de las secuencias de acciones.
- Actor
 - Conjunto coherente de roles que los usuarios de los *casos de uso* juegan al interactuar con estos.
 - Puede ser una persona, un dispositivo hardware o incluso otro sistema que interactúa con el sistema objetivo.
- Límite del sistema
 - Representa el límite entre el sistema físico y los actores que interactúan con el sistema.

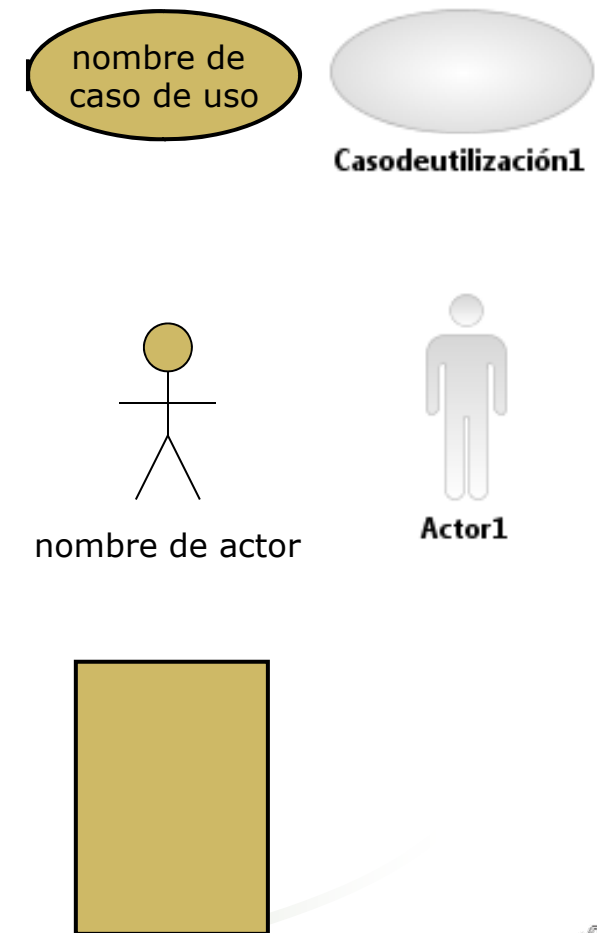


Diagrama de casos de uso: relaciones

- Asociación

- Participación de un actor en un caso de uso
 - La instancia de un actor se comunica con instancias de un caso de uso.



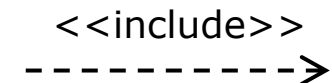
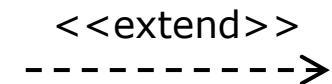
- Generalización

- Relación taxonómica entre un caso de uso más general y otro más específico

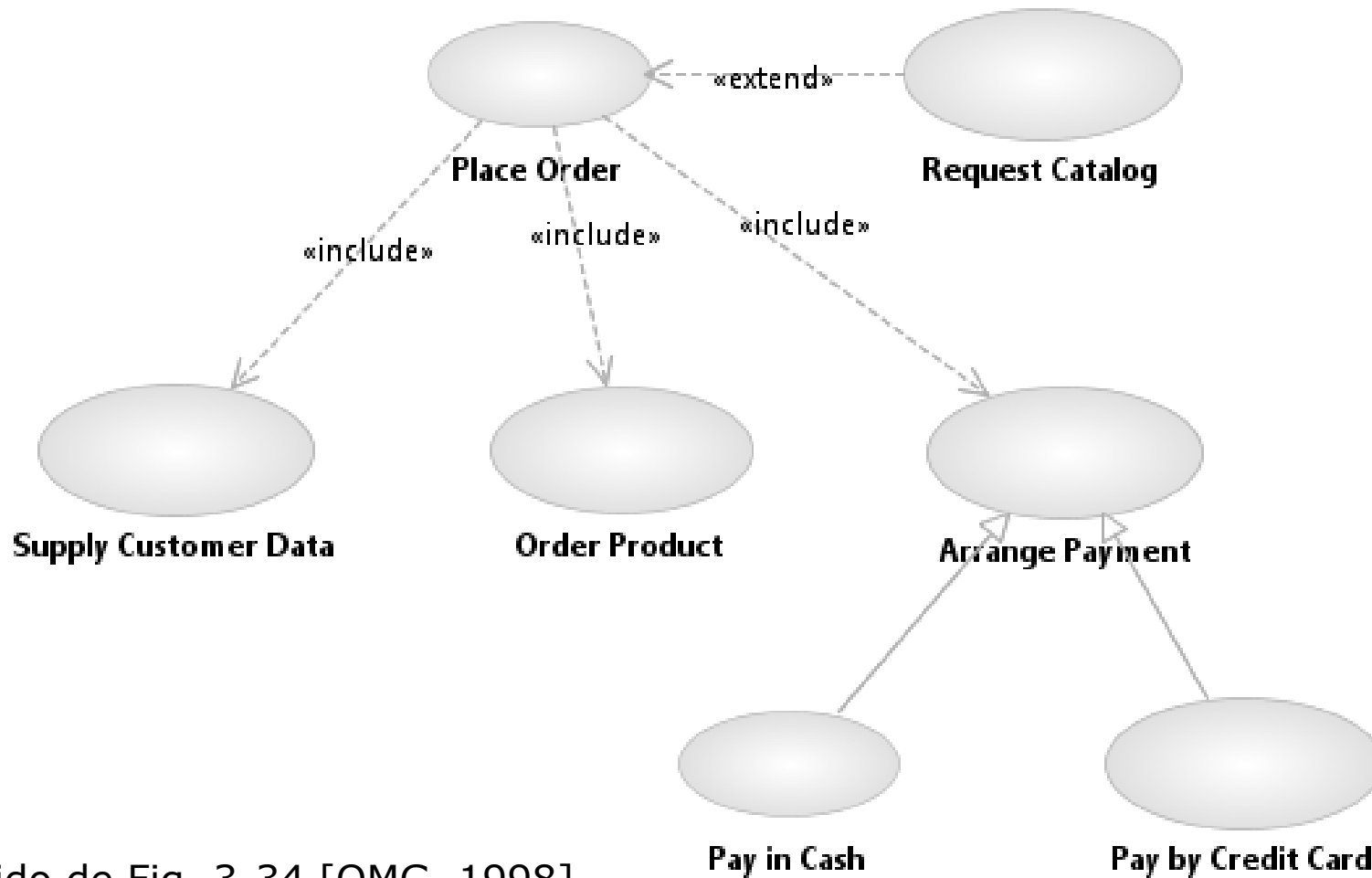


- Dependencia

- Extensión (opcionalidad)
 - El comportamiento del caso de uso puede extender el del caso de uso objetivo.
- Inclusión (obligatoriedad)
 - El comportamiento del caso de uso siempre incluye el del caso de uso objetivo.



Ejemplo: asociaciones de dependencia y generalización



Extendido de Fig. 3-34 [OMG, 1998]



Ejemplo: extensión

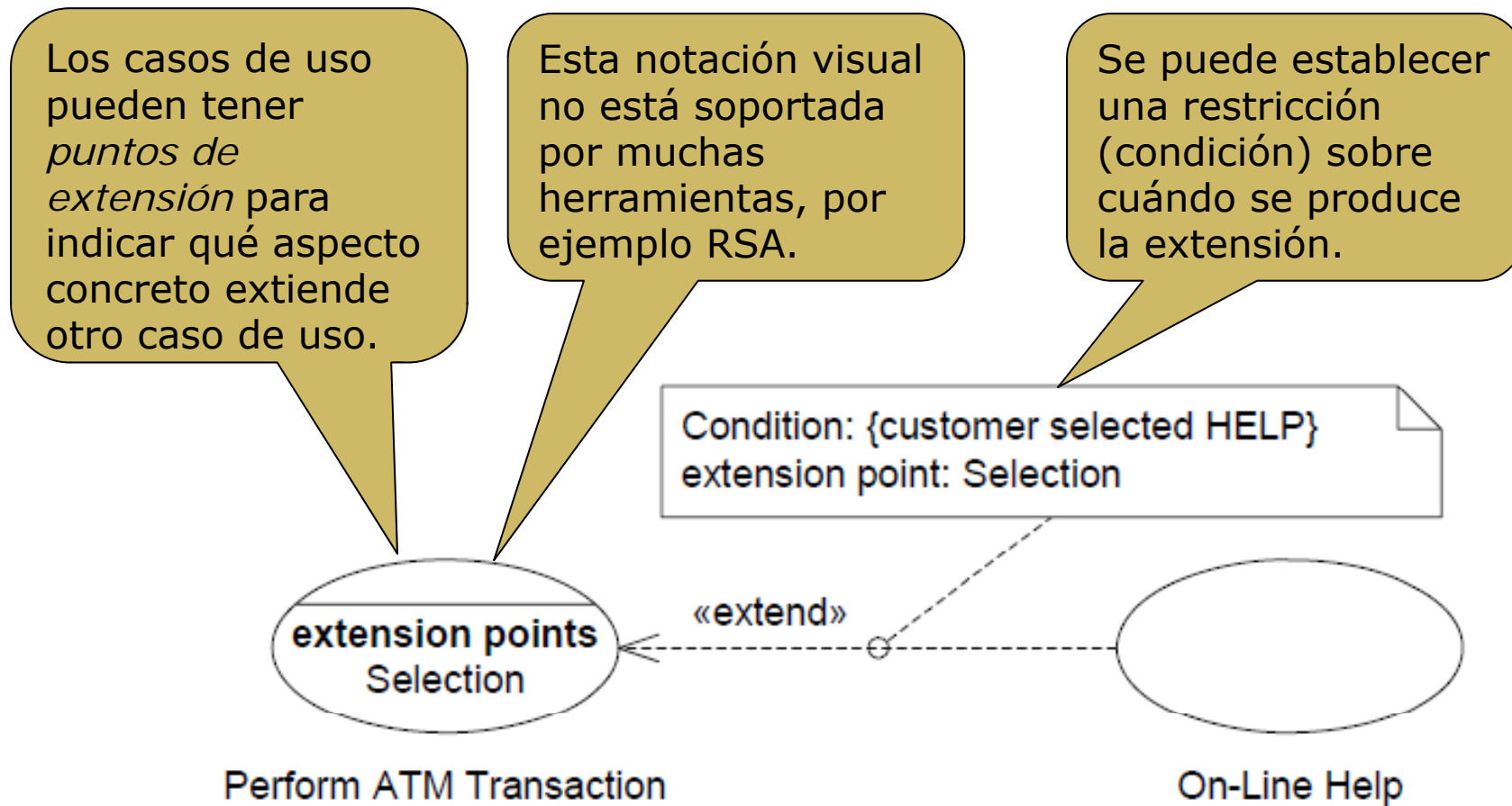


Fig. 16.3 [OMG, 2011b]

Ejemplo: límite del sistema

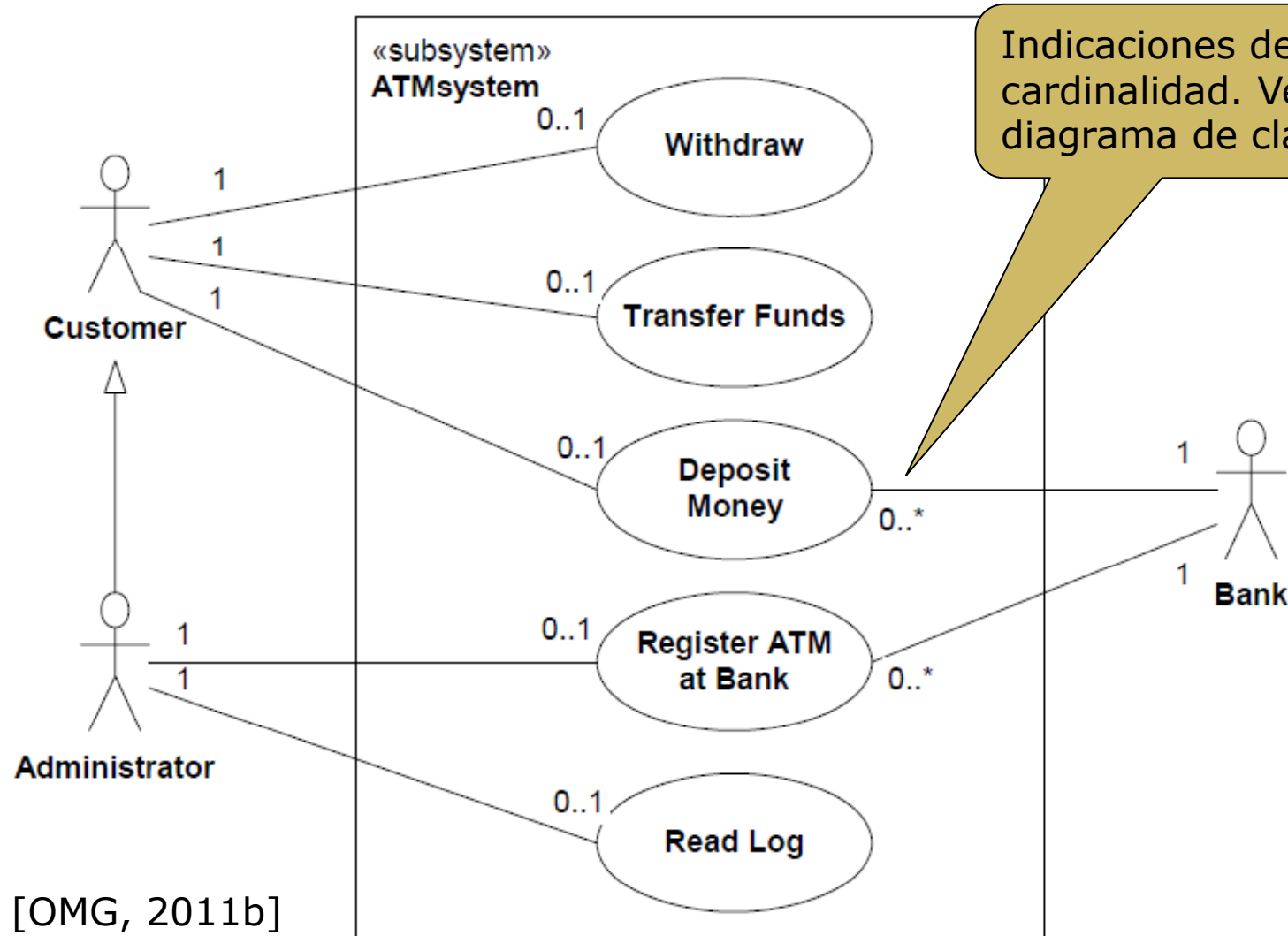


Fig. 16.5 [OMG, 2011b]



Descripción textual de casos de uso

- Generalmente se usa una plantilla:
 - Actores
 - Precondiciones y postcondiciones
 - Cómo y cuándo empieza y acaba el caso de uso respectivamente
 - Flujo principal (normal) y flujos alternativos
 - Cuándo y cómo interactúan el sistema y los actores
 - Acciones de los actores ↔ Respuesta del sistema
 - Distintos escenarios y posibles excepciones
 - Las excepciones se pueden clasificar, por ejemplo, por datos incompletos, datos erróneos, comportamiento alternativo...
- No obstante, el estándar [OMG, 2011b] aconseja igualmente otras representaciones textuales y diagramas.
 - Pre y post condiciones, diagramas de actividades, interacción y máquinas de estado, y *colaboraciones* con los actores y los casos de uso tipando sus partes.





Ejemplo: descripción textual de caso de uso

- Dar de alta libro
 - Flujo de eventos principal:
 1. Introducir título, autor, ISBN, número de ejemplares.
 2. Introducir en la BD el ejemplar.
 - 2.1. Si error de validación de los datos
 3. Generar signature.
 4. Generar código para cada ejemplar.
 5. Si quiere el usuario, imprimir etiquetas con signature y código de ejemplar para cada ejemplar.
 - Flujo de eventos alternativo:
 - En cualquier momento se puede cancelar la operación.
 - 2.1. Si algún dato no es correcto se comunica al usuario.
 - Precondición: El bibliotecario tiene autorización.
 - Postcondición: No hay dos ejemplares distintos con el mismo identificador.



Descripción con diagrama de máquina de estados de casos de uso

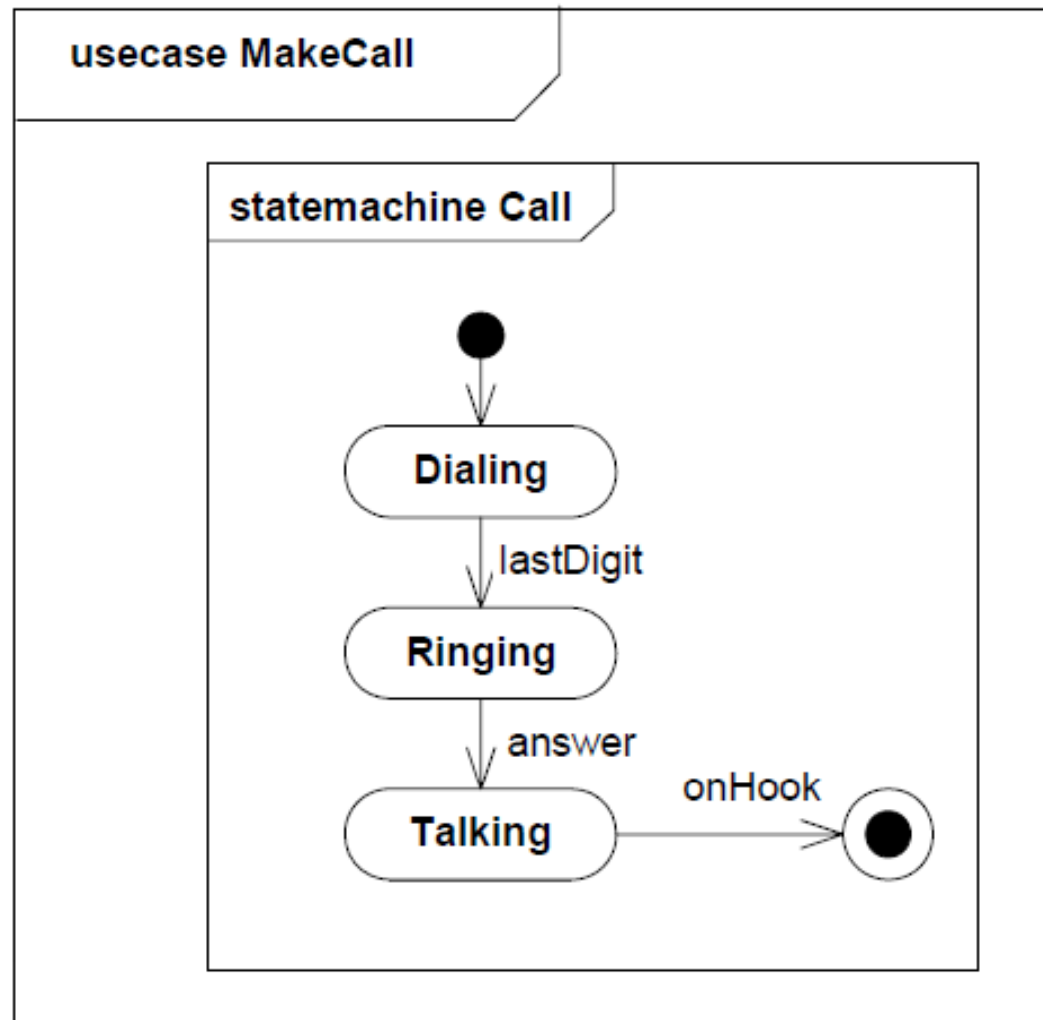


Fig. 16.6 [OMG, 2011b]





Cómo identificar casos de uso

- A partir de los actores
 - Identificar los actores relacionados con el sistema o la organización.
 - Para cada actor, identificar los procesos que inicia o en los que participa.
- A partir de los eventos
 - Identificar los eventos externos a los que puede responder el sistema.
 - Relacionar los eventos con actores y casos de uso.





Consejos para un buen modelo de casos de uso

- Asegurarse de que cada caso de uso describe una parte significativa del funcionamiento del sistema.
- Evitar un número excesivo de casos de uso.
 - Un caso de uso no es un paso, operación o actividad individual en un proceso.
 - Un caso de uso describe un proceso completo que incluye varios pasos.
- Los casos de uso tienen que ser entendibles tanto por desarrolladores software como por expertos del dominio.
 - Es una descripción de alto nivel del sistema.
 - Evitar conceptos de diseño.





Ejercicio

- Definir el modelo de casos de uso para un sitio web de reservas de hotel.
 - Identificar actores
 - Realizar los diagramas de casos de uso
 - Describir los casos de uso





MODELO ESTRUCTURAL





Modelo estructural

- Visión del sistema que describe la estructura de los objetos, incluyendo su clasificación, relaciones, atributos y operaciones.
 - Desarrollado por analistas, diseñadores y programadores
- Muestra la estructura estática del sistema.
 - Las entidades que existen → vocabulario del sistema
 - Ej. clases, interfaces, componentes y nodos.
 - Su estructura interna
 - Su relación con otras entidades
- Se define mediante diagramas estructurales estáticos.
 - Diagrama de clases
 - Diagrama de objetos
 - Diagrama de estructura compuesta





Diagramas estructurales estáticos

- Muestran un grafo de elementos clasificados (con tipo) conectados por relaciones estáticas.
- Tres tipos:
 - Diagrama de clases
 - Proporciona una visión de clasificación de las entidades del sistema.
 - Qué tipos de elementos existen.
 - Cómo se relacionan los tipos de elementos.
 - Diagrama de objetos
 - Proporciona una visión de instanciación del sistema.
 - Qué elementos / objetos existen.
 - De qué tipos / clases son.
 - Cómo se relacionan los elementos / objetos.
 - Diagrama de estructura compuesta
 - Describe la estructura interna de una clase.
 - Qué tipos de elementos existen.
 - Cómo se relacionan los tipos de elementos.





DIAGRAMA DE CLASES





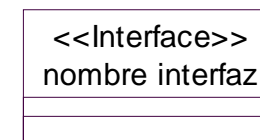
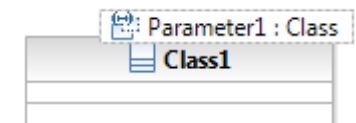
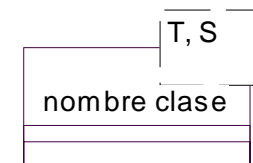
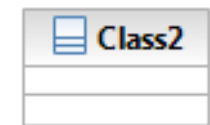
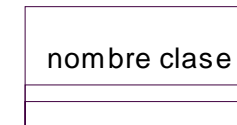
Diagrama de clases

- Un *diagrama de clases* es un diagrama estructural estático que muestra un conjunto de *clases*, *interfaces* y sus relaciones.
- Proporciona una visión de clasificación de las entidades del sistema.
 - Qué tipos de elementos existen.
 - Qué relaciones estáticas asocian a estos tipos de elementos.



Diagrama de clases: entidades

- Clase
 - Descripción de un conjunto de objetos que comparten los mismos atributos, operaciones y semántica.
- Clase plantilla
 - Clase parametrizada que necesita ser instanciada antes de utilizarla.
- Interfaz
 - Colección coherente de características y obligaciones que se usa para especificar un servicio de un clasificador.
 - Puede incluir operaciones, miembros de datos, restricciones, pre y post-condiciones y protocolos.



Esta definición es muy similar a la de clase abstracta, por lo que muchas veces se restringe su especificación a operaciones y restricciones.



Diagrama de clases: relaciones (1/2)

- Generalización
 - Relación entre un clasificador general (superclase / padre) y un tipo más específico de ese clasificador (subclase / hijo).
 - Se suele aplicar a clases e interfaces.
- Asociación
 - Relación estructural que especifica que las instancias de un clasificador se conectan a instancias de otro.
 - Dada una asociación, se puede navegar desde una instancia de un clasificador hasta una instancia del otro y viceversa.
 - Puede haber restricciones (ver navegabilidad).
 - Se suele aplicar a clases e interfaces.
- Agregación
 - Forma de asociación que especifica una relación todo-parte entre un agregado (el todo) y las partes que lo componen.



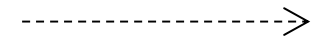
Muchas relaciones se definen como especializaciones de la *asociación*. Por tanto, lo que se aplica a ella se aplica a las demás.



Diagrama de clases: relaciones (2/2)

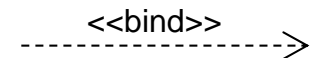
- Dependencia

- Relación de uso, la cual determina que un cambio en la especificación de un elemento (el proveedor) puede afectar a otro elemento que lo utiliza (el cliente), pero no necesariamente a la inversa.
- Se suele aplicar a clases e interfaces.



- Instanciación

- Denota el proceso por el cual se instancia una clase plantilla.

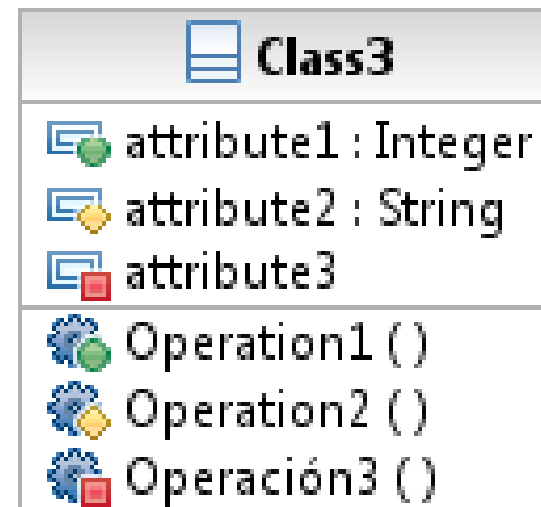
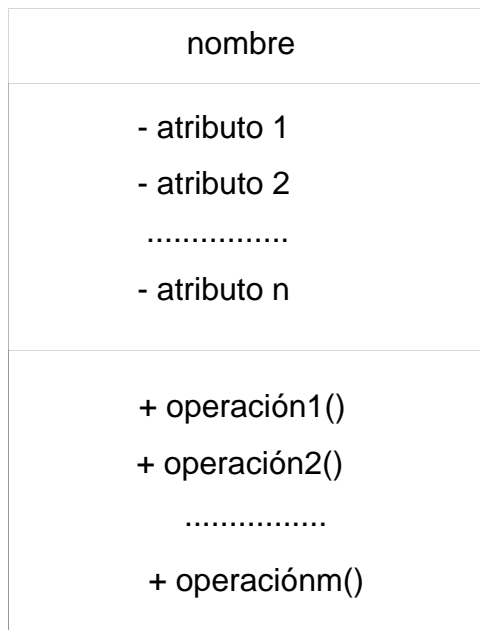


- Realización

- Relación entre una especificación y su implementación.
- De esta forma la implementación se compromete a cumplir las responsabilidades de los contratos implementados.
- Se suele aplicar entre clases e interfaces, pero también para indicar, por ejemplo, refinamientos en modelos, transformaciones u optimizaciones.



Clase



Iconos de clases con atributos, métodos y visibilidad





Adornos

- Visibilidad de clases, interfaces, atributos y operaciones
 - *Public* → +
 - *Protected* → #
 - *Private* → –
- Alcance de atributos
 - Indica si un atributo es *clave*.
 - Identifica a su objeto de forma unívoca.
 - Los atributos clave se subrayan.

RSA lo representa
con iconos propios



Ejemplo: clase

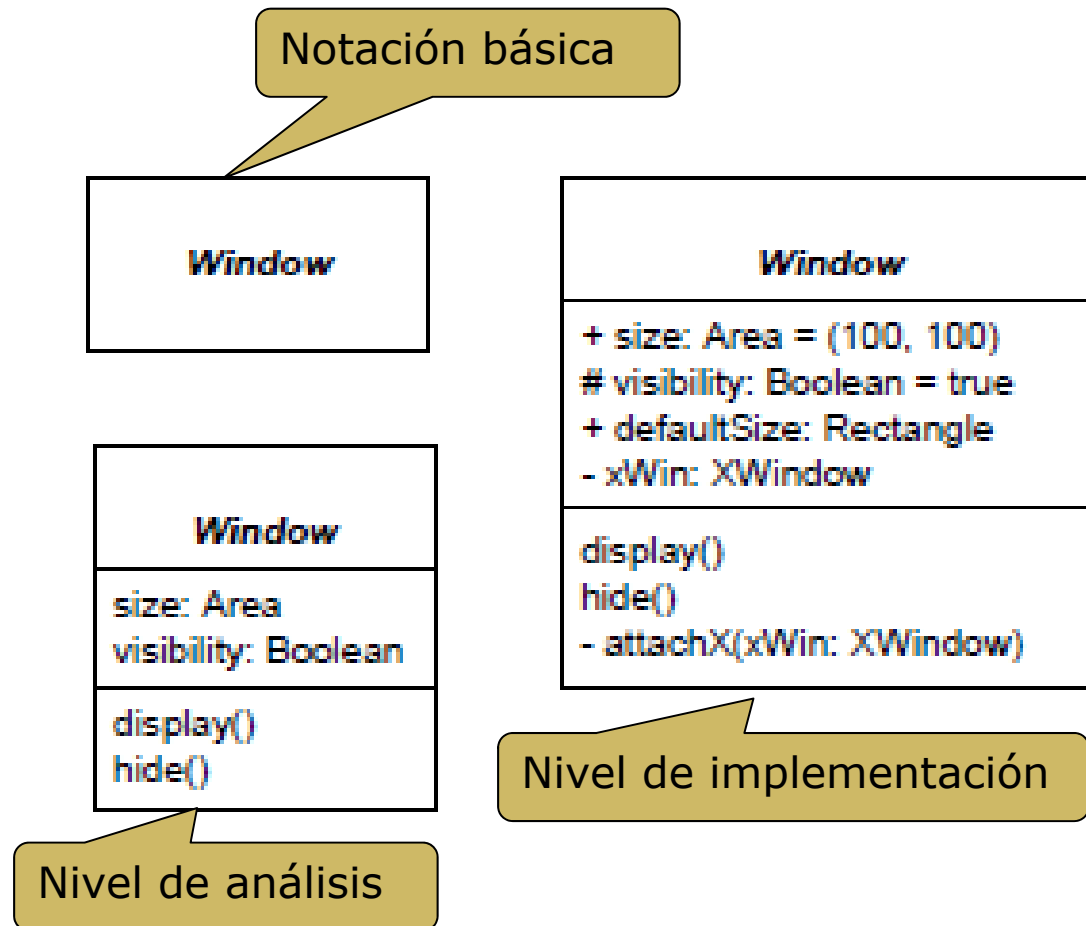
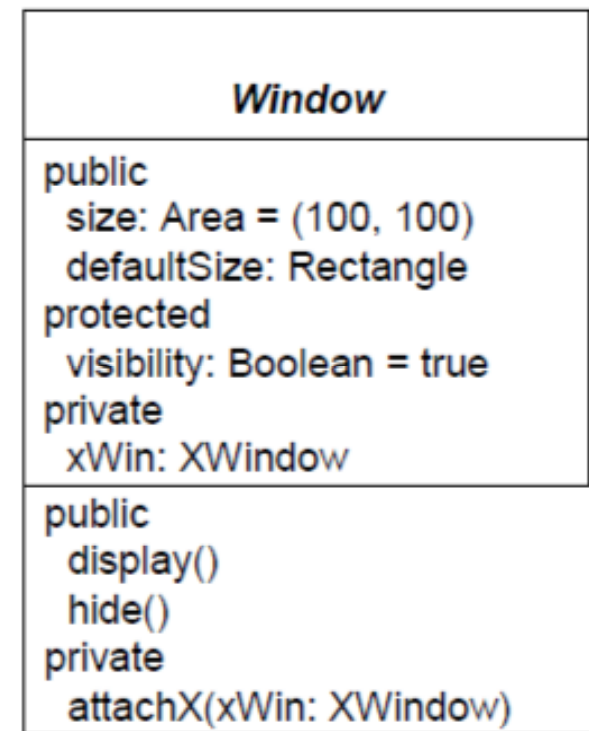


Fig. 7.28 [OMG, 2011b]



Atributos y métodos agrupados por visibilidad

Fig. 7.29 [OMG, 2011b]

Clase: métodos

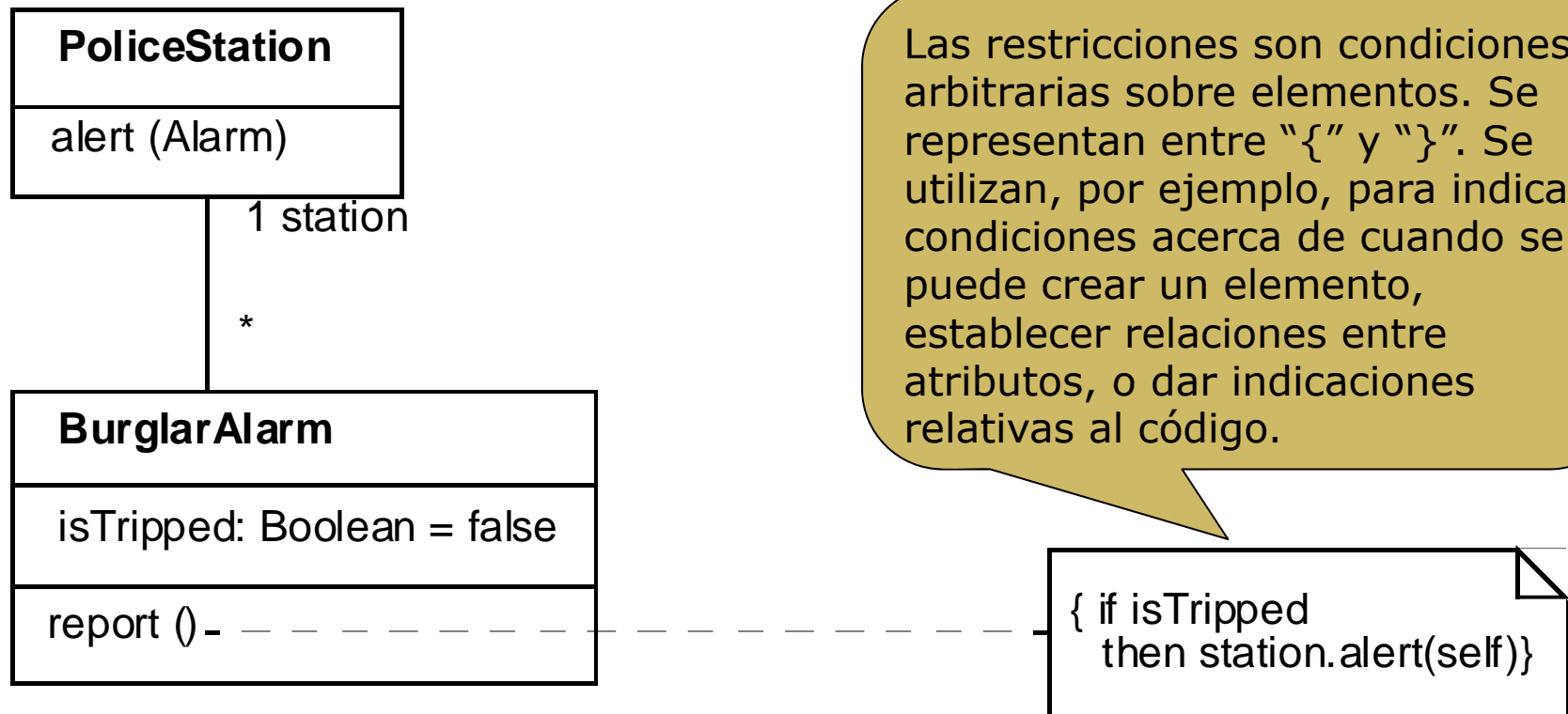


Fig. 3-24, *UML Notation Guide*



Atributos derivados y asociaciones

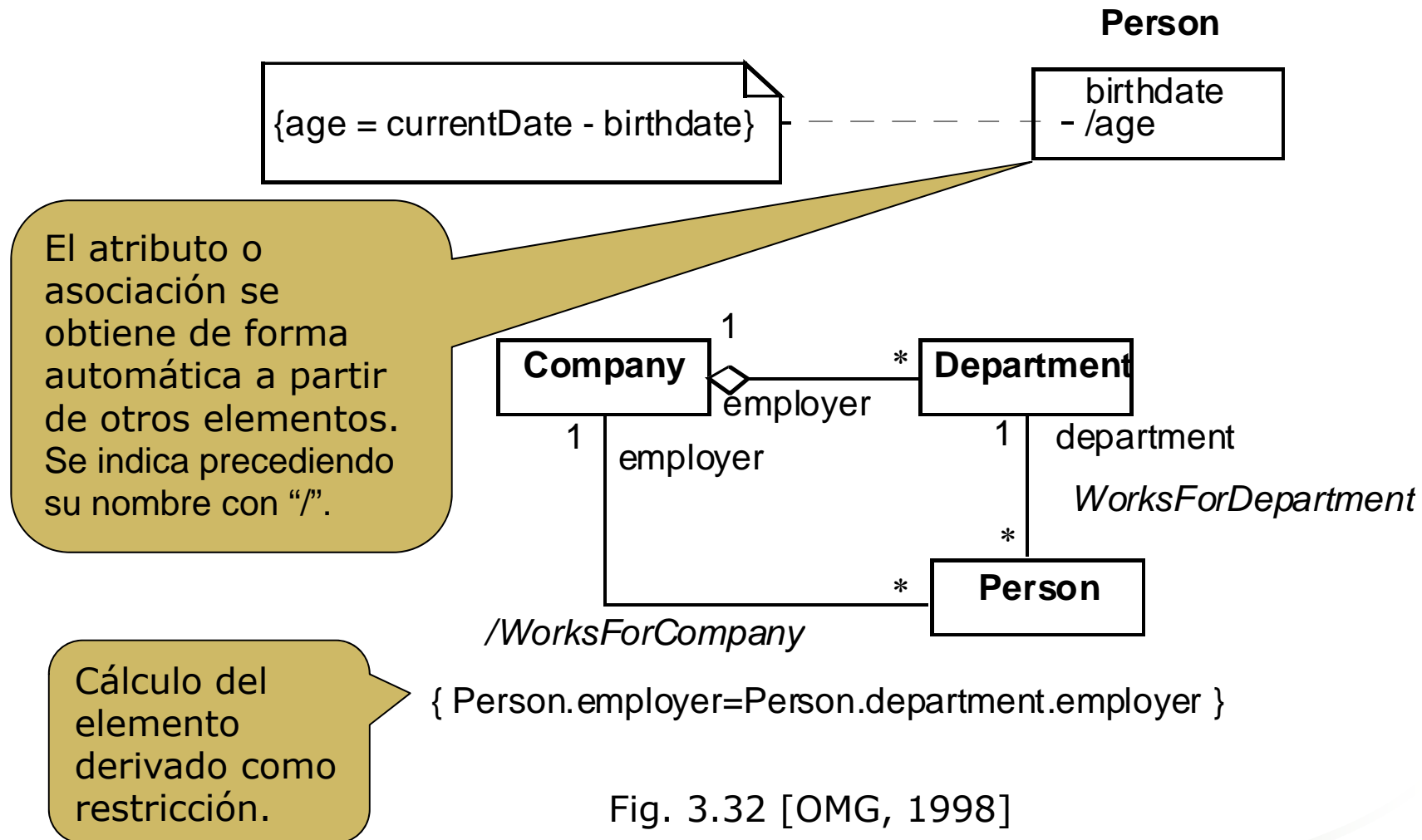


Fig. 3.32 [OMG, 1998]



Interfaz

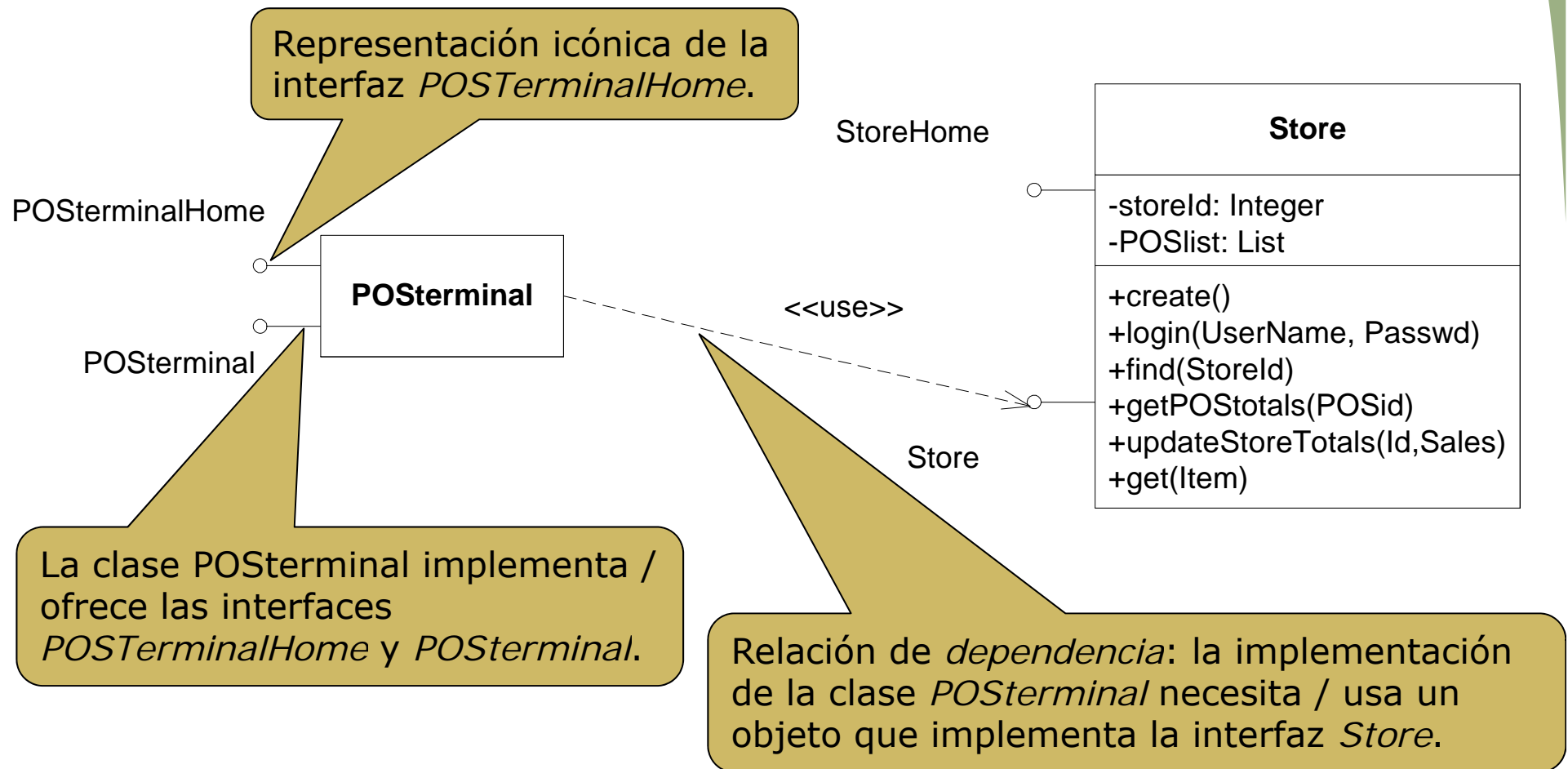


Fig. 3-29, UML Notation Guide



Asociación

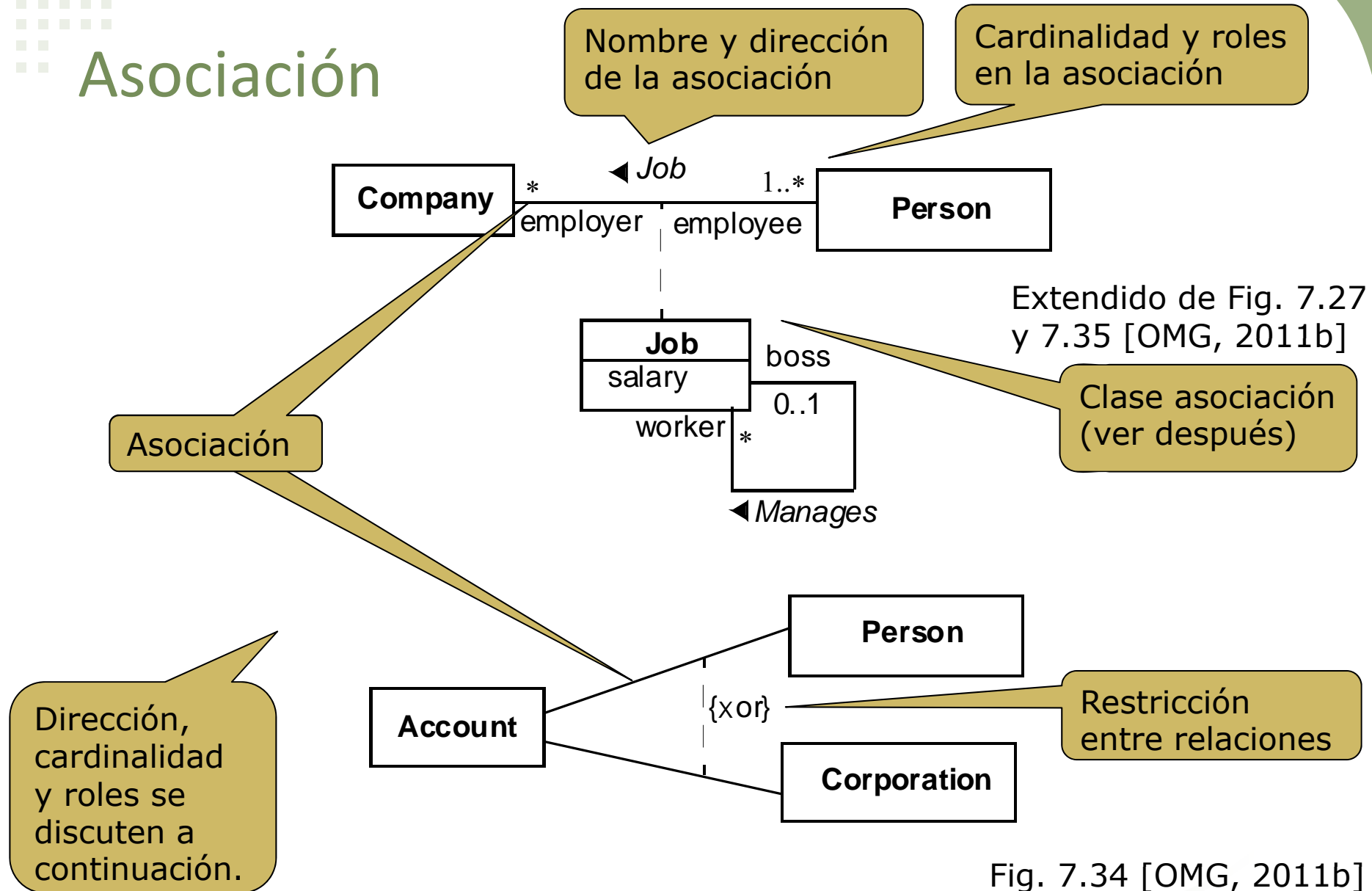


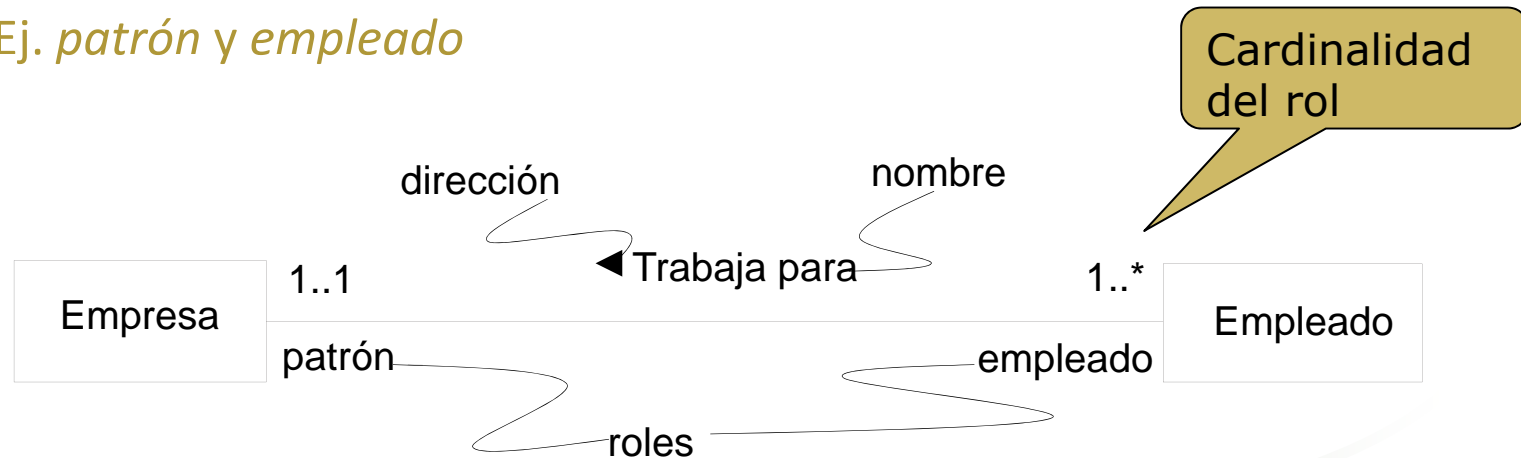
Fig. 7.34 [OMG, 2011b]



Adornos de relación: nombre, dirección y rol

- El *nombre* es la identificación de la relación.
 - Ej. *Trabaja para*
- La *dirección* indica el sentido de “lectura” de la relación.
 - Ej. el *Empleado Trabaja para* la *Empresa*
- El *rol* o *extremo de asociación* es el papel que juega cada extremo en la relación.
 - Ej. *patrón* y *empleado*

Se aplica a cualquier relación, no sólo a las asociaciones.



Adornos de relación: multiplicidad

- La *multiplicidad* o *cardinalidad* asignada a un extremo de relación declara el número de instancias que pueden satisfacer el puesto definido por ese extremo.
- Puede ser un valor o un rango.
 - Ej. 1, 5, * (0 o más), n (0 o más), N (0 o más), *Mín..Máx* (con $Mín \leq Máx$)
 - Ej.



Se aplica a todas las relaciones, menos a las generalizaciones.

A nivel asociación prohíbe tuplas de la forma:

(Caja Madrid, Lavapiés 8) y *(Banesto, Lavapiés 8)*

porque una *Sucursal* sólo se puede relacionar con un *Banco*.

Pero sí admite tuplas de la forma:

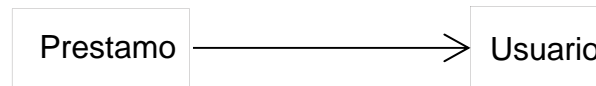
(Caja Madrid, Lavapiés 8) y *(Caja Madrid, Atocha 62)*

porque un *Banco* se puede relacionar con una o más *Sucursales*.



Adornos de asociación: navegabilidad

- Las *asociaciones* pueden o no ser navegables en cada dirección.
 - Una asociación *navegada* de la clase A a la B indica que se puede llegar de manera directa de la clase A a la B, pero no de B a A, aunque sí es posible que se pueda llegar de B a A de manera indirecta.
- La navegabilidad se indica con los correspondientes adornos en los extremos de la asociación.
 - “>” indica navegable en esa dirección y
 - “X” indica no navegable en esa dirección.
 - Ej.



De *Préstamo* se puede llegar directamente a *Usuario*, es decir, todo objeto de la clase *Préstamo* referencia a su objeto de la clase *Usuario*.

También se aplica en otras relaciones como las agregaciones.



Ejemplo: navegabilidad

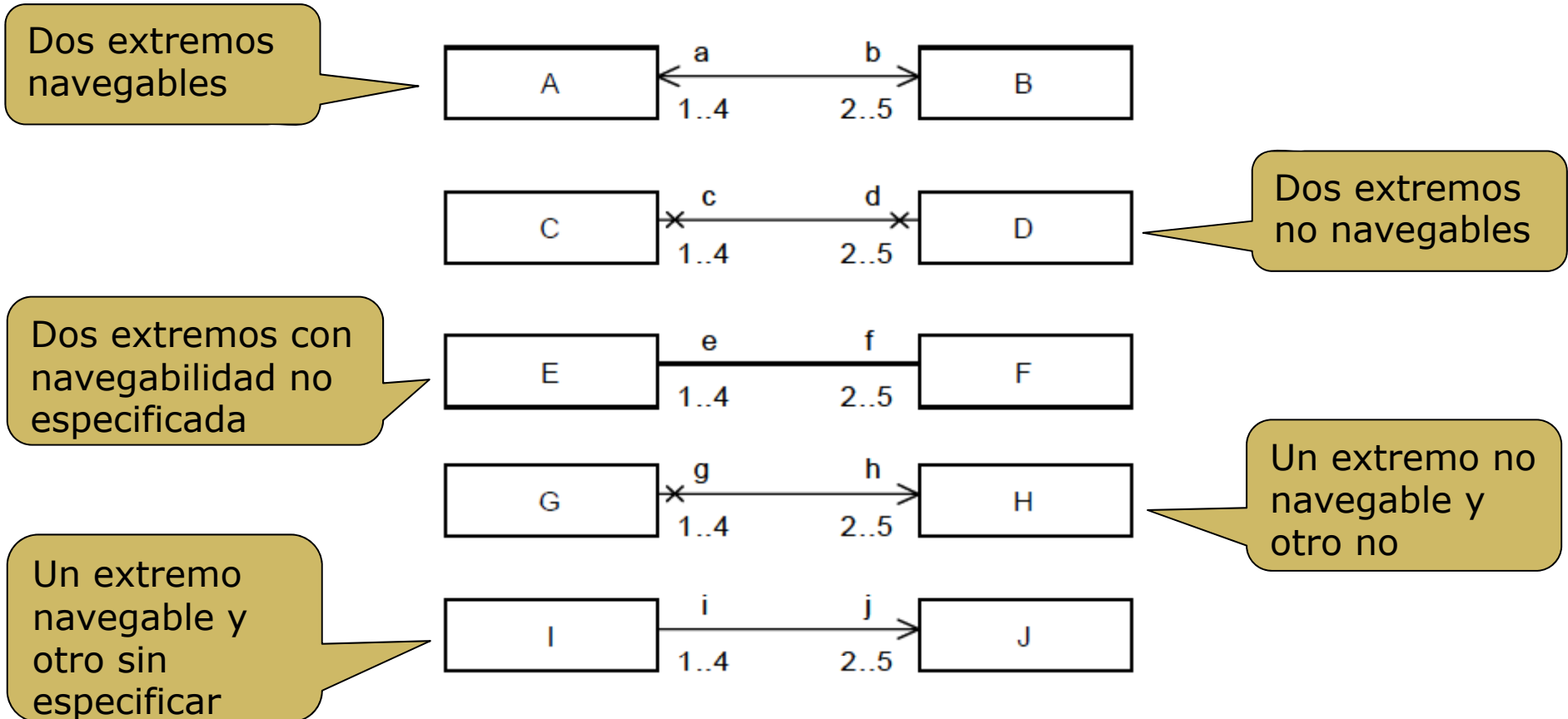
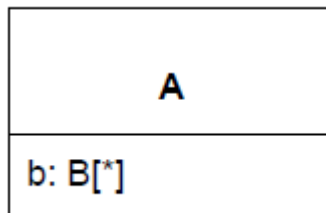


Fig. 7.23 [OMG, 2011b]

Asociación: representaciones alternativas

- Las asociaciones admiten dos representaciones equivalentes en UML.



Asociación n-aria

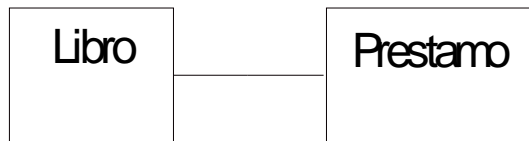


Fig. 3-44, *UML Notation Guide*

Extendido de Fig. 7.21 [OMG, 2011b]

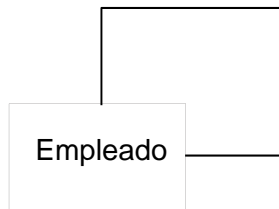


Asociación: implementación



```
class Libro {
    public Prestamo p;
};
```

```
class Prestamo {
    public Libro l;
};
```



```
class Empleado {
    public Empleado e;
};
```

```
class Sucursal {
    public Banco b;
};
```

```
class Banco {
    // de Sucursal
    public Vector sucursales = new Vector();
};
```



Clase asociación

- Una *clase asociación* representa propiedades intrínsecas a la propia asociación.
 - Ej. el *sueldo* (*salary*) no es función de la *Empresa* (*Company*) o el *Empleado* (*Person*) individualmente, sino del *Empleo* que un cierto *Empleado* tiene en una *Empresa*.

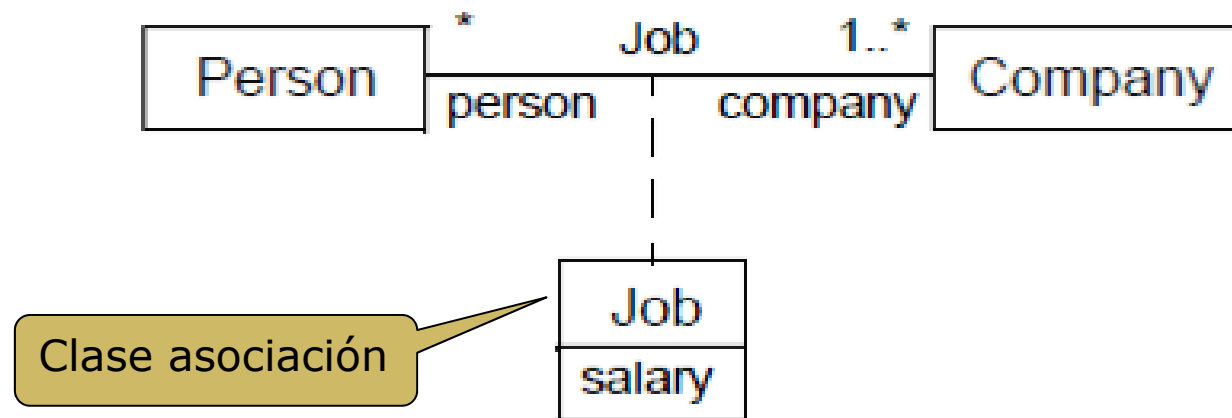


Fig. 7.27 [OMG, 2011b]





Agregación

- La *agregación* es una forma de *asociación* que especifica una relación todo-parte entre un agregado (el todo) y las partes que lo componen.

– Ej.



```
class Biblioteca {
    public Ejemplares ejs;
};
```

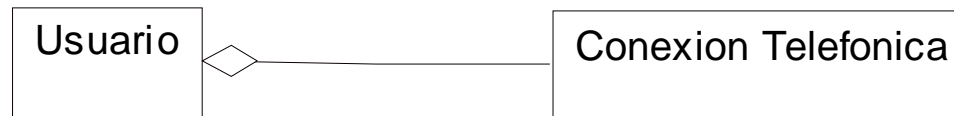
- La diferencia entre *asociación* con dirección y la *agregación* es semántica.
 - A nivel de código son iguales.



Agregación y composición

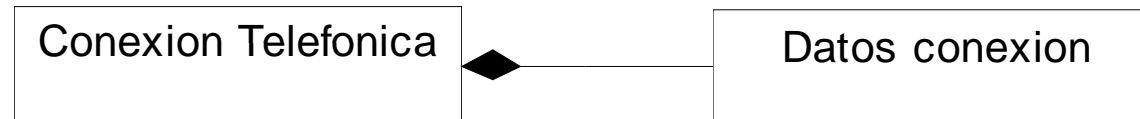
- La agregación puede denotar una vinculación entre la vida del agregado y sus componentes:
 - Si no es así, *agregación simple o agregación*.

- Ej.



- En caso contrario tenemos *agregación compuesta o composición*.

- Ej.



- En la práctica esta característica distingue entre la contención por referencia o por valor.
 - En lenguajes como Java no tiene mucho sentido esta distinción.



Extremos de agregación

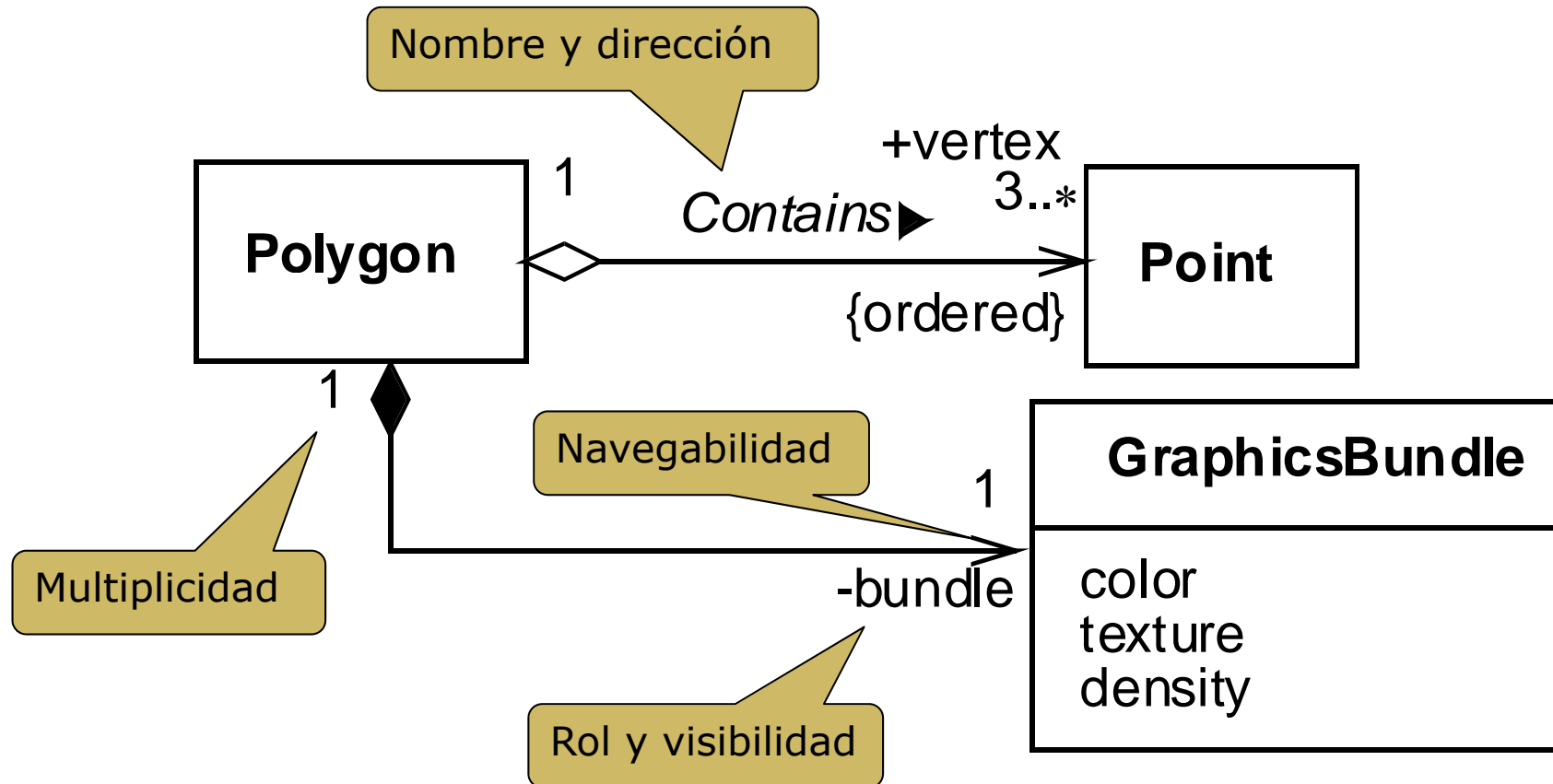


Fig. 3.21 [OMG, 1998]



Composición: representaciones alternativas

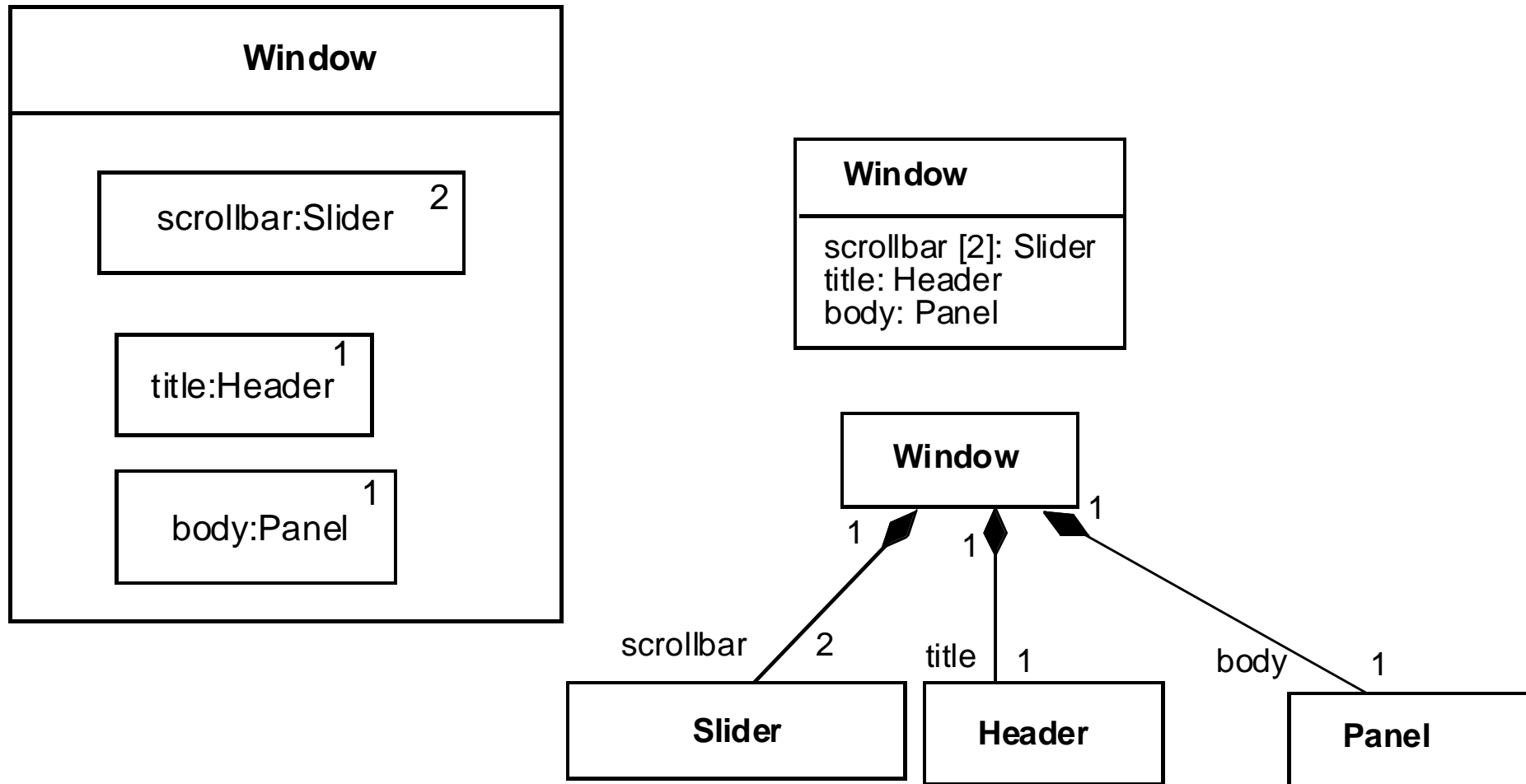


Fig. 3.25 [OMG, 1998]



Generalización

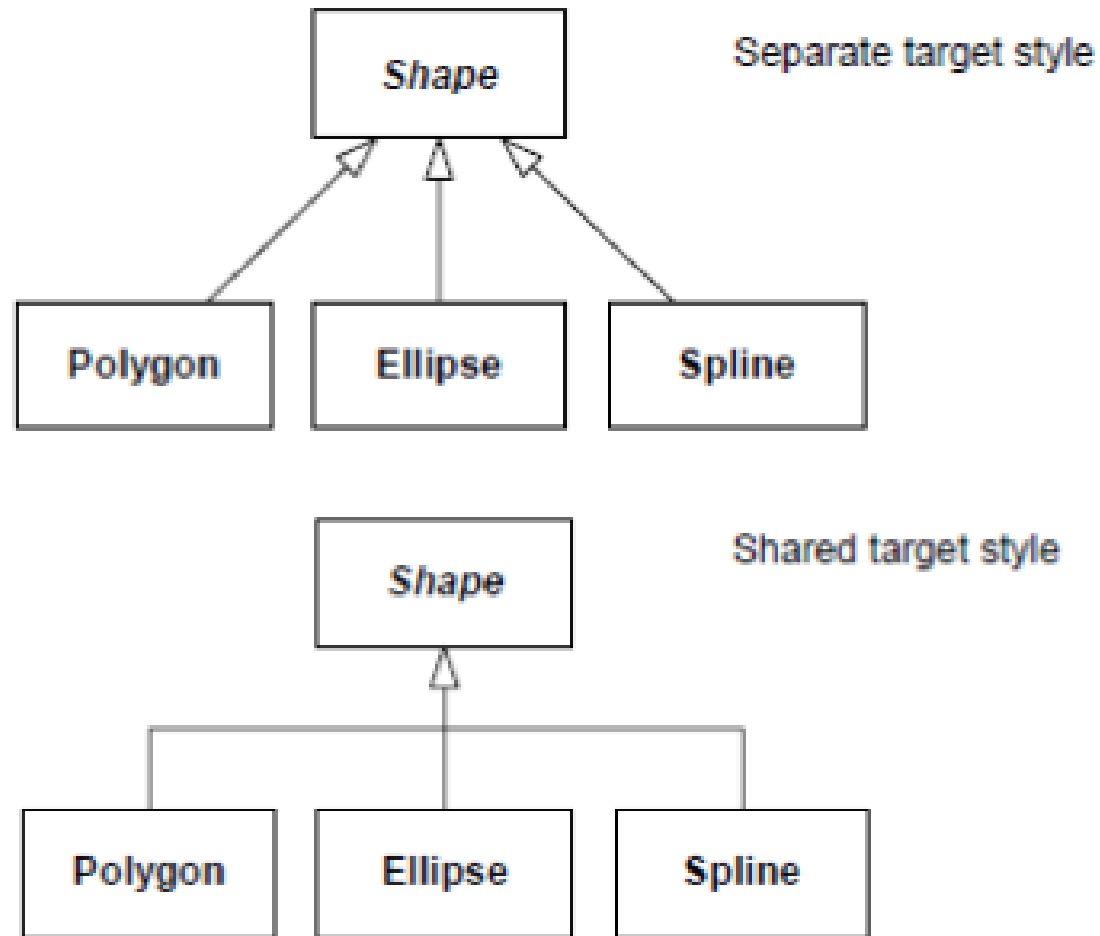


Fig. 7.23 [OMG, 2011b]

Generalización: particiones por subtipo

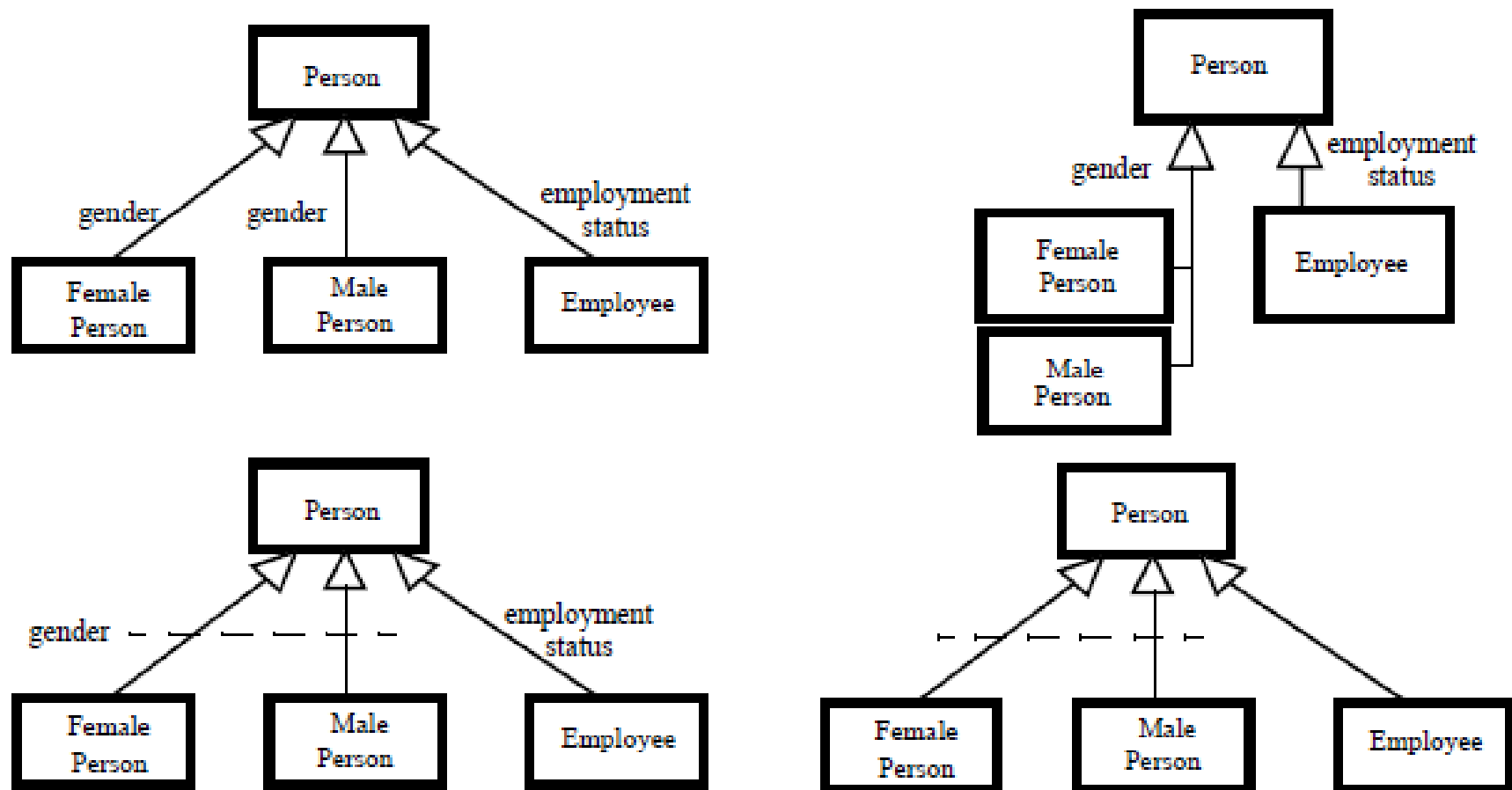
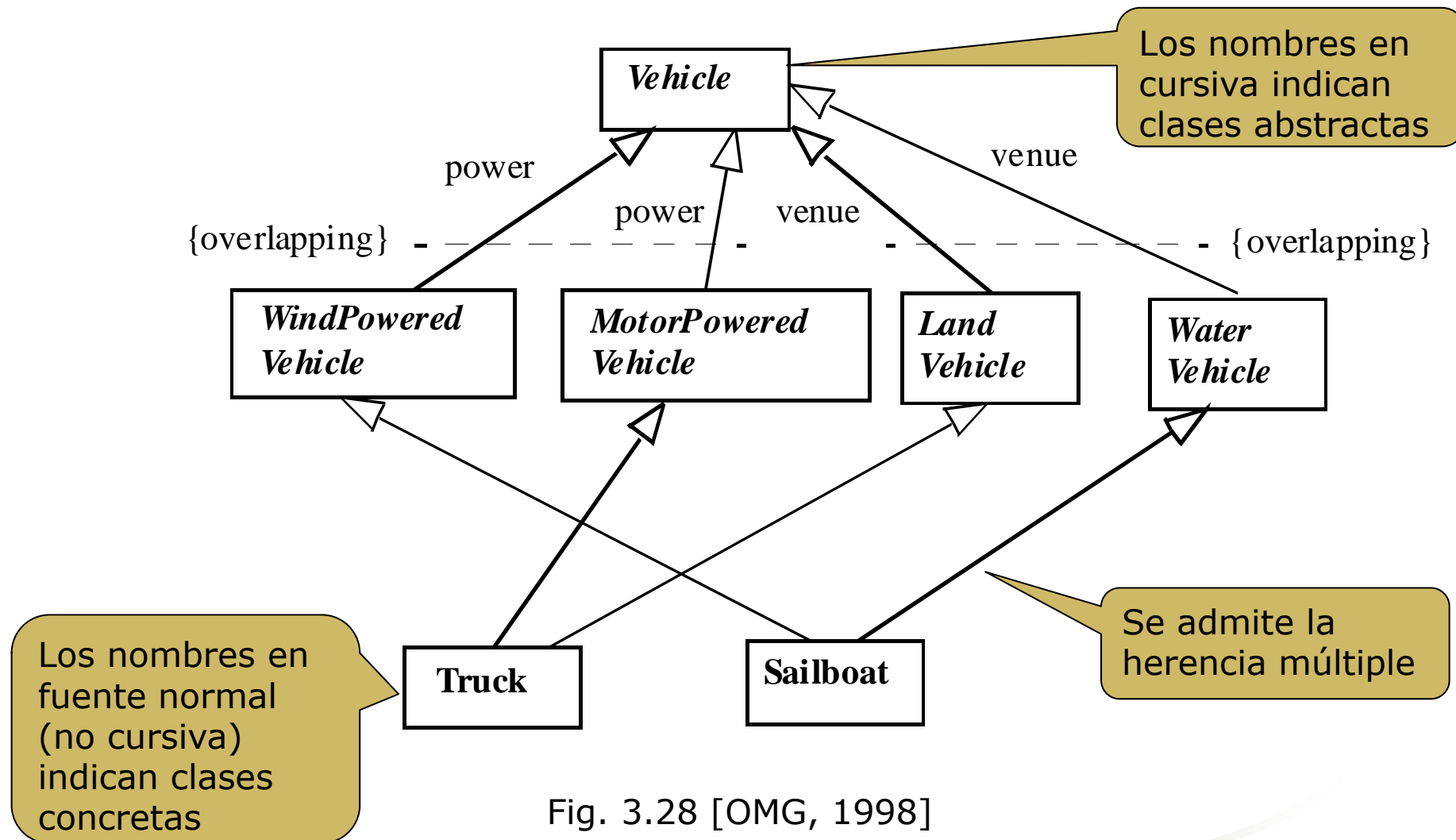


Fig. 7.44 [OMG, 2011b]

Generalización: particiones por subtipo



Dependencia

- Una *dependencia* es una relación de uso, la cual determina que un cambio en la especificación de un elemento (el proveedor) puede afectar a otro elemento que lo utiliza (el cliente), pero no necesariamente a la inversa.

— Ej.

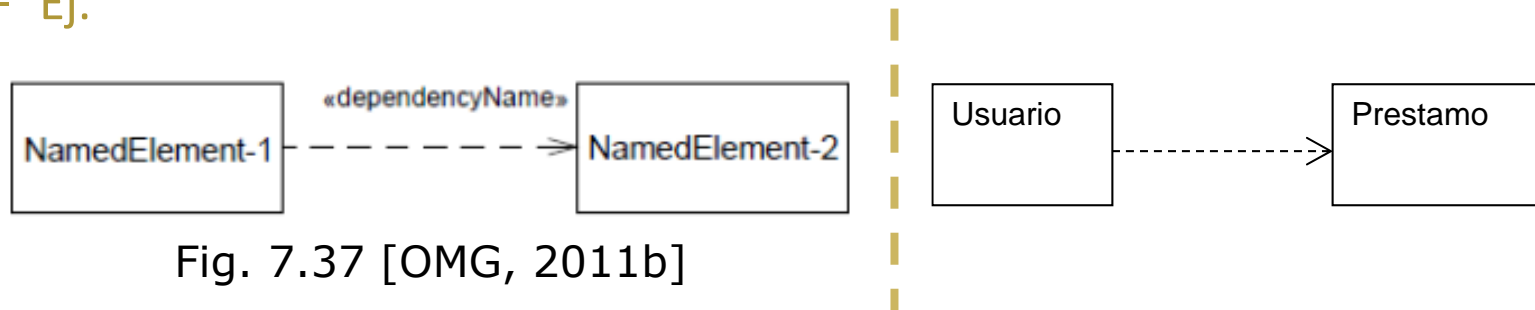


Fig. 7.37 [OMG, 2011b]

- La mayoría de las veces, la dependencia entre clases denota una relación de *uso*, por la cual un elemento requiere la presencia de otro para su correcto funcionamiento.



Dependencia: implementación

- La relación de dependencia se suele plasmar en el código con que el proveedor es:
 - Global al cliente
 - Ej. un *singleton*
 - Parámetro de una función del cliente
 - Está definido en el cuerpo de una función del cliente
 - Ej. constructora

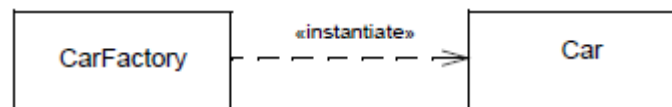
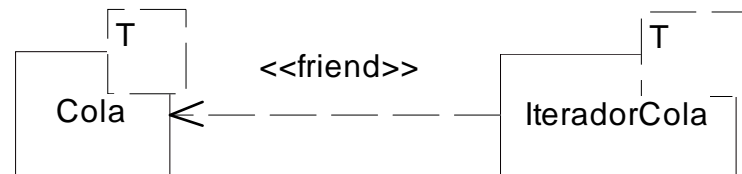
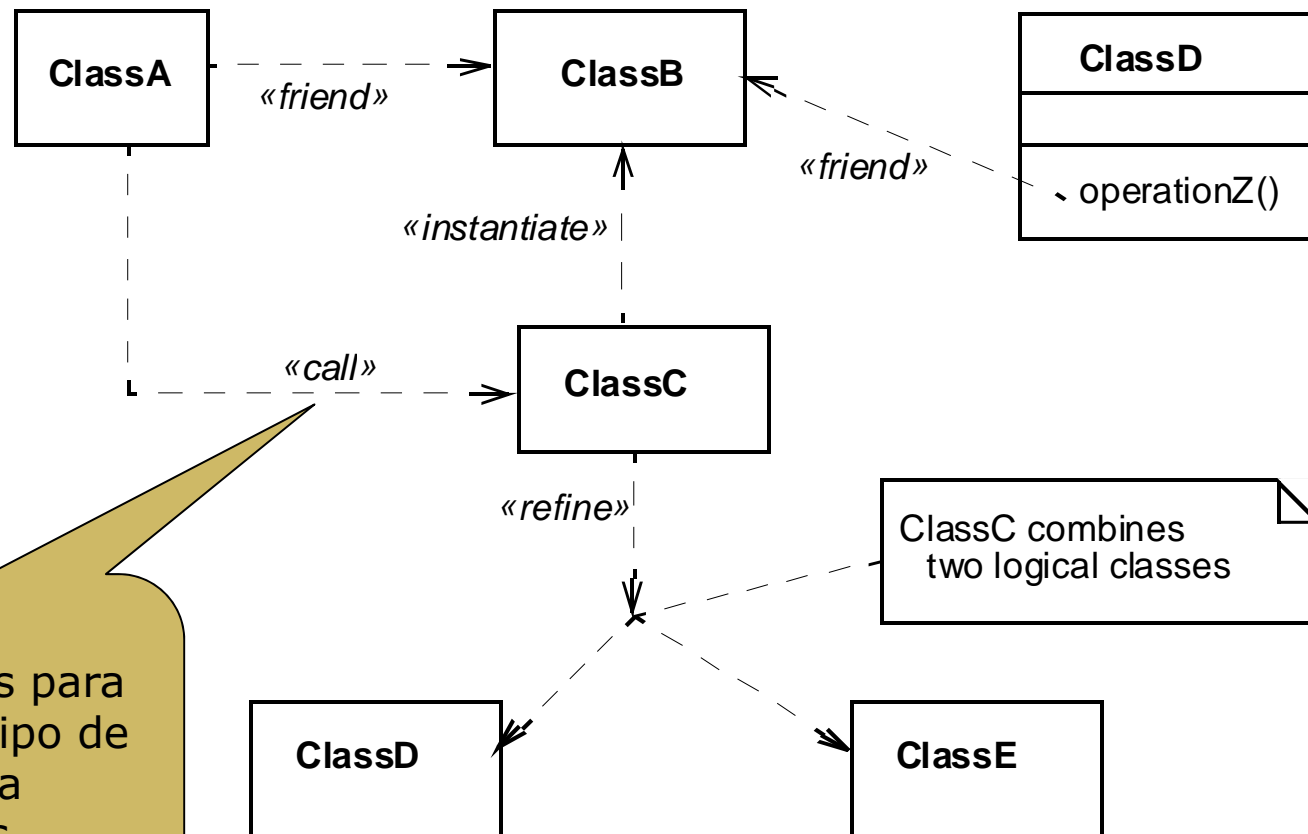


Fig. 7.37 [OMG, 2011b]

- También puede denotar permiso de acceso
 - Ej. dependencia como permiso de acceso *friend*



Dependencia: estereotipos



Uso de estereotipos para detallar el tipo de dependencia entre clases. También puede tener un nombre.

Extendido de Fig. 3.30 [OMG, 1998]

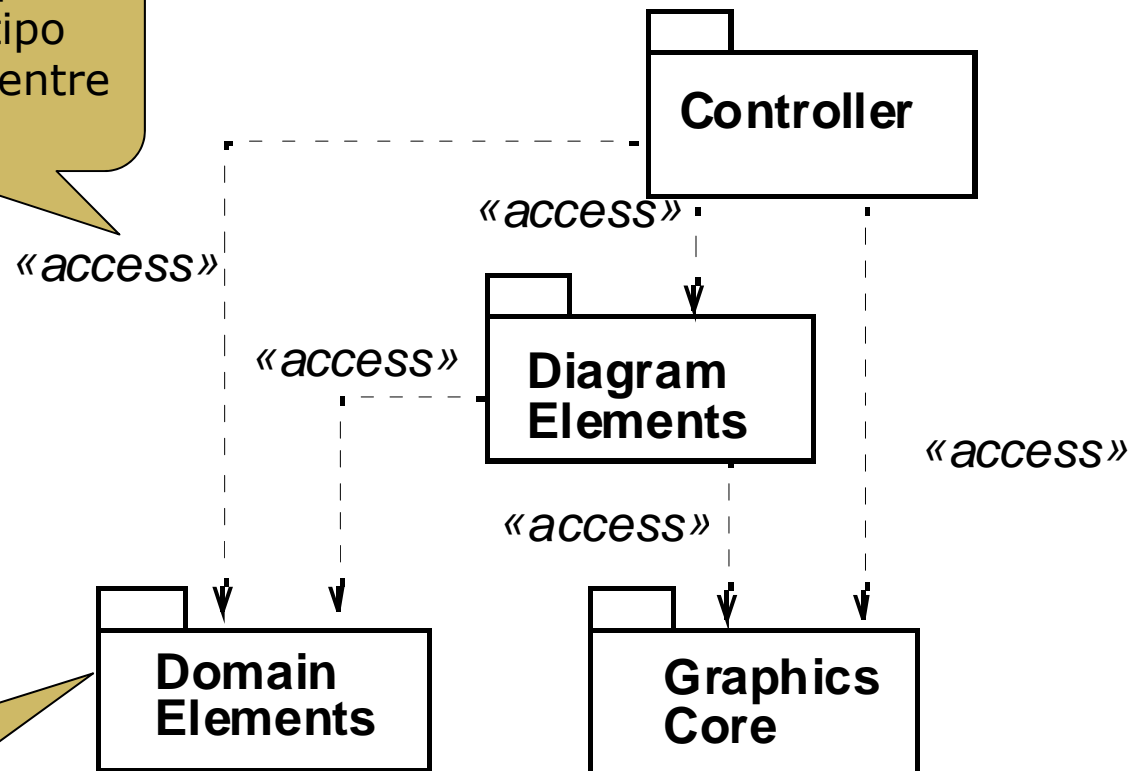
Fig. 3-50, *UML Notation Guide*

Ingeniería del Software



Dependencia: estereotipos

Uso de estereotipos para detallar el tipo de dependencia entre paquetes.



Los paquetes se verán más adelante.

Extendido de Fig. 3.31 [OMG, 1998]

Fig. 3-51, *UML Notation Guide*



Dependencia: interfaces

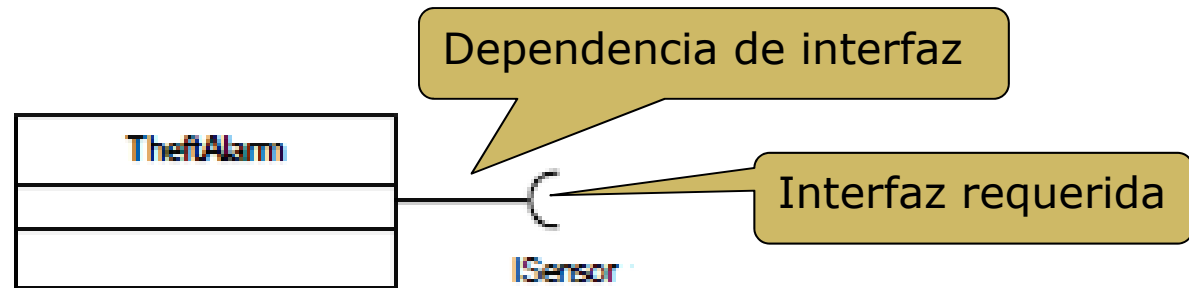


Fig. 7.56 [OMG, 2011b]

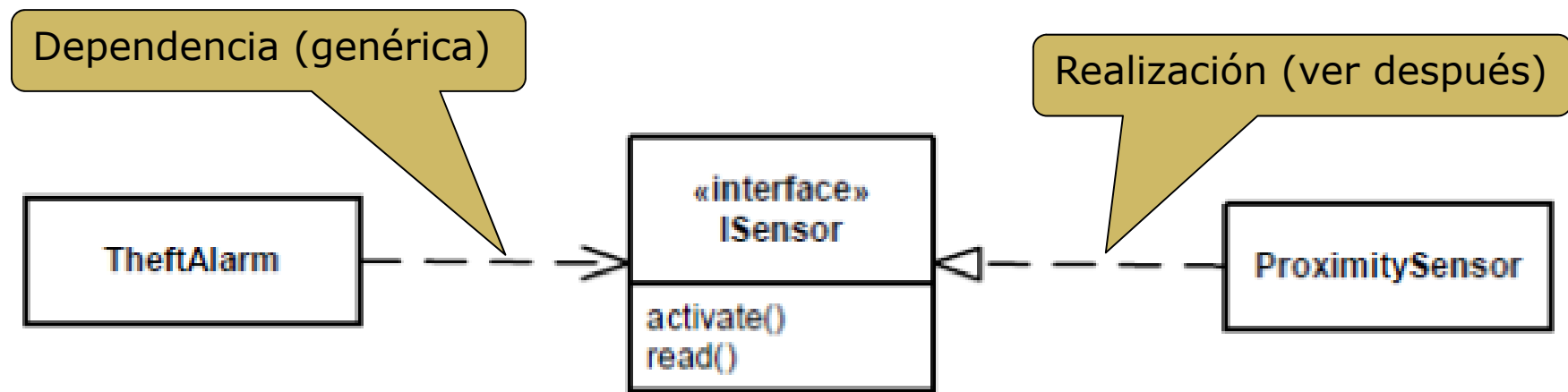


Fig. 7.57 [OMG, 2011b]



Realización

- La *realización* es una relación entre una especificación y su implementación.
 - De esta forma la implementación se compromete a cumplir las responsabilidades de los contratos implementados.
- En particular, decimos que una *clase realiza* una *interfaz* si y sólo si implementa todas sus operaciones.
 - Interfaz como conjunto de operaciones sin atributos.
- Nótese que:
 - Una interfaz puede ser realizada por más de una clase.
 - Una clase puede realizar más de una interfaz.



Realización: tipos e implementación de clases

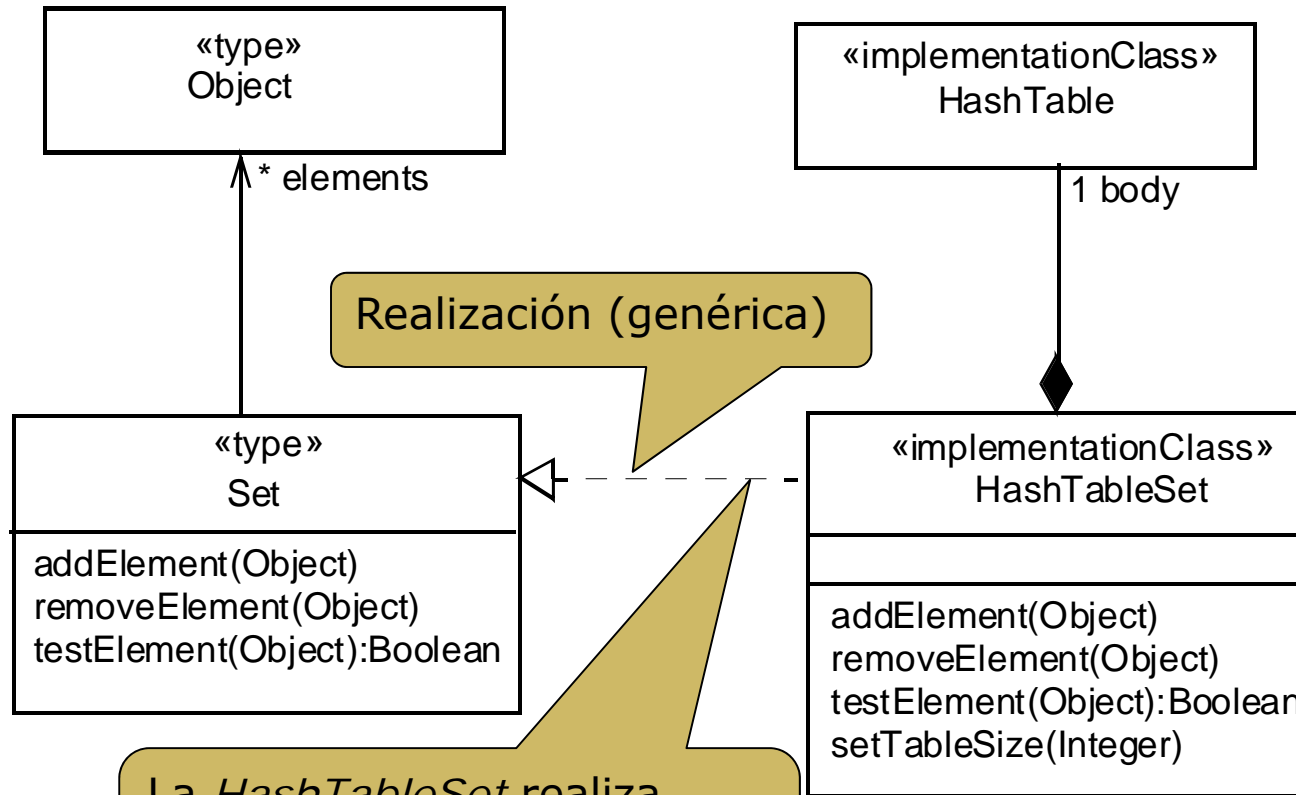


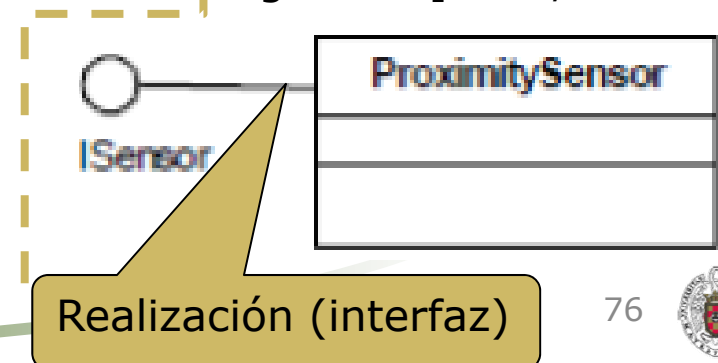
Fig. 3.12 [OMG, 1998]

La *HashSetSet* realiza (implementa) la especificación abstracta *Set*.

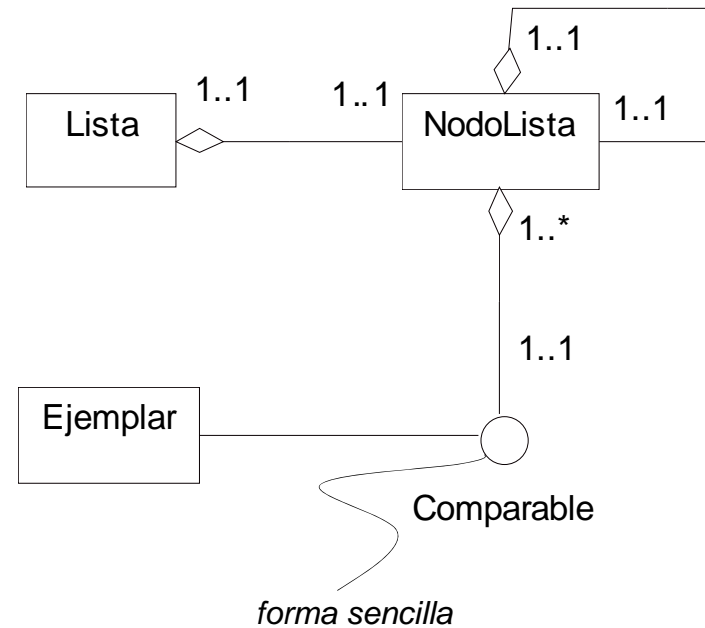
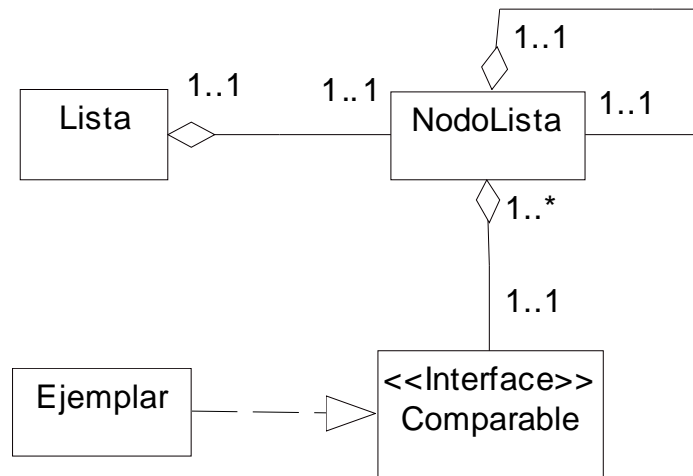
Los estereotipos de este ejemplo son del perfil estándar L2 [OMG, 2011b].

el Software

Fig. 7.55 [OMG, 2011b]



Ejemplo: realización



¿Son correctos los diagramas?



Instanciación

- La *instanciación* denota el proceso por el cual se instancia una clase plantilla.
 - Ej.

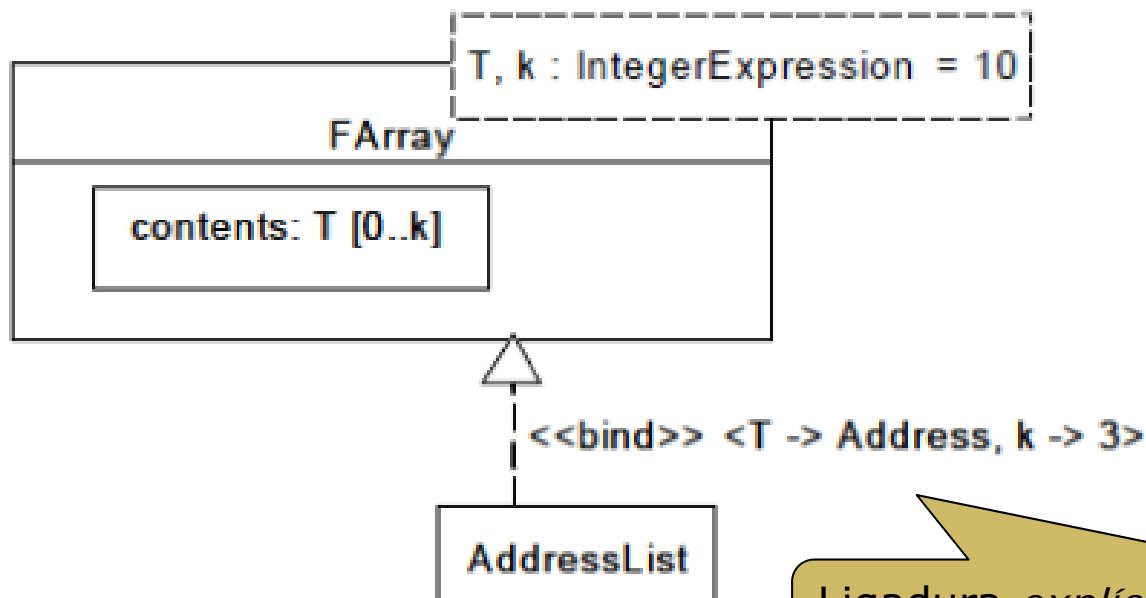


Fig. 17.14 [OMG, 2011b]

Ligadura *implícita* o *anónima*

`FArray <T -> Point>`

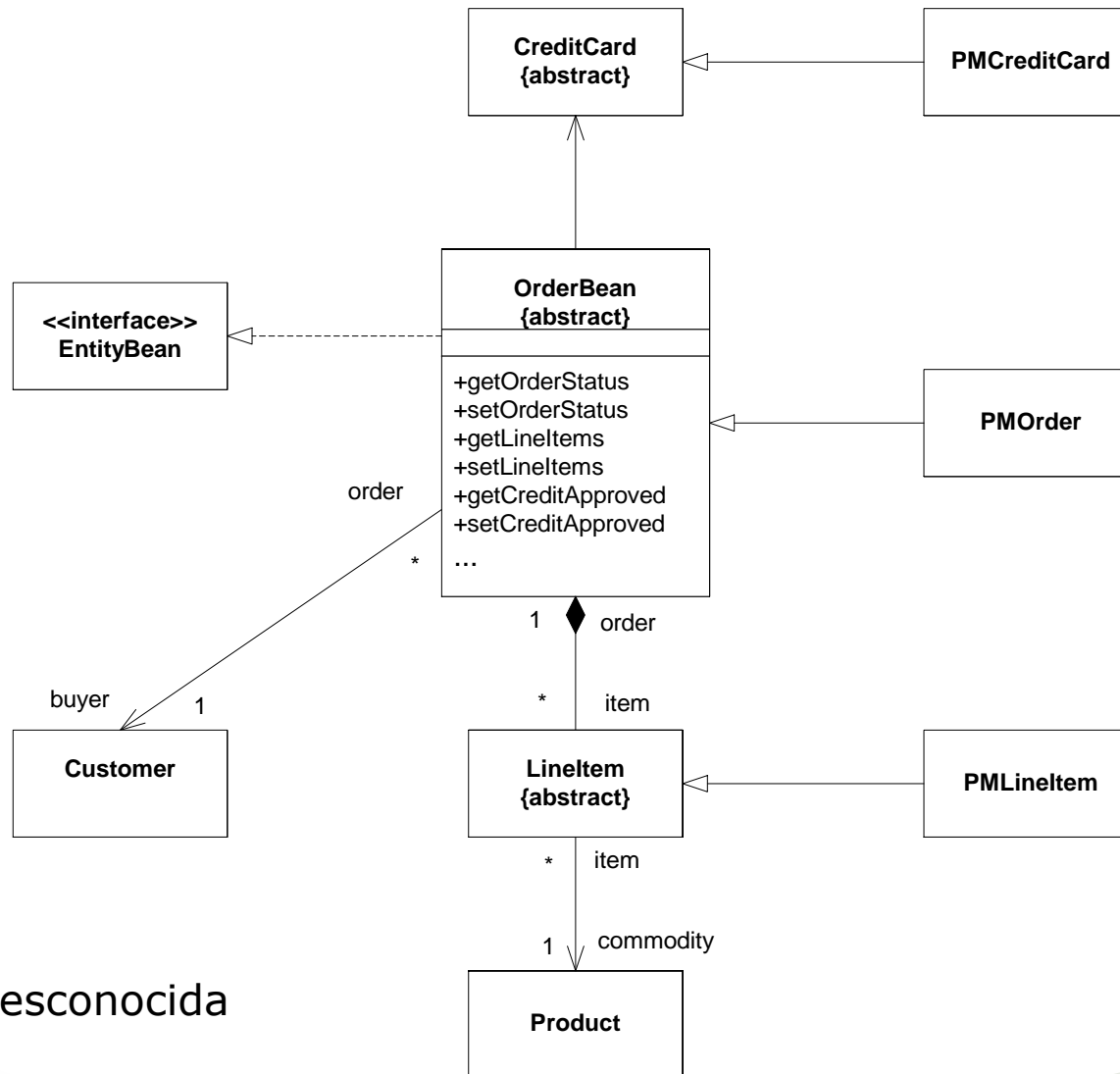
Fig. 17.15 [OMG, 2011b]

Ligadura *explícita*

```
typedef FArray<Address, 3> AddressList
```



Ejemplo: diagrama de clases



Referencia desconocida



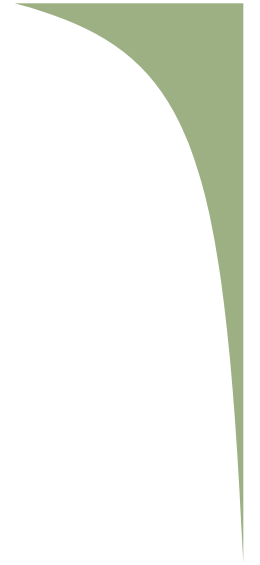


DIAGRAMA DE OBJETOS





Diagrama de objetos

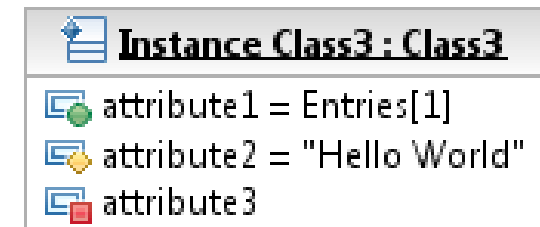
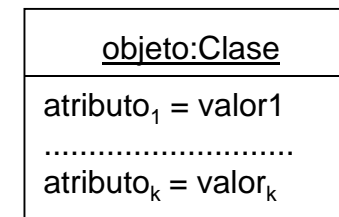
- Un *diagrama de objetos* es un diagrama estructural estático que muestra un conjunto de *objetos* y sus relaciones en un momento concreto.
 - Estos diagramas modelan instancias de los elementos contenidos en los diagramas de clases.
 - No muestran la evolución del sistema en el tiempo.
- Proporciona una visión de instanciación de las entidades del sistema.
 - Qué elementos / objetos existen.
 - De qué tipos / clases son.
 - Cómo se relacionan los elementos / objetos.



Diagrama de objetos: entidades y relaciones

- Objeto

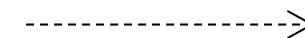
- Un *objeto* o *instancia* es una manifestación concreta de una abstracción (*clasificador*).
- Este *clasificador* suele ser una clase o interfaz, aunque puede ser de otros tipos como nodos, enumerados, componentes o tipos.
- Se le puede aplicar un conjunto de operaciones, y posee un estado que almacena el efecto de las operaciones.



- Enlace (*asociación*)

- Conexión entre 2 o más objetos.

- Dependencia



Objetos

Instancia y clasificador
con valores de atributos

triangle: Polygon

center = (0,0)
vertices = ((0,0),(4,0),(4,3))
borderColor = black
fillColor = white

triangle: Polygon

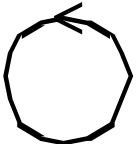
Instancia y
clasificador

Sólo identificador de
la instancia (objeto)

triangle

:Polygon

Objeto
anónimo (sin
nombre). Sólo
identifica la
abstracción
(clasificador)
de la que es
instancia
":abstracción".


scheduler

Objeto anónimo. El
icono corresponde a
una clase de control.

Fig. 3.18 [OMG, 1998]



Ejemplo: diagrama de objetos

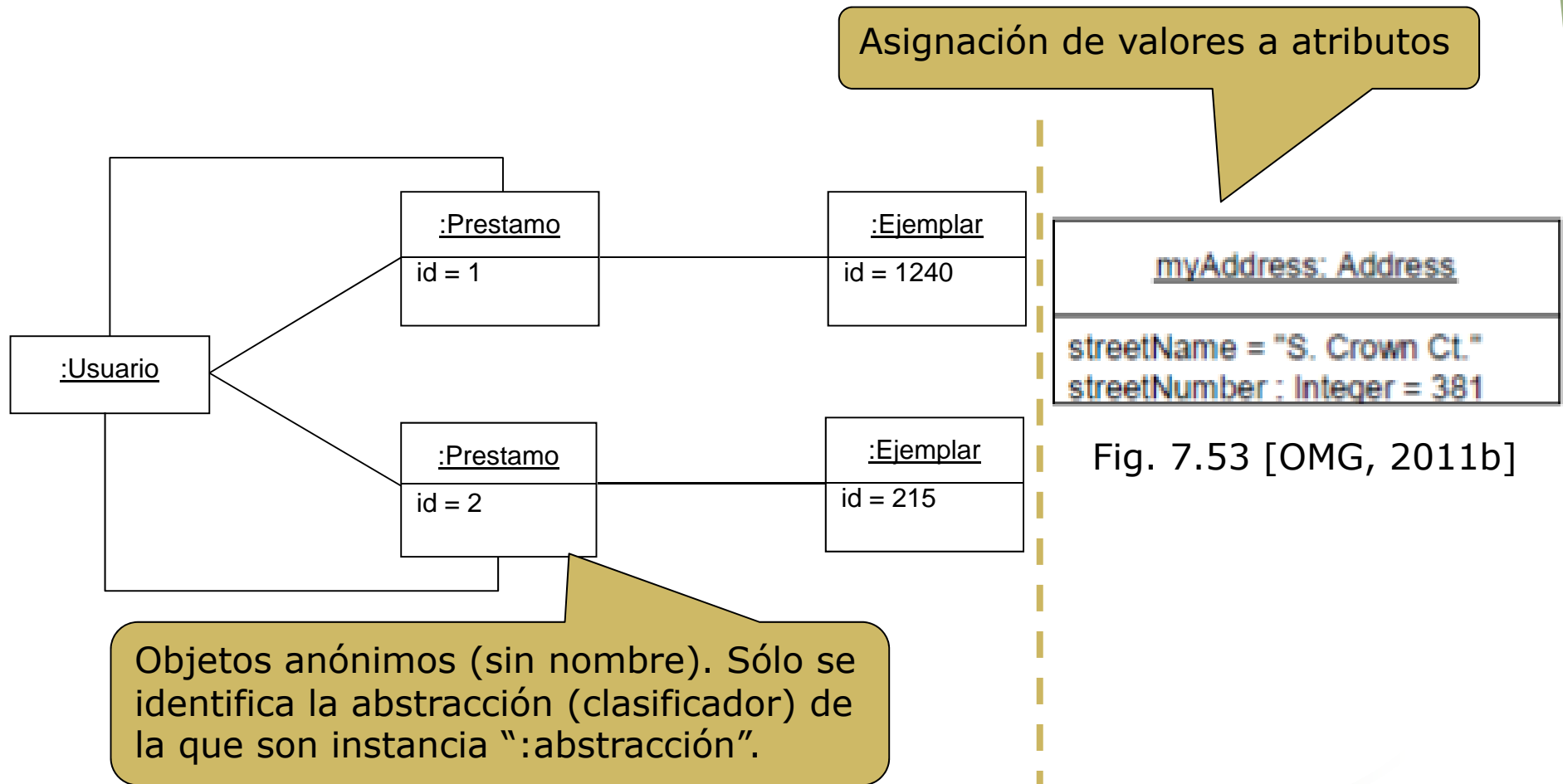


Fig. 7.53 [OMG, 2011b]



Restricciones y comentarios

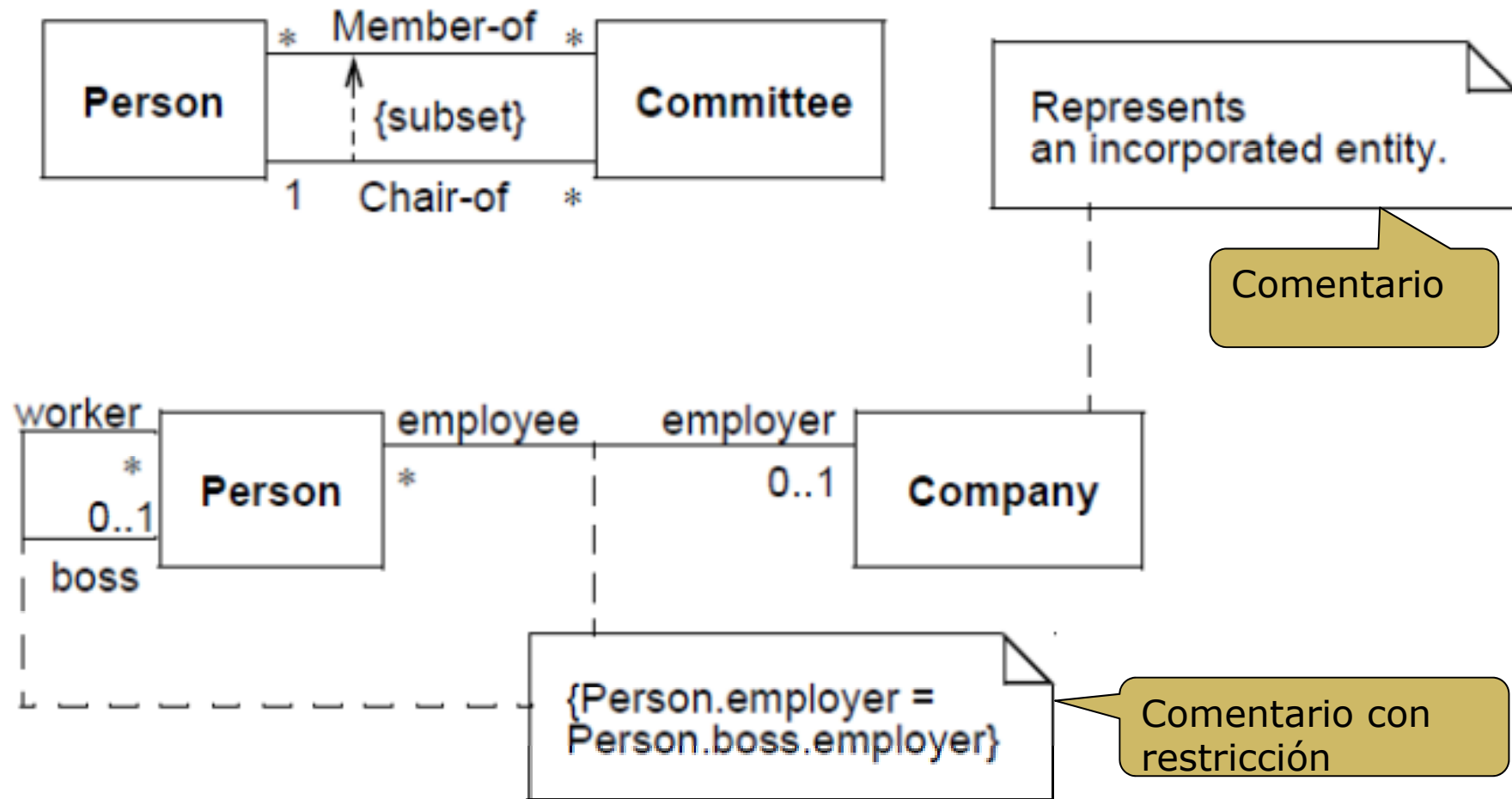


Fig. 3.5 [OMG, 1998]

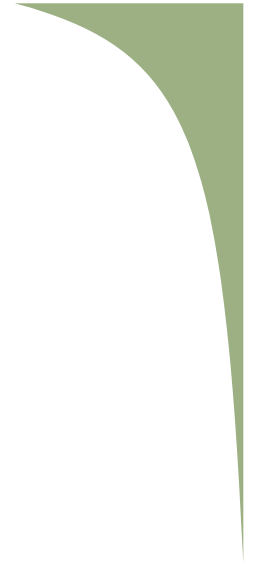


DIAGRAMA DE ESTRUCTURA COMPUESTA





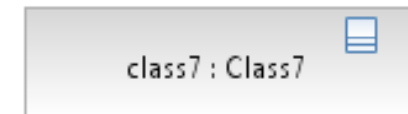
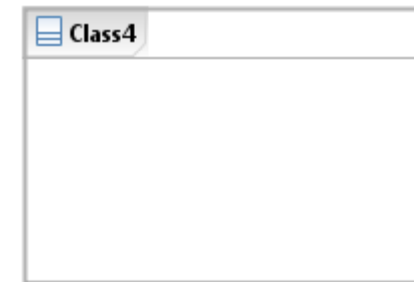
Diagrama de estructura compuesta

- El *diagrama de estructura compuesta* es un diagrama estructural estático que muestra
 - la estructura interna de un clasificador y las colaboraciones que ésta facilita.
 - el uso de colaboraciones (menos habitual).
- Proporciona una visión de clasificación de los elementos de una clase.
 - Qué tipos de elementos existen.
 - Cómo se relacionan los tipos de elementos.
 - Qué puertos y conectores ofrecen y necesitan.



Diagrama de estructura compuesta: entidades (1/2)

- Clasificador estructurado
 - Elemento del que se representa la estructura interna.
 - En el diagrama contiene sus componentes.
- Parte
 - Rol jugado por una instancia o colección de instancias de un clasificador.
- Puerto
 - Punto de interacción de una parte.
 - Marca las interfaces proporcionadas y requeridas.



El puerto es el cuadrado. Los otros símbolos representan las interfaces proporcionadas y requeridas.





Diagrama de estructura compuesta: entidades (2/2)

- Colaboración
 - Interacción entre partes que juegan roles.
 - Más abstracta que el clasificador estructurado.
- Uso de colaboración
 - Instanciación de una colaboración con las instancias específicas que juegan sus roles.





Diagrama de estructura compuesta: relaciones

- Conector

- Línea que une dos elementos.
- Indica que los elementos unidos pueden interactuar en tiempo de ejecución.



- Vinculación de roles

- Relaciona una colaboración con las instancias que juegan los roles de la misma.





Puerto

- Un *puerto* agrupa un conjunto semánticamente cohesivo de interfaces proporcionadas y requeridas.
- Indica un punto de interacción entre un clasificador estructurado o parte y su entorno.
- El puerto permite aislar el comportamiento interno de un clasificador de su entorno.
 - De esta forma es posible centrarse en el comportamiento del clasificador, obviando el entorno de desarrollo.
 - Siempre que el entorno cumpla las especificaciones del puerto, el clasificador funcionará.



Ejemplo: puerto

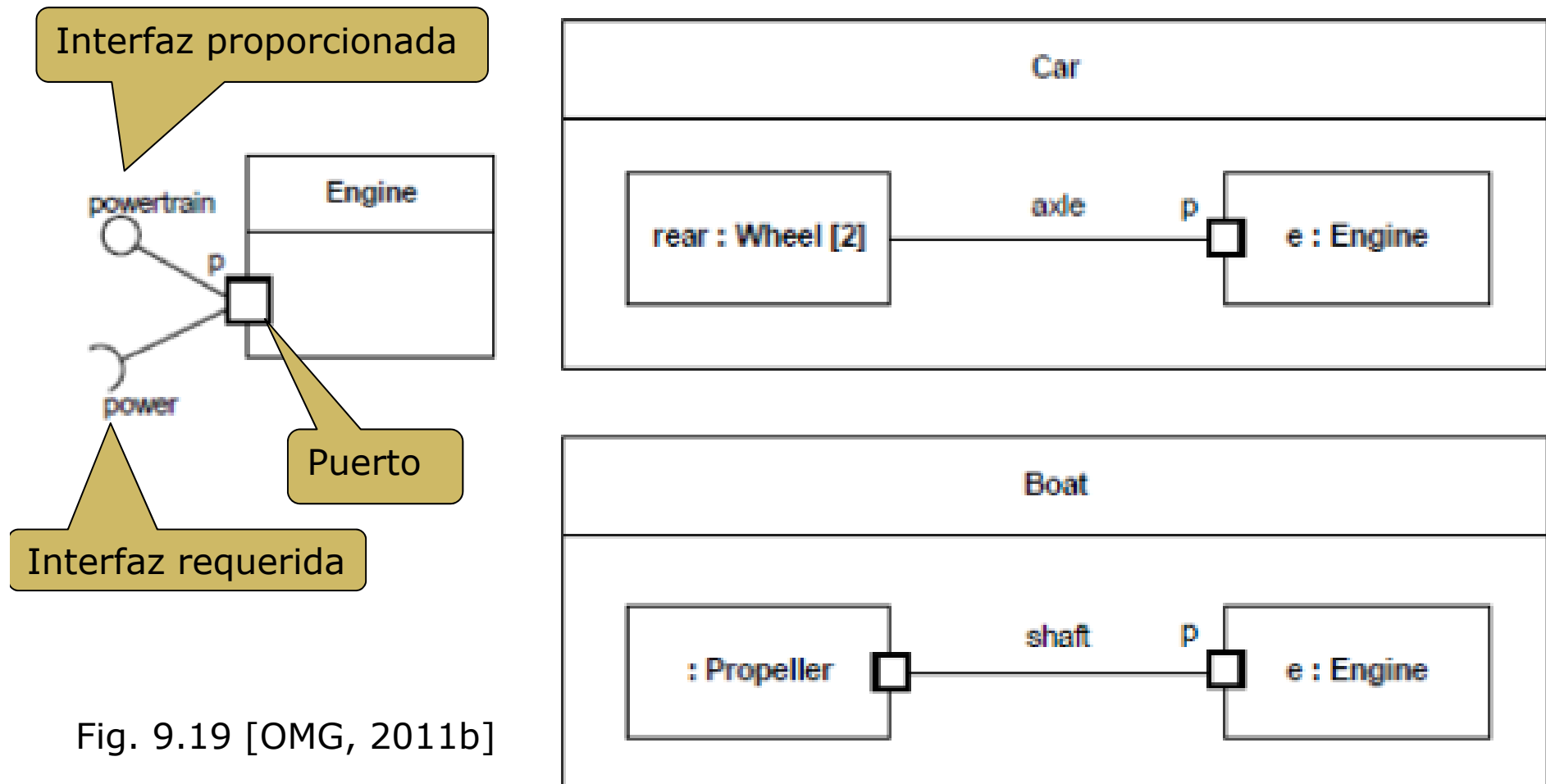
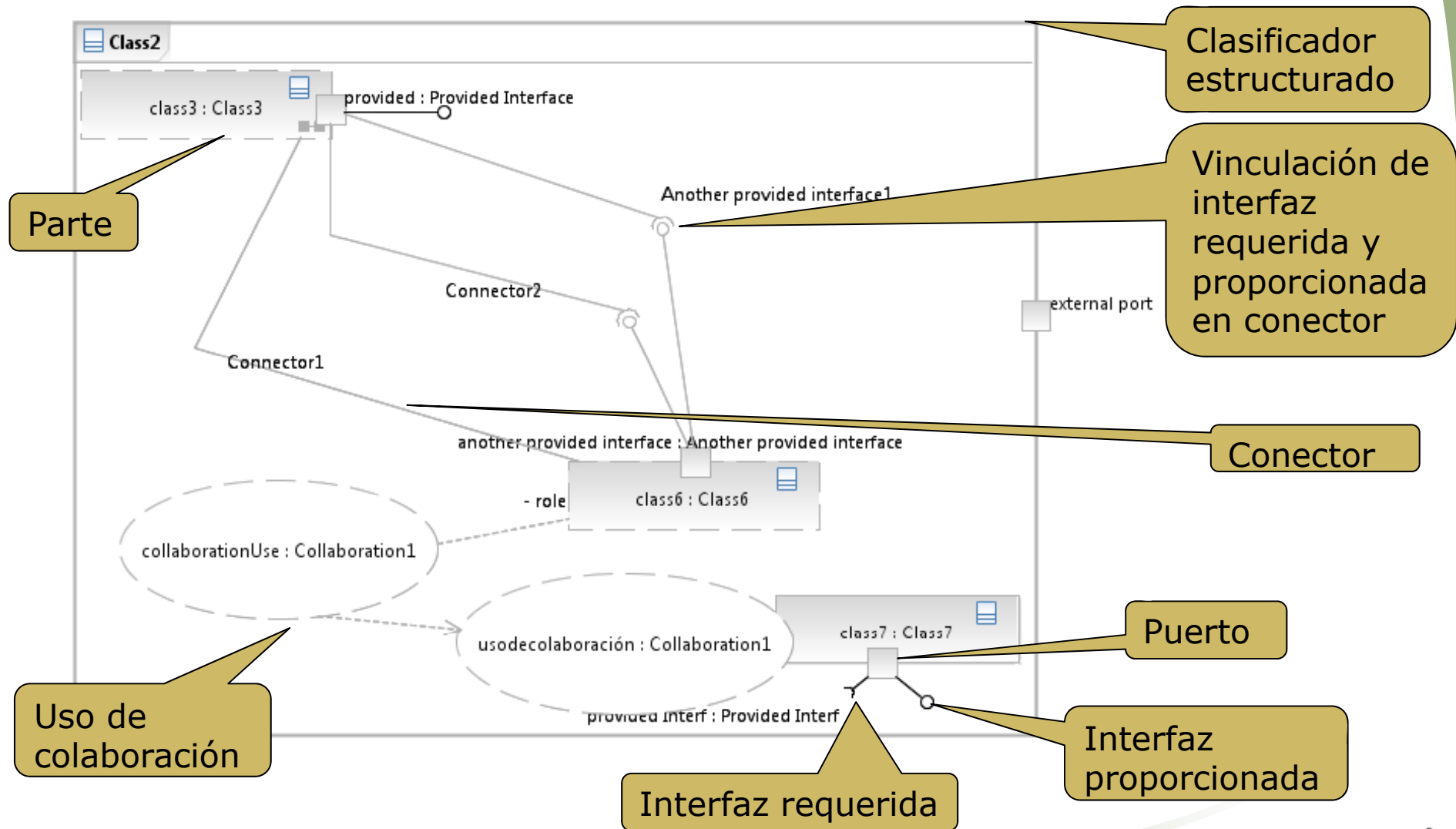
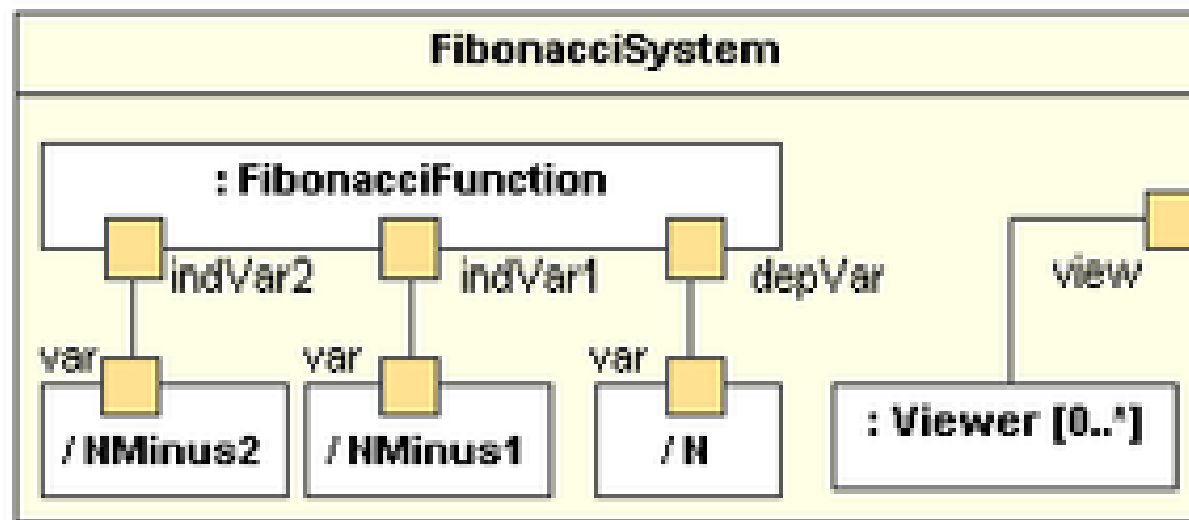


Fig. 9.19 [OMG, 2011b]

Ejemplo: diagrama de estructura compuesta



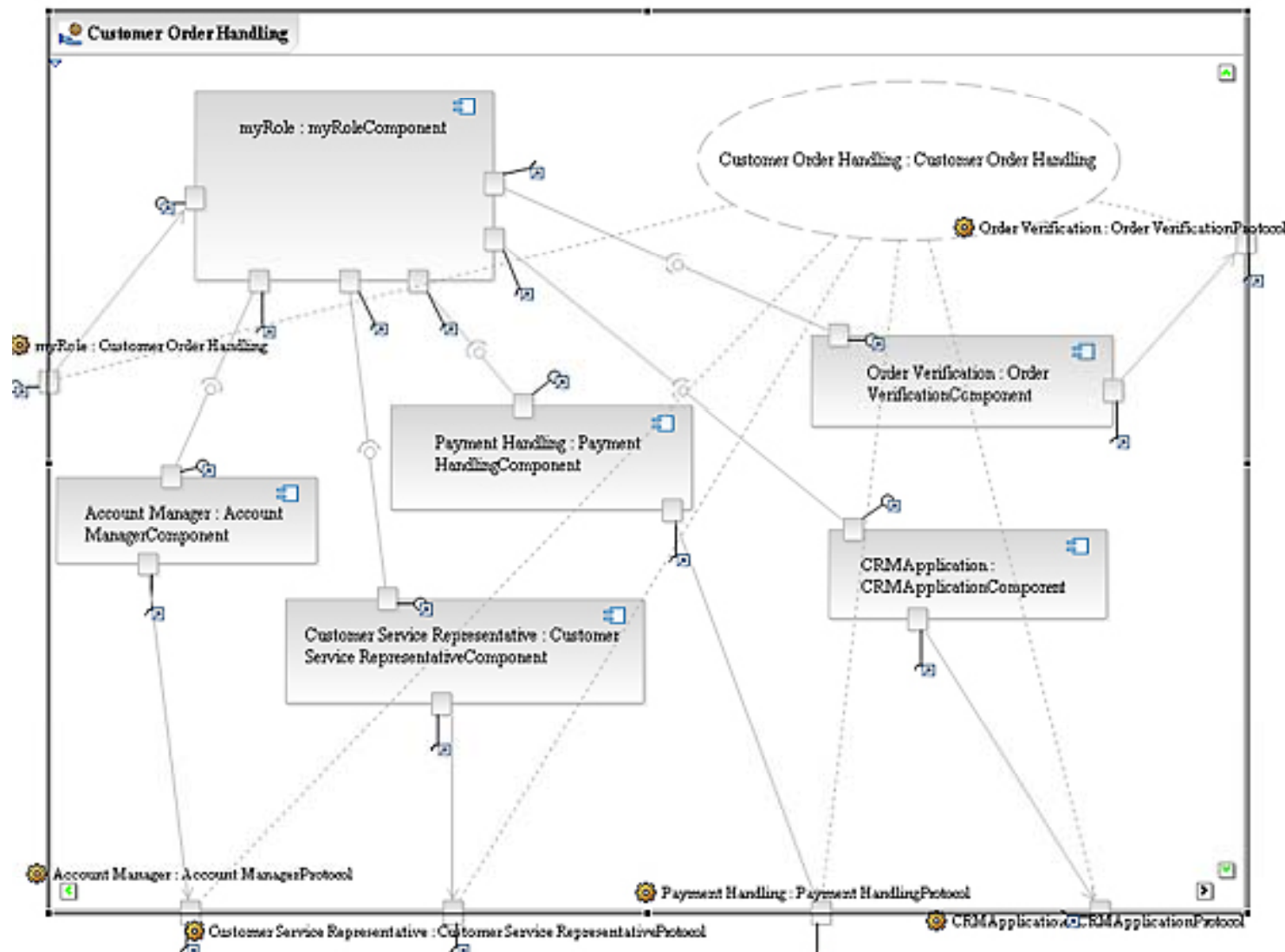
Ejemplo: función de Fibonacci

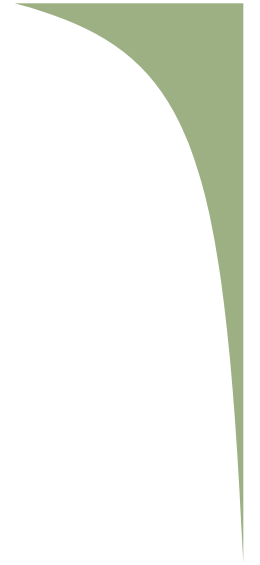


Fuente: Wikipedia,
http://en.wikipedia.org/wiki/Composite_structure_diagram



Ejemplo: gestión de peticiones de cliente





PAUTAS PARA MODELOS ESTRUCTURALES





Consejos para un buen modelo estructural

- Definir un esqueleto que pueda extenderse y refinarse a medida que se aprende más sobre el dominio.
- Utilizar construcciones básicas.
 - Dejar la notación avanzada para los detalles, si es necesaria.
- Dejar los detalles de implementación para el final.
 - Ver el modelo de implementación más adelante.
- Cada diagrama estructural debería:
 - Mostrar un aspecto particular del modelo estructural.
 - Contener elementos del mismo nivel de abstracción.
- Si hay un gran número de elementos, organizarlos en paquetes.





Cómo hacer un modelo estructural

- Utilizar planteamientos de análisis y diseño ascendente (*bottom-up*) y descendente (*top-down*).
 - Especificar la estructura de alto nivel usando clases que tengan significado arquitectural y elementos que gestionen la complejidad (ej. paquetes y subsistemas).
 - Especificar la estructura de bajo nivel a medida que se descubren detalles, se reclasifica, y se definen relaciones.
- Si se conoce bien el dominio, se puede empezar con un modelo estructural, si no:
 - Si se utiliza un método dirigido por casos de uso, asegurarse de la consistencia del modelo estructural con el de casos de uso.
 - Si se utiliza un método dirigido por colaboraciones (basado en roles), asegurarse de que el modelo estructural es consistente con el modelo de colaboración.

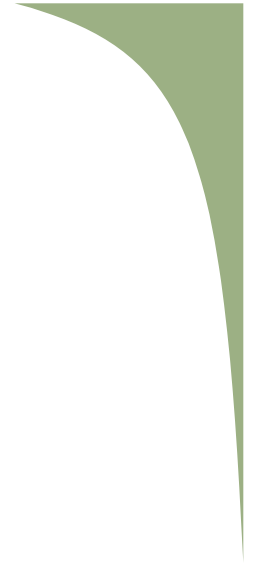




Ejercicio de modelo estructural

- Realizar un diagrama de clases para el ejemplo anterior.
- Realizar el diagrama de estructura compuesta de algunos de sus elementos que participa en múltiples relaciones.





MODELO DE COMPORTAMIENTO





Modelo de comportamiento

- Visión del sistema que describe las interacciones entre sus elementos y la evolución del estado del sistema y sus componentes.
 - Desarrollado por analistas, diseñadores y programadores
- Muestra el comportamiento dinámico del sistema.
 - Las entidades que participan
 - Ej. clases, interfaces, componentes y nodos
 - Las acciones que realizan
 - Ej. pasos de mensajes
 - Los argumentos involucrados
- Se define mediante:
 - Diagramas de interacción
 - Diagrama de secuencia
 - Diagrama de comunicación
 - Diagrama de máquina de estados
 - Diagrama de actividad





Modelo de comportamiento

- Muestran un grafo de elementos clasificados (con tipo) conectados por relaciones (estáticas o dinámicas), posiblemente con flujos de información o control asociados, en el tiempo (durante una colaboración).
- Cuatro tipos:
 - Diagrama de secuencia
 - Proporciona una visión cronológica del paso de mensajes en una interacción.
 - Diagrama de comunicación
 - Da una visión del paso de mensajes en una interacción centrada en las instancias participantes y sus relaciones.
 - Diagrama de máquina de estados
 - Describe la evolución interna de una instancia durante su vida.
 - Diagrama de actividad
 - Describe las actividades realizadas en el sistema a lo largo de varios escenarios por múltiples instancias.





DIAGRAMAS DE INTERACCIÓN





Diagramas de interacción

- Los *diagramas de interacción* muestran una *interacción*.
- Una *interacción* es un comportamiento que comprende un conjunto de *mensajes* intercambiados entre un conjunto de *objetos* dentro de un *contexto* para lograr un propósito.
 - Un *mensaje* es la especificación de una comunicación entre objetos.
 - Transmite información.
 - Implica la expectativa de desencadenar una actividad al ser recibido (evento).
 - Esa actividad suele consistir en ejecutar un método y cambiar de estado.
 - Un *contexto* es un conjunto de objetos que están relacionados para un propósito.
- Una *interacción* se describe mediante:
 - el conjunto de objetos que participa en ella
 - sus relaciones
 - los mensajes que se pueden enviar entre los objetos a través de las relaciones





Diagramas de interacción: tipos

- Hay dos tipos de diagramas de interacción:
 - Diagramas de secuencia
 - Diagramas de comunicación
- Los diagramas de secuencia y de comunicación describen una información similar, pero la muestran de maneras diferentes.
 - Los *diagramas de secuencia* muestran la secuencia explícita de mensajes ordenada en el tiempo.
 - Los *diagramas de comunicación* muestran las relaciones entre objetos y el paso de mensajes.



Ejemplo: diagramas de interacción

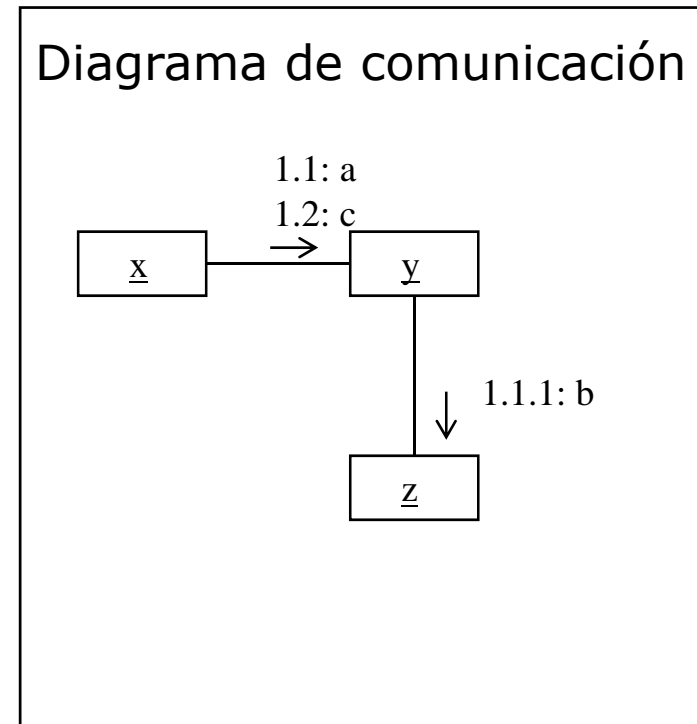
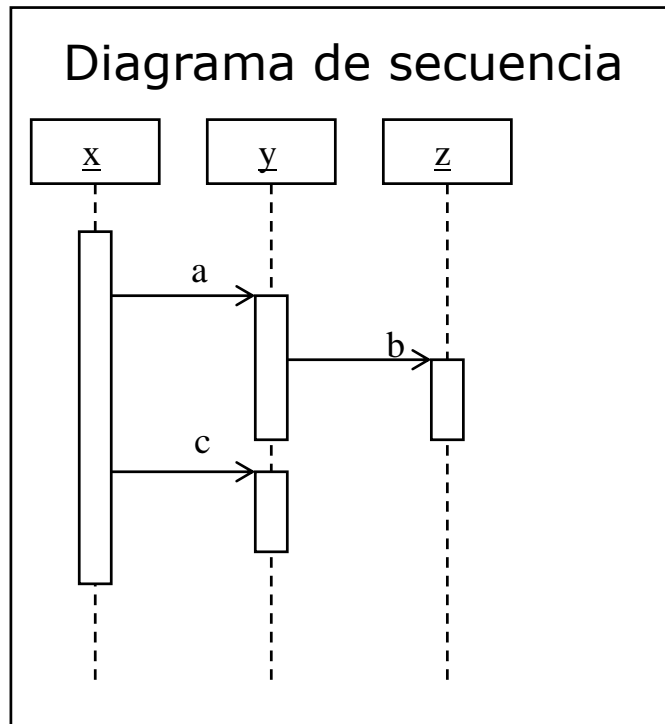




DIAGRAMA DE SECUENCIA





Diagramas de secuencia

- Un *diagrama de secuencia* muestra el desarrollo de una interacción en base al intercambio de mensajes entre objetos organizado en una secuencia temporal.
 - Objetos participantes en la interacción
 - Paso de mensajes entre objetos en orden cronológico
- Son los diagramas de interacción apropiados para especificaciones de tiempo real y escenarios complejos.



Diagrama de secuencia: entidades (1/2)

- Marco
 - Contenedor opcional del diagrama que incluye un compartimento con su nombre e indicaciones del tipo.
 - Ver fragmentos combinados para las indicaciones del tipo.
- Línea de vida
 - Entidad con identidad única.
 - Se le pueden aplicar un conjunto de operaciones (enviar señales).
 - Tiene un estado y almacena los efectos de las operaciones (las señales).
 - Puede ser un objeto, valor de datos o instancia de componente.

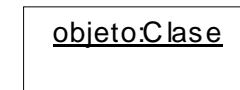
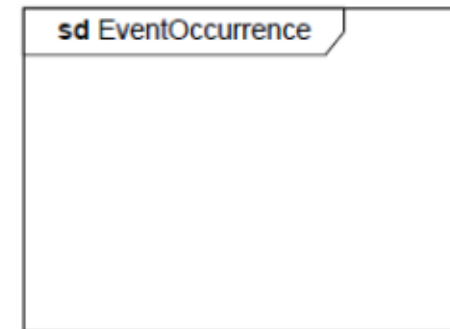
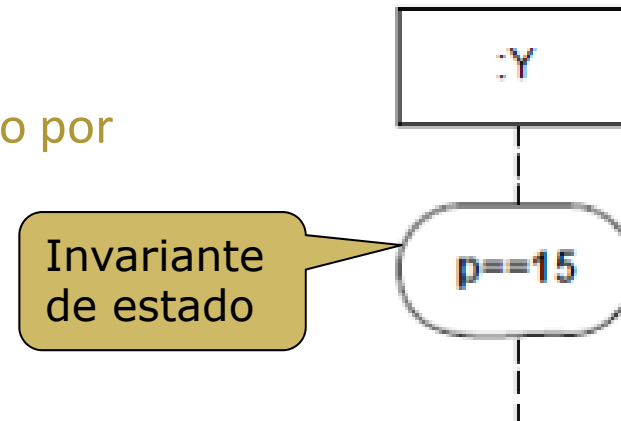


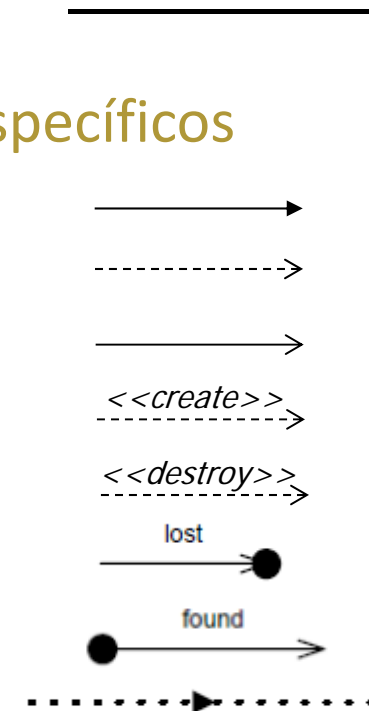
Diagrama de secuencia: entidades (2/2)

- Invariante de estado
 - Condición satisfecha en todo momento por una línea de vida.

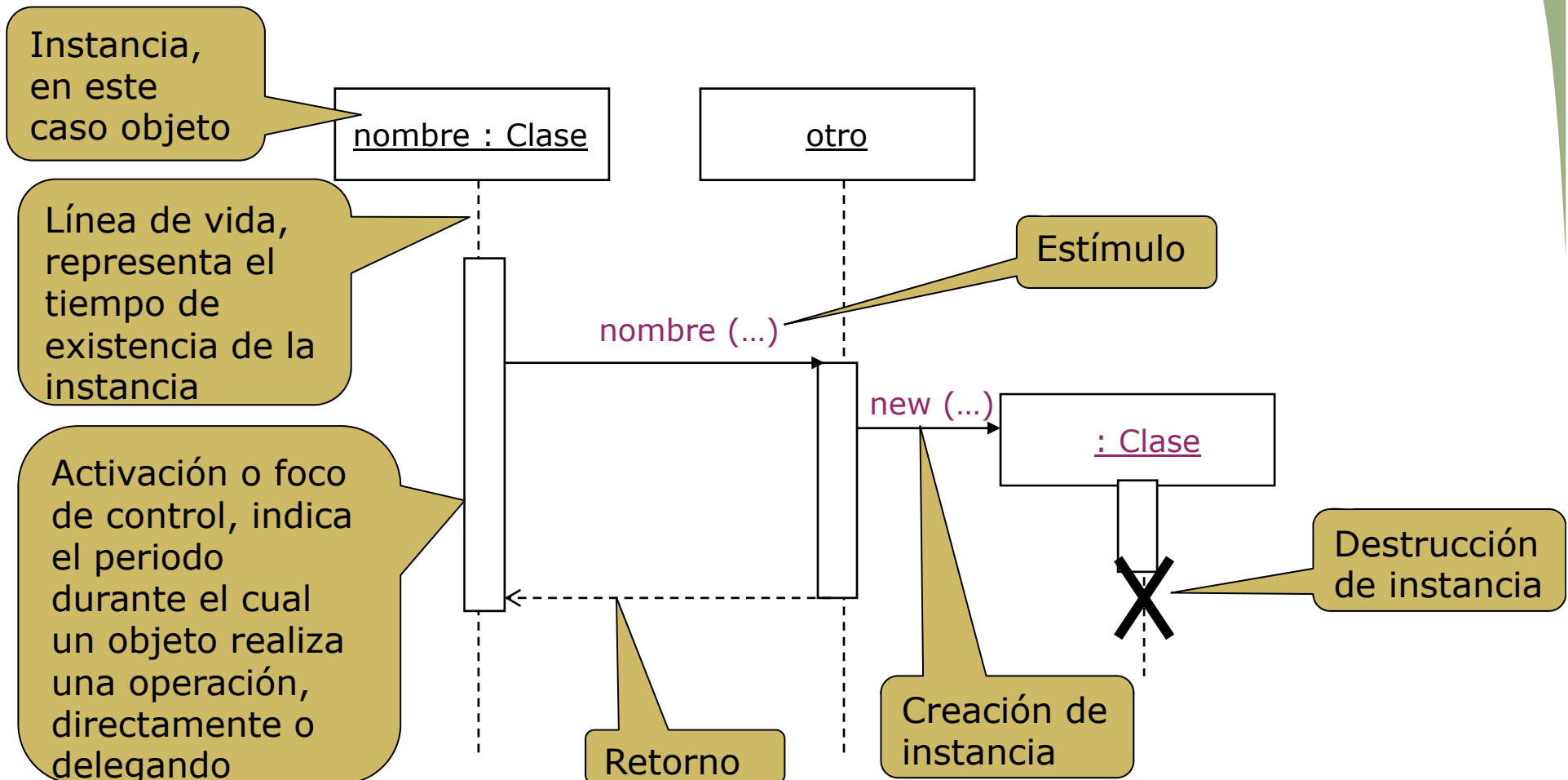


Diagramas de secuencia: relaciones

- Enlace
 - Conexión entre instancias.
 - Los enlaces pueden tener atributos con valor.
- Los enlaces pueden ser de múltiples tipos específicos
 - Invocación síncrona
 - Retorno de invocación síncrona
 - Invocación asíncrona
 - Creación de instancia
 - Destrucción de instancia
 - Pérdida de mensaje
 - Mensaje encontrado
 - Orden general
 - Ej. para eventos o mensajes



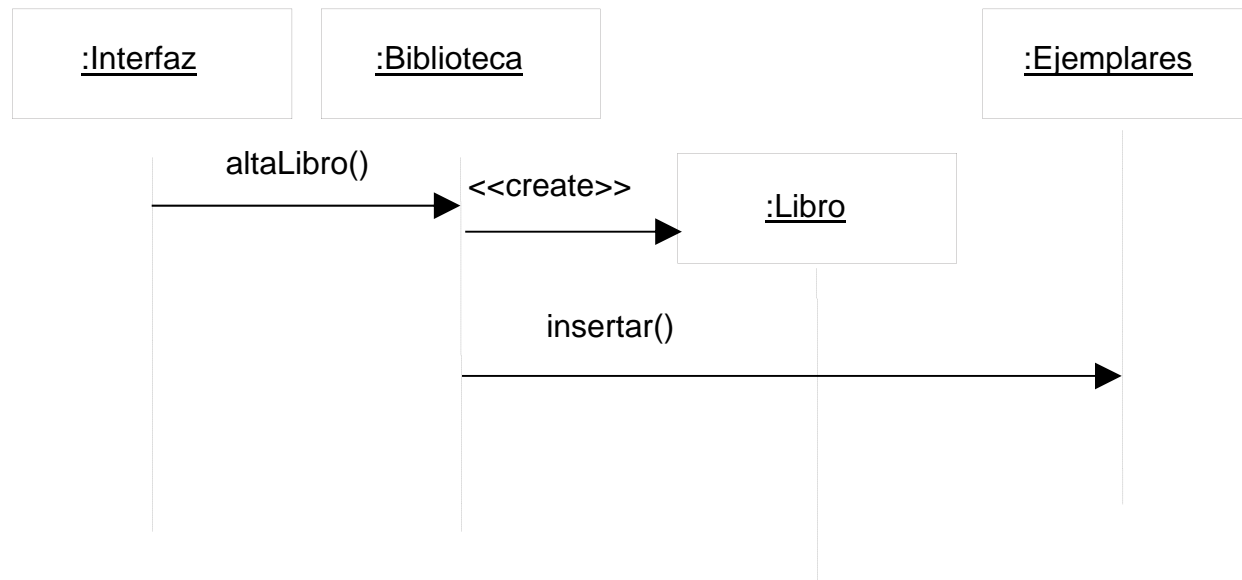
Ejemplo: diagrama de secuencia



El estímulo se puede poner *[condición] variable := mensaje(params)*



Ejemplo: diagrama de secuencia



Interacciones

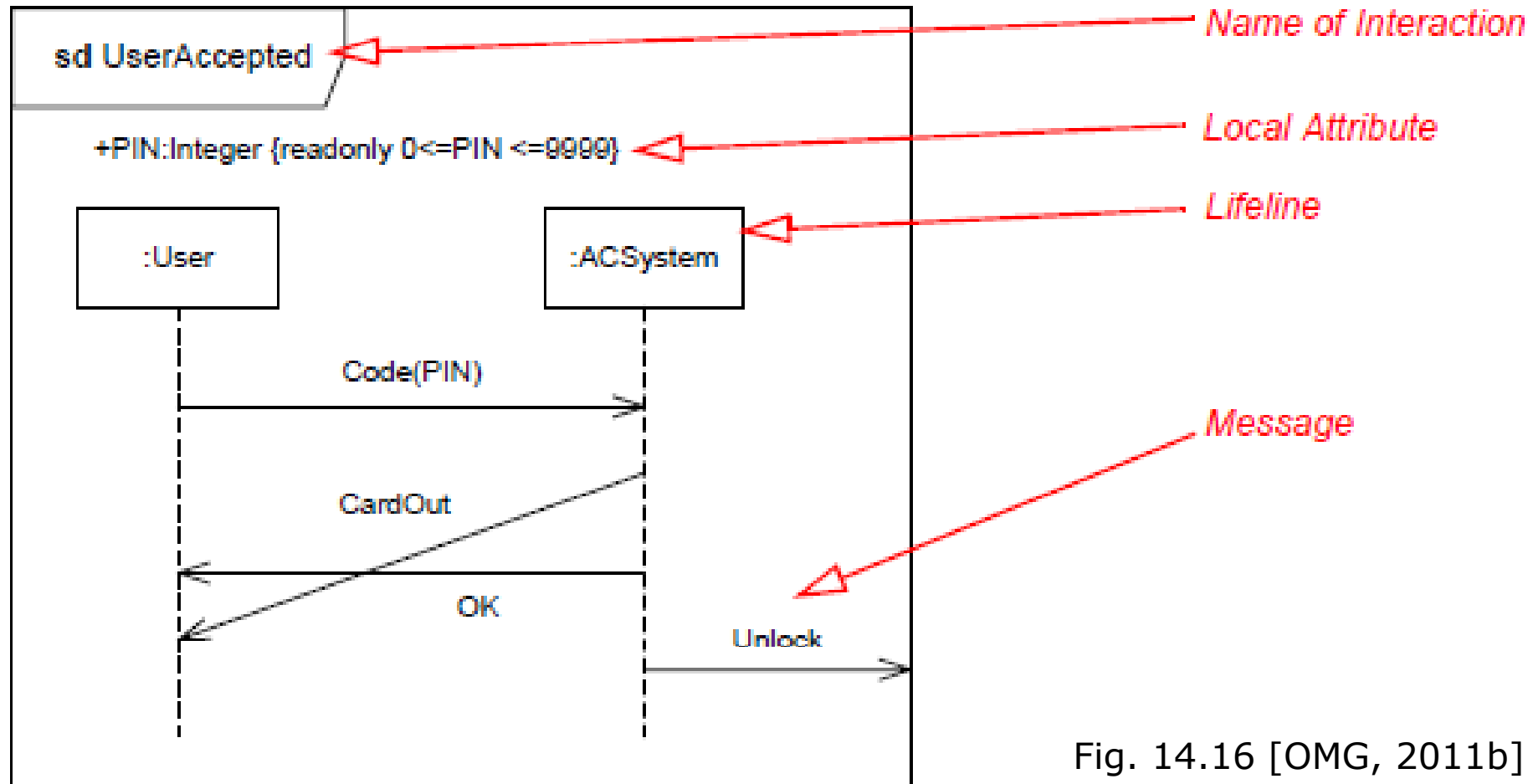


Fig. 14.16 [OMG, 2011b]



Corregión

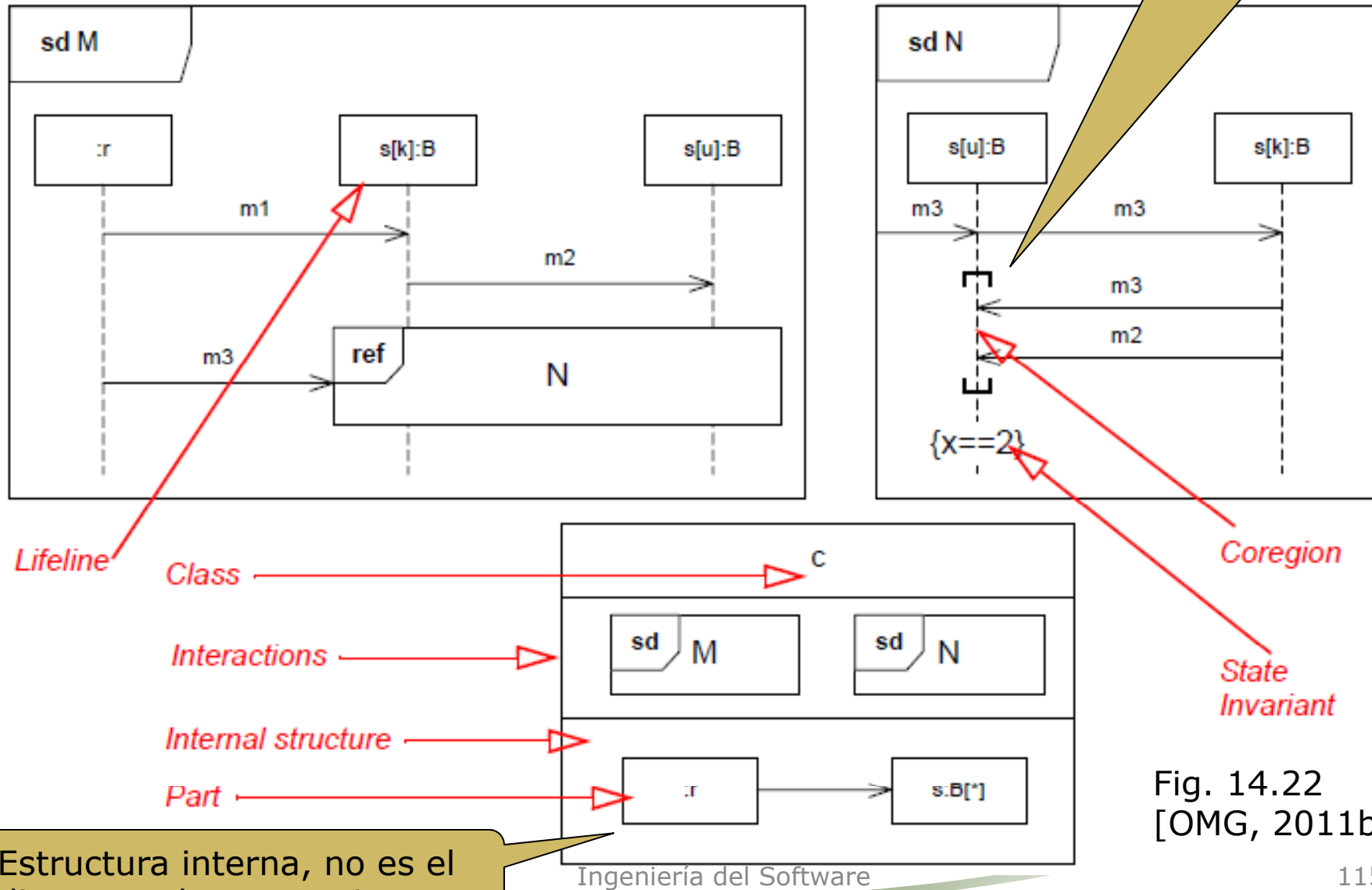
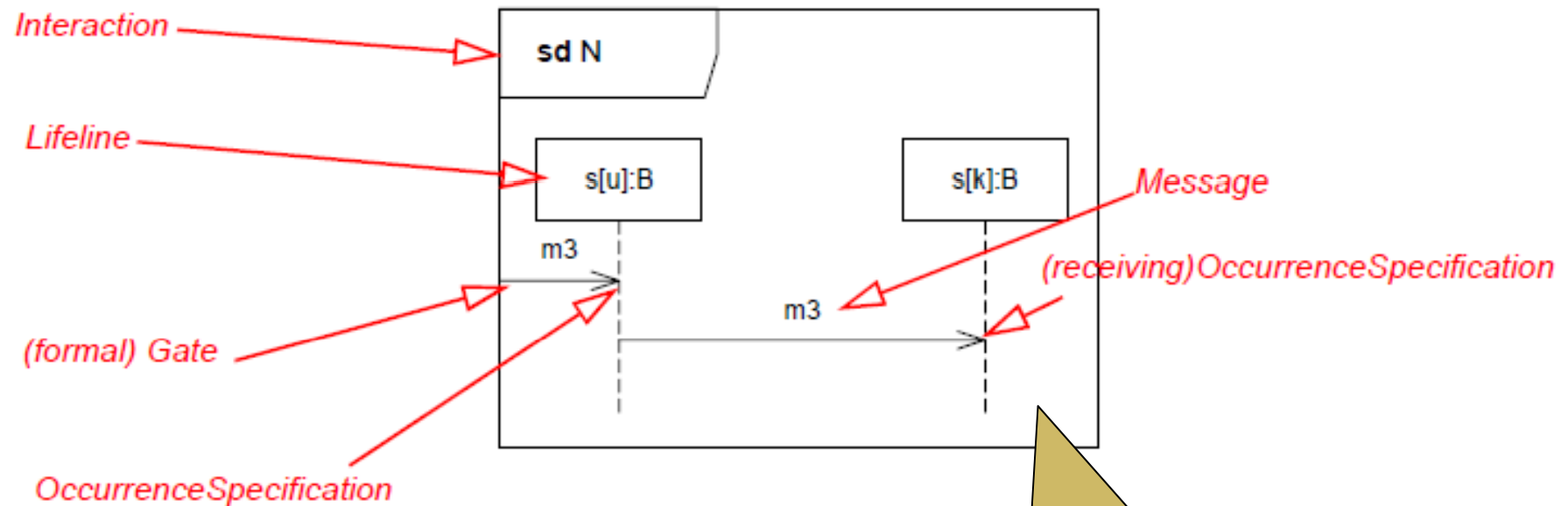


Fig. 14.22
[OMG, 2011b]



Especificaciones de ocurrencias



Las interacciones pueden definir puertas (*gates*) como puntos de entrada / salida con parámetros. Estos luego serán vinculados a los elementos actuales que ocurran en la interacción.

Las ocurrencias de especificación representan eventos, como llegadas de mensajes

Fig. 14.23 [OMG, 2011b]



Restricciones temporales

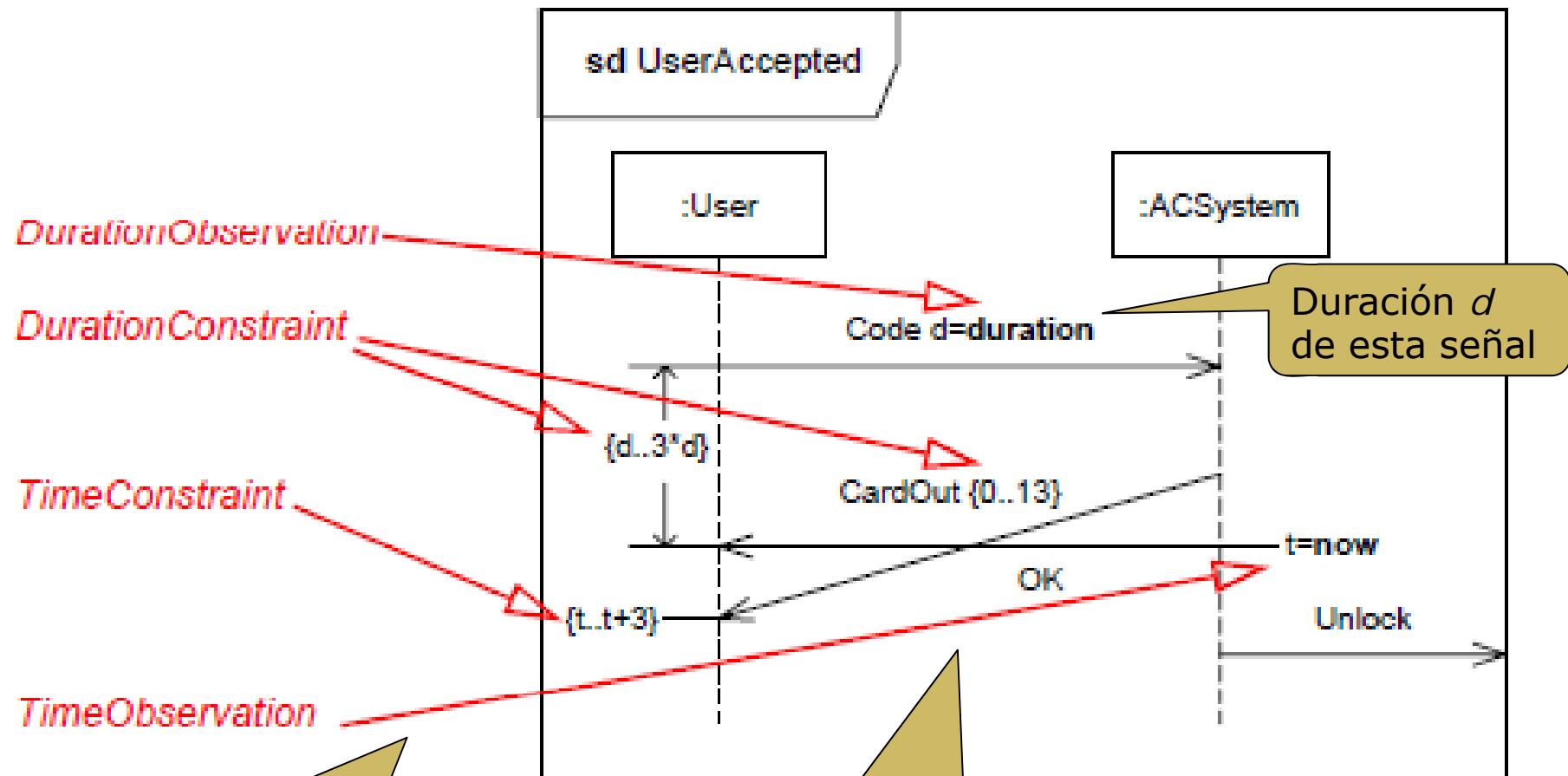


Fig. 14.26 [OMG, 2011b]

Restricción del intervalo de tiempo entre señales entre d y $3 \cdot d$.

La duración de CardOut ha de estar entre 0 y 13 unidades de tiempo





Fragmentos combinados

- Los *fragmentos combinados* u *operadores de control estructurados* son elementos de organización de los diagramas de secuencia.
 - Encapsulan porciones de un diagrama de secuencia, que se corresponden con el ámbito del operador de control definido en dicho fragmento.
 - Sólo desde UML 2.x
- Cada fragmento combinado tiene un *operador*, uno o más *operandos*, y cero o más *condiciones de guarda*.
 - El *operador* determina como se ejecutan los *operandos*.
 - La *guarda* es una condición booleana que determina si se ejecuta su *operando*.



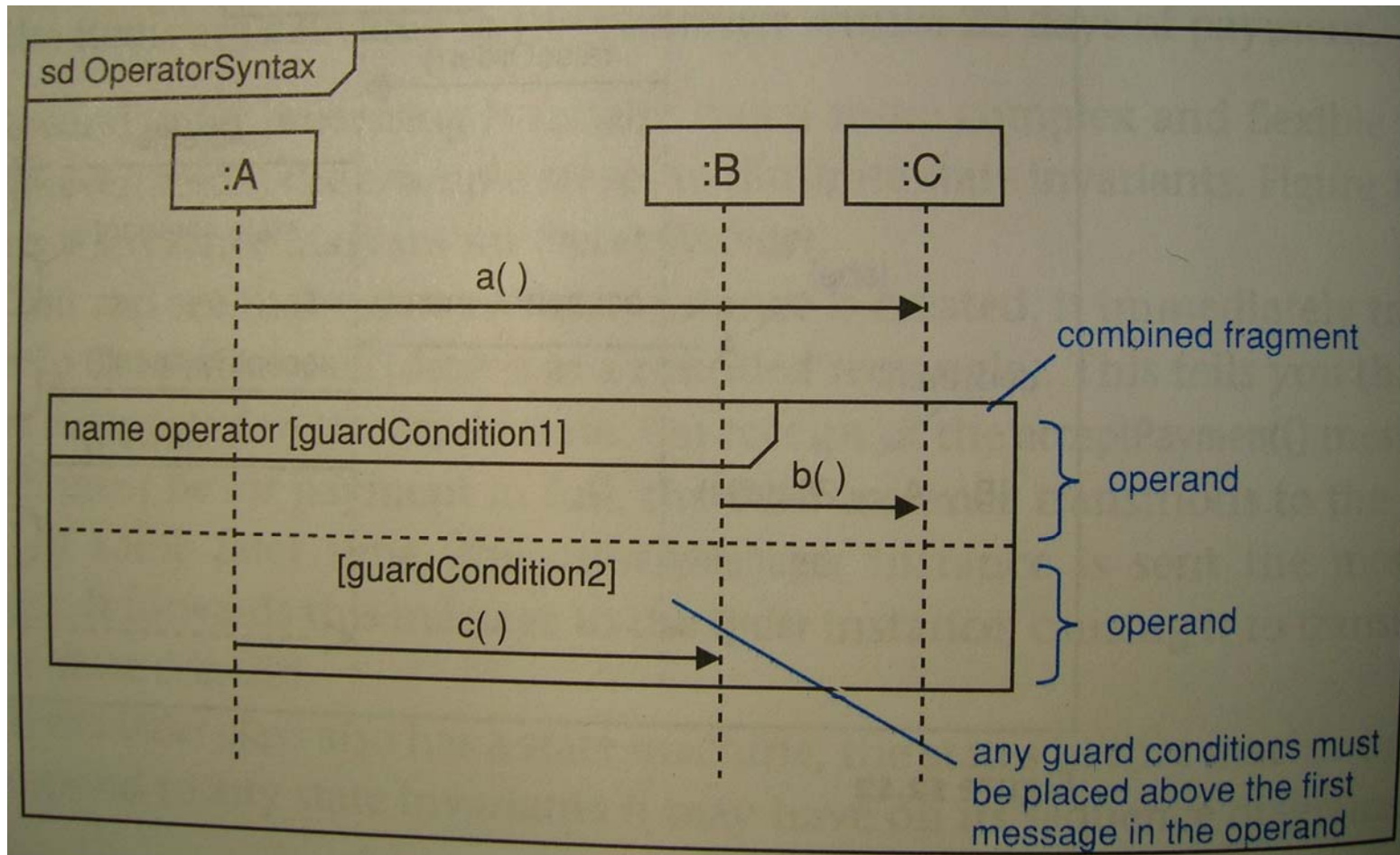


Operadores

- Los operadores más comunes son:
 - *opt* → ejecución opcional, que corresponde a un *if*
 - *alt* → ejecución condicional, que corresponde a un *switch*
 - *loop* → ejecución iterativa, que corresponde a un *for/while*
 - *par* → ejecución paralela
 - *critical* → región crítica que ha de ser tratada atómicamente



Ejemplo: fragmento combinado



Ejemplo: fragmento combinado – *opt*

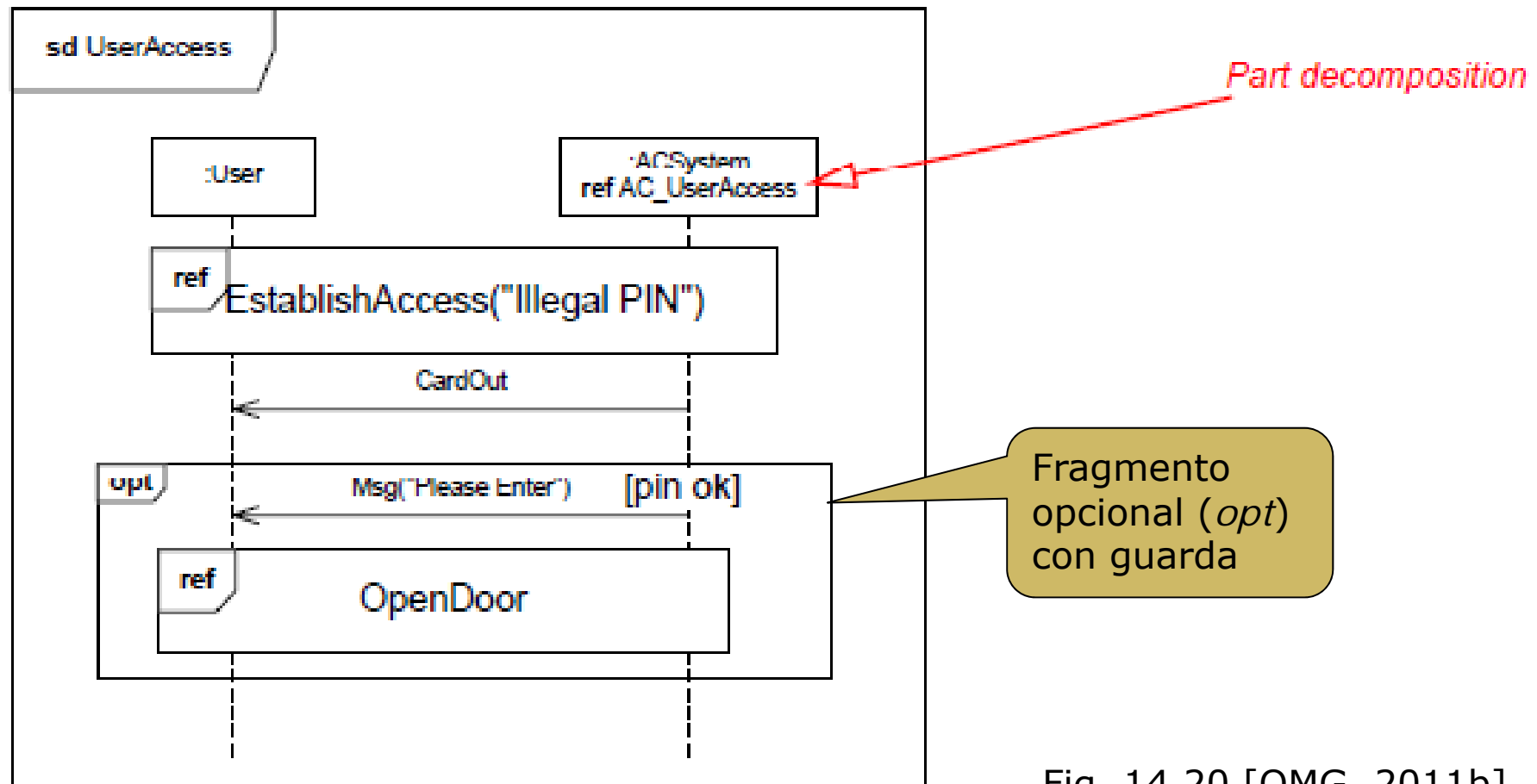
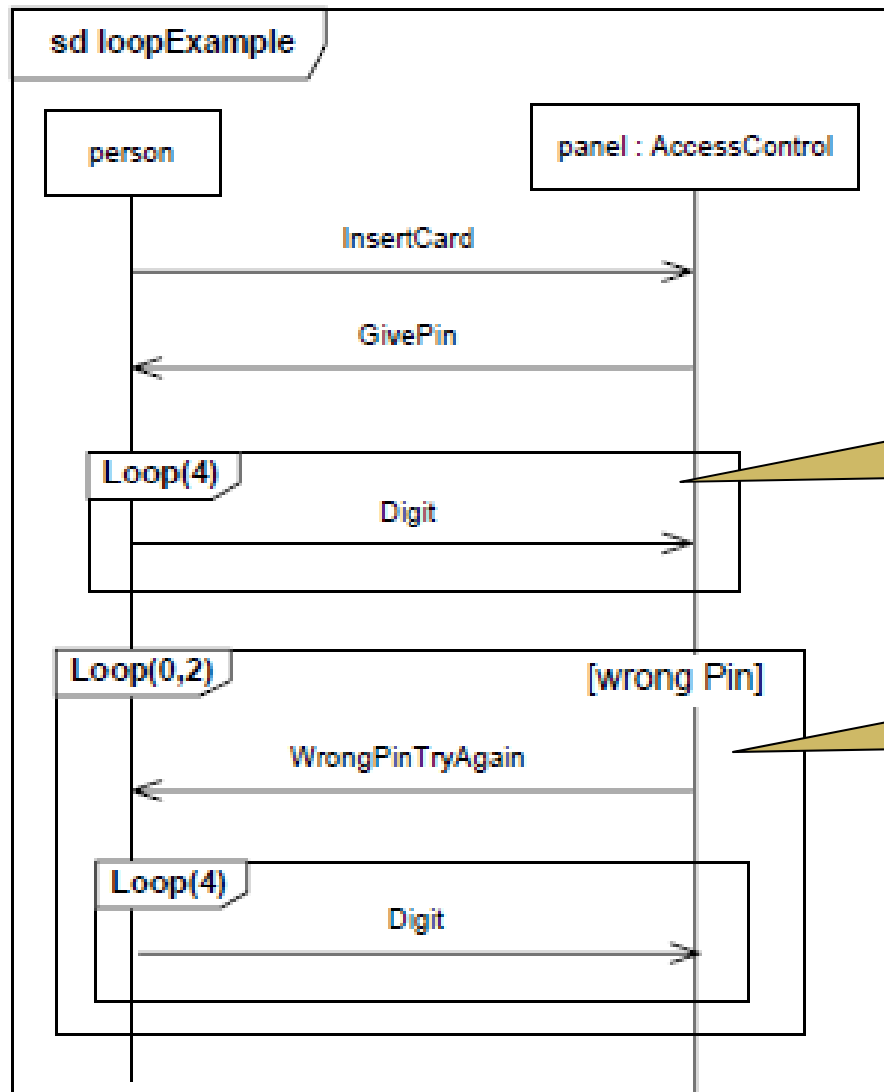


Fig. 14.20 [OMG, 2011b]



Ejemplo: fragmento combinado – *loop*



Fragmento iterativo (*loop*) con número fijo de iteraciones

Fragmento iterativo con rango de iteraciones y guarda

Fig. 14.10 [OMG, 2011b]



Ejemplo: fragmento combinado – *alt*

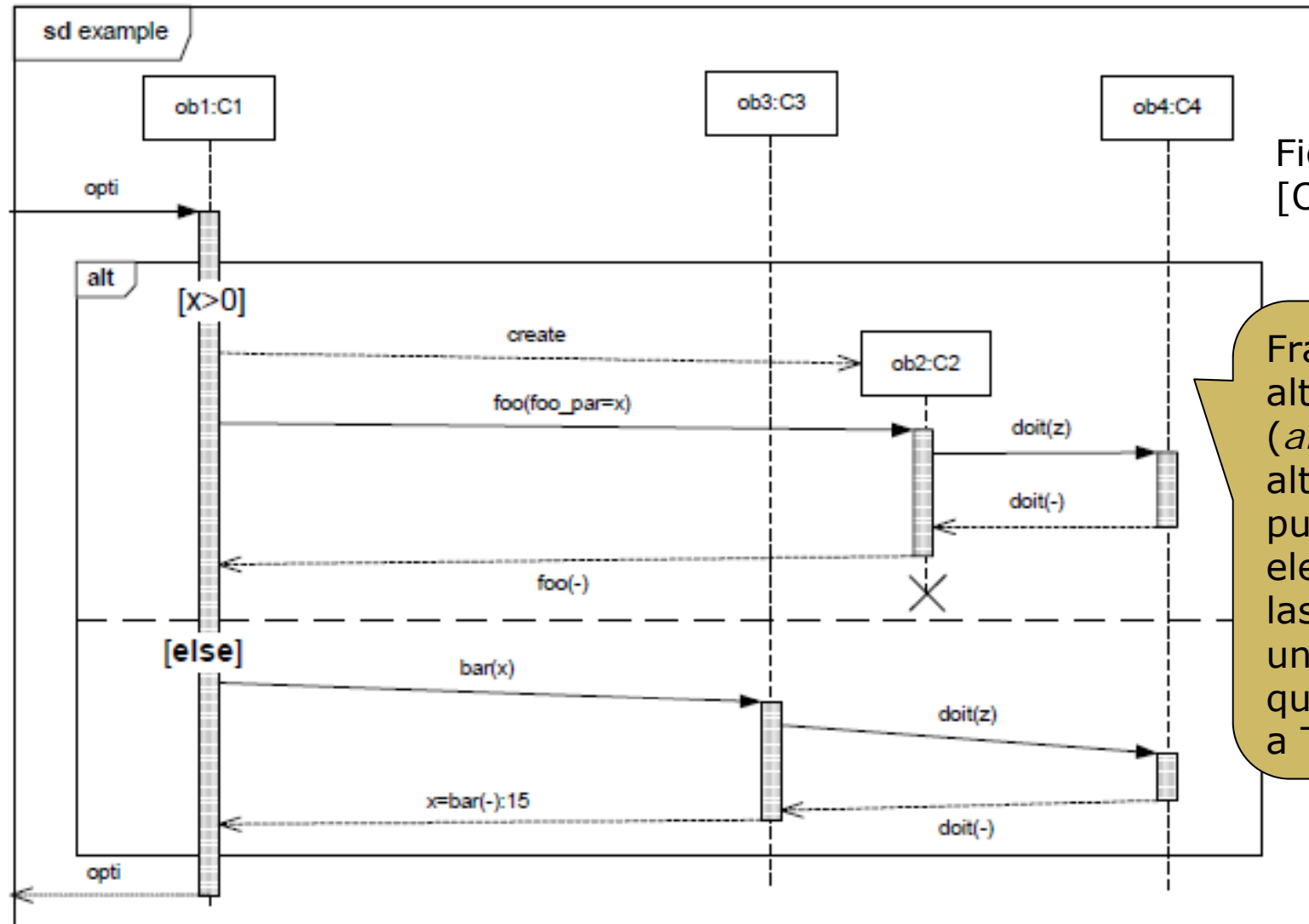


Fig. 14.11
[OMG, 2011b]

Fragmento alternativo (*alt*). Sólo una alternativa puede ser elegida entre las que tienen una guarda que se evalúa a TRUE.



Ejemplo: fragmento combinado – *par* y *critical*

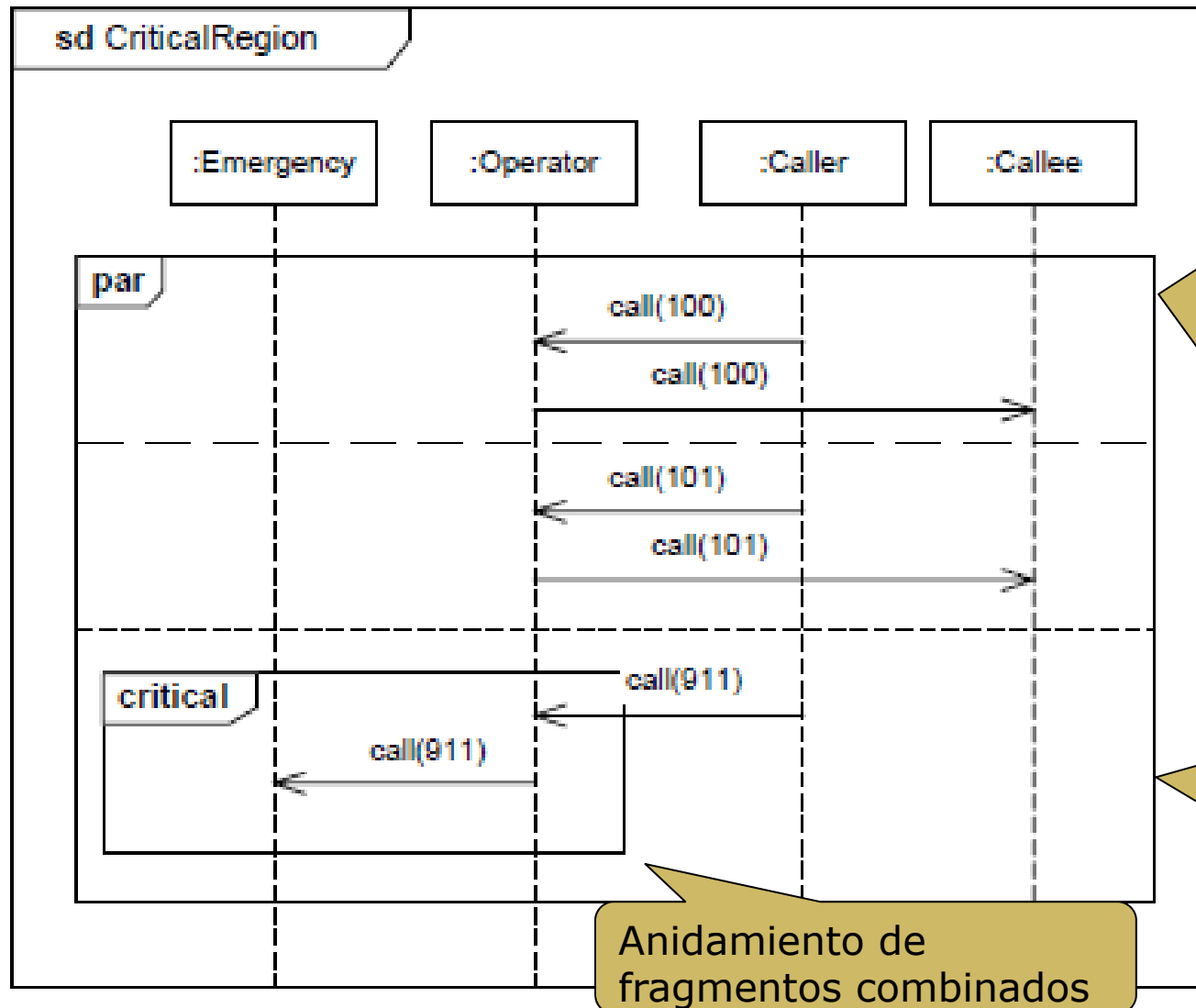


Fig. 14.9
[OMG, 2011b]

Fragmento paralelo (*par*). Los fragmentos que lo componen se intercalan de cualquier forma, aunque los componentes de cada operando respetan su orden.

Fragmento de región crítica (*critical*) que se ejecutará atómicamente.

Anidamiento de fragmentos combinados





DIAGRAMA DE COMUNICACIÓN





Diagramas de comunicación

- Un *diagrama de comunicación* muestra las interacciones entre objetos destacando su organización.
 - Objetos participantes en la interacción
 - Relaciones entre ellos
 - En menor medida, los mensajes que intercambian los objetos
- Son los diagramas de interacción apropiados para comprender todos los efectos que tiene un objeto y para el diseño de procedimientos.
- Respecto a los diagramas de secuencia:
 - Permiten indicar la visibilidad del enlace.
 - Hay que indicar explícitamente el número de secuencia del mensaje.
 - No permiten mostrar la información de retorno.
- En UML 1.x se denominaban de diagramas de colaboración.





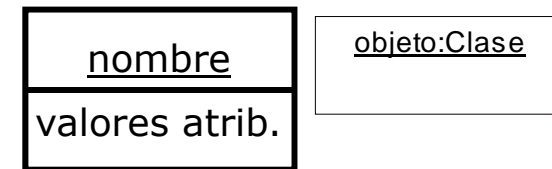
Diagramas de comunicación vs diagramas de objetos

- Aunque visualmente son similares, los diagramas de comunicación y de objetos tienen significados distintos:
 - Diagrama de objetos → foto del sistema / objeto prototípico
 - Diagrama de comunicación → interacciones entre los objetos del sistema para llevar a cabo una funcionalidad



Diagramas de comunicación: entidades

- Instancia



- Estímulo /mensaje

- Llamada o envío de información

- Ambos tienen el mismo significado que en los diagramas de secuencia.



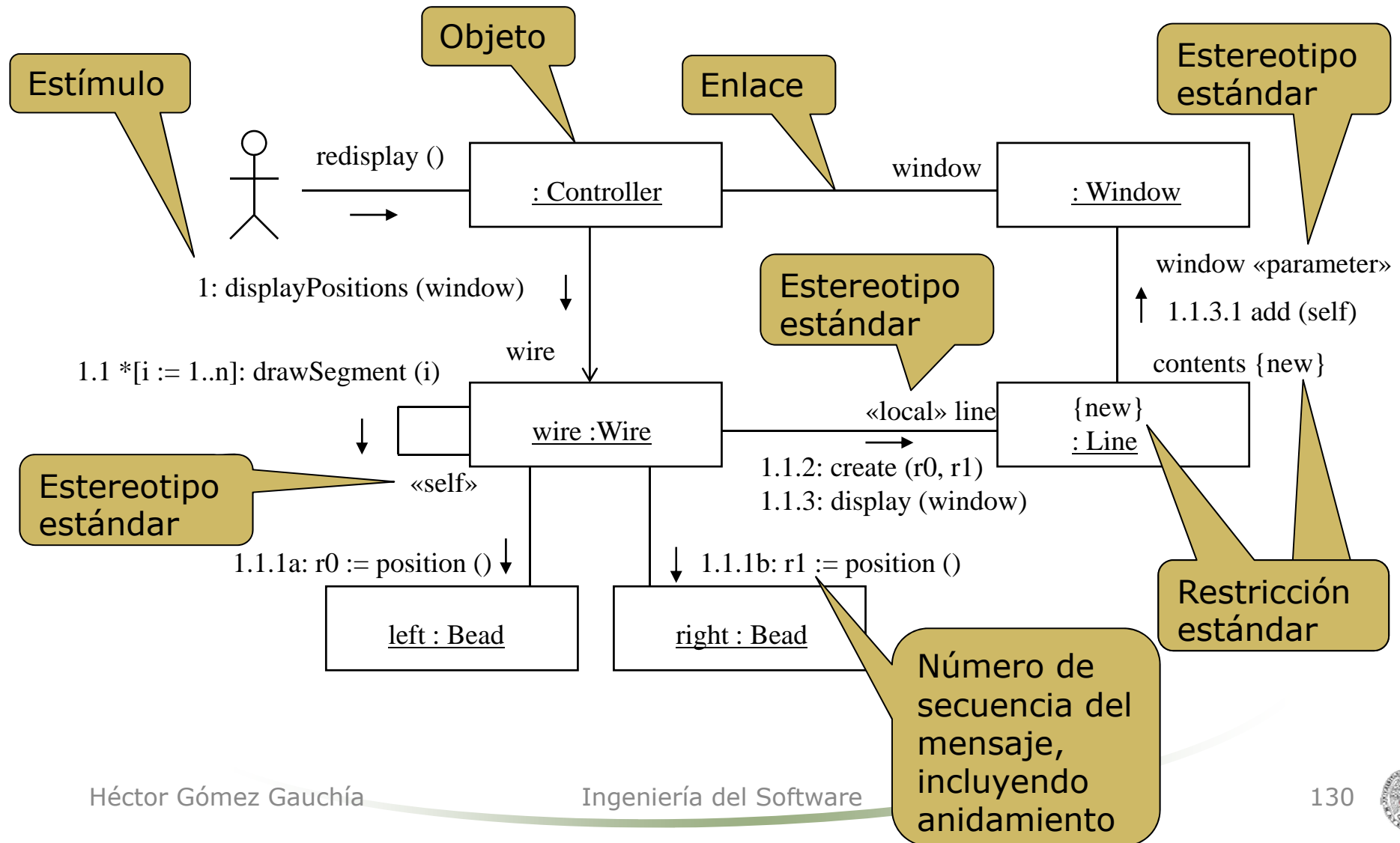


Diagramas de comunicación: relaciones

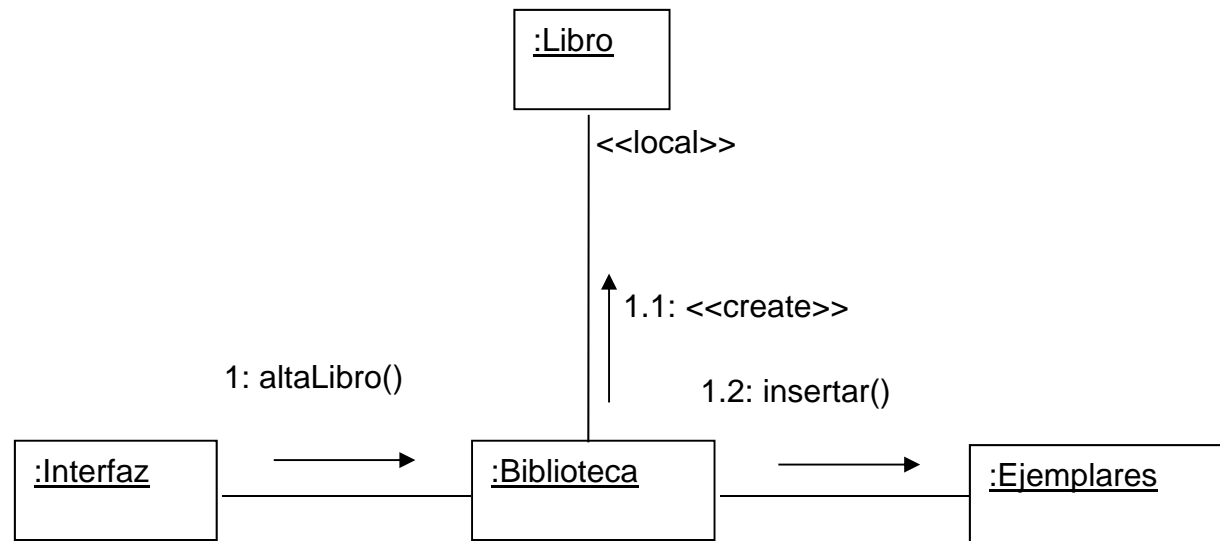
- Enlace
 - Conexión entre instancias
 - Los enlaces pueden tener atributos con valor.

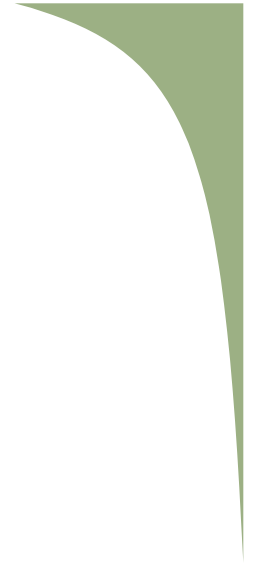


Ejemplo: diagrama de comunicación



Ejemplo: diagrama de comunicación





PAUTAS PARA MODELOS DE INTERACCIÓN





Consejos para un buen diagrama de interacción

- Indicar el contexto de la interacción.
- Expresar el flujo de izquierda a derecha y de arriba abajo.
- Poner las instancias activas en la izquierda arriba y las pasivas a la derecha más abajo.





Cuándo hacer un diagrama de interacción

- General
 - Especificar cómo interaccionan unas instancias con otras.
 - Identificar las interfaces de las clases.
 - Distribuir los requisitos y las responsabilidades.
- Diagramas de secuencia
 - Ilustrar escenarios típicos con el orden explícito de los estímulos.
 - Modelar tiempo real.
- Diagramas de colaboración
 - Coordinación de la estructura y control de los objetos.
 - Para concentrarse en los efectos sobre las instancias.
 - Ayudan a agrupar objetos en módulos ya que las relaciones entre objetos son visibles.





Ejercicio de diagramas de interacción

- Supóngase un sistema de reservas y el siguiente caso de uso:
 - Actores: Viajero, Sistema de Reservas Aéreas, Base de Datos de Clientes
 - Precondiciones: El viajero ha entrado en el sistema (login)
 - Escenario normal:
 - El viajero selecciona la opción “Cambiar itinerario de vuelo”.
 - El sistema obtiene la cuenta del viajero y el itinerario de vuelo de la base de datos de clientes.
 - El sistema presenta al viajero el itinerario de vuelo y le pide que seleccione el segmento del itinerario que quiere cambiar.
 - El viajero selecciona el itinerario.
 - El sistema pide al viajero información nueva de salida y destino.
 - El viajero le da esa información.
 - Si hay vuelos disponibles, entonces ...
 - ...
 - El sistema muestra un resumen de la transacción.
 - Escenario alternativo
 - Si no hay vuelos disponibles, entonces ...



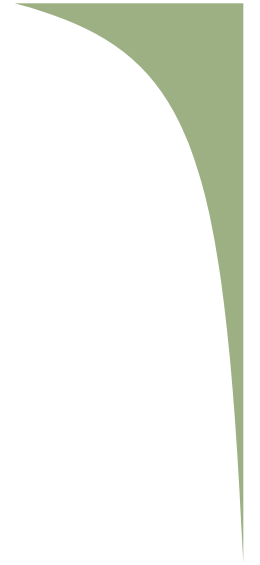


DIAGRAMA DE MÁQUINA DE ESTADOS





Diagrama de máquina de estados

- Un *diagrama de estados* muestra una máquina de estados (autómata), destacando el flujo de control entre estados.
 - Una *máquina de estados* es una especificación de las secuencias de *estados* por las que pasa un objeto a lo largo de su vida en respuesta a *eventos*, junto con las respuestas a esos *eventos*.
- Son los diagramas apropiados para representar el comportamiento interno de una clase o del sistema.



Diagrama de máquina de estados: entidades

- Estado
 - Es una condición o situación en la vida de un objeto durante la cual satisface una condición, realiza alguna actividad, o espera algún evento.
- Estado inicial
 - Realmente es un pseudoestado.
- Estado final
 - Realmente es un pseudoestado.
- Evento
 - Es la especificación de un acontecimiento significativo que ocupa un lugar en el tiempo y en el espacio.
 - Es la aparición de un estímulo.

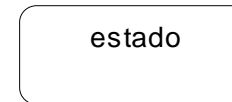




Diagrama de máquina de estados: relaciones

- Transición

- Es una relación entre dos *estados* que indica que un objeto que esté en el primer estado realizará ciertas acciones y entrará en el segundo estado cuando ocurra un evento especificado y se satisfagan unas condiciones específicas.

evento[condición]/acción →



Ejemplo: diagrama de máquina de estados

- Protocolo para puerta

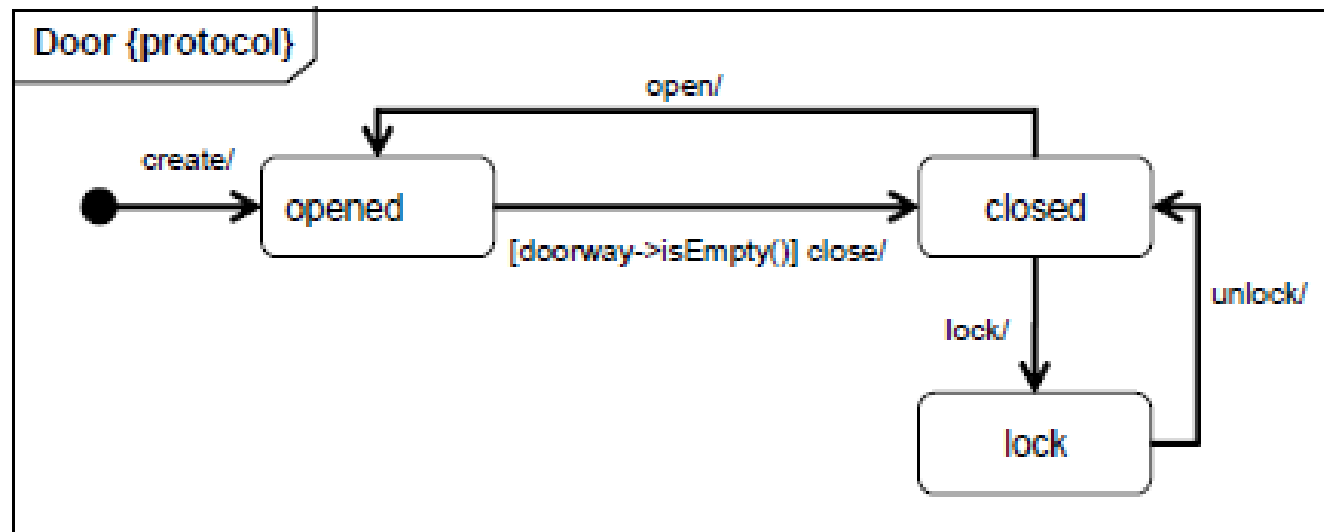
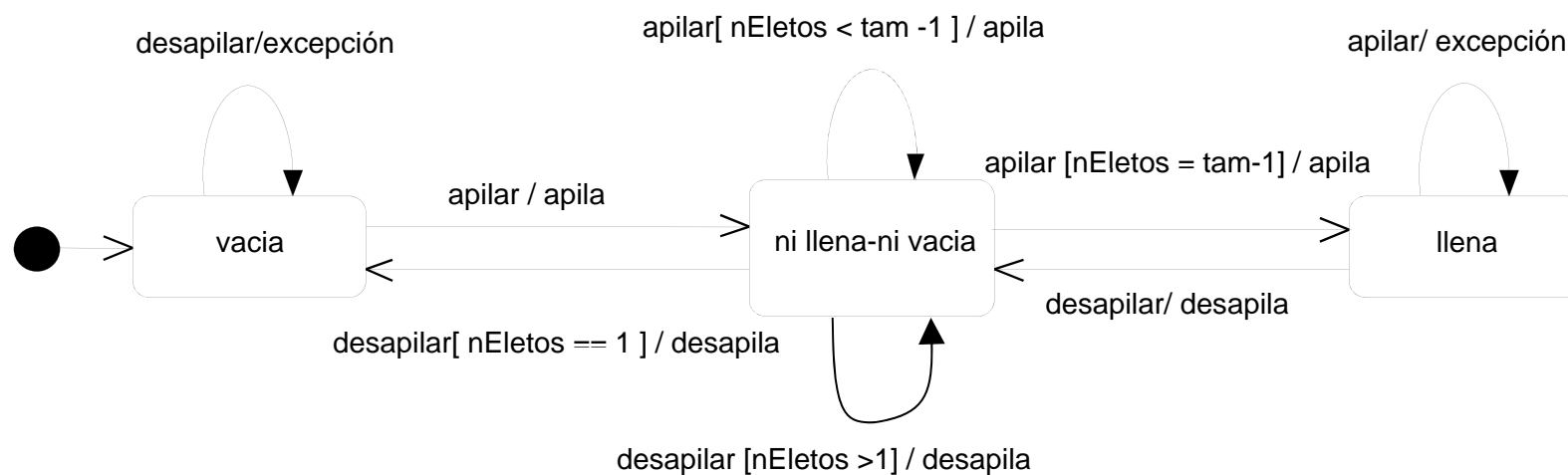


Fig. 15.12 [OMG, 2011b]

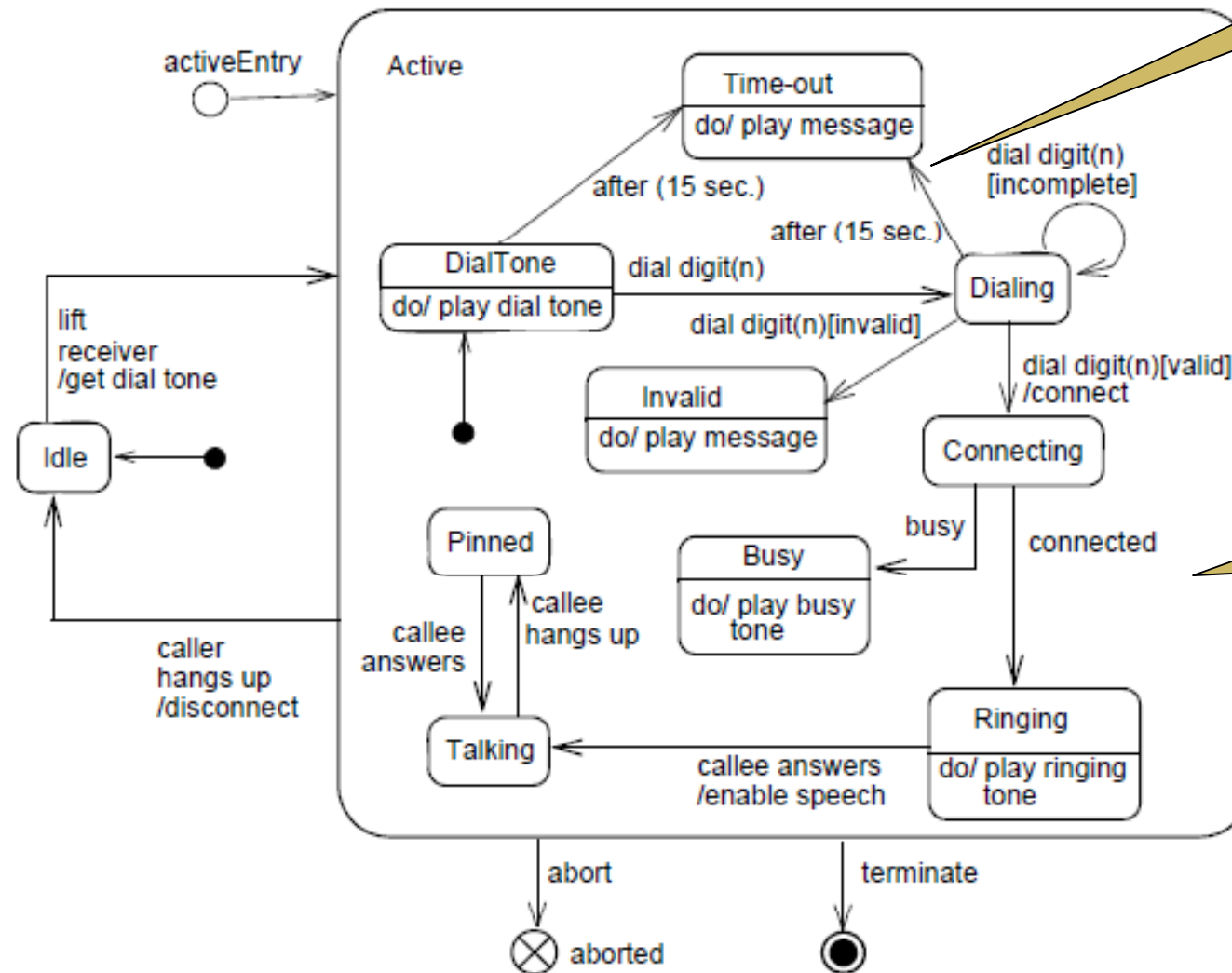


Ejemplo: diagrama de máquina de estados

- Pila estática



Estados anidados



Transición
disparada por
temporización

Estados
anidados

Fig. 15.41
[OMG, 2011b]





Concurrencia

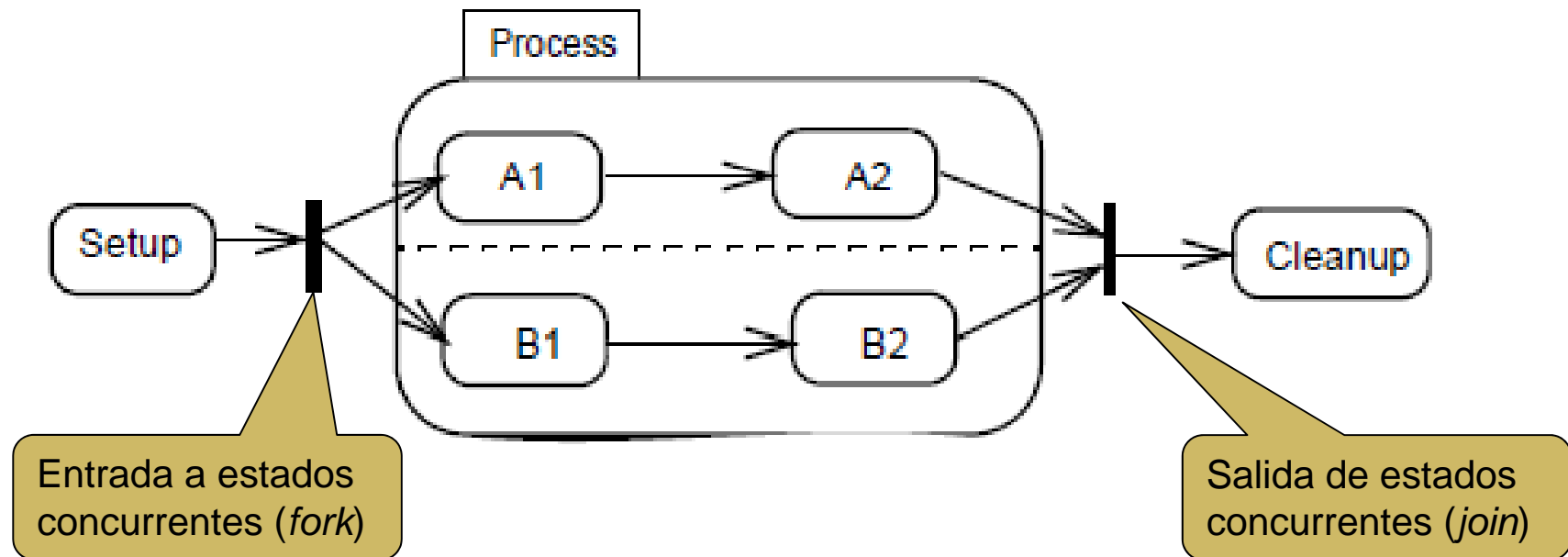


Fig. 15.25 [OMG, 2011b]





Cuándo usar diagramas de máquina de estados

- Para capturar el comportamiento dinámico de una entidad.
 - Cuando tiene cierta complejidad.
- Especialmente adecuado para modelar sistemas reactivos.
 - La mejor forma de caracterizar su comportamiento es señalar su respuesta a los eventos lanzados desde fuera de su contexto.
 - Orientados a eventos
 - Interfaces de usuario
 - Dispositivos
 - Protocolos de comunicación
 - Modelo de servidor
- Objetos cuyo comportamiento dependa de su *pasado*.
 - Su comportamiento varíe significativamente en función de su estado actual.





DIAGRAMA DE ACTIVIDADES





Diagrama de actividades

- Un *diagrama de actividades* muestra un flujo de actividades.
 - Mostrar las acciones realizadas por una operación.
 - Mostrar el trabajo interno de un objeto.
 - Mostrar cómo se desarrollan un conjunto de acciones y cómo afectan a los objetos que las realizan.
 - Mostrar cómo se realiza una instancia de un caso de uso en términos de acciones y cambios de estado de los objetos.
 - Mostrar un modelo de negocio en términos de trabajadores (actores), flujos de trabajo, organización y objetos (factores físicos e intelectuales usados en los negocios).
- Su representación es similar a los diagramas de máquinas de estado pero no hay que confundirlos.
 - Sus transiciones suceden al finalizar una actividad y no activadas por un evento externo.
 - Soportan concurrencia dinámicamente.



Diagrama de actividades: entidades (1/3)

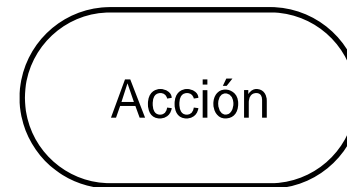
- Actividad

- Es una ejecución no atómica.
 - Las actividades producen finalmente *acciones*.
 - Contenedor de otros elementos
- Es interrumpible por eventos.



- Acción

- Es una ejecución atómica que produce un cambio en el estado del modelo o que devuelve un valor.
 - Ej. Invocar una operación de un objeto o ejecutar una acción del usuario.
- No es interrumpible por eventos.



- Llamada de actividad

- Inicia otro grafo de actividad sin usar una operación.
- Usado para descomposición funcion

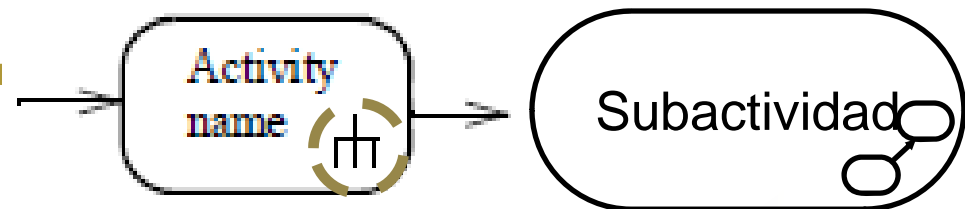


Diagrama de actividades: entidades (2/3)

- Estado inicial
 - Estado de inicio del grafo de acciones.
- Estado final
 - Estado de terminación del grafo de acciones.
 - No se atienden más eventos y finalizan todos los flujos en la actividad si hay varios.
- Decisión
 - Bifurcación de la secuencia de acciones basada en condiciones.
- *Fork*
 - Bifurcación incondicional de la secuencia de acciones en múltiples flujos concurrentes.
- *Join*
 - Unión incondicional de múltiples flujos de acciones concurrentes.

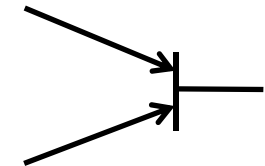
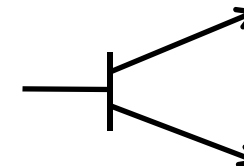
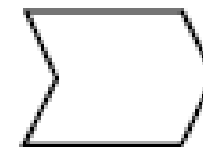
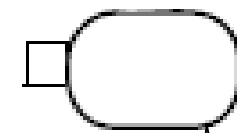
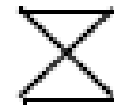
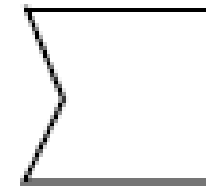


Diagrama de actividades: entidades (3/3)

- **Acción de aceptación de evento**
 - Acción especial que comienza cuando lo hace la actividad contenedora.
 - Permanece habilitada después de aceptar un evento.
- **Nodos objeto**
 - Representan pasos de instancias de clasificadores.



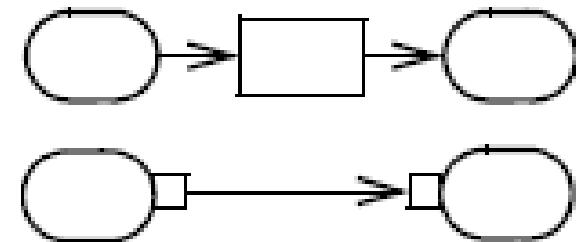
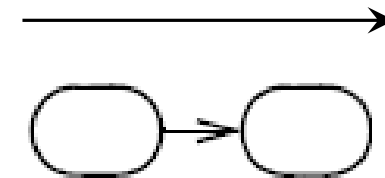
Señal

Acción con
nodo objeto



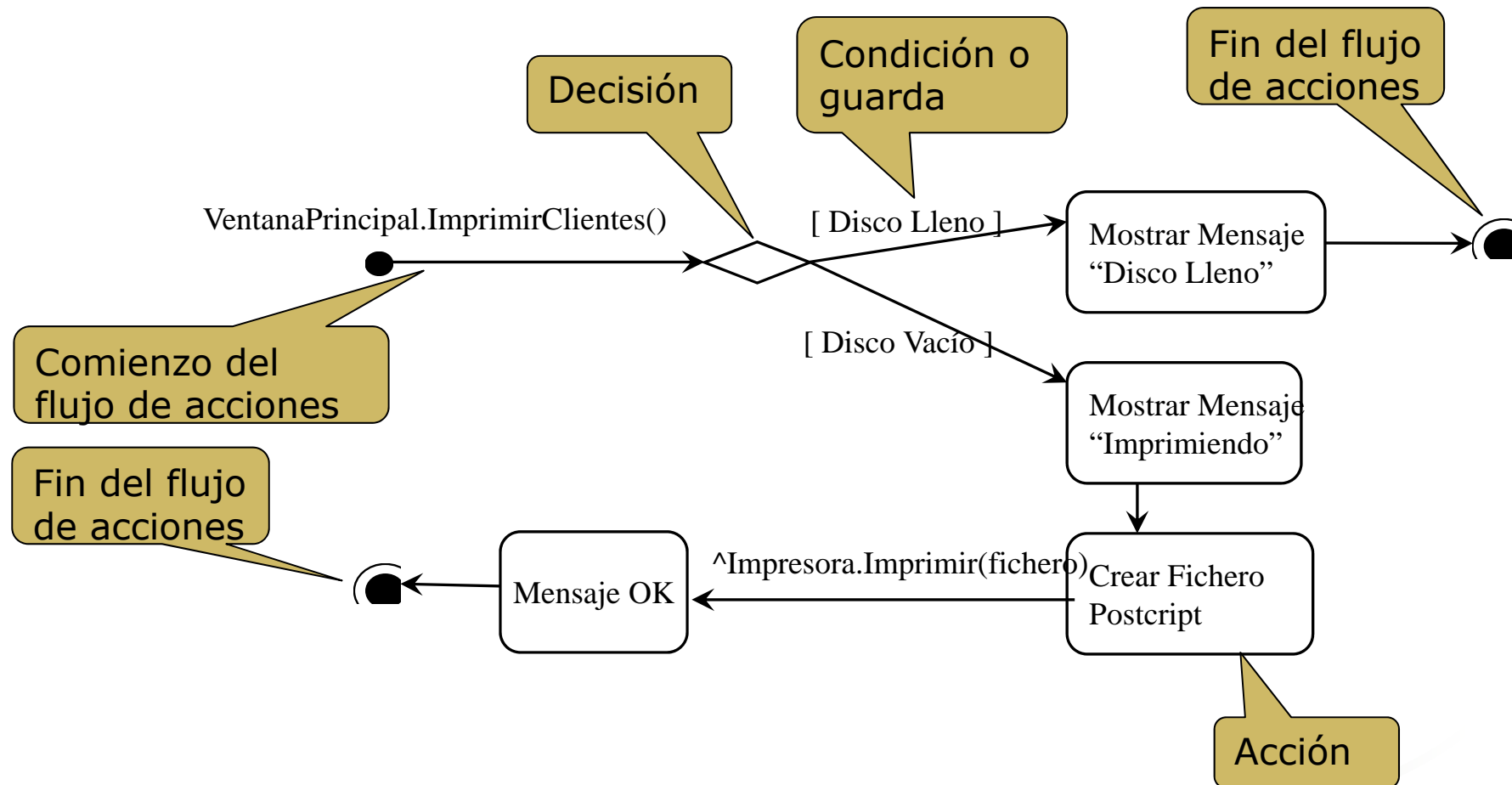
Diagrama de actividades: relaciones

- Flujo de control
 - Es una relación entre dos *acción* que indica que se puede pasar del fin de la ejecución de la primera acción al comienzo de la ejecución de la segunda.
- Flujo de objeto



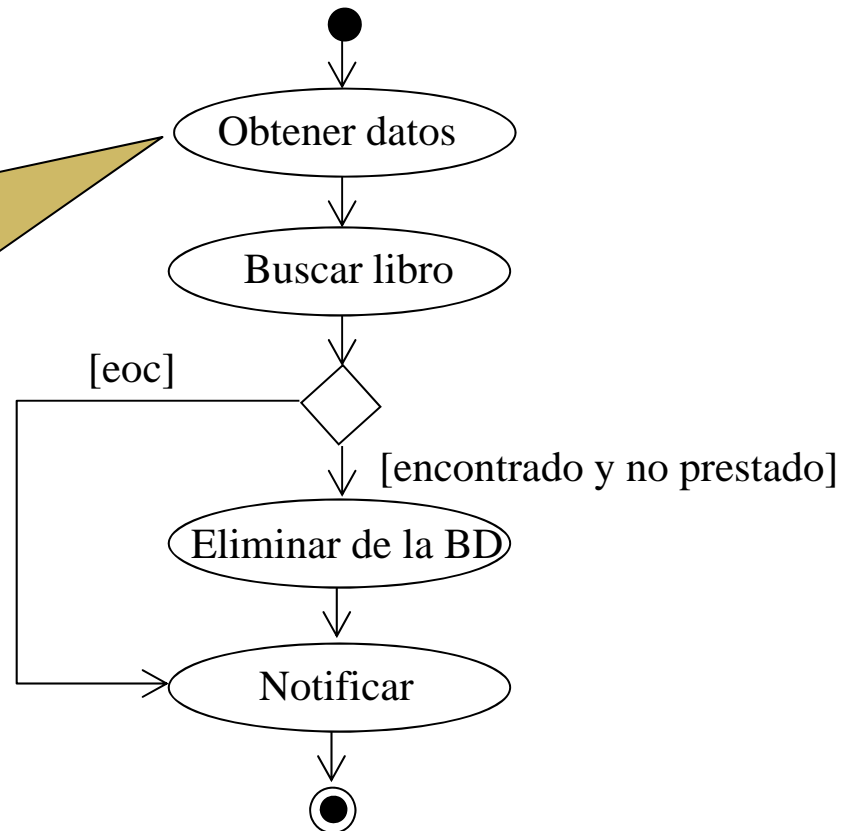
Ejemplo: diagrama de actividades

- Realización de una operación en un objeto



Ejemplo: diagrama de actividades

Las acciones se dibujan como óvalos frecuentemente para diferenciarlas visualmente de los estados. El estándar no lo hace.



Ejemplo: parámetros de entrada y salida

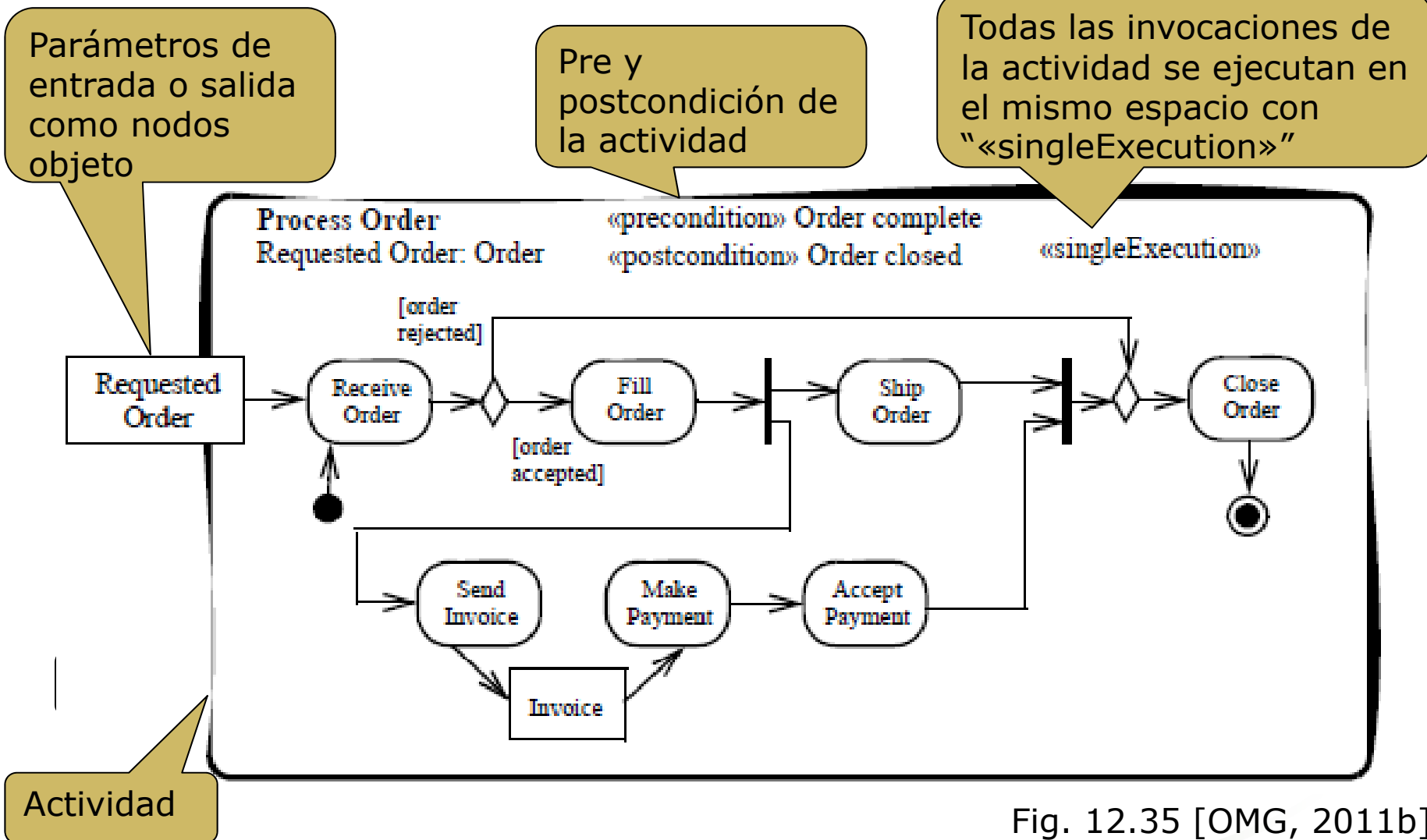


Fig. 12.35 [OMG, 2011b]



Ejemplo: señales y acciones de aceptación de eventos

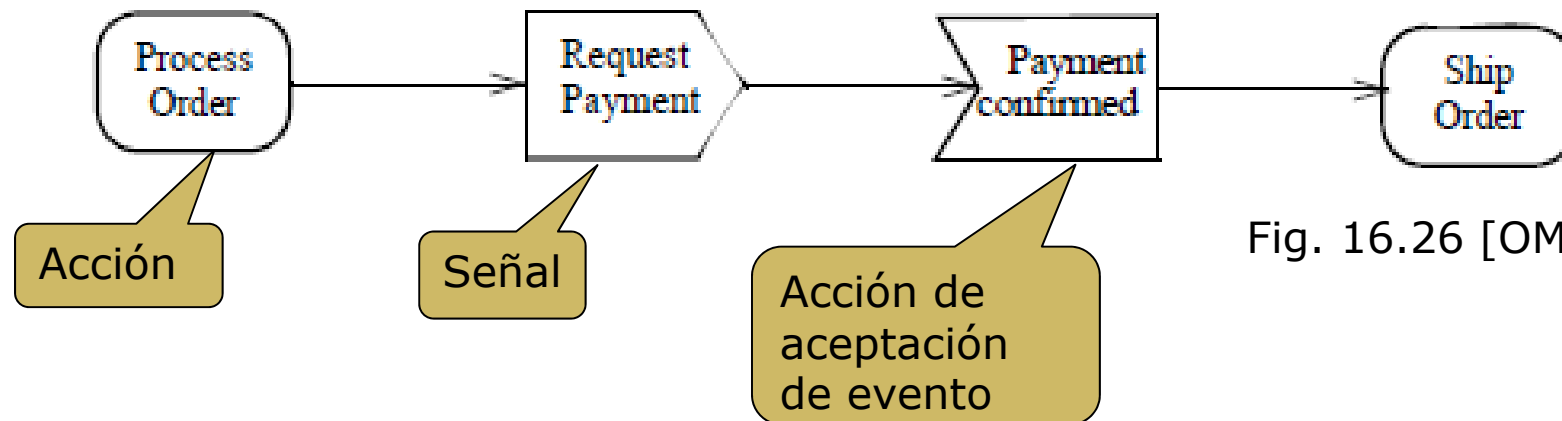
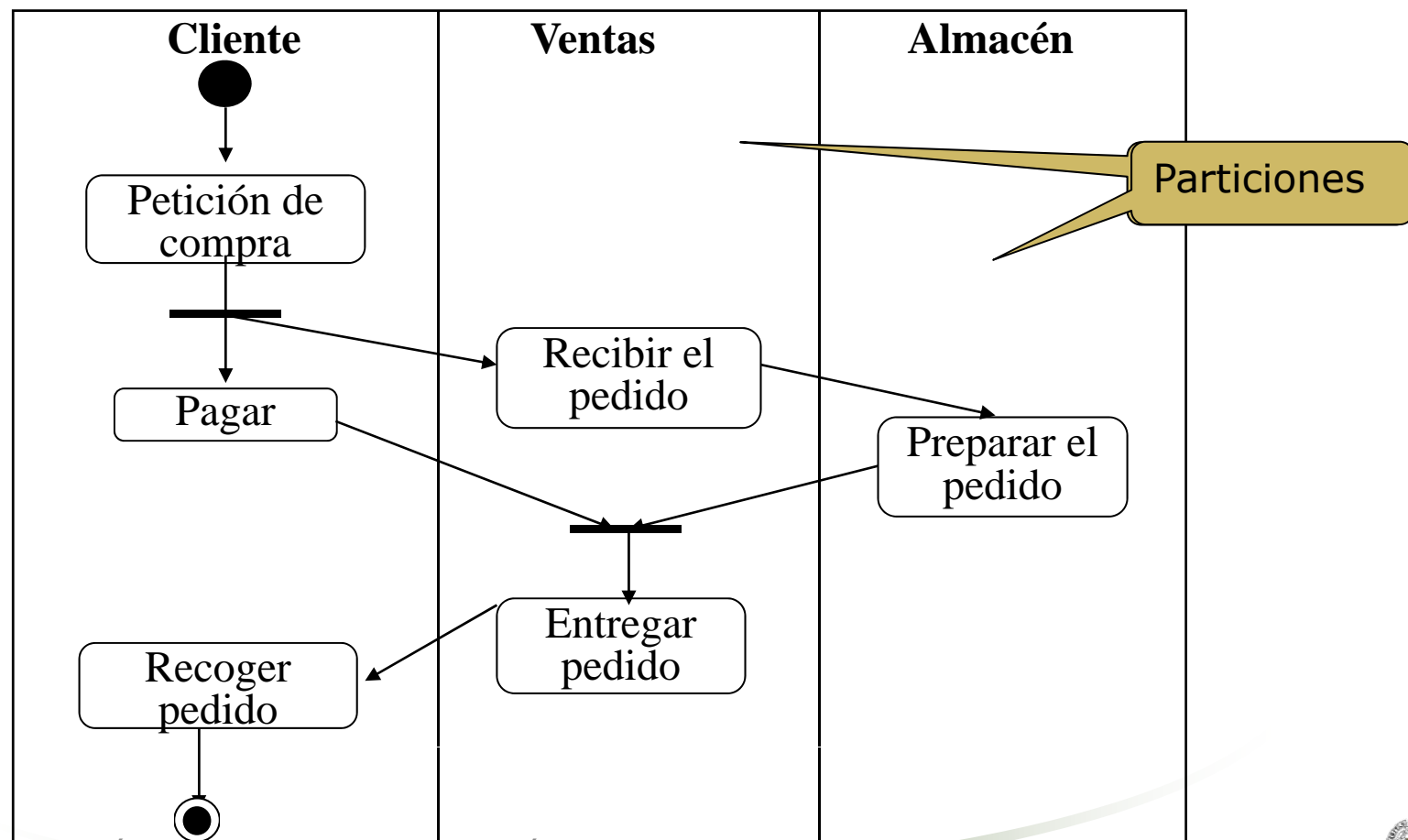


Fig. 16.26 [OMG, 2011b]



Particiones

- Particiones (*swimlanes*)
 - Definición de una transacción en múltiples objetos



Ejemplo: particiones multidimensionales

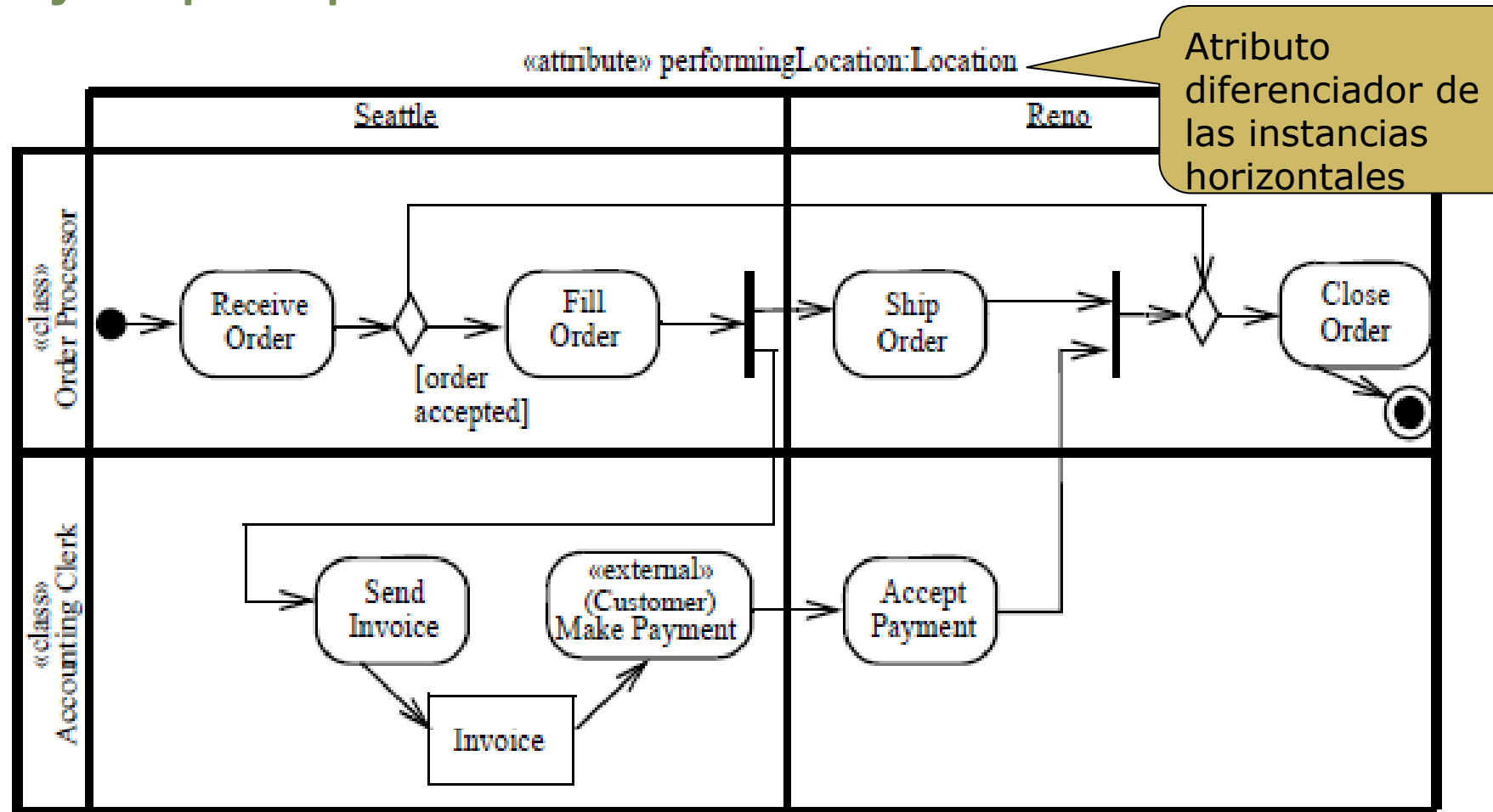


Fig. 12.61 [OMG, 2011b]





Consejos para un buen diagrama de actividad

- El flujo de control y el flujo de objetos no están separados.
 - Ambos se modelan con transición entre actividades.
 - Se pueden mezclar flujos de control/objetos en el mismo diagrama.
- Definir diagramas bien anidados.
 - Ello asegura que habrá una máquina de estados (actividades) ejecutable correspondiente.
 - Evitar el *goto*.



Consejos para un buen diagrama de actividad

Diagrama mal anidado

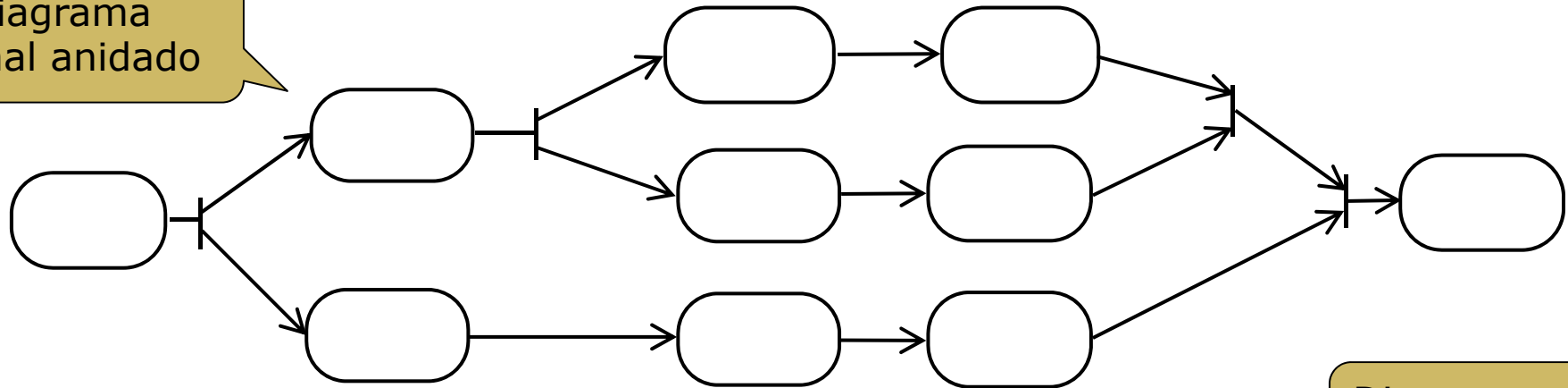
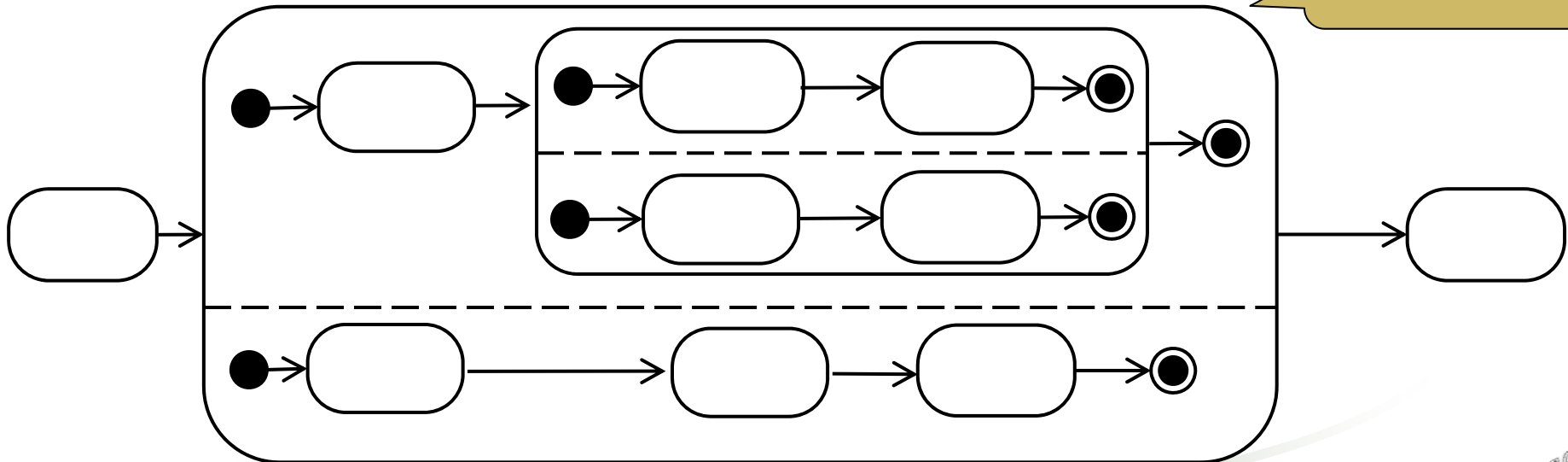


Diagrama bien anidado





Cuándo usar diagramas de actividad

- Aplicaciones que necesiten modelos de flujo de control o de flujo de datos/objetos.
 - en vez de modelos dirigidos por eventos (para los que son más apropiadas las máquinas de estado).
 - Ej. modelos de procesos de negocio y *workflows*.
- Cuando el comportamiento
 - no depende mucho de eventos externos
 - los pasos se ejecutan hasta acabar, y no son interrumpidos por eventos
 - requiere flujo de objetos/datos entre los pasos
- Se construyen normalmente durante el análisis para ver qué actividades ocurren en vez de qué objetos son responsables de ellas.
 - Posteriormente se pueden usar particiones para asignar esas responsabilidades.





Ejercicio: Diagrama de actividad

- Definir un diagrama de actividad para la operación de reserva de plaza de hotel.





PAUTAS PARA MODELOS DE COMPORTAMIENTO

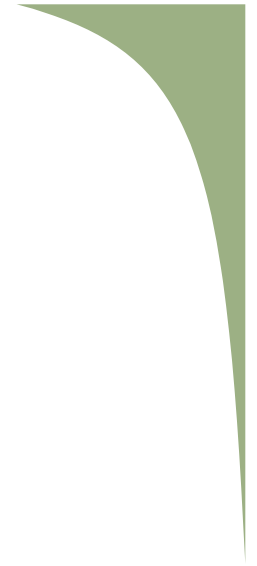




Qué diagramas de comportamiento utilizar

- Utilizar *diagramas de interacción* (secuencia o comunicación) cuando haya interés en conocer el comportamiento de varios objetos dentro de un caso de uso.
- Utilizar *diagramas de estado* cuando haya interés en conocer el comportamiento de un único objeto dentro de múltiples casos de uso.
- Utilizar *diagramas de actividad* para observar el comportamiento a lo largo de varios *casos de uso*, o muchos *hilos de control*.





MODELO DE IMPLEMENTACIÓN





Modelo de implementación

- Visión de implementación y despliegue del sistema.
 - Desarrollado por analistas, diseñadores y programadores.
- Muestra la estructura del sistema entregado y su entorno de ejecución.
 - Componentes del producto
 - Infraestructura del despliegue del producto
 - La estructura de los anteriores
 - Sus relaciones, tanto entre componentes del producto y de la infraestructura como entre los elementos de estos grupos
- Se definen mediante diagramas de implementación:
 - Diagrama de componentes
 - Diagrama de despliegue (o implantación)





Diagramas de implementación

- Muestran los aspectos de implementación del modelo.
 - Estructura del código fuente y objeto
 - Estructura de la implementación en ejecución
- Dos tipos:
 - Diagrama de componentes
 - Muestra la organización y dependencias entre los componentes software.
 - Clases de implementación
 - Artefactos binarios, ejecutables y ficheros de scripts
 - Diagrama de despliegue (o implantación)
 - Configuración de los elementos de proceso (*nodo*) en tiempo de ejecución.
 - Descripción de los componentes software, procesos y objetos que viven en ellos.
 - Distribución de componentes en los elementos de proceso.





ARTEFACTOS





Artefactos

- Un *artefacto* representa un elemento concreto del mundo real.
 - Es la especificación de dicho elemento concreto.
- Ejemplos de artefactos son:
 - Ficheros fuente
 - Ficheros ejecutables
 - Scripts
 - Tablas de bases de datos
 - Documentos
- Por su naturaleza, los artefactos pueden aparecer tanto en *diagramas de componentes* como en *diagramas de despliegue*.
- Una *instancia de artefacto* es una instancia particular de un *artefacto* concreto en una determinada instancia de *nodo*.





Artefactos

- Un artefacto puede proporcionar la manifestación física de cualquier elemento UML.
 - Dicha manifestación se representa como una dependencia estereotipada con `manifest`.
- Los artefactos pueden presentar dependencias entre ellos.



Ejemplo: artefacto

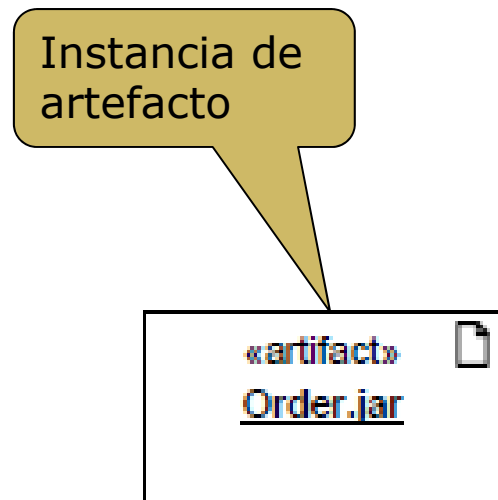


Fig. 10.6 [OMG, 2011b]

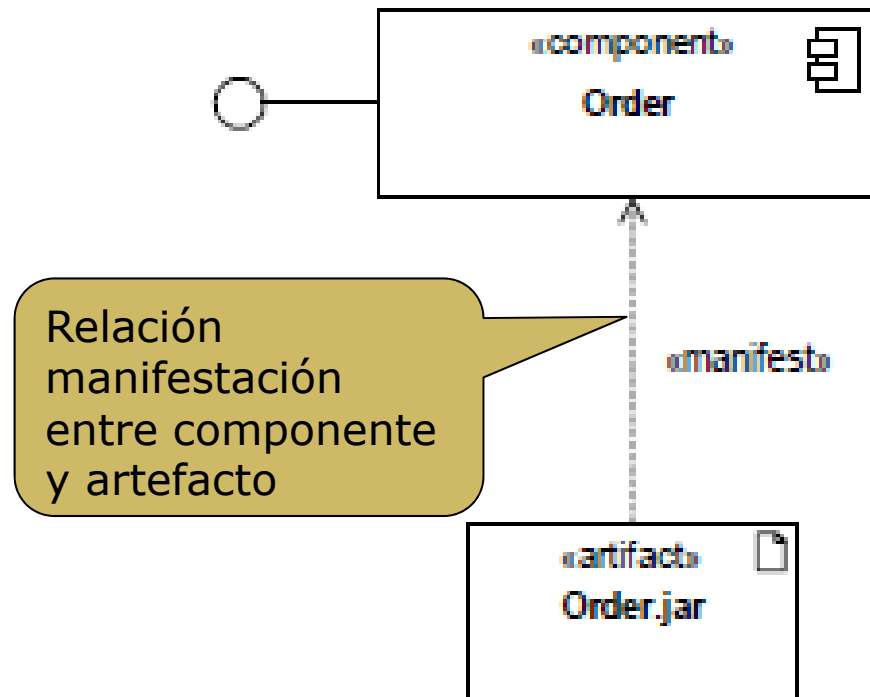


Fig. 10.7 [OMG, 2011b]



Artefactos estándar en UML 2.x

- `file` → archivo físico
- `deployment spec` → especificación de detalles de despliegue
 - Ej. `web.xml` en J2EE
- `document` → fichero genérico que contiene información
- `executable` → fichero ejecutable
- `library` → librería estática o dinámica
- `script` → script ejecutable por un intérprete
- `source` → fichero compilable en ejecutable





DIAGRAMA DE COMPONENTES





Diagrama de componentes

- Un *diagrama de componentes* muestra las partes internas, los conectores y los puertos que implementan los componentes.
- Un *componente* es una parte modular de un sistema que encapsula sus contenidos y cuya manifestación es reemplazable dentro de su entorno.
 - Actúa como una caja negra cuyo comportamiento externo está completamente definido por sus interfaces proporcionadas e implementadas.
 - A causa de esto, un componente puede ser reemplazado por cualquier otro que soporte su mismo protocolo.





Componente

- Los componentes pueden representar:
 - Algo que puede ser instanciado en ejecución, como un EJB.
 - Una construcción lógica, como un subsistema.
- Los componentes:
 - Se manifiestan físicamente como uno o más *artefactos*.
 - Pueden tener atributos y operaciones.
 - Pueden participar en relaciones de asociación y generalización.
- Aunque la notación no varía en exceso entre UML 1.x y UML 2.x, la semántica sí:
 - Componentes UML 1.x están más cercanos al concepto de *artefacto*.
 - Componentes UML 2.x son elementos del diseño caracterizados por sus puertos, y que en ejecución se instanciarán (ej. EJBs) o se corresponderán con varios objetos (ej. subsistemas).



Diagrama de componentes: entidades (1/2)

- **Componente**
 - Representa una parte modular de un sistema que encapsula sus contenidos y cuya manifestación es reemplazable dentro de su entorno.
- **Puerto**
 - Punto de interacción de una parte del sistema.
 - Ej. subsistema, componente o clase.
 - Indica las interfaces proporcionadas y requeridas.
- **Interfaces**
 - Requeridas
 - Proporcionadas

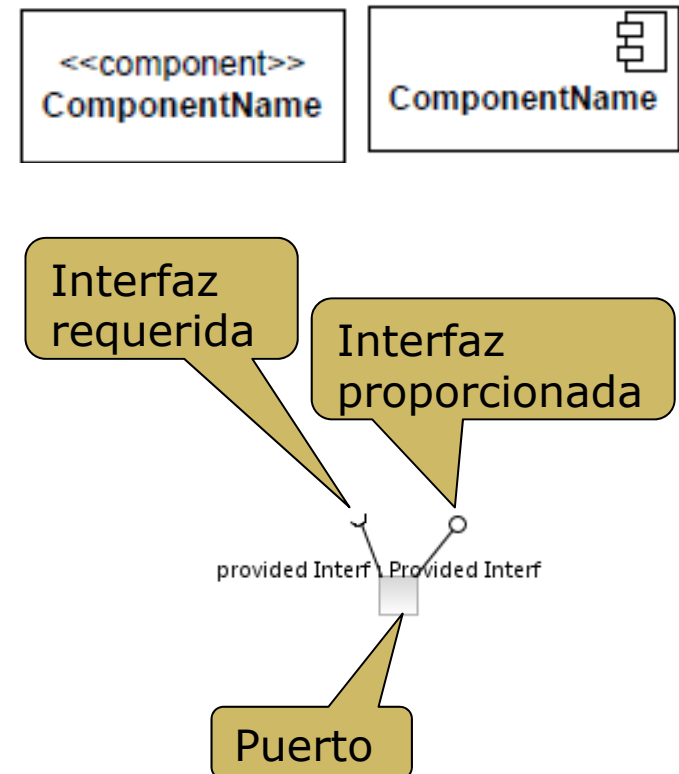


Diagrama de componentes: entidades (2/2)

- Artefactos
 - Manifestación de un elemento concreto del mundo real.
- Clases

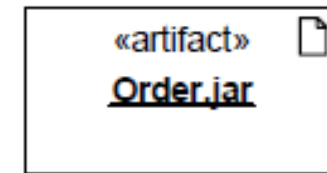
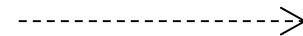


Diagrama de componentes : relaciones

- Conector
 - Relaciona dos elementos vinculados en ejecución.
- Conector de ensamblaje
 - Relaciona una interfaz requerida con otra proporcionada.
- Realización
 - Relaciona un componente con sus clasificadores internos o su especificación (ej. interface).
- Dependencia



Ejemplo: diagrama de componentes

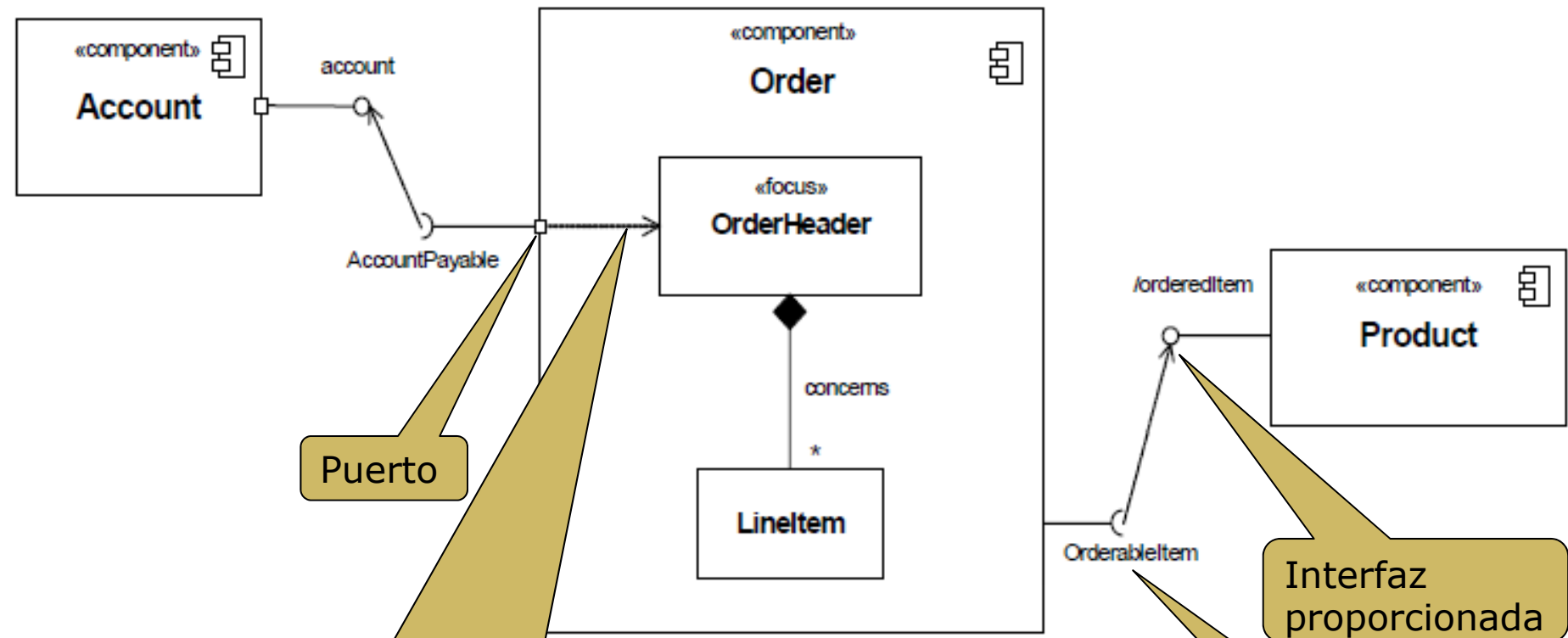


Fig. 8.14 [OMG, 2011b]

Conector de delegación. Indica una dependencia: la interfaz requerida por el puerto lo es a causa del clasificador



Conector múltiple

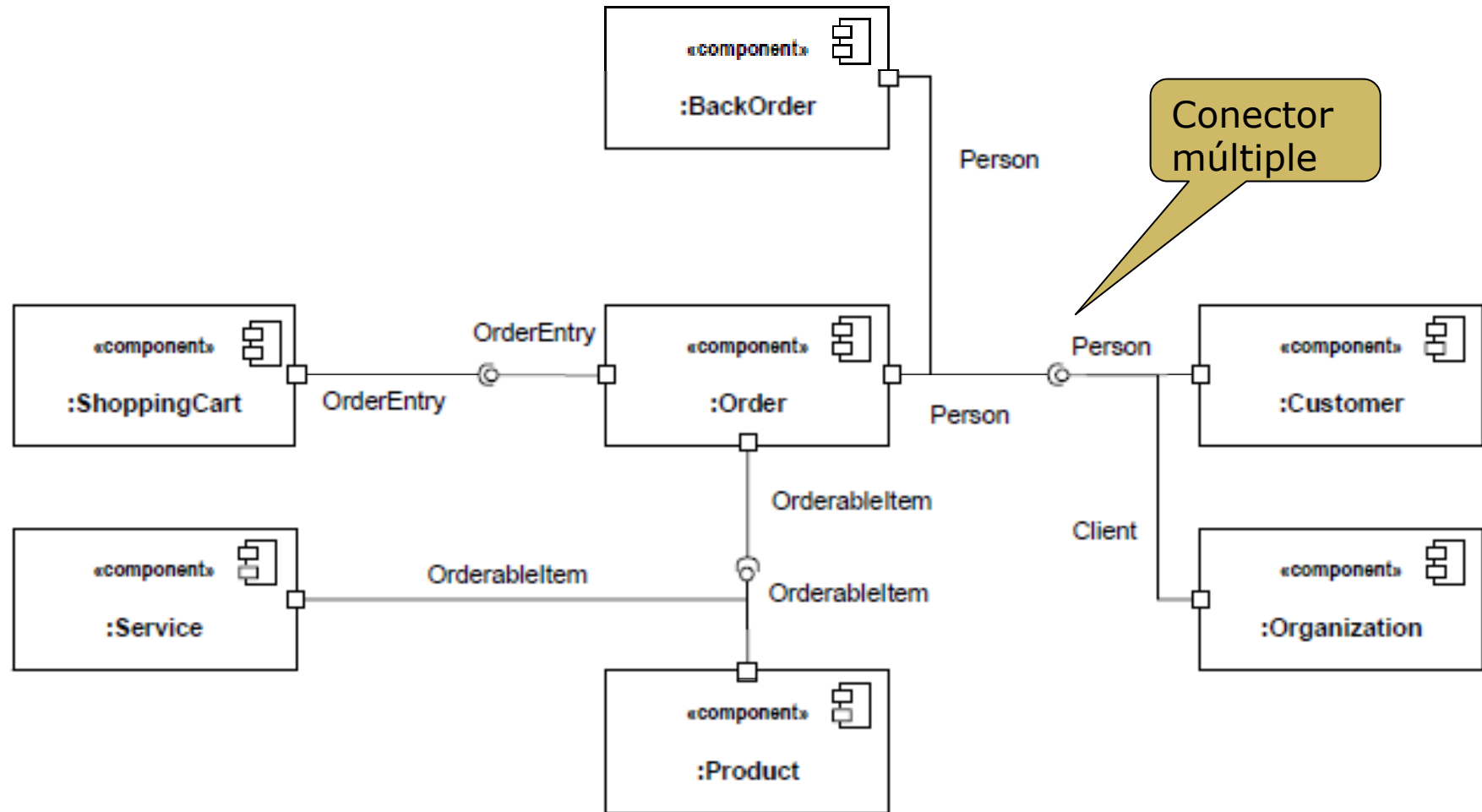


Fig. 8.15 [OMG, 2011b]





DIAGRAMA DE DESPLIEGUE





Diagrama de despliegue

- Un *diagrama de despliegue* es un diagrama de implementación que muestra la configuración de los *nodos* que participan en la ejecución y de los *artefectos* que residen en ellos.
- Un *nodo* es un elemento físico computacional que existe en tiempo de ejecución.
 - Representa un recurso computacional que generalmente tiene alguna memoria, y a menudo, capacidad de procesamiento.



Diagrama de despliegue: entidades

- **Nodo**
 - Elemento físico que existe en tiempo de ejecución y representa un recurso computacional.
- **Artefacto**
 - Elemento concreto que manifiesta una abstracción de los diagramas.
 - Ej. especificación de despliegue
- **Nodo con artefactos desplegados**

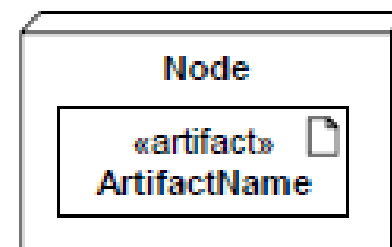
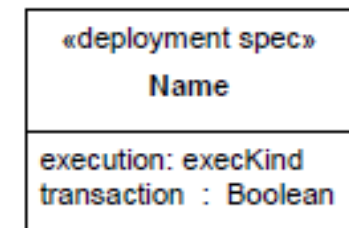
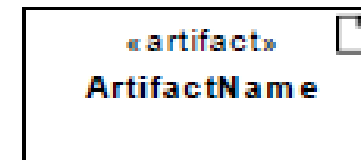
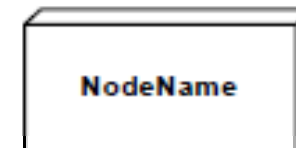
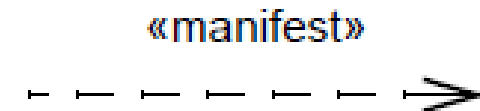
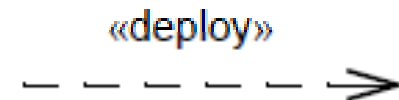
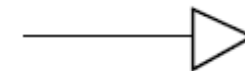
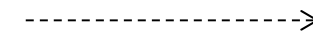
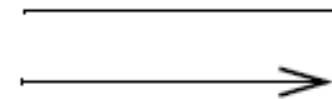


Diagrama de despliegue: relaciones

- Asociación
 - La asociación denota conexión física entre nodos.
- Dependencia
 - Se establece entre los elementos que pudieran aparecer dentro de los nodos.
- Generalización
- Despliegue
 - Indica el despliegue de un artefacto en un nodo.
- Manifestación
 - Relaciona un artefacto con el elemento de modelo que implementa.



Ejemplo: despliegue

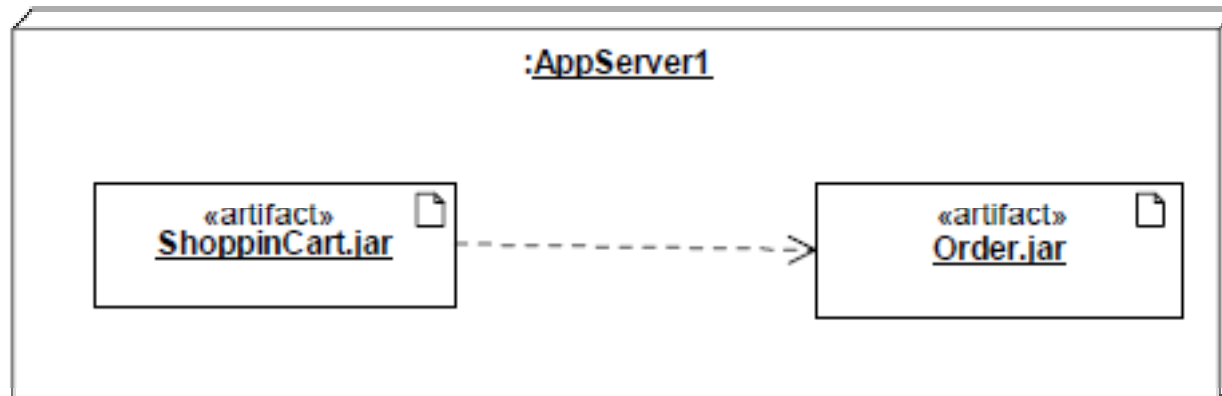


Fig. 10.8 [OMG, 2011b]

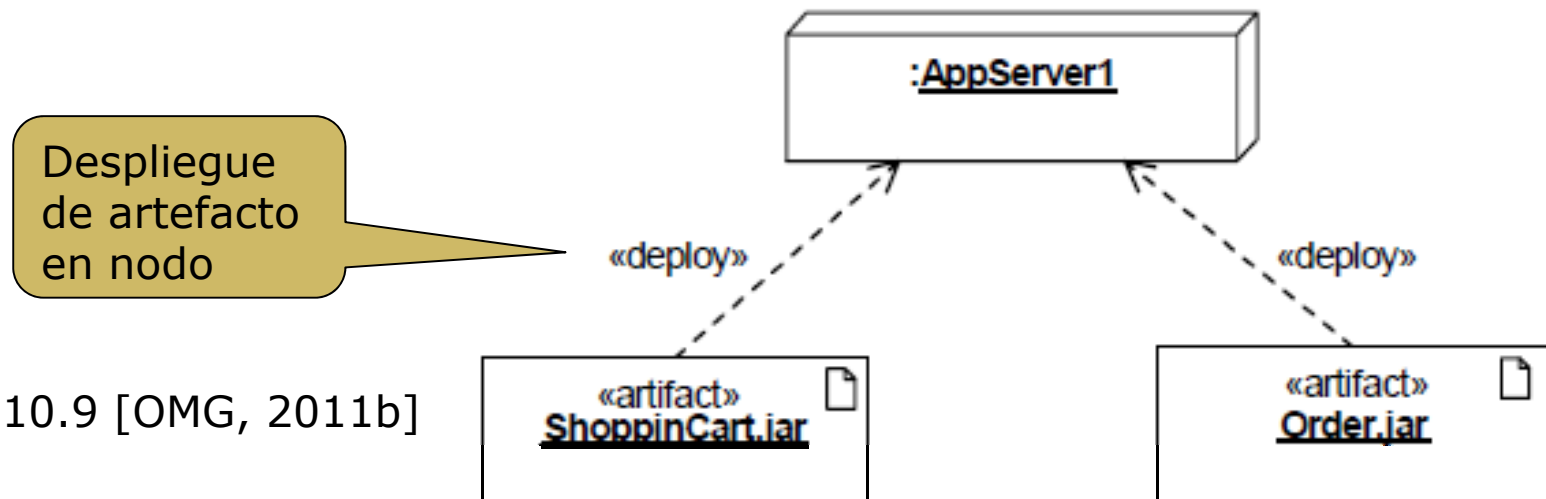


Fig. 10.9 [OMG, 2011b]

Ejemplo: especificación de despliegue

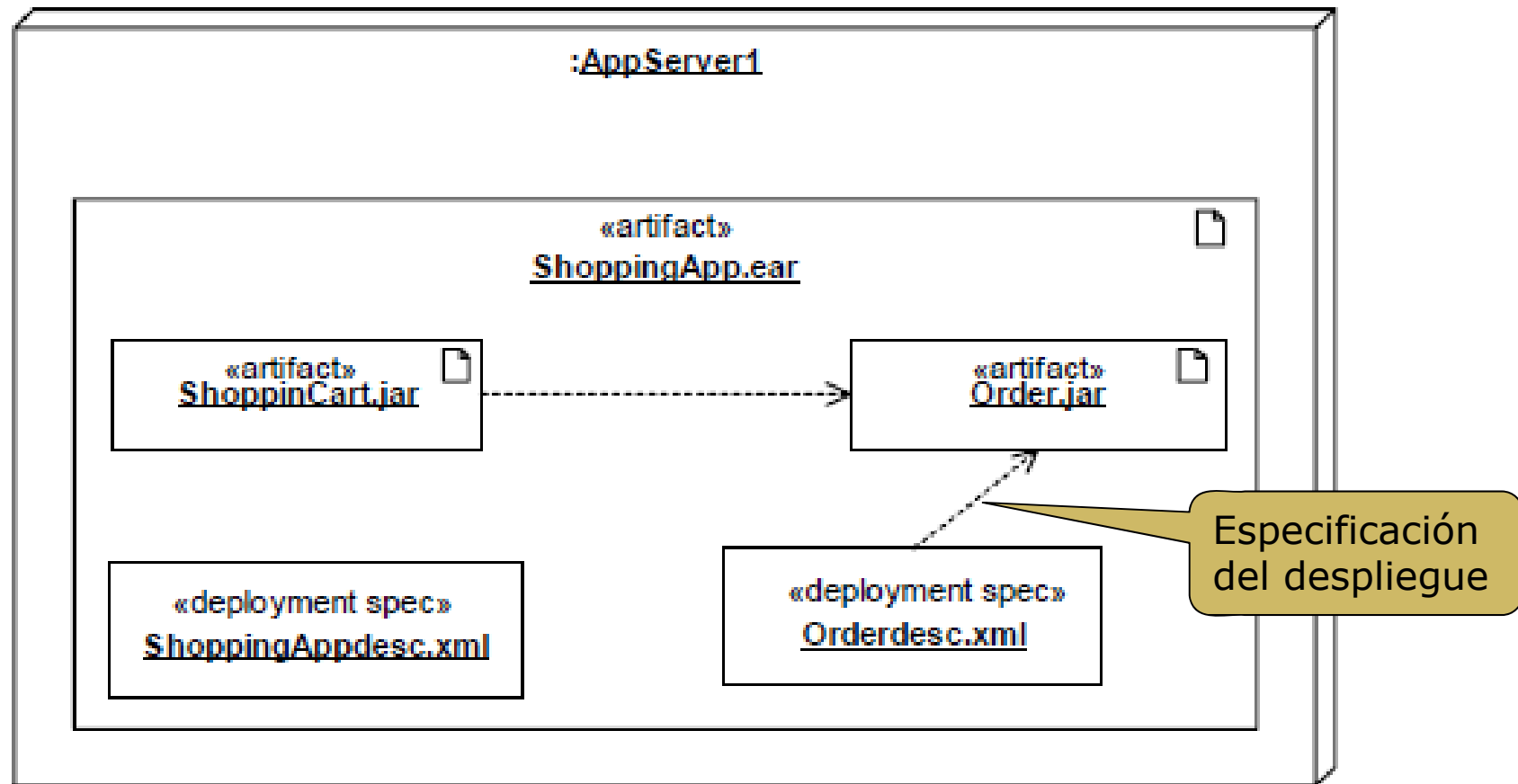


Fig. 10.12 [OMG, 2011b]

Ejemplo: especificación de despliegue

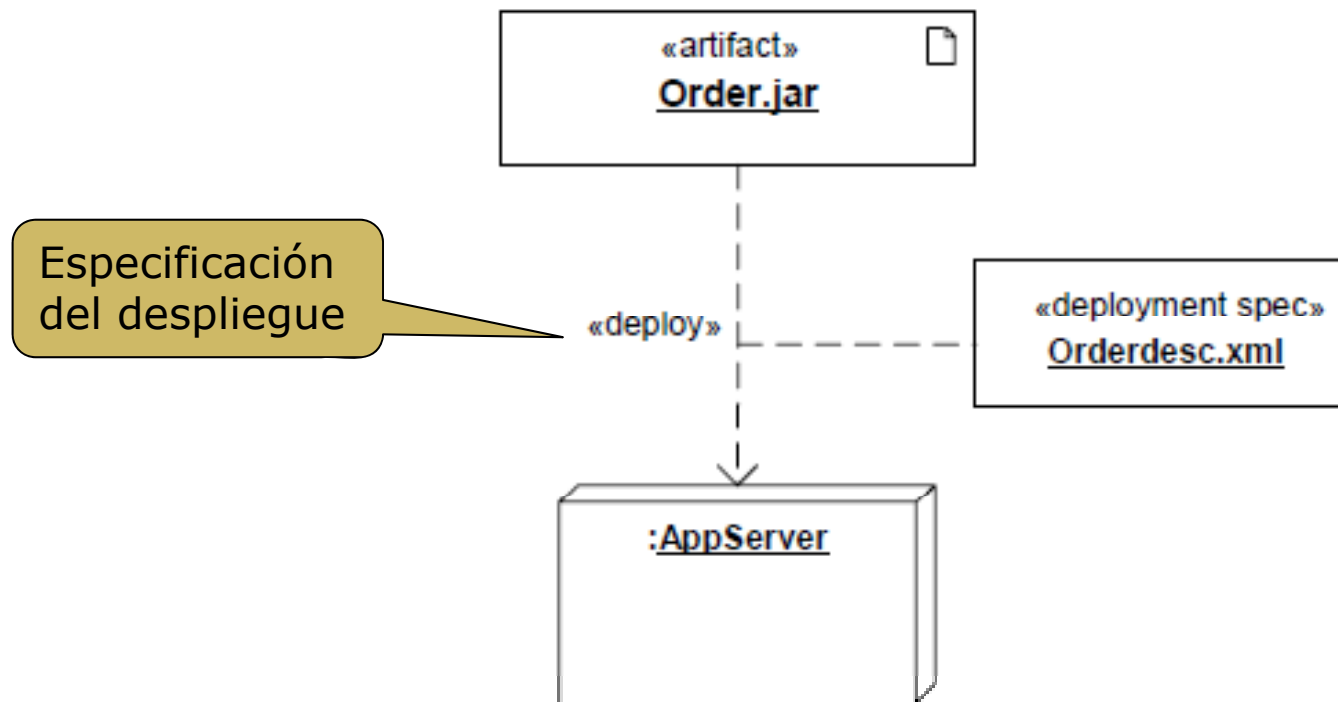


Fig. 10.13 [OMG, 2011b]



Dispositivo

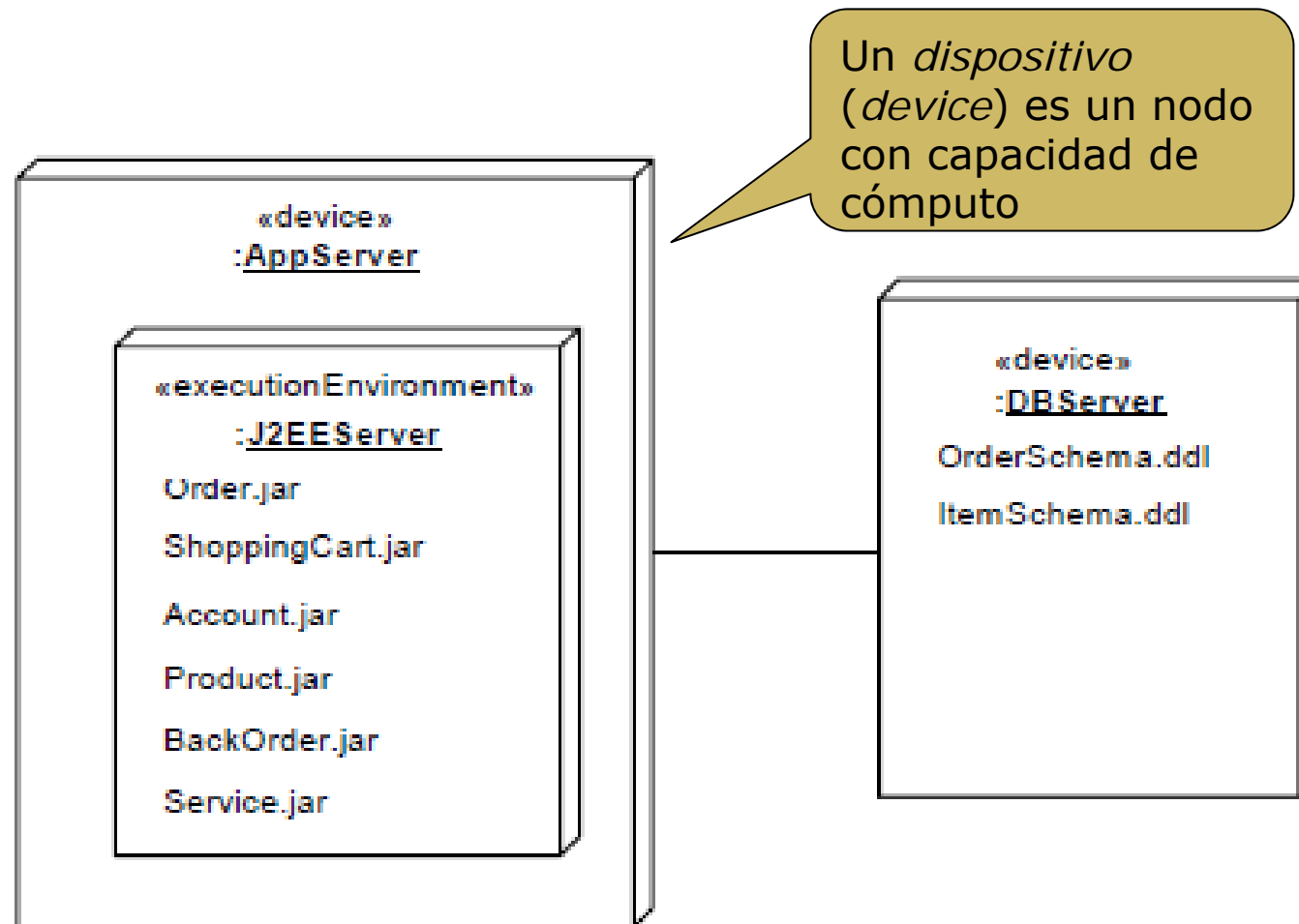


Fig. 10.14 [OMG, 2011b]



Enlaces de comunicación entre nodos

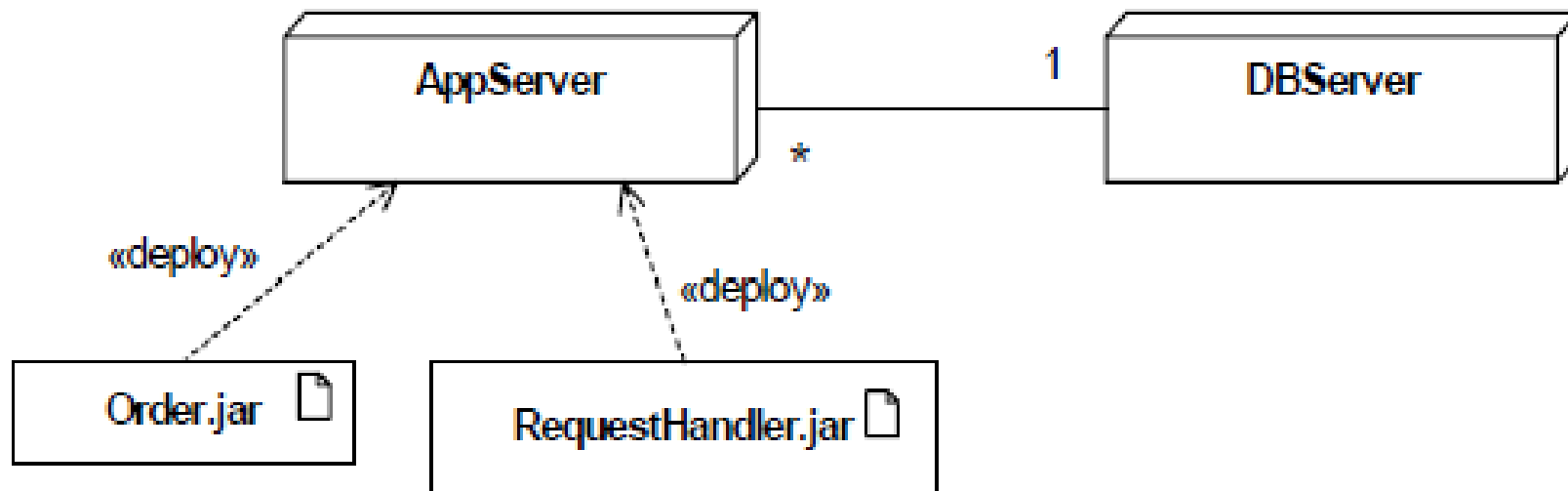


Fig. 10.17 [OMG, 2011b]



PAQUETES





Paquetes

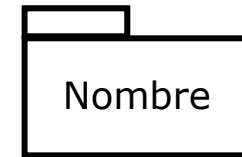
- Un *paquete* es una agrupación de elementos de modelo.
 - Puede contener elementos de modelo de distintos tipos.
 - Incluyendo otros paquetes → jerarquías de paquetes
 - También diagramas UML
- Define un espacio de nombres para sus contenidos.
 - Similar a lo que ocurre con los paquetes de Java.
- Los paquetes se utilizan para:
 - Organizar un modelo grande.
 - Agrupar elementos relacionados.
 - Separar espacios de nombres.
 - Crear una visión general de un conjunto de modelos grande.





Paquete: entidades

- Paquete
 - Es una agrupación de elementos de modelo.





Paquete: relaciones

- Importación
 - Dependencia que indica que el contenido *público* del paquete de destino se añade al espacio de nombres del paquete origen.
- Acceso
 - Dependencia que indica que el contenido *privado* del paquete de destino está disponible en el espacio de nombres del paquete origen.
- Generalización
- Dependencia

«import»
----->

«access»
----->

----->

----->





Visibilidad

- Todo elemento contenido por un paquete tiene una visibilidad relativa al paquete que lo contiene.
 - Un elemento *public* es visible a todos los elementos fuera del paquete.
 - Se indica con '+'
 - Un elemento *protected* es visible sólo en los paquetes que heredan.
 - Se indica con '#'
 - Un elemento *private* no es visible a los elementos fuera del paquete.
 - Se indica con '-'
- Es la misma sintaxis de visibilidad que se utiliza con los atributos y operaciones en las clases.



Ejemplo: importación de paquetes

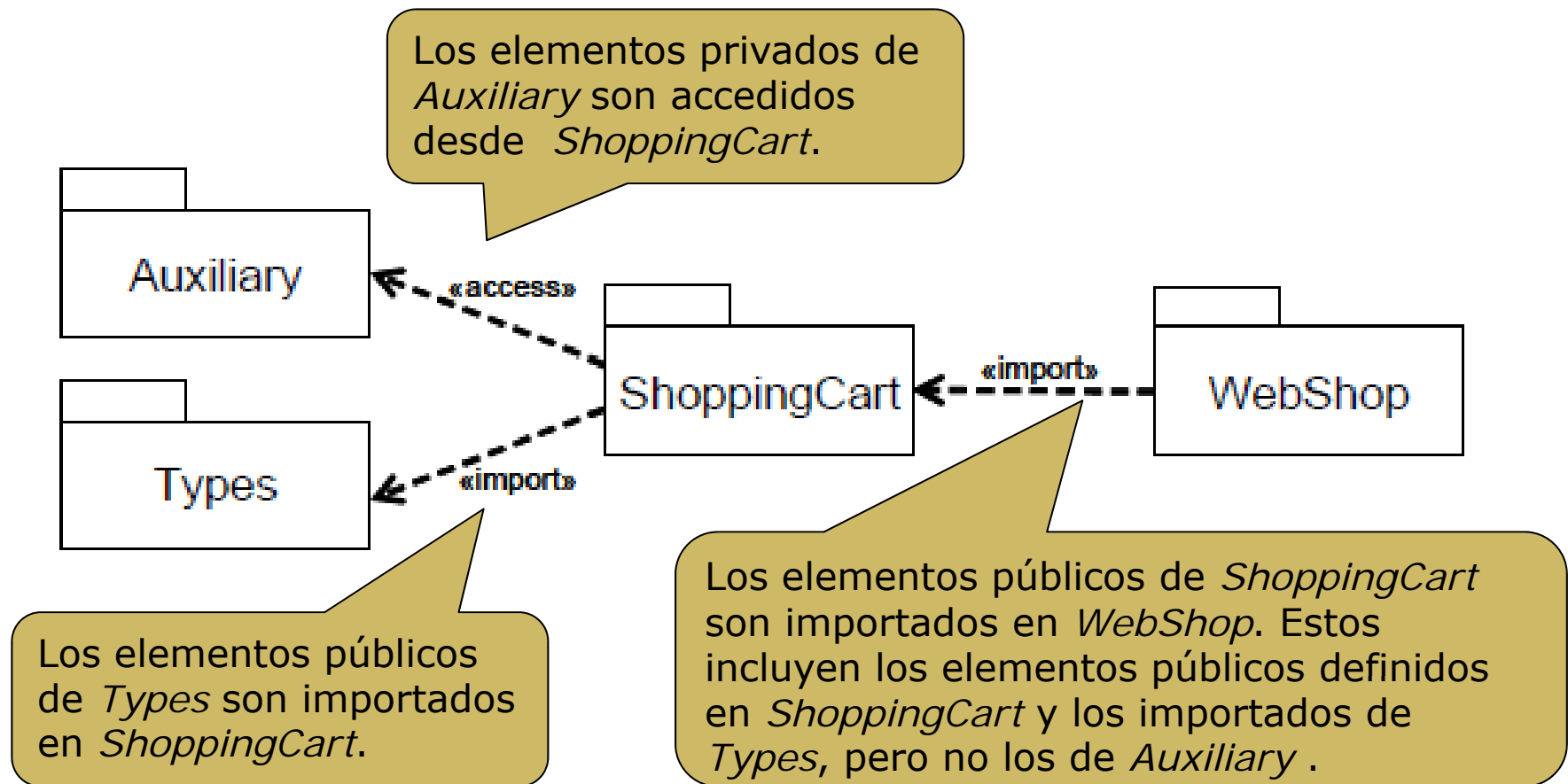
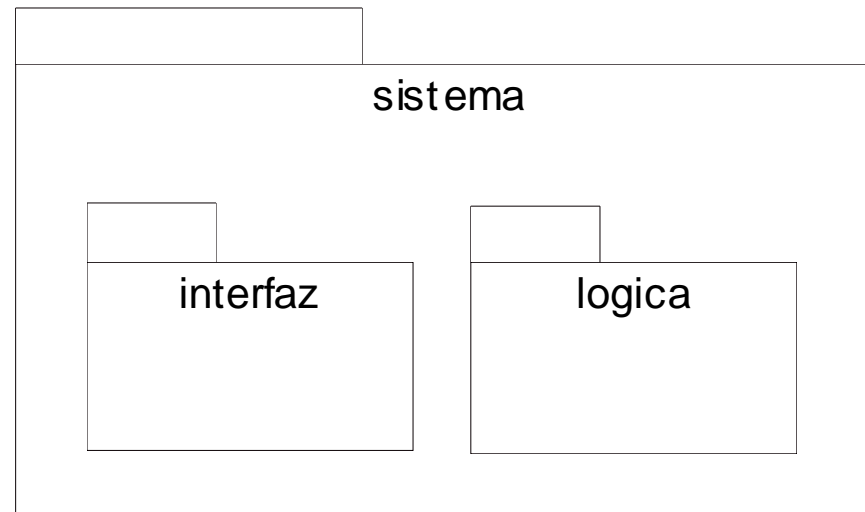


Fig. 7.64 [OMG, 2011b]





Ejemplo: paquetes anidados



Ejemplo: paquetes y diagramas

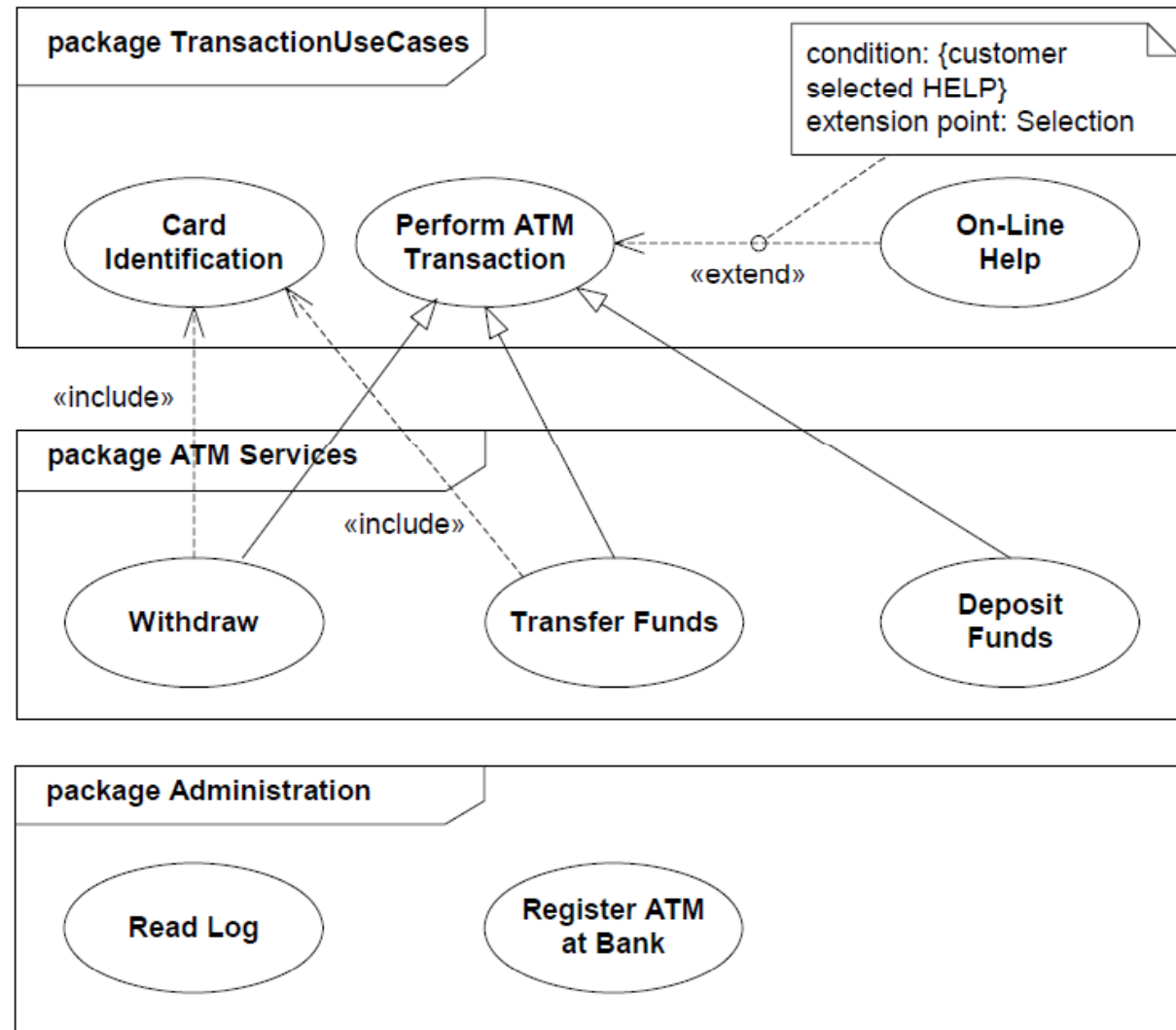


Fig. 16.7
[OMG, 2011]



Consejos para definir paquetes

- Juntar elementos de modelo que tengan una gran cohesión en un paquete.
- Guardar elementos de modelo con baja cohesión en distintos paquetes.
- Minimizar las relaciones, especialmente asociaciones, entre elementos de modelos de distintos paquetes.
- Una implicación del espacio de nombres:
 - Un elemento importado en un paquete *no sabe* cómo se va a usar en el paquete importado.
 - Cuidado con las definiciones.

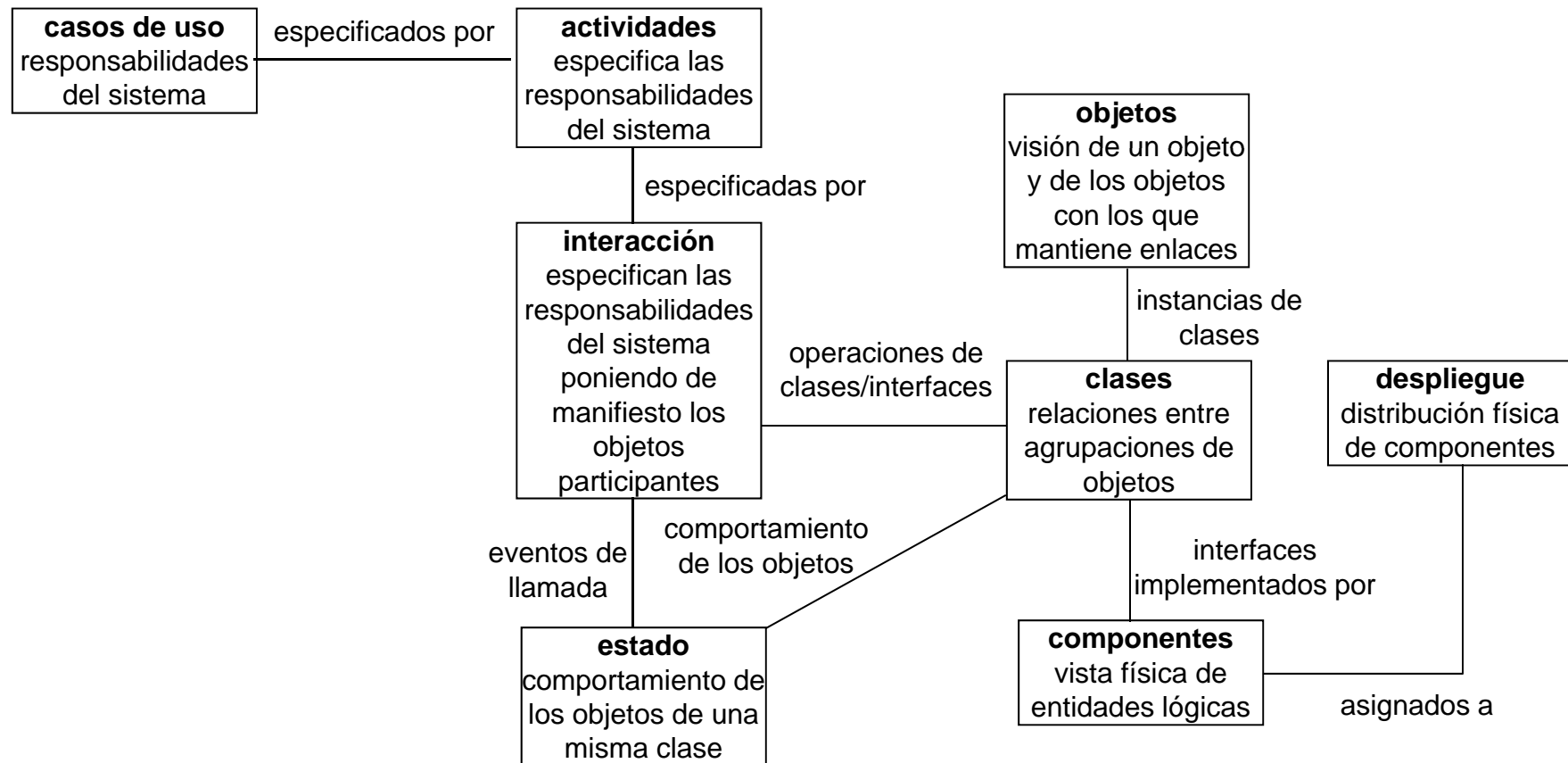




USO INTEGRADO DE DIAGRAMAS



Racionalidad de uso de UML 1.x



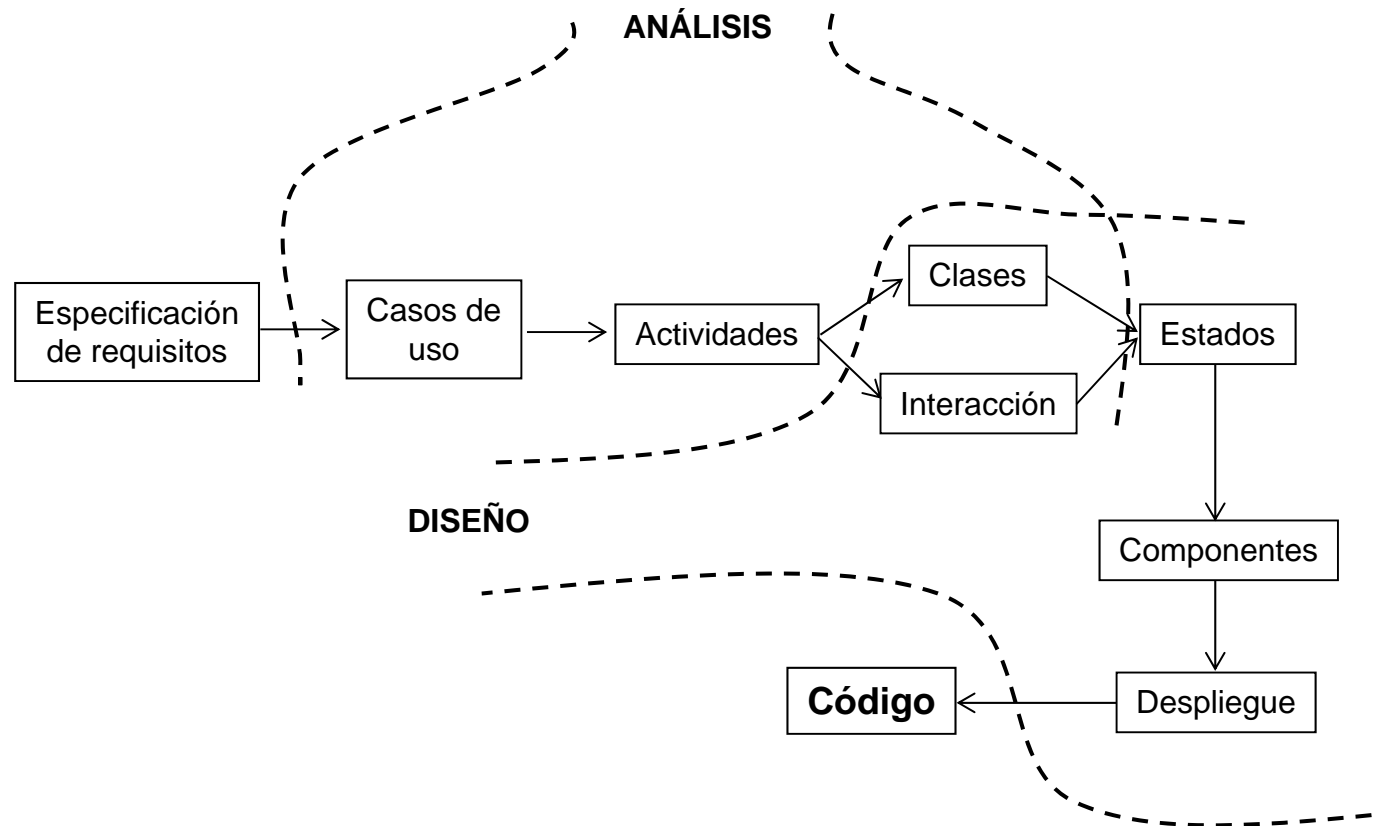


Racionalidad de uso: UML 1.x vs UML 2.x

- Con respecto a UML 2.x la racionalidad de uso es similar salvo que:
 - Los componentes serían vistas arquitectónicas de alto nivel.
 - Se usarían artefactos en lugar de componentes.



Análisis y diseño OO



Posible interpretación de los flujos de trabajo del Proceso Unificado de Desarrollo en términos de los diagramas UML generados





CONCLUSIONES





Conclusiones

- UML es un lenguaje de modelado OO de propósito general.
 - Se puede adaptar a usos específicos con estereotipos.
 - También cambiando el metamodelo, aunque esto no se ha visto.
- Una especificación con UML se puede componer de múltiples diagramas de varios tipos.
 - Diagramas de Casos de uso, Clase, Objetos, Estructura compuesta, Secuencia, Comunicación, Máquina de estados, Actividades, Componentes, Despliegue
 - Y otros que no hemos visto como los de Tiempo o Vista general de interacciones
- UML cubre el ciclo completo de desarrollo.
 - No hay diagramas específicos para una parte del desarrollo.
 - Depende de qué información se vuelca.





Glosario

- CASE = *Computer-Aided Software Engineering*
- EJB = *Enterprise JavaBean*
- MIS = *Management Information System*
- Objecteering = *Object-Oriented Software Engineering*
- OMG = *Object Management Group*
- OMT = *Object Modelling Technique*
- OO = Orientado a Objetos
- RSA = *Rational Software Architect*
- UML = *Unified Modelling Language*





Referencias

- R. Pressman: Ingeniería del Software. Un enfoque práctico, 7ª edición. McGraw-Hill, 2010.
 - Capítulo 4-13
- I. Sommerville: Ingeniería del Software, 7ª edición. Addison Wesley, 2007.
 - Capítulo 8
- J. Arlow, I. Neudstadt: UML 2. Anaya Multimedia, 2006.
- OMG: OMG Unified Modeling Language Specification, v1.1. OMG, 1998.
- OMG: OMG Unified Modeling Language (OMG UML), Infrastructure, v2.4.1. OMG, 2011a.
- OMG: OMG Unified Modeling Language (OMG UML), Superstructure v2.4.1. OMG, 2011b.

