INGENIERÍA DE SOFTWARE I

Tema 8 – UML. Unified Modeling Language

2º G.I.I.
Fecha de última modificación: 26-2-2020

Dr. Francisco José García Peñalvo / fgarcia@usal.es
Dra. María N. Moreno García / mmg@usal.es
Dra. Alicia García Holgado / aliciagh@usal.es
Andrea Vázquez Ingelmo / andreavazquez@usal.es

Departamento de Informática y Automática
Universidad de Salamanca

Resumen

Resumen	Este es un tema de referencia y consulta. En él se presentan las diferentes vistas del Lenguaje Unificado de Modelado UML: Vista estática, Vista de gestión del modelo, Vista de casos de uso, Vista de interacción, Vista de actividad, Vista de máquina de estados, Vista de diseño, Vista de despliegue. Es un contenido de referencia común a las asignaturas Ingeniería de Software I e Ingeniería de Software II. En el caso de Ingeniería de Software I se pone el énfasis en la Vista estática, la Vista de casos de uso y la Vista de interacción
Descriptores	UML; Vista estática; Vista de gestión del modelo; Vista de casos de uso; Vista de interacción; Vista de actividad; Vista de máquina de estados; Vista de diseño; Vista de despliegue
Bibliografía	[Booch et al., 2007] [Rumbaugh et al., 2005]

Contenidos

1. Introducción
2. Vista estática
3. Vista de gestión del modelo
4. Vista de casos de uso
5. Vista de interacción
6. Vista de actividad
7. Vista de máquina de estados
8. Vista de diseño
9. Vista de despliegue
10. Perfiles
11. Bibliografía

Introducción

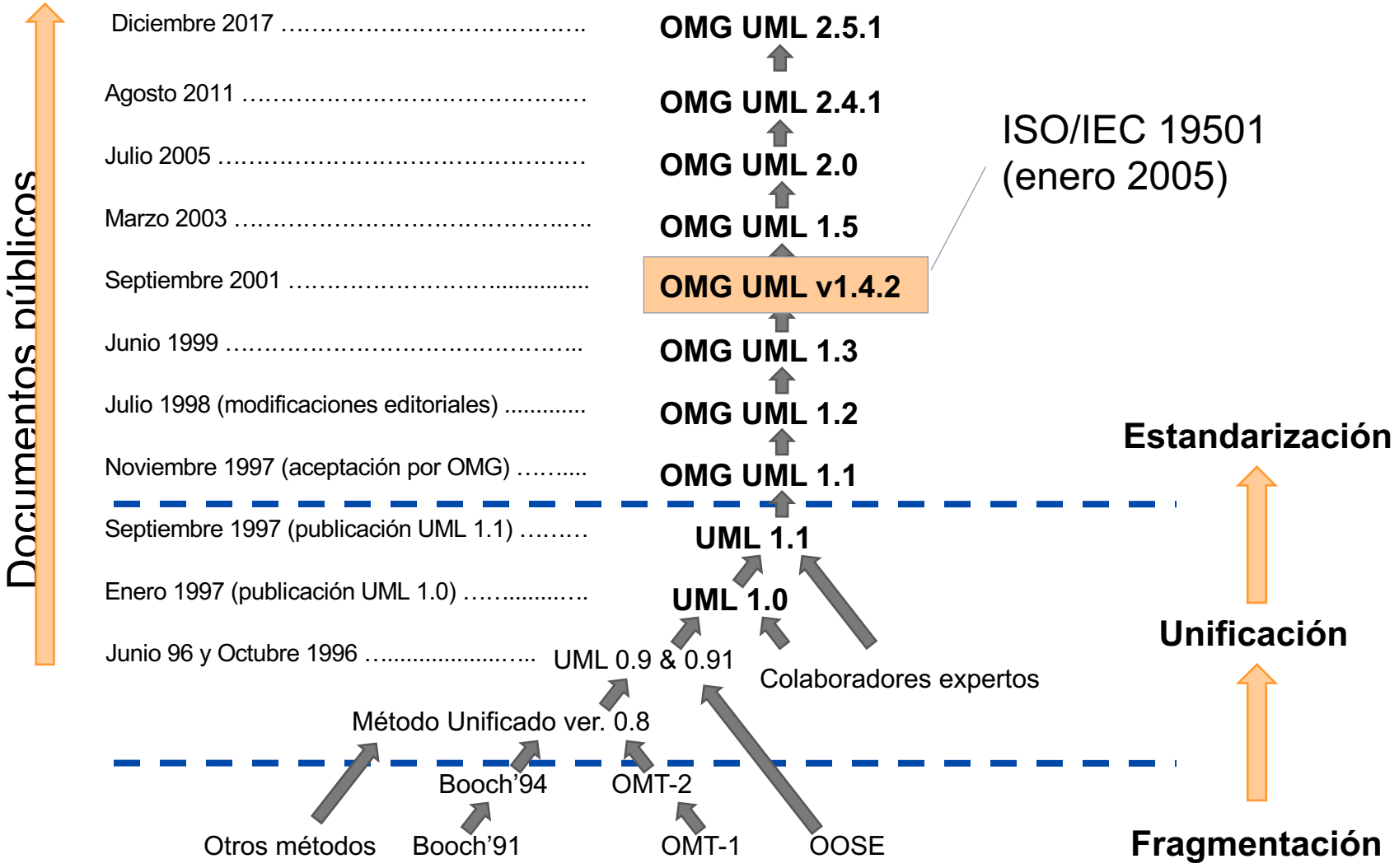
Características

- UML es un lenguaje de modelado para visualizar, especificar, construir y documentar partes de un sistema *software* desde distintos puntos de vista
  - Puede usarse con cualquier proceso de desarrollo, a lo largo de todo el ciclo de vida y puede aplicarse a todos los dominios de aplicación y plataformas de implementación
  - También puede usarse en tras áreas, como la ingeniería de negocio y modelado de procesos gracias a los mecanismos de adaptación/extensión mediante perfiles
- Lo que no es
  - UML no es una notación propietaria
  - UML no es un método, ni un proceso ni una metodología
- El objetivo de UML es la unificación de los métodos de modelado de objetos (Booch, OMT y OOSE) por medio de la
  - Identificación y definición de la semántica de los conceptos fundamentales y elección de una representación gráfica con una sintaxis simple, expresiva e intuitiva
- La especificación UML se define usando el enfoque de un metamodelo

Génesis y evolución

- En 1994 Rumbaugh y Booch crean El Método Unificado
- En 1995 se incorpora Jacobson y los tres autores publican un documento titulado Unified Method V0.8 [Booch y Rumbaugh, 1995]
- El método unificado se reorienta hacia la definición de un lenguaje universal para el modelado de objetos, transformándose en UML (Unified Modeling Language for Object-Oriented Development)
- En 1996 se crea un consorcio de colaboradores para trabajar en la versión 1.0 de UML
- En 1997 se produce la estandarización de UML 1.0 por la OMG [Booch et al., 1997]
- La siguiente versión oficial de UML es la versión 1.1 [Rational et al., 1997]
- En julio de 1998 aparece una revisión interna de UML que recoge diversos cambios editoriales, pero no técnicos. Esta versión es la que se conoce como UML 1.2 [OMG, 1998]
- Casi un año más tarde, en junio de 1999 aparece OMG UML 1.3 [OMG, 1999] con algunos cambios significativos, especialmente en lo tocante a la semántica
- En septiembre de 2001 aparece UML 1.4 [OMG, 2001a] y en enero de 2005 OMG UML 1.4.2 (OMG document: formal/05-04-01) es aceptado como un estándar ISO (ISO/IEC 19501) [OMG, 2005a]
- En julio de 2005 se libera UML 2.0 [OMG, 2005b] siendo la última versión 2.5.1 [OMG, 2017]

Génesis y evolución



Diagramas y vistas

- UML define varios modelos para la representación de los sistemas que pueden verse y manipularse mediante un conjunto de diagramas diferentes
  - **Diagramas de estructura**
    - Diagrama de clases
    - Diagrama de estructuras compuestas
    - Diagrama de componentes
    - Diagrama de despliegue
    - Diagrama de objetos
    - Diagrama de paquetes
  - **Diagramas de comportamiento**
    - Diagrama de casos de uso
    - Diagrama de actividad
    - Diagramas de interacción
      - Diagrama de secuencia
      - Diagrama de comunicación o colaboración
      - Diagrama de visión global de la interacción
      - Diagrama de tiempo
    - Diagrama de maquina de estados

Diagramas y vistas

- Una vista es un subconjunto de las construcciones de modelado de UML que representa un aspecto del sistema
- Los diagramas UML se pueden organizar en las siguientes vistas [Rumbaugh et al., 2007]

  - **Vista estática**
    - Diagramas de clases
  - **Vista de casos de uso**
    - Diagramas de casos de uso
  - **Vista de interacción**
    - Diagramas de secuencia
    - Diagramas de comunicación
  - **Vista de actividad**
    - Diagramas de actividad
  - **Vista de la máquina de estados**
    - Diagramas de máquina de estados

  - **Vista de diseño**
    - Diagramas de estructuras compuestas
    - Diagramas de colaboración
    - Diagramas de componentes
  - **Vista de despliegue**
    - Diagramas de despliegue
  - **Vista de gestión del Modelo**
    - Diagramas de paquetes
  - **Perfiles**
    - Diagramas de paquetes

Diagramas y vistas

- Las vistas se pueden agrupar en áreas conceptuales

Área	Vista
Estructural	Vista estática
	Vista de diseño
	Vista de casos de uso
Dinámica	Vista de máquina de estados
	Vista de actividad
	Vista de interacción
Física	Vista de despliegue
Gestión	Vista de gestión del modelo
	Perfiles

Vista estática

Características

- Modela conceptos del dominio de la aplicación sus propiedades internas y sus relaciones
- Se denomina vista estática porque no modela comportamiento del sistema dependiente del tiempo
- Componentes principales
  - **Clases**: describen conceptos del dominio de la aplicación o de la solución
  - **Relaciones**
    - Asociación
    - Generalización
    - Dependencia: realización y uso
- Diagramas utilizados:
  - **Diagrama de clases**: colección de elementos de modelado estáticos como clases, tipos, sus contenidos y relaciones

Clases y objetos

- Una **clase** es un clasificador que describe un conjunto de objetos que comparten la misma especificación de características, restricciones y semántica
  - Una clase describe las propiedades y comportamiento de un grupo de objetos
- Un **objeto** es una instancia de una clase
  - Un objeto es una entidad discreta con identidad, estado y comportamiento invocable
  - Los objetos representan entidades software del mundo real
- **Diagramas de clases**: describen la vista estática de un sistema en forma de clases y relaciones entre ellas
- **Diagramas de objetos**: muestra las instancias de las clases y los enlaces específicos entre esas instancias en un momento determinado

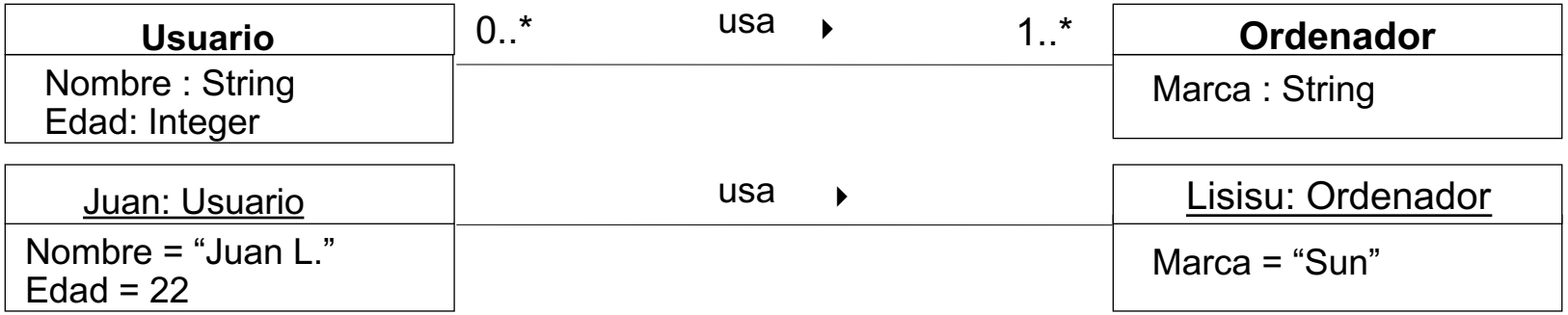
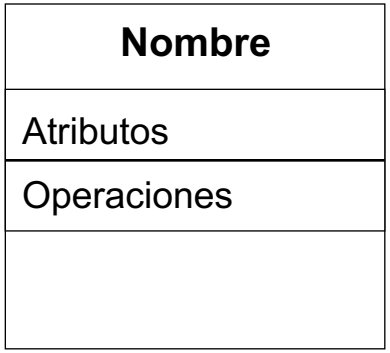


Diagrama de clases y diagrama de objetos

Clases y objetos

- **Características de las clases**:
  - Tienen un nombre único dentro de su contenedor, que es generalmente un paquete, aunque puede ser también otra clase
  - Tienen una visibilidad con respecto a su contenedor, la cual especifica cómo puede ser utilizada por otras clases externas al contenedor
  - Se representan por rectángulos compartimentados
    - **Compartimento del nombre**: contiene al menos el nombre de la clase (inicial en mayúscula) en negrita y centrado. Puede contener también estereotipos (ej. <<enumeration>>) y propiedades (ej. {persistent})
      - **Compartimento de los atributos**: contiene el nombre de los atributos (inicial en minúscula) alineado a la izquierda y opcionalmente información adicional
      - **Compartimento de las operaciones**: contiene el nombre de las operaciones (inicial en minúscula) alineado a la izquierda y opcionalmente información adicional
      - **Compartimentos adicionales**: para mostrar propiedades definidas por el usuario



Clases y objetos

- **Compartimento de atributos (I)**
  - Sintaxis de descripción de un atributo:

visibilidad / nombre:tipo [multiplicidad] = valorPorDefecto {cadena de propiedades}

    - **Visibilidad**: expresa si los atributos son visibles a otros objetos

+ Visibilidad Pública

 # Visibilidad Protegida

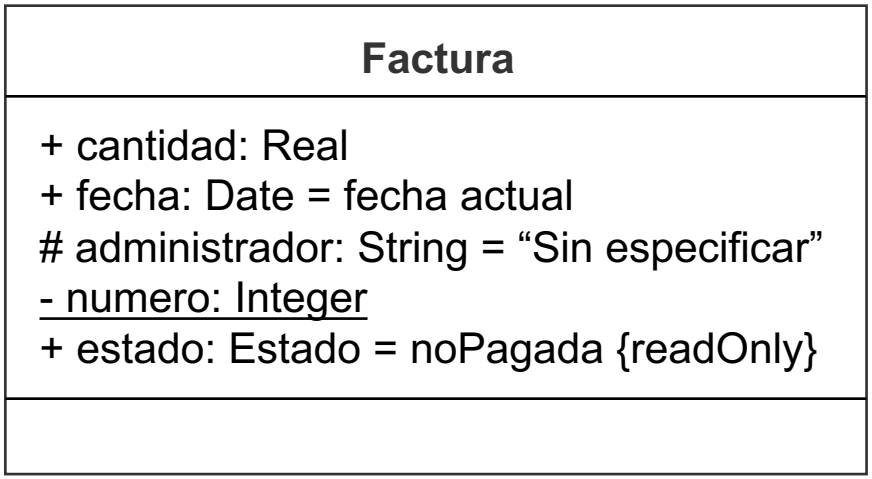
- Visibilidad Privada

~ Visibilidad de Paquete

      - No hay visibilidad por defecto
      - Se pueden definir otros tipos de visibilidad usando perfiles
    - **/** : indica que el atributo es derivado
    - **Nombre**: es una cadena que sirve para identificar al atributo
    - **Tipo**: Indica el tipo o dominio del atributo
    - **Multiplicidad**: número de instancias del atributo (ej. [0..1])
    - **ValorPorDefecto**: si no se da este valor se omite el signo igual
    - **Cadena de propiedades**: lista separada por comas de las propiedades de un atributo (readOnly, ordered, sequence…)

Clases y objetos

- **Compartimento de atributos (II)**
  - Un atributo con alcance de clase se expresa mediante una cadena subrayada



Clases y objetos

- **Compartimento de operaciones (I)**
  - Sintaxis de descripción de una operación

visibilidad nombre (listaParametros) : tipoRetorno {cadena de propiedades}

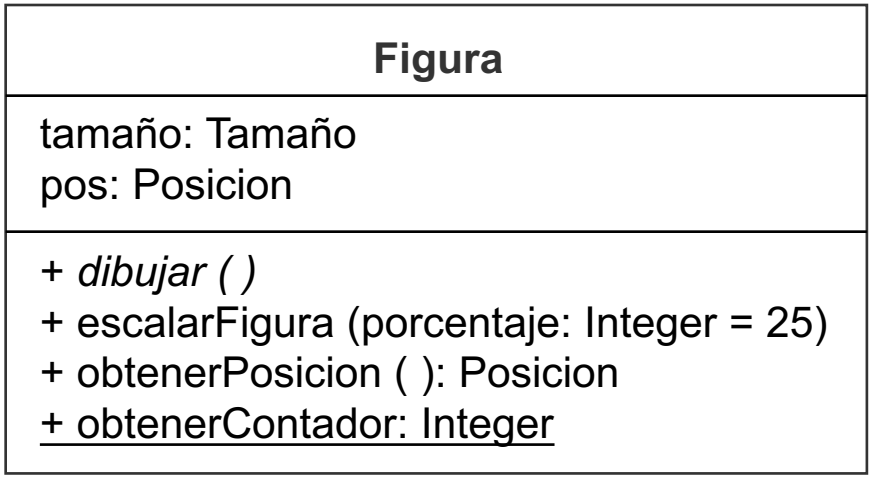
- **Visibilidad**: igual que para los atributos
- **Signatura de la operación**: nombre, listaParametros y tipoRetorno
- **listaParametros**: lista separada por comas de los parámetros formales. Sintaxis:

direccion nombre : tipo [multiplicidad] = valorPorDefecto {cadena de propiedades}

  - **Direccion:** indica si el parámetro se envía dentro o fuera de la operación (in, out, inout)
- **Cadena de propiedades**: lista separada por comas de las propiedades o restricciones de una operación (isQuery, isPolymorphic…)

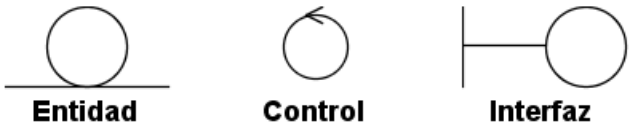
Clases y objetos

- **Compartimento de operaciones (II)**
  - Una operación con alcance de clase se expresa mediante una cadena subrayada
  - Si la operación es abstracta se muestra en cursiva



Clases y objetos

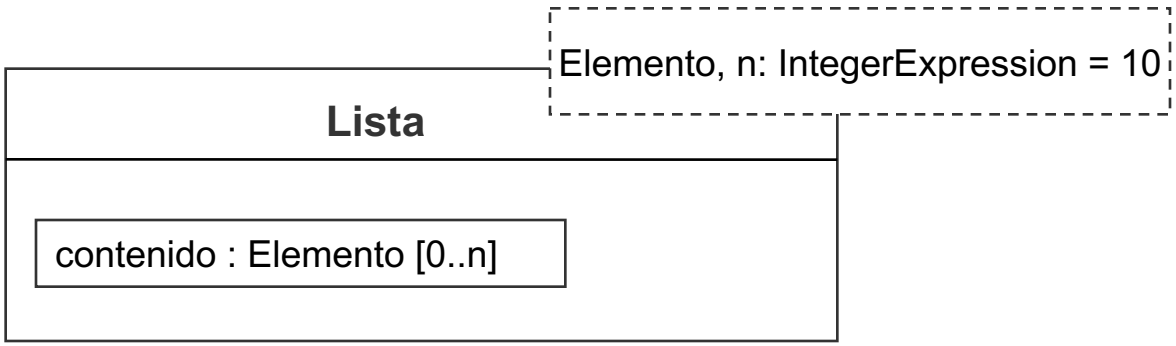
- **Clases entidad, control e interfaz**
  - Las clases **entidad** (***entity***) representan objetos de negocio normalmente persistententes que se almacenan en el sistema
  - Las clases de **control** (***control***) se ocupan del procesamiento de la información
  - Las clases **interfaz** (***boundary***) se encargan de presentar y comunicar la información al usuario o a otros sistemas



Estereotipos utilizados para representar las clases entidad, control e interfaz

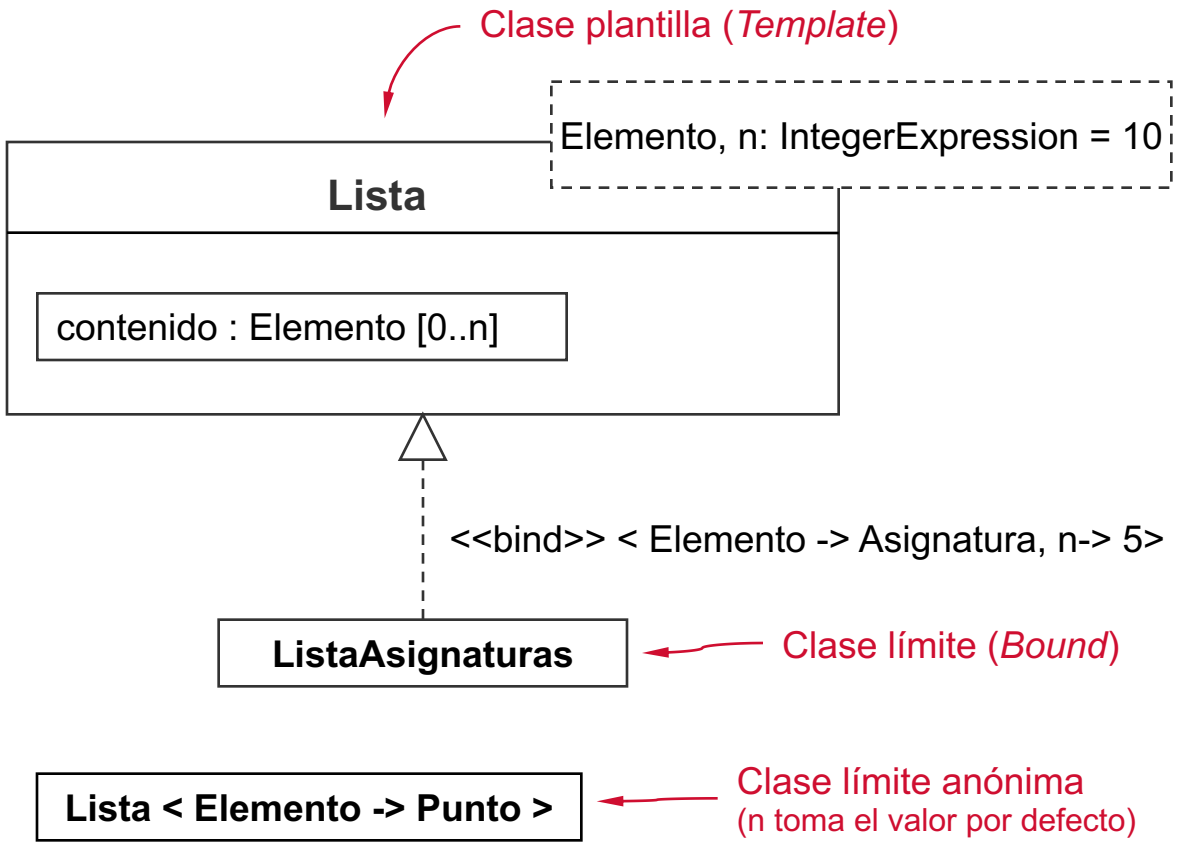
Clases y objetos

- **Clases plantilla (*Template Class*) (I)**
  - Una plantilla es un descriptor de una clase con uno o más parámetros formales sin especificar
    - Define una familia de clases
    - Cada clase se especifica asociando valores a los parámetros
  - Los parámetros pueden ser:
    - Clasificadores (normalmente clases)
    - Tipos primitivos
  - La clase que se produce a partir de una plantilla se denomina **clase límite *(Bound class)***



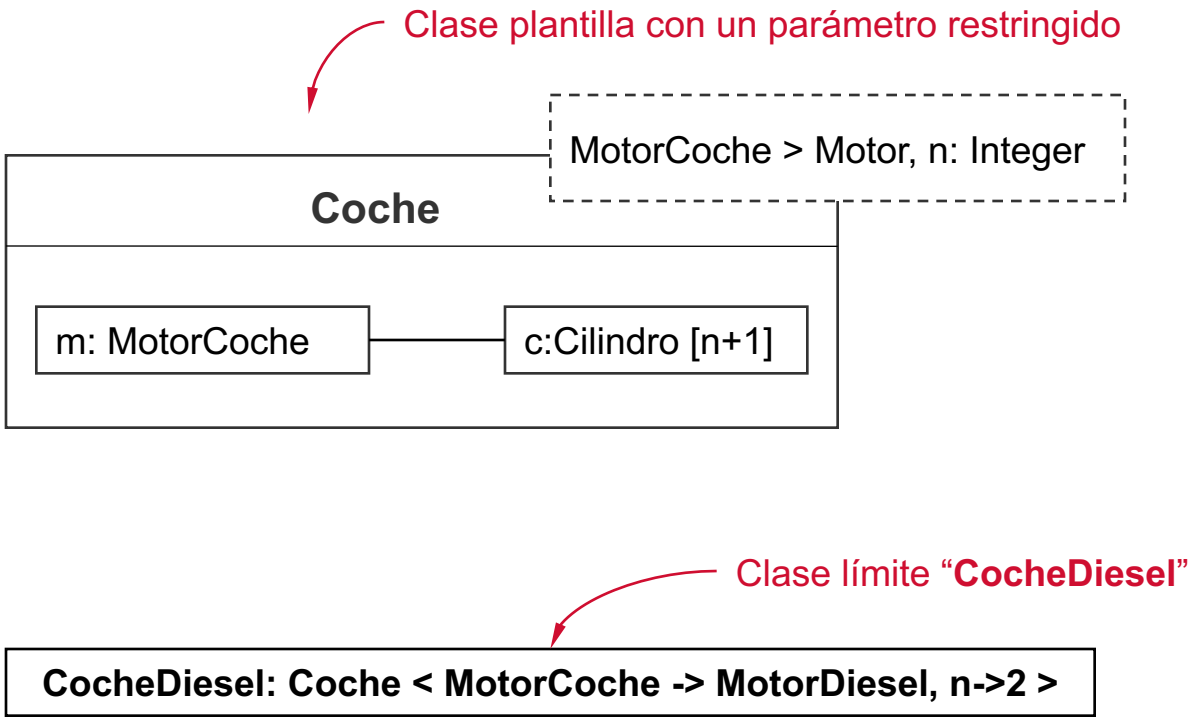
Clases y objetos

- **Clases plantilla (*Template Class*) (II)**



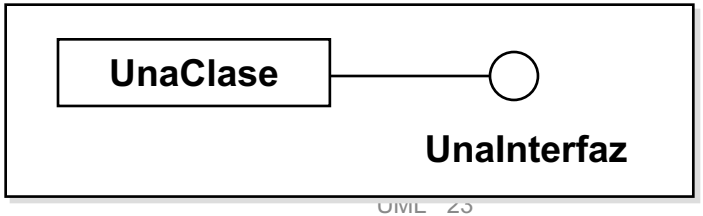
Clases y objetos

- **Clases plantilla (*Template Class*) (III)**



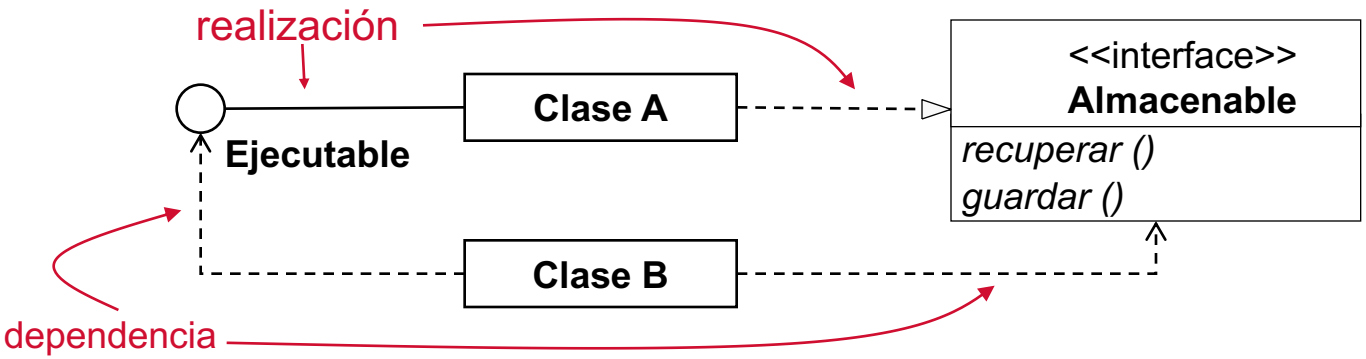
Interfaces

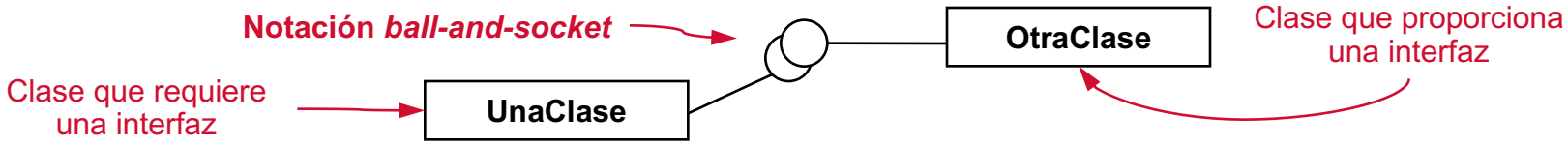
- Una interfaz es un descriptor de las operaciones visibles externamente de una clase u otra entidad que no especifica la estructura interna
  - No contienen atributos ni la implementación de las operaciones
  - Las clases (o componentes) que realizan una interfaz tienen que implementar todas las operaciones de la interfaz (directamente o por derivación)
  - Una clase puede admitir muchas interfaces, cuyos efectos podrán ser disjuntos o solapados
  - Una interfaz no puede tener una asociación que sea navegable partiendo de la interfaz
  - Las interfaces pueden tener relaciones de generalización
  - Todas las operaciones de una interfaz tienen visibilidad pública
  - Notación:
    - Como una clase con el estereotipo << interface>>
    - Mediante un pequeño círculo con el nombre de la interfaz situado debajo del símbolo



Interfaces

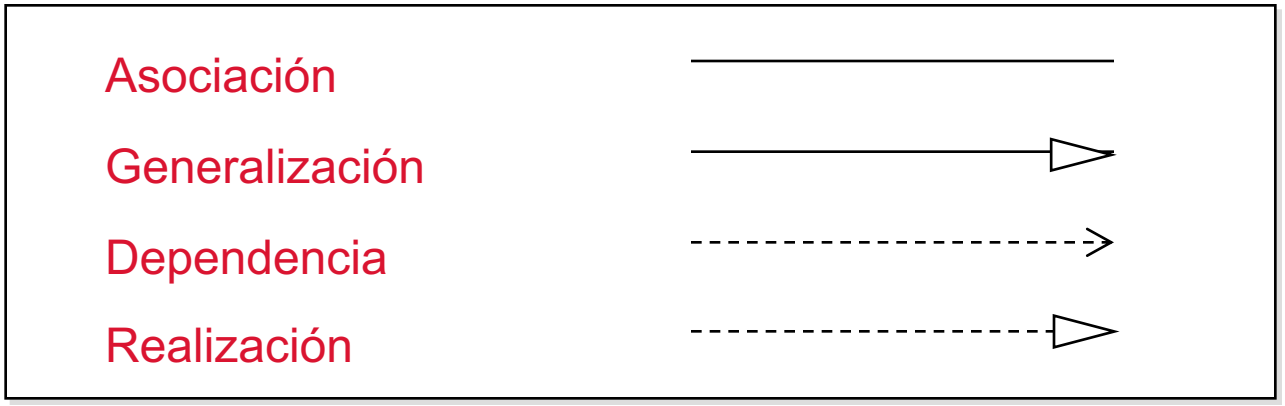
- **Relaciones entre clases e interfaces**
  - **Dependencia**: una clase utiliza o requiere operaciones proporcionadas por la interfaz
  - **Realización**: indica que la clase implementa todas las operaciones definidas en la interfaz (directamente o por derivación)
  - Cuando una clase requiere una interfaz y otra clase proporciona esa interfaz se puede usar la **notación "ball-and-socket"**





Relaciones

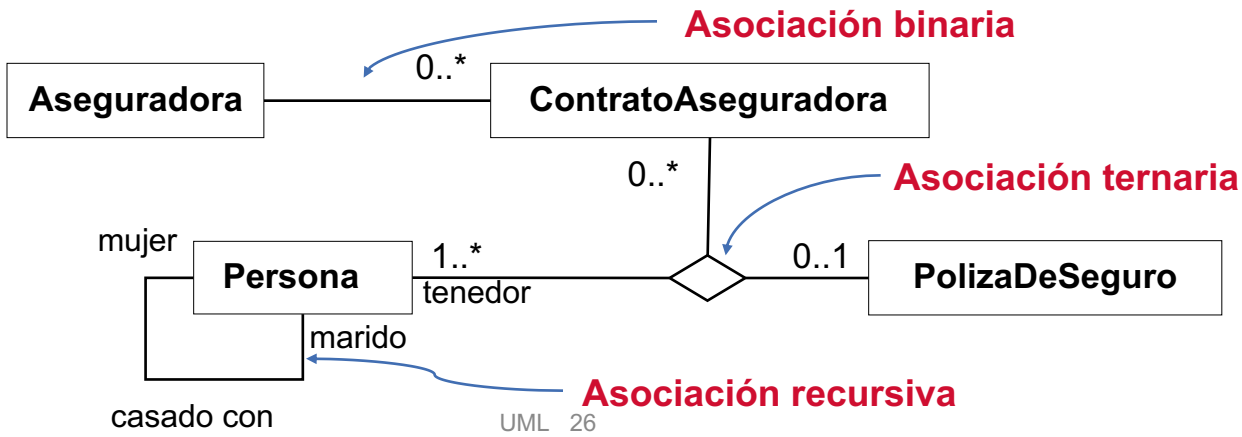
- Una relación es una conexión semántica entre elementos de un modelo
- Pueden ser de varios tipos:
  - **Asociación**: relación que describe un conjunto de enlaces entre objetos
  - **Generalización**: relación entre un elemento más general y otro más específico
  - **Dependencia**: relación entre elementos, uno dependiente y otro independiente. Un cambio en el independiente afectará al dependiente
  - **Abstracción/realización**: relación entre dos descripciones de una misma cosa a diferentes niveles



Notación de los diferentes tipos de relaciones

Relaciones

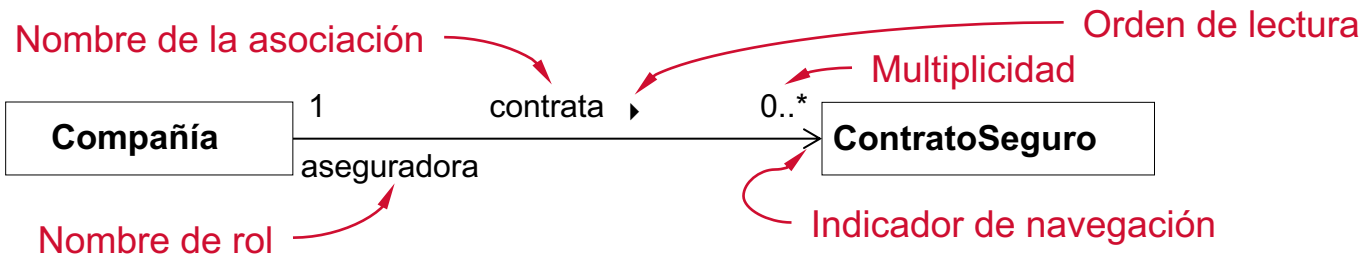
- **Asociaciones**
  - Una asociación describe un conjunto de tuplas cuyos valores se refieren a instancias de un tipo (clase). Una instancia de una asociación es un **enlace**
  - Las asociaciones llevan la información sobre relaciones entre objetos en un sistema
  - Una asociación puede ser
    - **Recursiva**: conecta una clase consigo misma (conexión semántica entre objetos de la misma clase)
    - **Binaria**: conexión entre dos clases
    - **Ternaria** o de **orden superior** (**n-arias**): conexión entre tres o más clases
  - Cada conexión de una asociación a una clase se llama **extremo de la asociación** (***association end***)

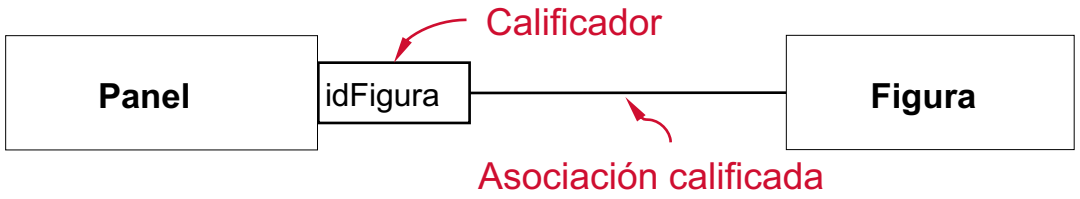


Relaciones

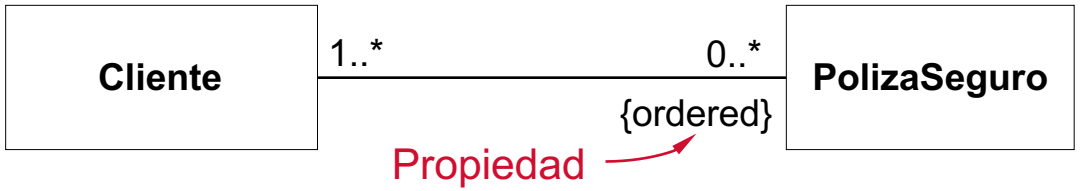
- **Adornos de una asociación**
  - La línea de asociación puede tener en el centro el **nombre de la asociación** y un triángulo sólido (orden de lectura)
  - El **extremo de la asociación** puede contar con los siguientes *adornos*
    - **Símbolo de agregación**
    - **Nombre del extremo/rol** : representa el subconjunto de instancias del clasificador que participa en la asociación
    - **Indicador de navegación**: Una flecha indica que la asociación es navegable en un determinado sentido. Una x indica que el extremo en el que se sitúa no es navegable
    - **Multiplicidad**: Indica cuantos objetos pueden conectarse a través de una instancia de la asociación. En cada extremo de la asociación se puede indicar
      - uno: 1
      - cero o uno: 0..1
      - muchos: *
      - uno o más: 1..*
      - un número exacto: 3 (por ejemplo)
    - **Visibilidad**: se puede limitar la visibilidad en la navegación entre los objetos de una clase de la asociación y los objetos de la otra clase
    - **Calificación**: particionamiento de un conjunto de objetos relacionados con otro objeto mediante un atributo de la asociación (calificador)
    - **Cadena de propiedades**

Relaciones

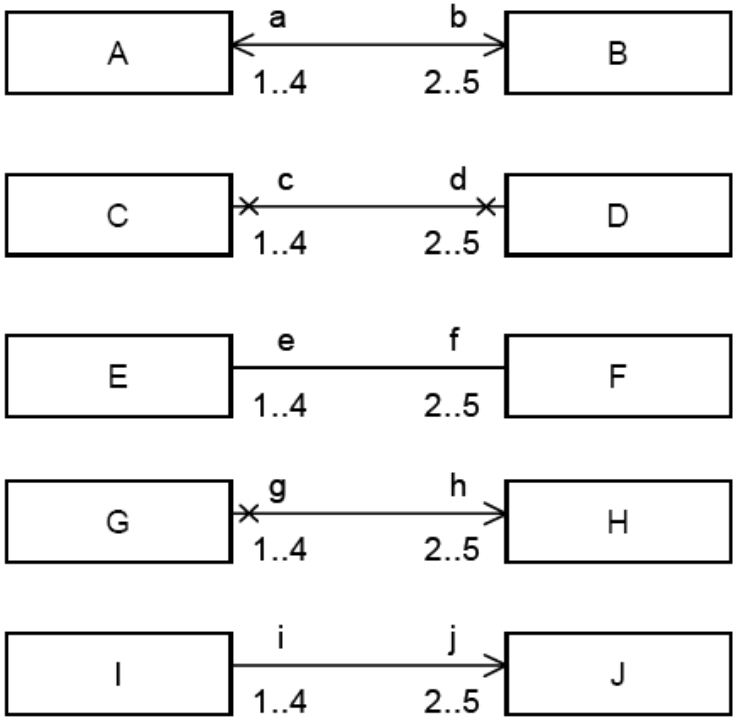

Nombre de la asociación
Orden de lectura
Multiplicidad
Compañía
1
contrata
0..*
ContratoSeguro
aseguradora
Nombre de rol
Indicador de navegación


Calificador
Panel
idFigura
Figura
Asociación calificada

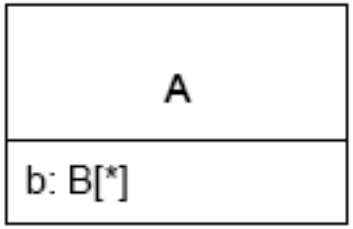
GrupoUsuarios
*
*
Usuario
1
*
clave
Visibilidad
+ usuario
+ propietario
- clave


Cliente
1..*
0..*
PolizaSeguro
{ordered}
Propiedad

Relaciones



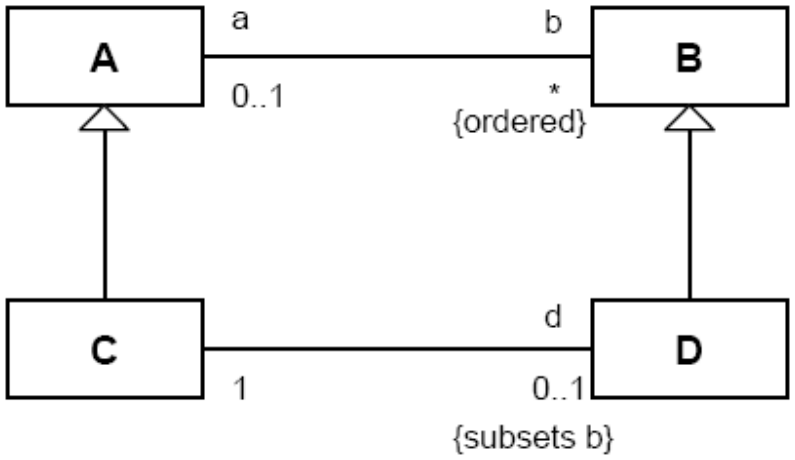
Ejemplos de extremos de navegación



Ejemplo de extremo navegable

Relaciones

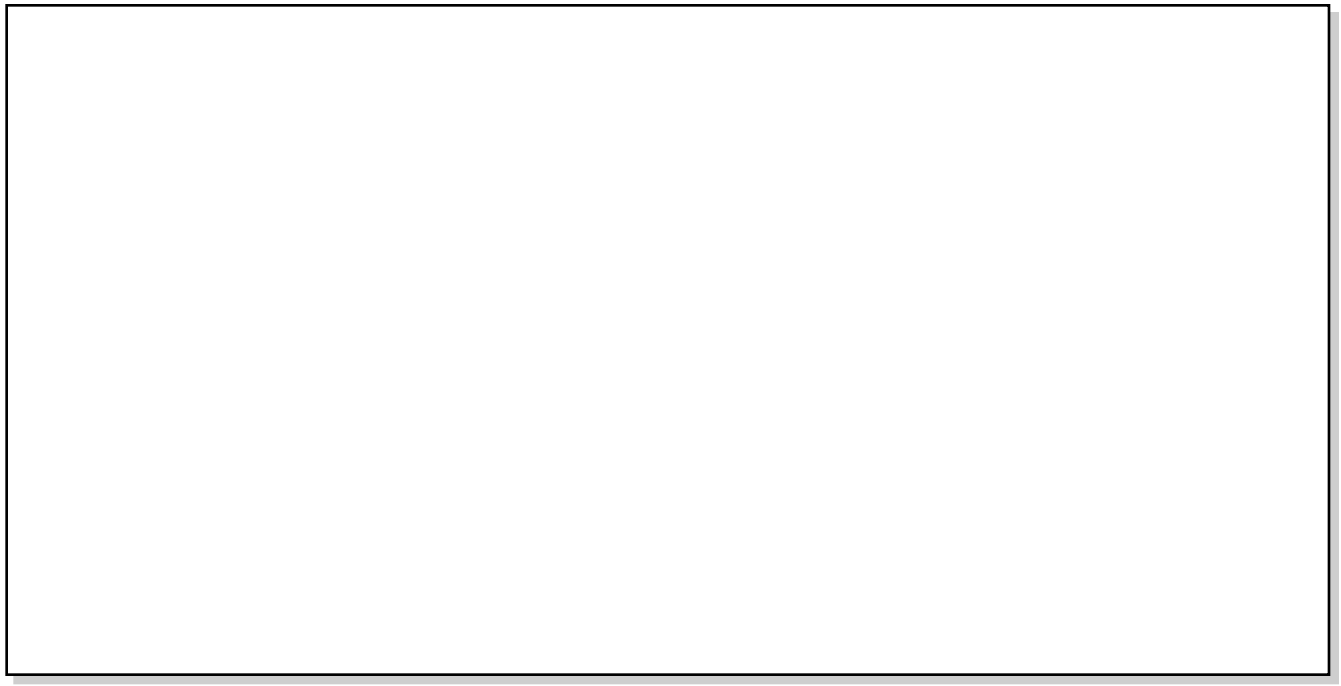
- **{subsets <nombrePropiedad>}** el extremo es un subconjunto de la propiedad llamada <nombrePropiedad>
- **{redefines <nombreExtremo>}** el extremo redefine otro extremo llamado <nombreExtremo>
- **{union}** el extremo se deriva de la unión de sus subconjuntos
- **{ordered}** el extremo representa un conjunto ordenado
- **{bag}** el extremo representa una bolsa
- **{sequence}** o **{seq}** el extremo representa una secuencia
- Si el extremo es navegable se le puede aplicar cualquier cadena de propiedades aplicable a los atributos



Cadenas de propiedades aplicables a extremos de una asociación

Relaciones

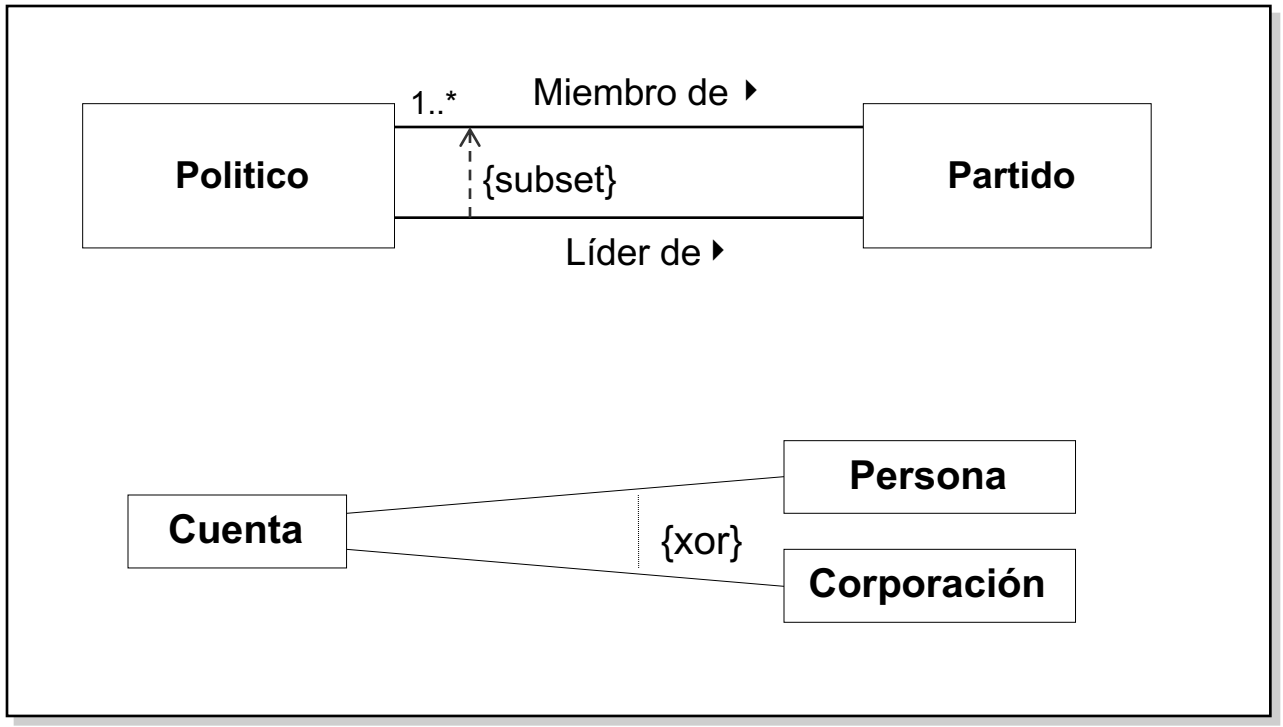
- **Relaciones entre asociaciones (I)**



Relaciones de generalización entre asociaciones

Relaciones

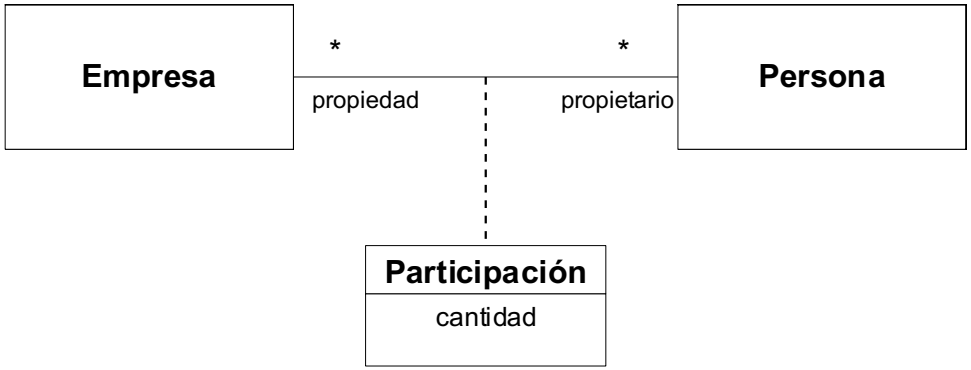
- **Relaciones entre asociaciones (II)**



Restricciones entre asociaciones

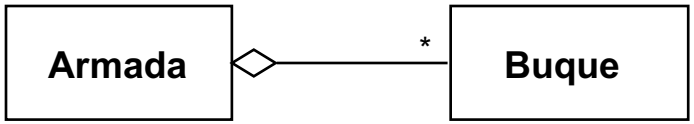
Relaciones

- **Clases asociación**: son asociaciones que también son clases
  - Tienen propiedades tanto de las clases como de las asociaciones
  - Se utilizan cuando cada enlace debe tener sus propios valores para los atributos, operaciones propias o sus propias referencias a objetos
  - Pueden tener operaciones que modifiquen los atributos del enlace o que añadan o eliminen enlaces al propio enlace
  - Pueden participar en otras asociaciones
  - Cada instancia de una clase asociación tiene referencias a objetos, así como valores para los atributos especificados por la parte de la clase

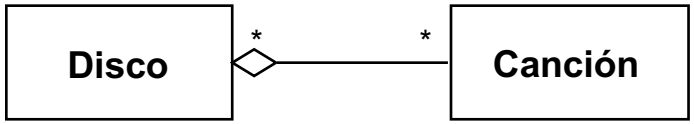


Relaciones

- **Agregación**
  - Es una forma de asociación que representa una relación ***todo-parte*** entre un agregado (el todo) y las partes que los componen

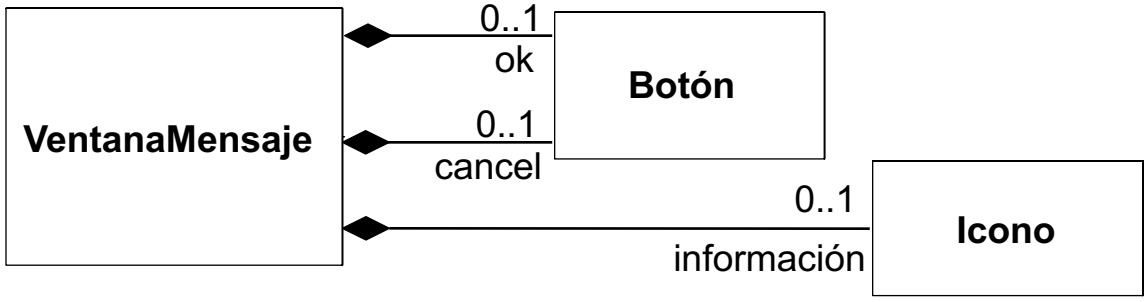


- **Agregación compartida**
  - Agregación en la cual las partes pueden pertenecer a cualquiera de los agregados



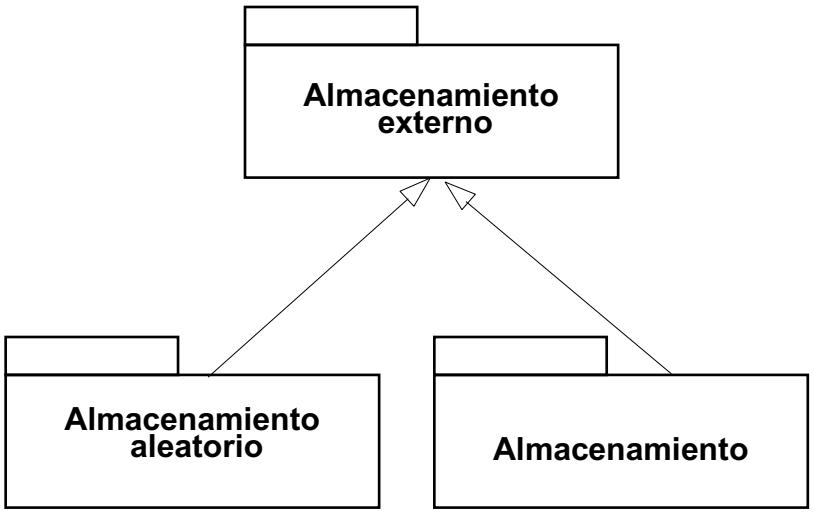
Relaciones

- **Agregación compuesta (composición)**
  - Es una asociación de agregación con las restricciones adicionales de que un objeto sólo puede ser parte de un compuesto a la vez y que el objeto compuesto es el único responsable de la disponibilidad de todas sus partes

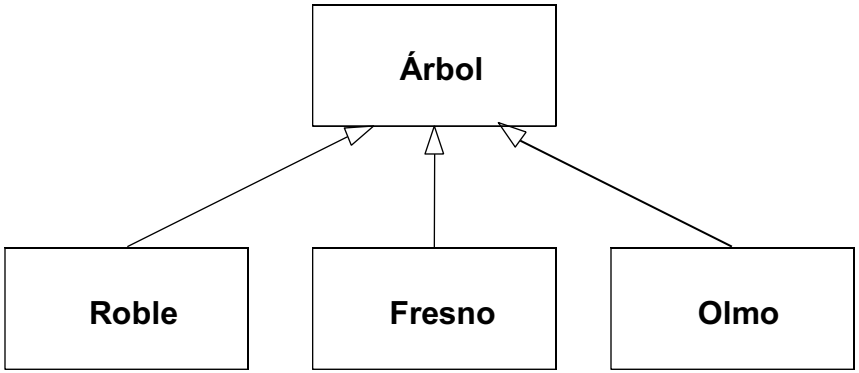


Relaciones

- **Generalización**
  - Es una relación de taxonomía entre un elemento general y otro más específico que es plenamente consistente con el primer elemento y que le añade información adicional
  - Cada instancia del clasificador específico es una instancia indirecta del clasificador general



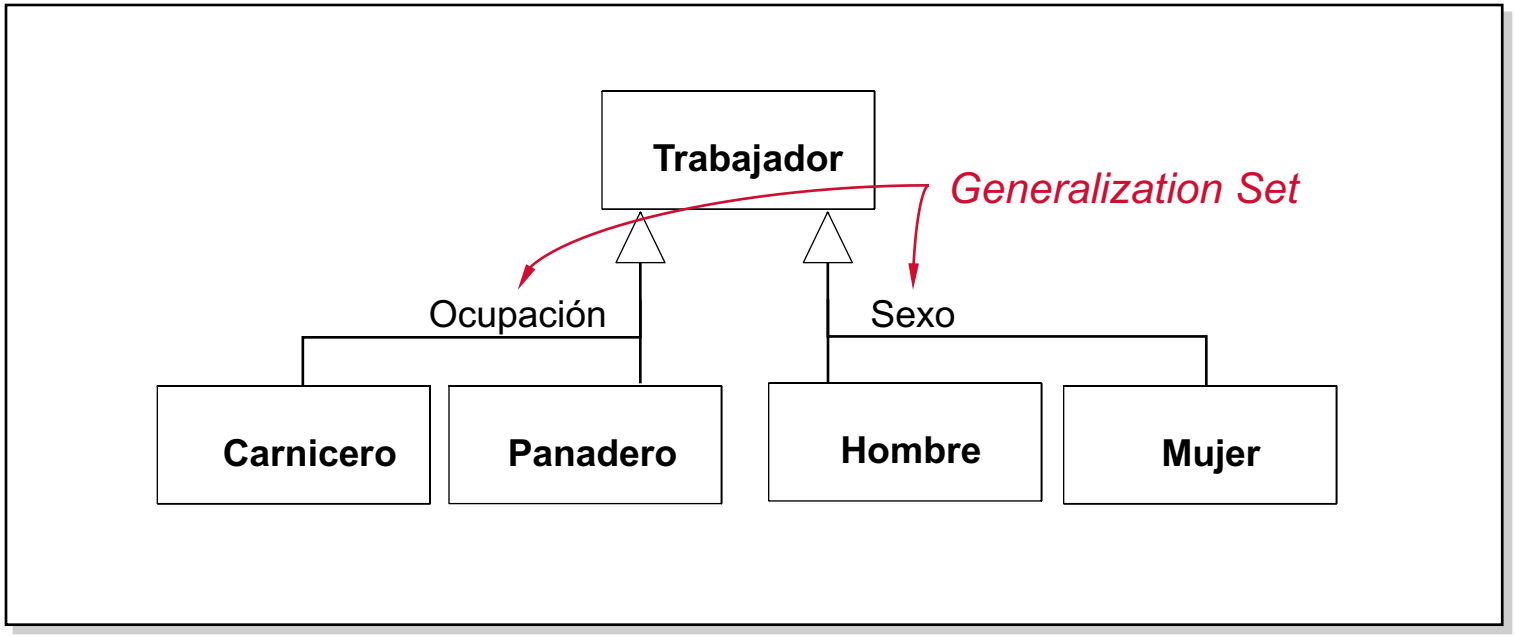
Relación de generalización entre paquetes



Relación de generalización entre clases

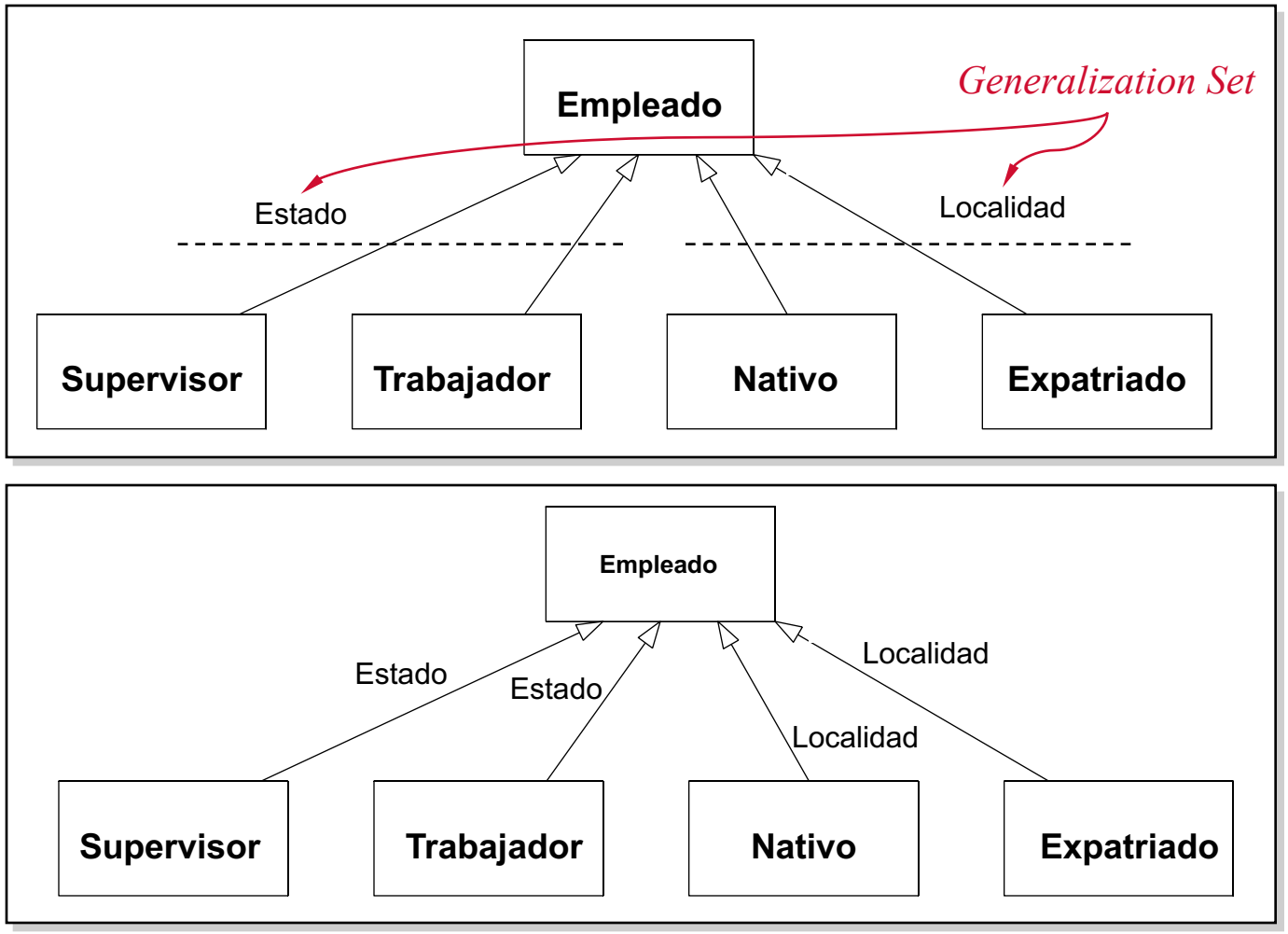
Relaciones

- **Conjunto de generalización *(Generalization Set)* (I)**
  - Se utiliza para diferenciar diferentes relaciones de generalización de un clasificador



Relaciones

- **Conjunto de generalización (II)**



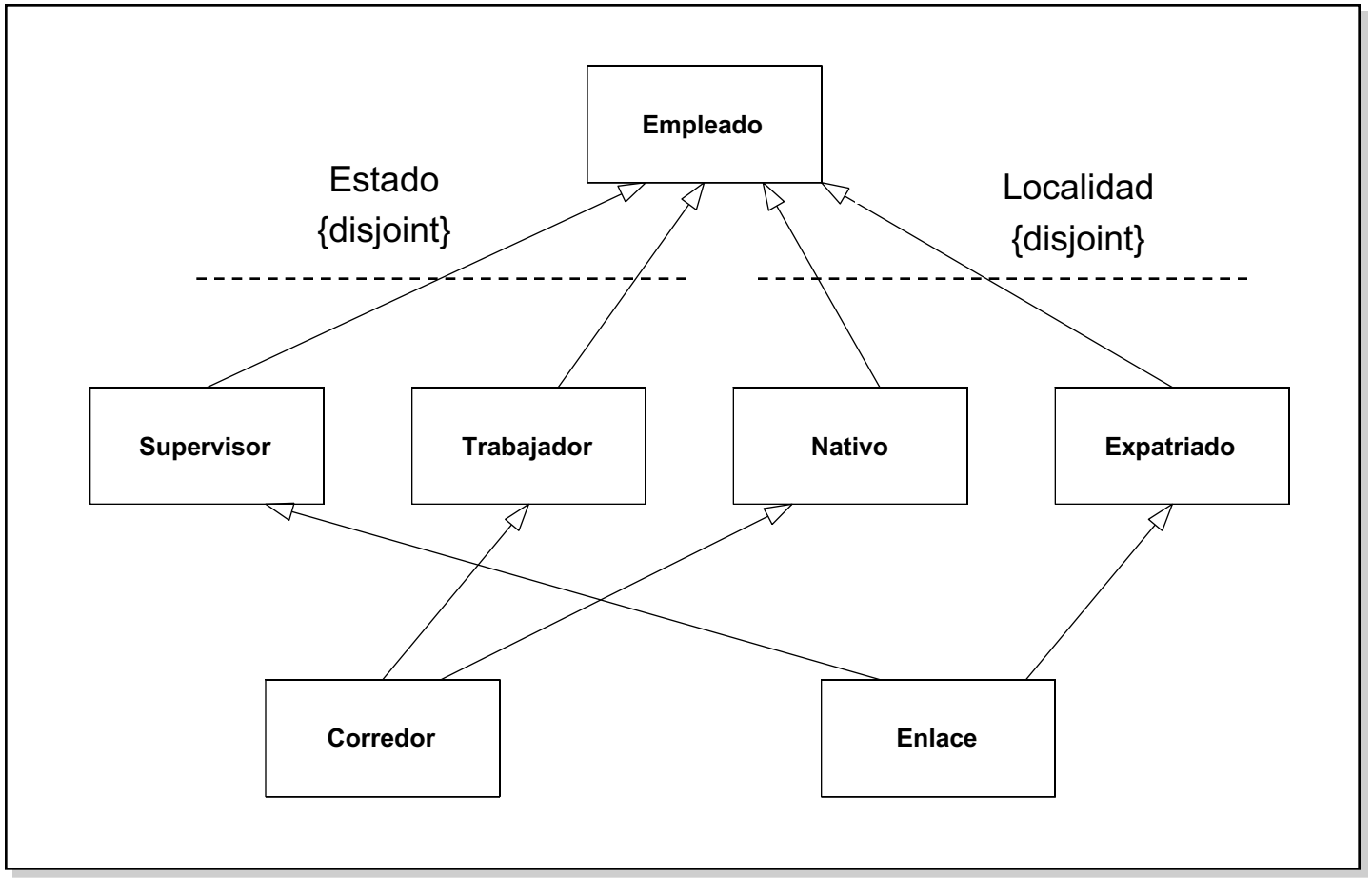
Relaciones

- **Restricciones de los conjuntos de generalización (I)**
  - ***Disjoint*** (disjunto) – Ningún elemento puede tener dos hijos en el conjunto como antecesores (en una situación de generalización múltiple). Ninguna instancia puede ser una instancia directa o indirecta de dos de los hijos (en una semántica múltiple de la clasificación)
  - ***Overlapping*** (solapado) – Un elemento puede tener dos o más hijos en el conjunto de antecesores. Una instancia puede ser una instancia de dos o más hijos
  - ***Complete*** (completo) – Todos los hijos posibles se han enumerado en el conjunto y no puede ser agregado ninguna más
  - ***Incomplete*** (incompleto) – No se ha enumerado todavía todos los hijos posibles en el conjunto. Se esperan más hijos o se conocen pero no se han declarado aún

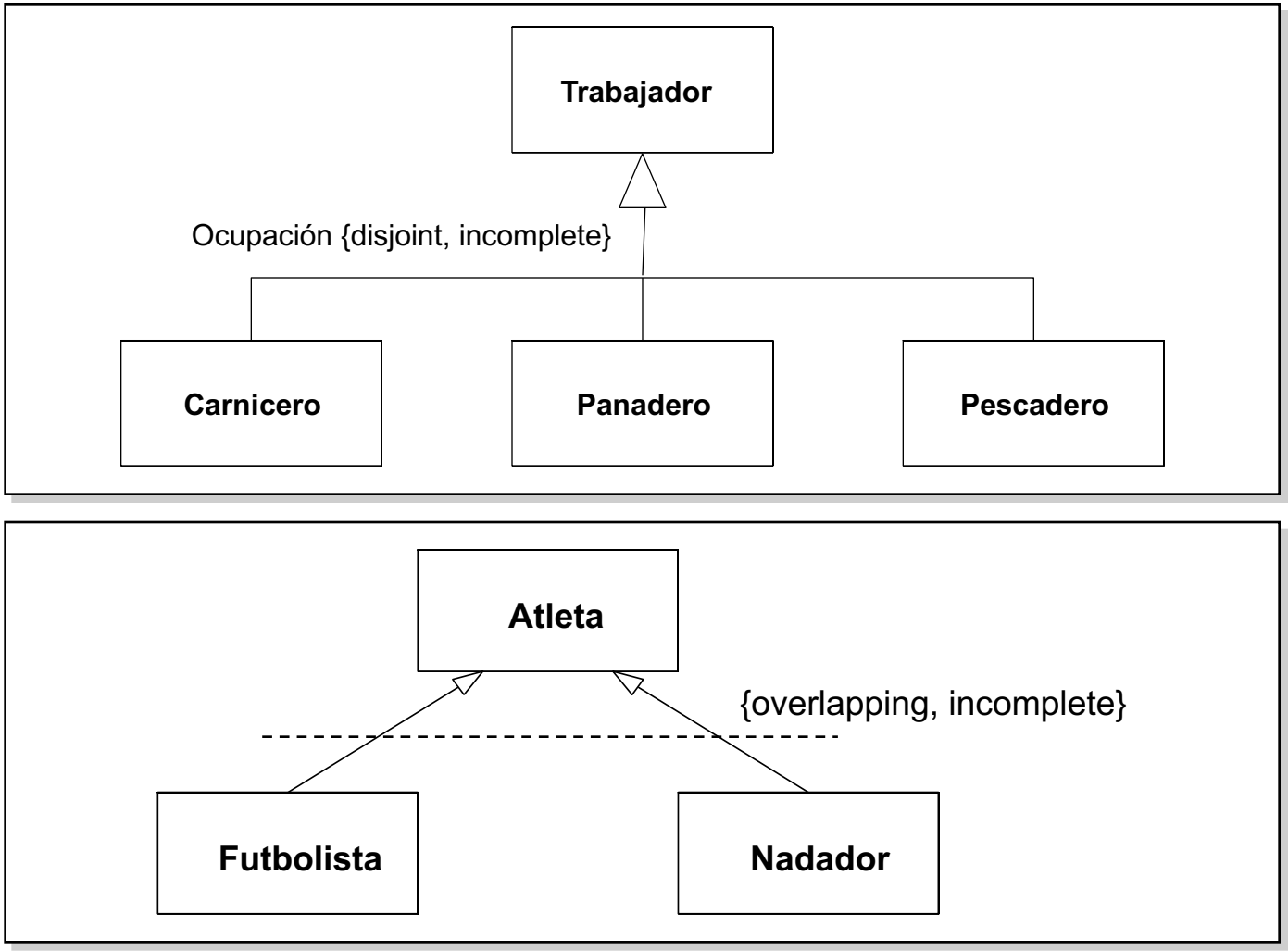
  Por defecto: **{incomplete, disjoint}**

Relaciones

- **Restricciones de los conjuntos de generalización (II)**



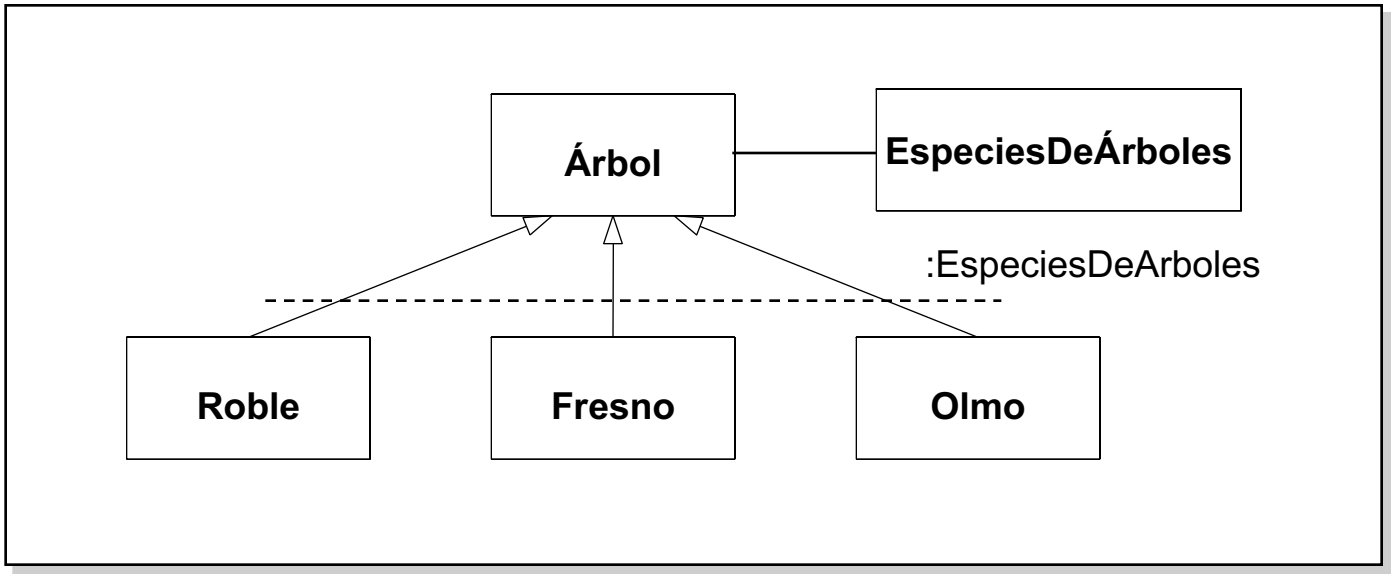
Relaciones



Formas de representar las restricciones de los conjuntos de generalización

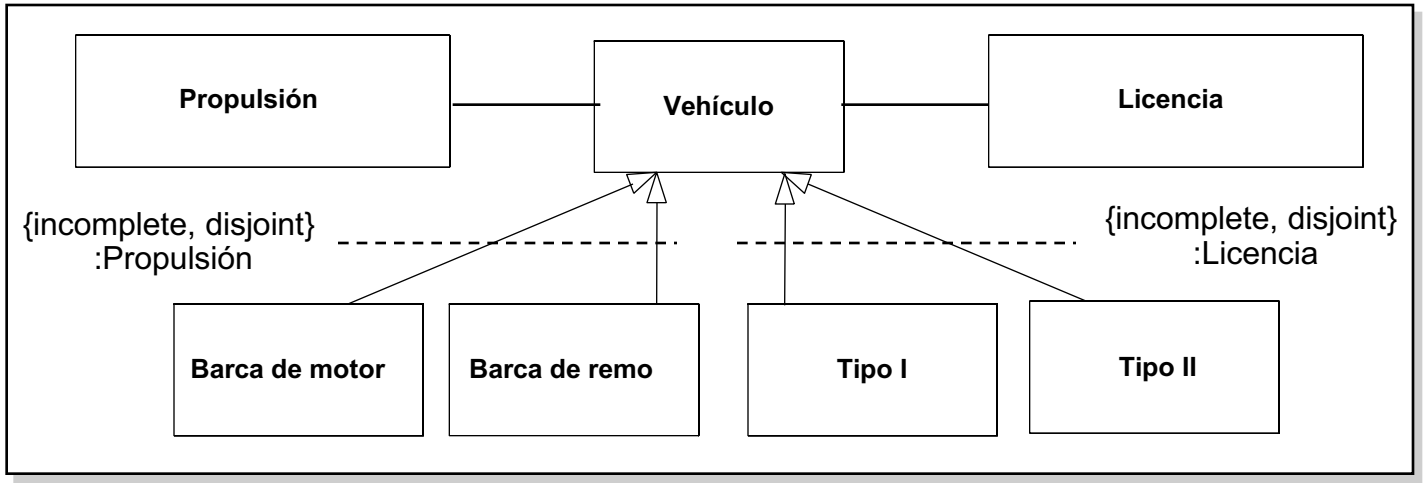
Relaciones

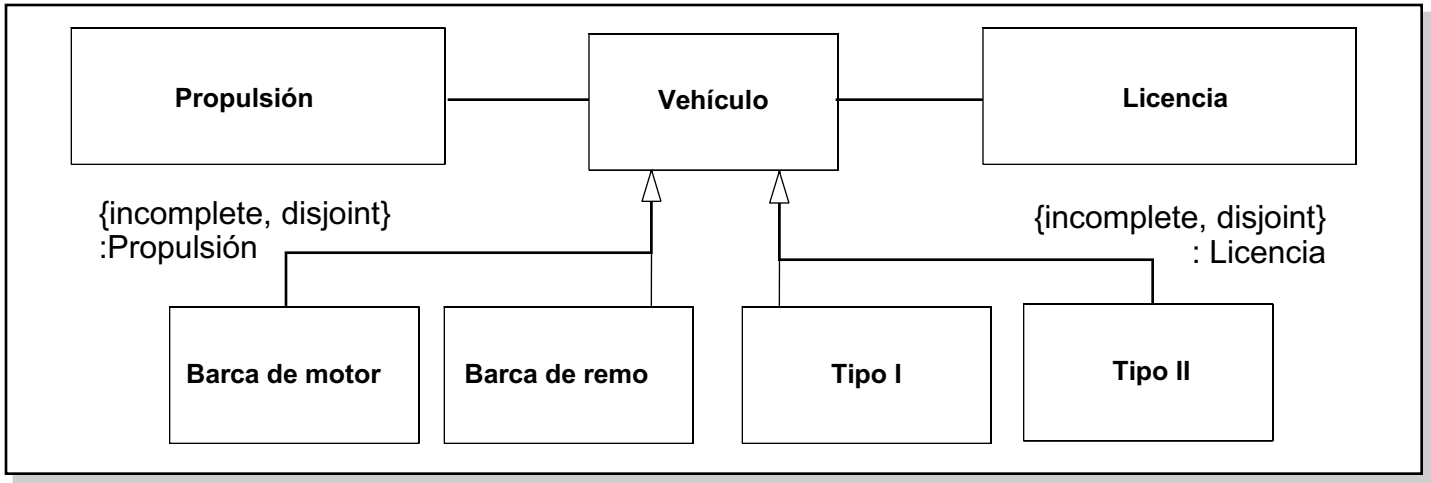
- **Supratipo (*powertype*) (I)**
  - Clasificador cuyas instancias son subclases de otro clasificador. Es una metaclase



Relaciones

- **Supratipo (II)**





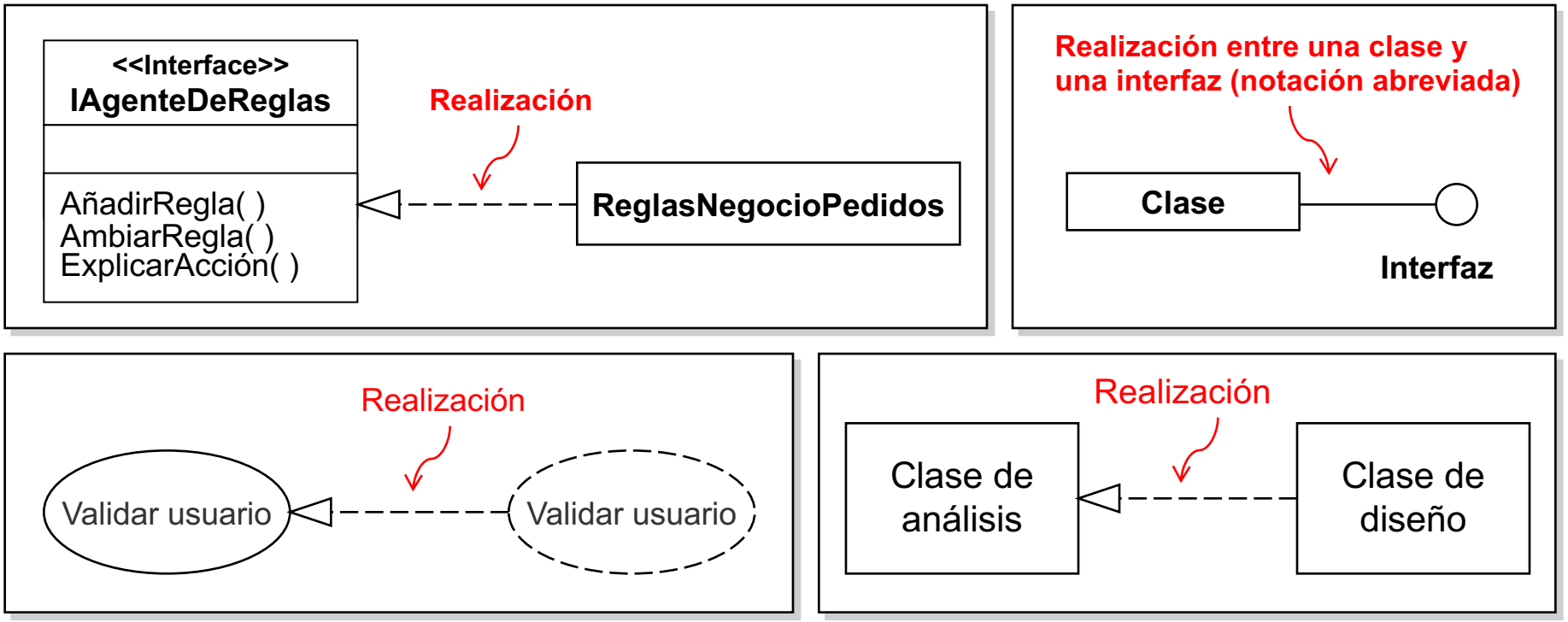
Relaciones

- **Realización (I)**
  - Es una relación entre una especificación y su implementación
    - El proveedor proporciona la especificación y el cliente la implementación de dicha especificación
    - El cliente debe soportar (por herencia o por declaración directa) todas las operaciones que el proveedor le proporciona
  - La **especificación** describe el comportamiento o la estructura de algo sin determinar cómo se implementará el comportamiento
  - Una **implementación** proporciona los detalles relativos a la forma de implementar el comportamiento
    - Un elemento puede realizar más de una especificación
  - El elemento **cliente** tiene que admitir todo el comportamiento del elemento proveedor, pero no tiene por qué satisfacer su estructura o su implementación
  - El **proveedor** de una realización indica qué operaciones tienen que estar presentes en el cliente, pero es el cliente el que tiene la responsabilidad de aportarlas

Relaciones

Realización (II)

- La relación de realización es una relación de dependencia con una notación especial:
  - Línea discontinua con una punta de flecha triangular hueca en el extremo del elemento que proporciona la especificación



Ejemplos de relaciones de realización

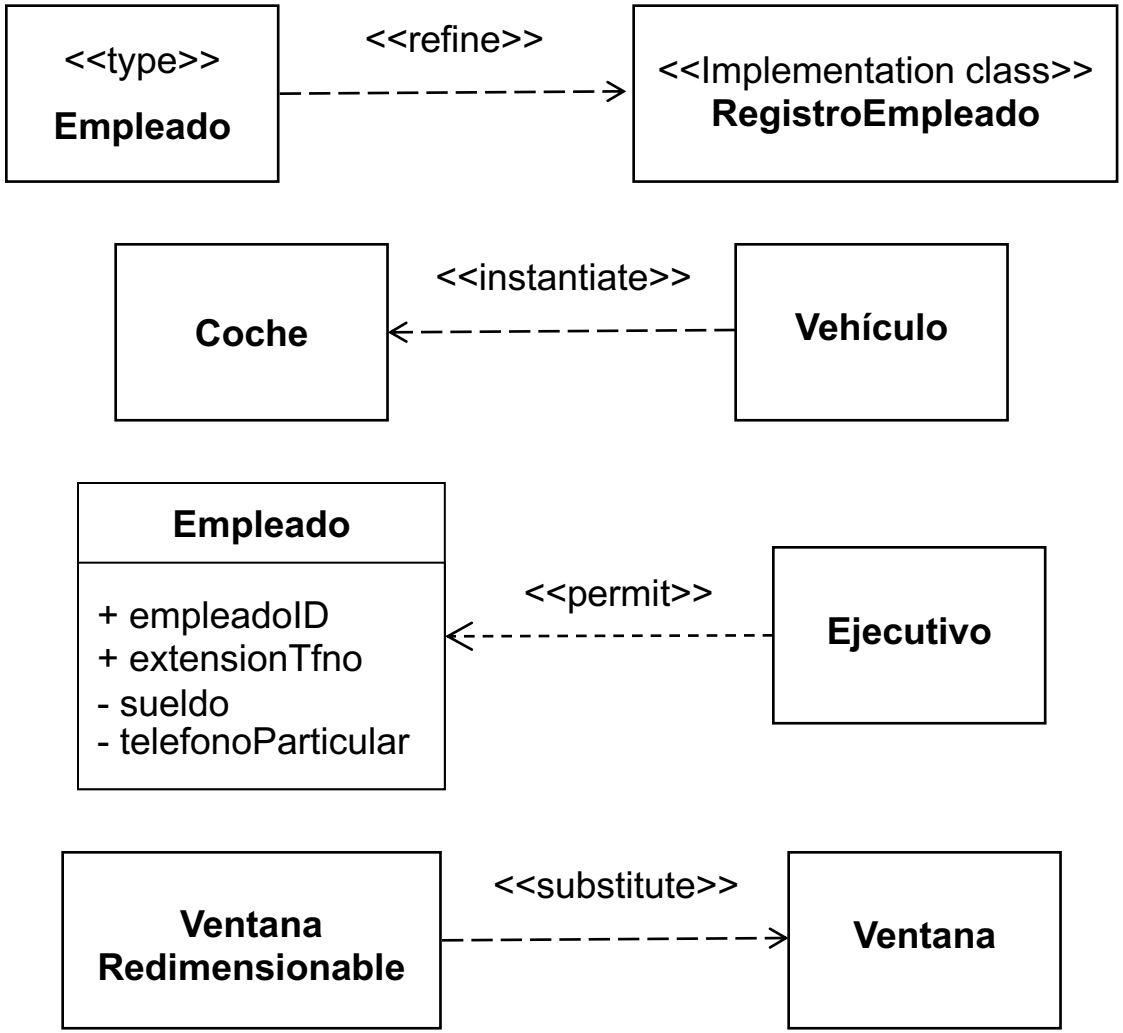
Relaciones

- **Dependencia**
  - Es una **relación semántica entre dos o más elementos del modelo**
  - Indica una situación en la que un **cambio en el elemento proveedor** requiere un cambio, o una indicación de cambio en el elemento cliente
  - Agrupa varios tipos de relaciones diferentes
  - Una dependencia puede tener:
    - Un **nombre** para indicar su **rol** en el modelo
    - Un **estereotipo** para establecer la **naturaleza** precisa de la dependencia
  - Una **dependencia entre dos paquetes** indica la presencia de, al menos, una dependencia del tipo indicado entre un elemento de cada paquete
  - Una dependencia **se representa** mediante **una flecha con línea discontinua** entre dos elementos del modelo
    - El elemento de modelo en la cola de la flecha (el cliente) depende del elemento en la punta de flecha (el proveedor)
    - La flecha se puede etiquetar con la palabra clave opcional, para indicar el tipo de dependencia, y un nombre opcional

Relaciones

- **Tipos de dependencia**
  - Las relaciones de dependencia pueden ser de varios tipos y pueden llevar asociados diferentes estereotipos
    - **Abstracción**
      - ***derive***
      - ***refine***
      - ***trace***
    - **Uso**
      - ***use***
      - ***call***
      - ***create***
      - ***instantiate***
      - ***send***
    - **Permiso**
      - ***permit***
    - **Sustitución**
      - ***substitute***

Relaciones

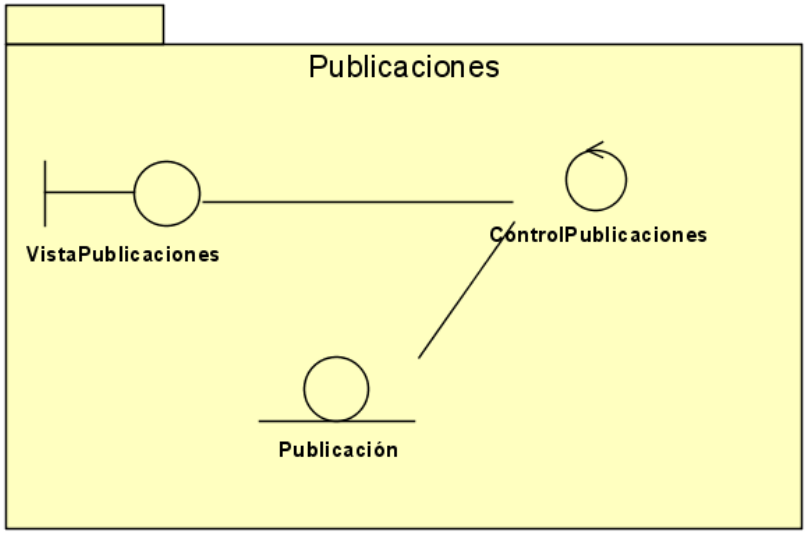


Ejemplos de tipos de dependencia

Vista de gestión del modelo

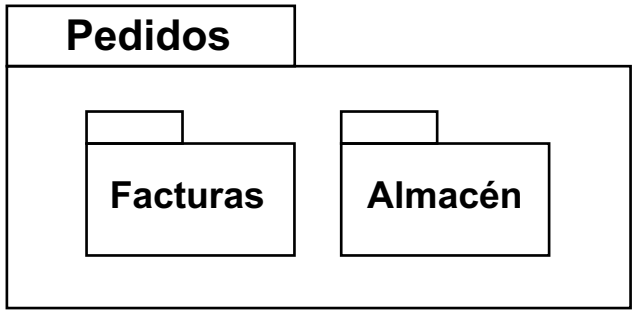
Características

- Modela la organización del modelo mismo mediante un conjunto de paquetes
- Un modelo abarca un conjunto de paquetes que contienen los elementos del modelo tales como clases, máquinas de estados y casos de uso
- Un elemento de modelado puede pertenecer a más de un paquete
- La agrupación de elementos de modelado en un paquete no es necesario que coincida con la agrupación física de elementos del sistema



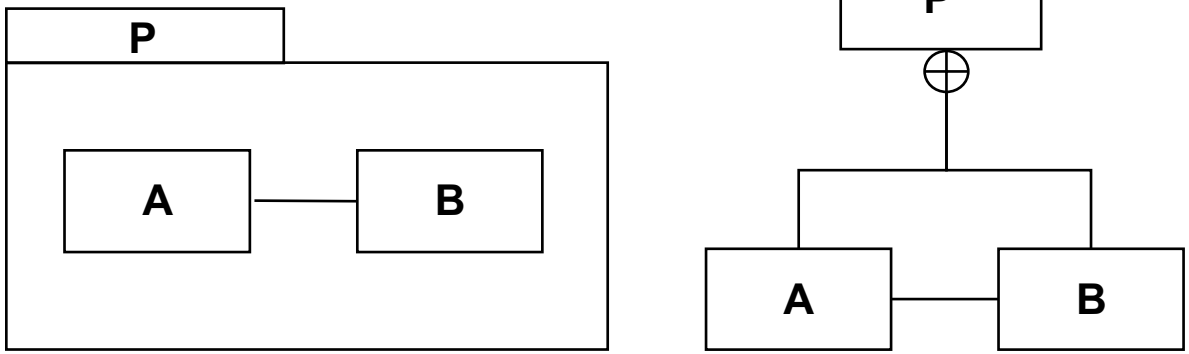
Paquetes

- Los paquetes constituyen un mecanismo de agrupación para organizar elementos UML
- Proporcionan un espacio de nombres a los elementos agrupados
- Los paquetes se organizan jerárquicamente, siendo el paquete raíz el que contiene todo el sistema
- Un paquete se representa como un rectángulo grande con un rectángulo pequeño sobre su esquina superior izquierda. El nombre puede aparecer en cualquiera de los dos rectángulos. Si se muestra el contenido del paquete, entonces el nombre aparece en el rectángulo pequeño



Paquetes

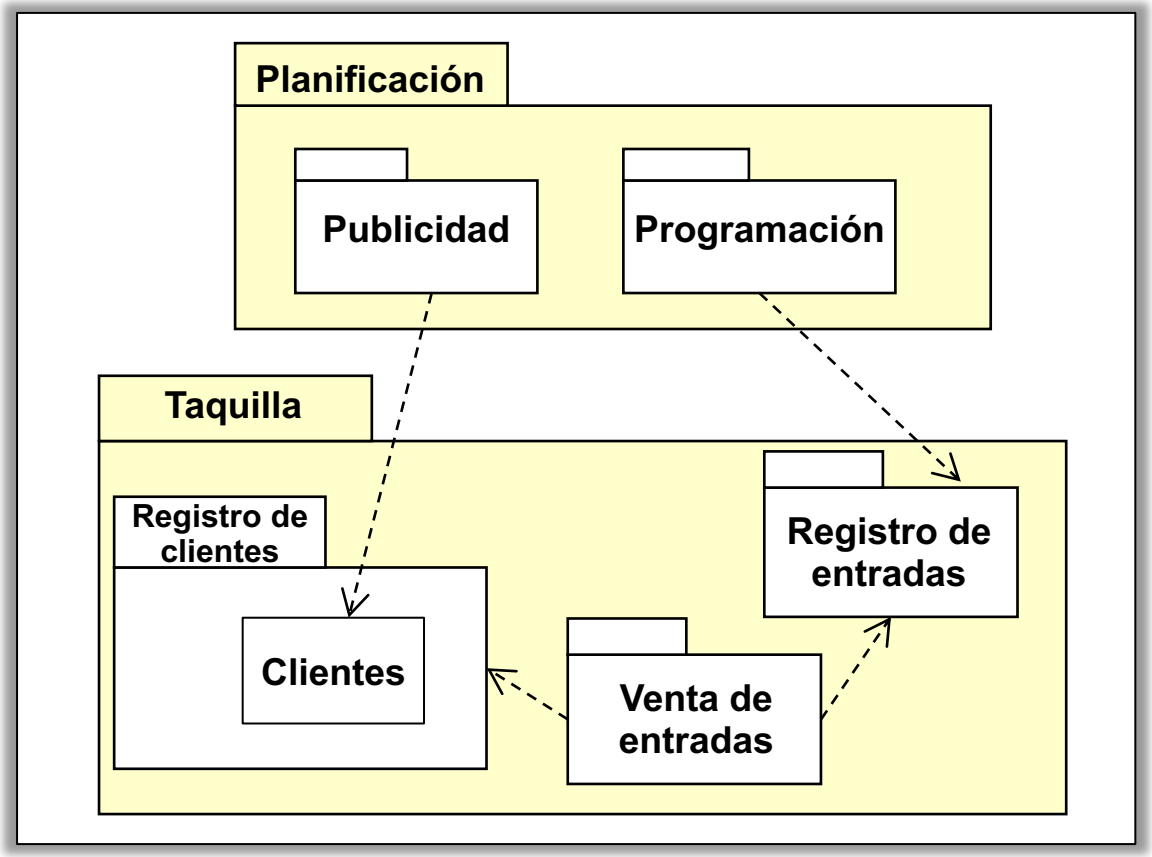
- **Contenido de un paquete**
  - **Elementos "empaquetables"** (***packageable elements***) son elementos con nombre que un paquete posee de forma directa
  - La visibilidad de los elementos de un paquete indica si son accesibles desde fuera del paquete. Se representa colocando delante del nombre del elemento los signos:
    - + para visibilidad pública
    - - para visibilidad privada
  - Los elementos que contiene un paquete se pueden representar de dos formas

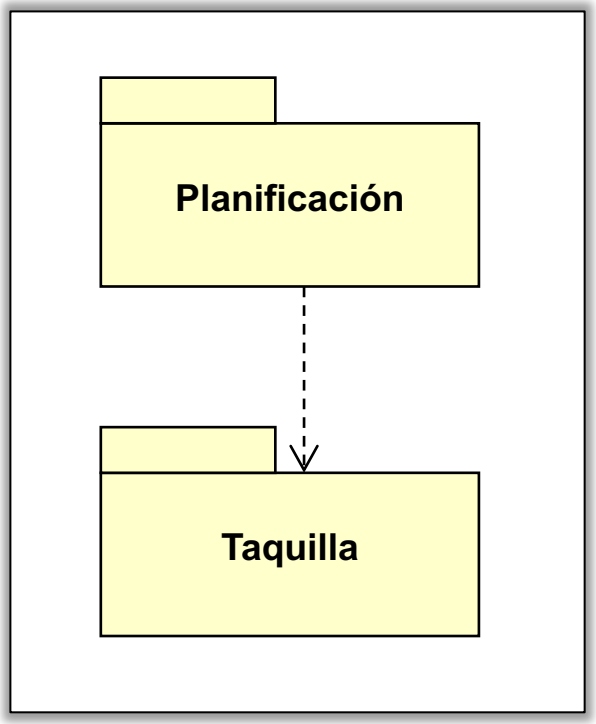


Notaciones diferentes para representar que el paquete P contiene las clases A y B

Paquetes

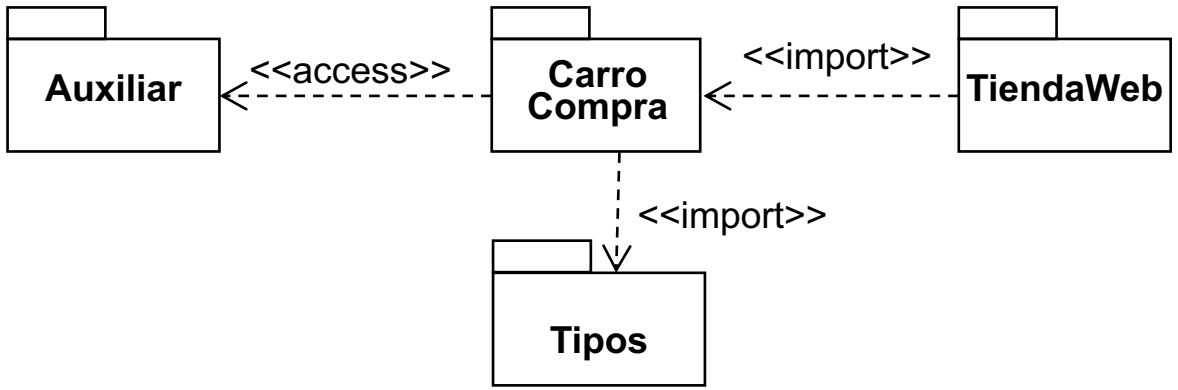
- **Relaciones entre paquetes (I)**
  - Las relaciones permitidas entre paquetes son ***generalización***, ***dependencia*** y ***refinamiento***
  - Las **dependencias** entre paquetes resumen las dependencias entre elementos individuales que están en ellos



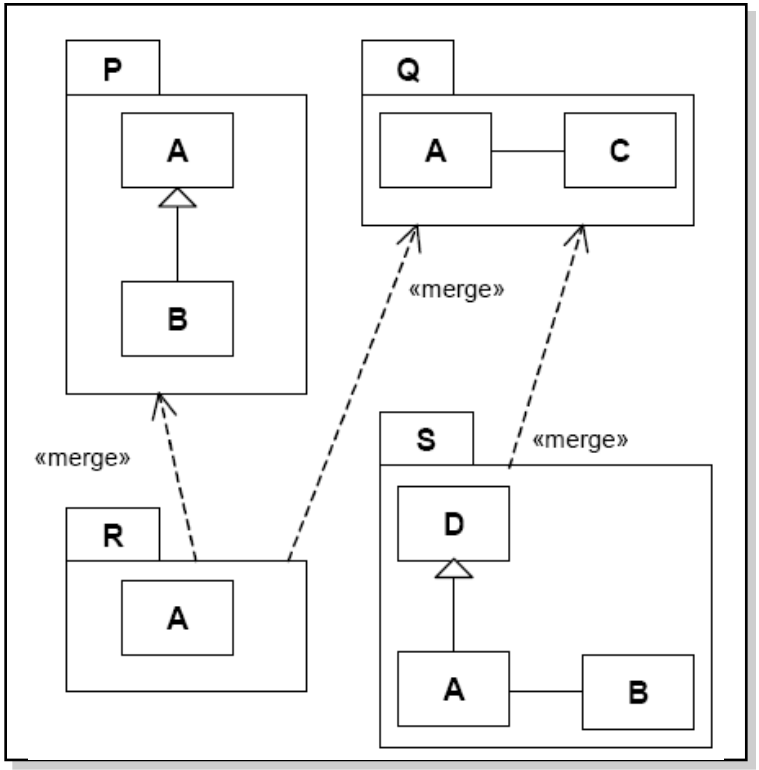


Paquetes

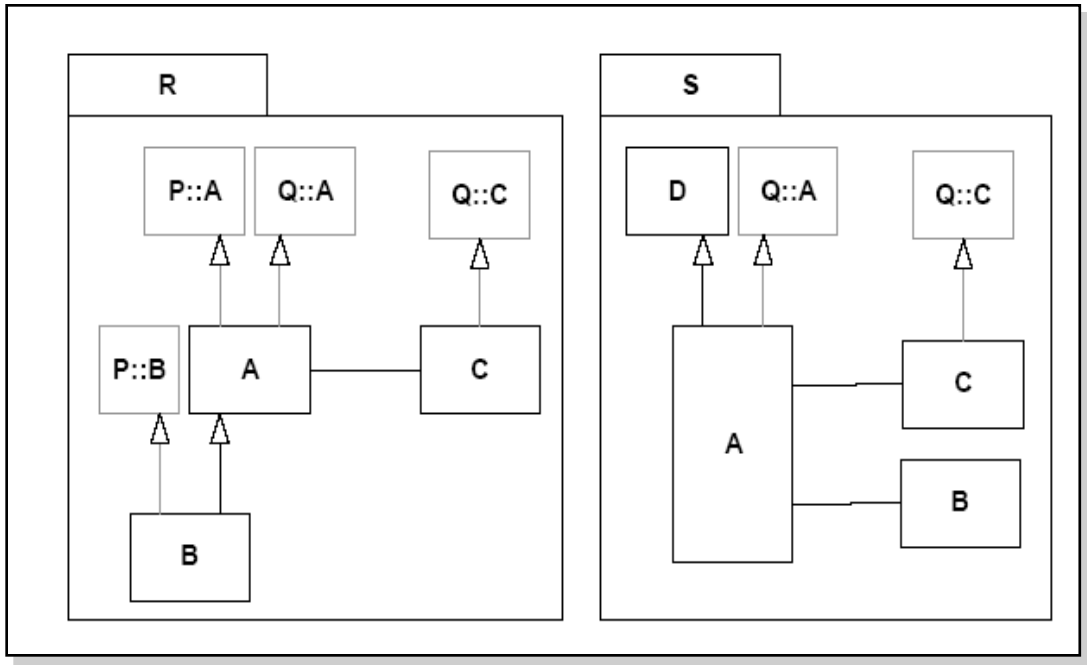
- **Relaciones entre paquetes (II)**
  - Existen tres tipos de dependencia específicos para paquetes
    - ***Importación***: permite a un paquete importar los elementos de otro y referenciarlos sin usar el nombre calificado. Se utiliza el estereotipo **<<import>>** para importación de un paquete público y **<<access>>** para un paquete privado
    - ***Fusión*** (***merge***): relación entre dos paquetes en la que el contenido del paquete objetivo es fusionado con el contenido del paquete fuente usando generalizaciones y redefiniciones. Se utiliza el estereotipo <<**merge**>>



Paquetes

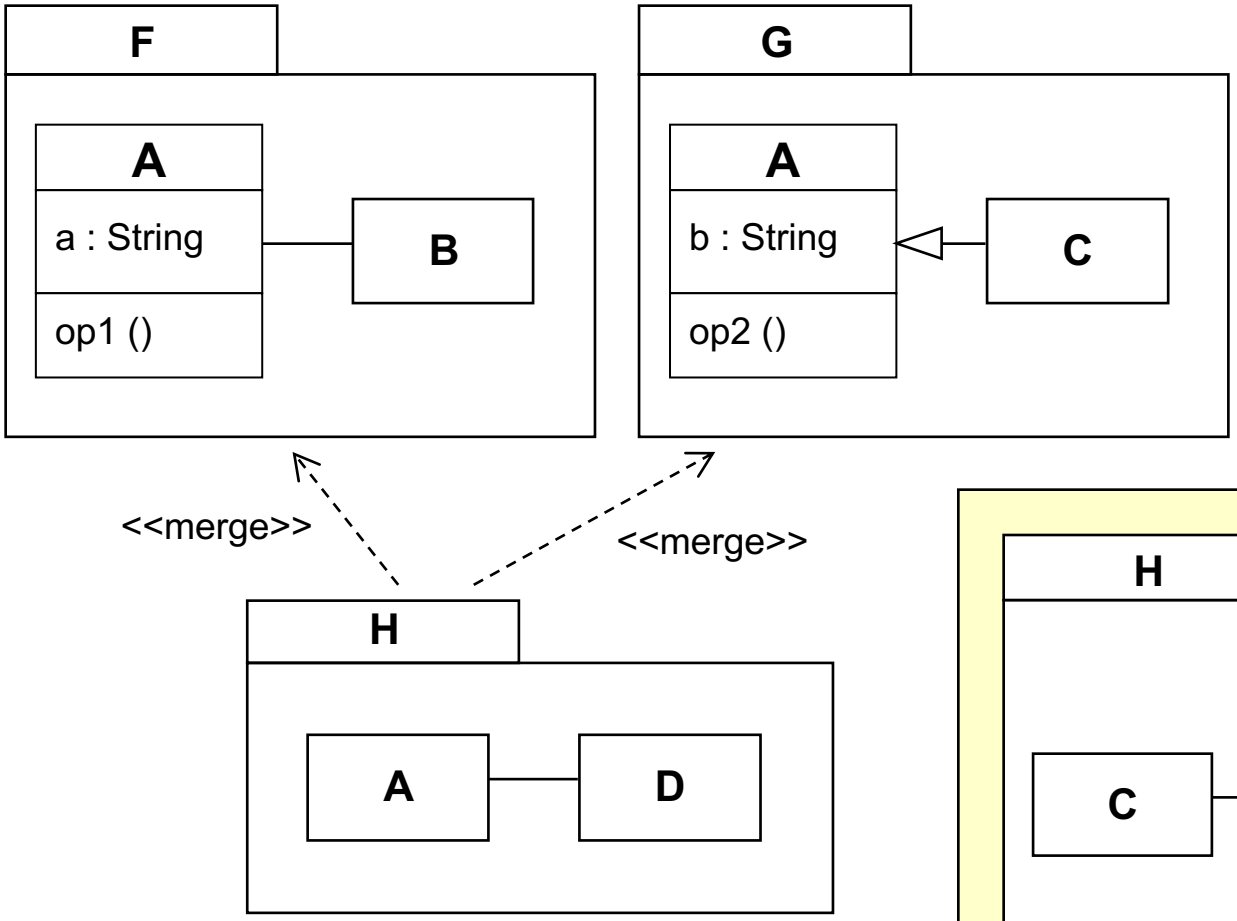


Ejemplo de relaciones <<merge>> entre paquetes

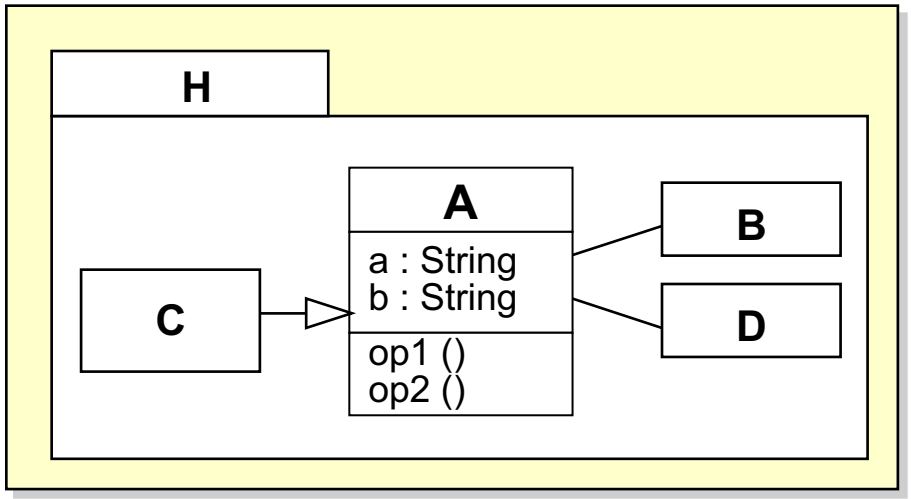


Resultado de la aplicación de las relaciones “merge” anteriores

Paquetes



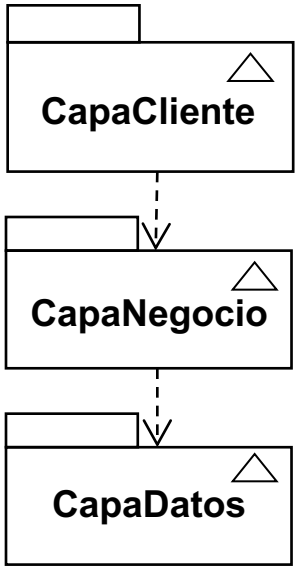
Relación <<merge>> entre paquetes
(H se fusiona con F y G)

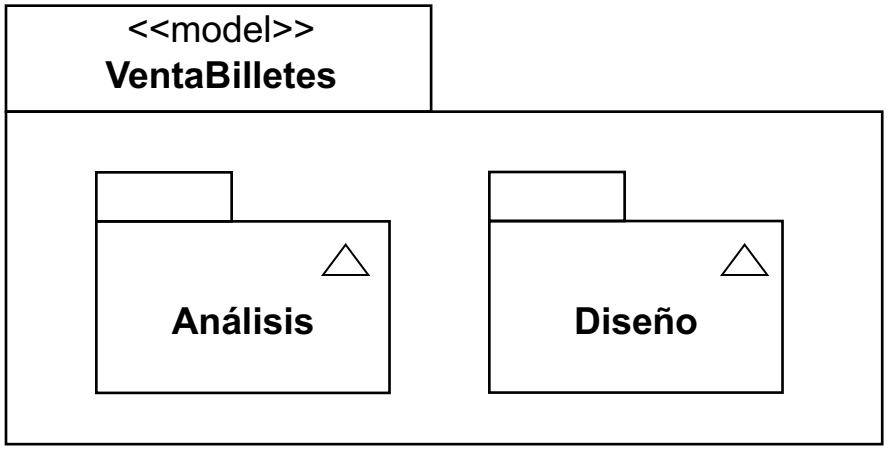


Paquete H transformado

Paquetes

- **Modelos (I)**
  - Un modelo es una abstracción del sistema físico que captura una vista particular del sistema para un determinado propósito
  - Se representa como un paquete que contiene un conjunto de elementos que juntos describen el sistema que se modela
  - La notación de un modelo es el símbolo del paquete con un pequeño triángulo en la esquina superior derecha del rectángulo grande. También se puede utilizar el estereotipo <<model>> encima del nombre del modelo





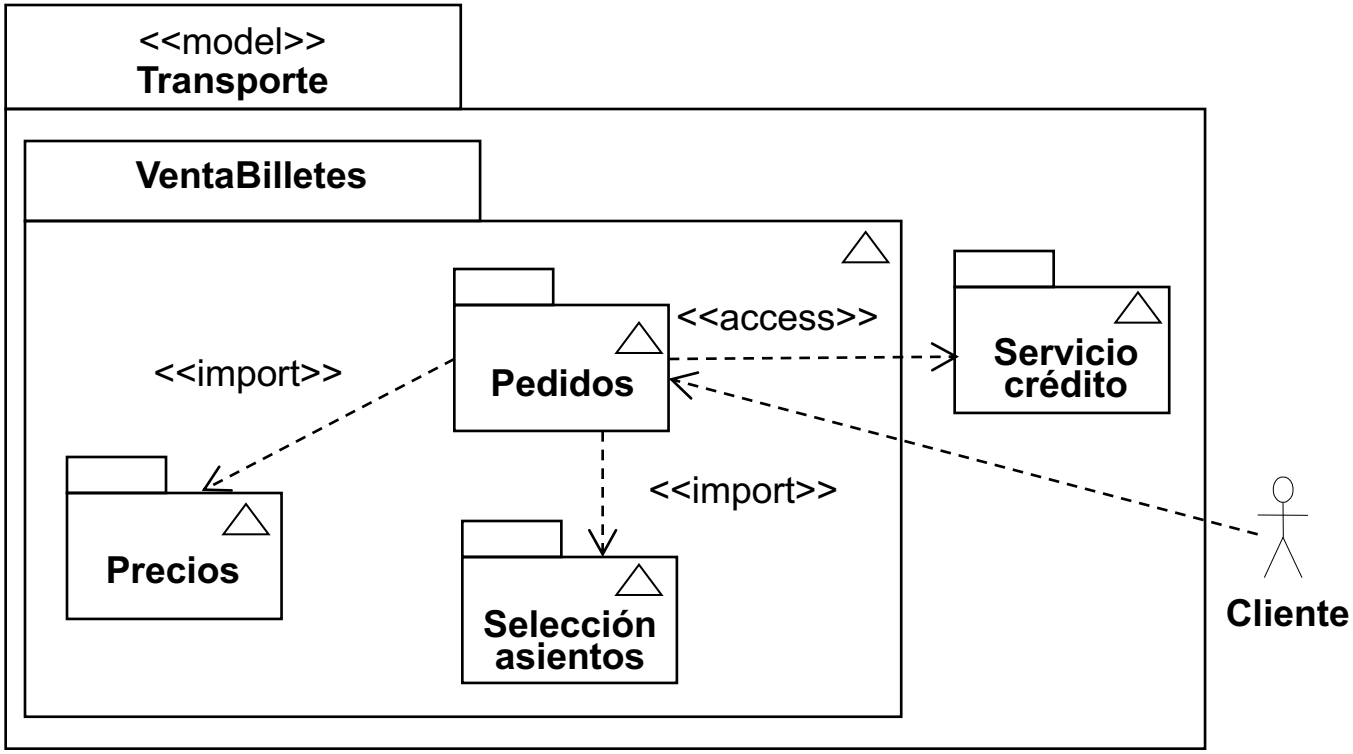
Ejemplos de representación de modelos

Paquetes

- **Modelos (II)**
  - Un modelo contiene o importa todos los elementos que necesita para representar completamente el sistema físico (de acuerdo a su propósito)
    - Los elementos se organizan en una jerarquía de contenedores donde el paquete raíz o subsistema representa los límites del sistema físico
    - Los elementos que describan partes relevantes del entorno del sistema (actores) pueden aparecer en el modelo pero fuera de la jerarquía de paquetes/subsistemas
  - Las relaciones de refinamiento o *mapping* entre modelos generalmente se descomponen en dependencias entre elementos contenidos en los modelos

Paquetes

- **Modelos (III)**



Anidamiento de modelos

Vista de casos de uso

Características

- Los casos de uso son una técnica para la especificación de requisitos funcionales propuesta inicialmente por **Ivar Jacobson** [Jacobson, 1987], [Jacobson et al. 1992] e incorporada a UML
- Modela la funcionalidad del sistema tal como la perciben los agentes externos, denominados actores, que interactúan con el sistema desde un punto de vista particular
- Sus componentes principales son:
  - **Sujeto**: sistema que se modela
  - **Casos de uso**: unidades funcionales completas
  - **Actores**: entidades externas que interactúan con el sistema
- El sujeto se muestra como una caja negra que proporciona los casos de uso
- El modelo de casos de uso se representa mediante los **diagramas de casos de uso**

Características

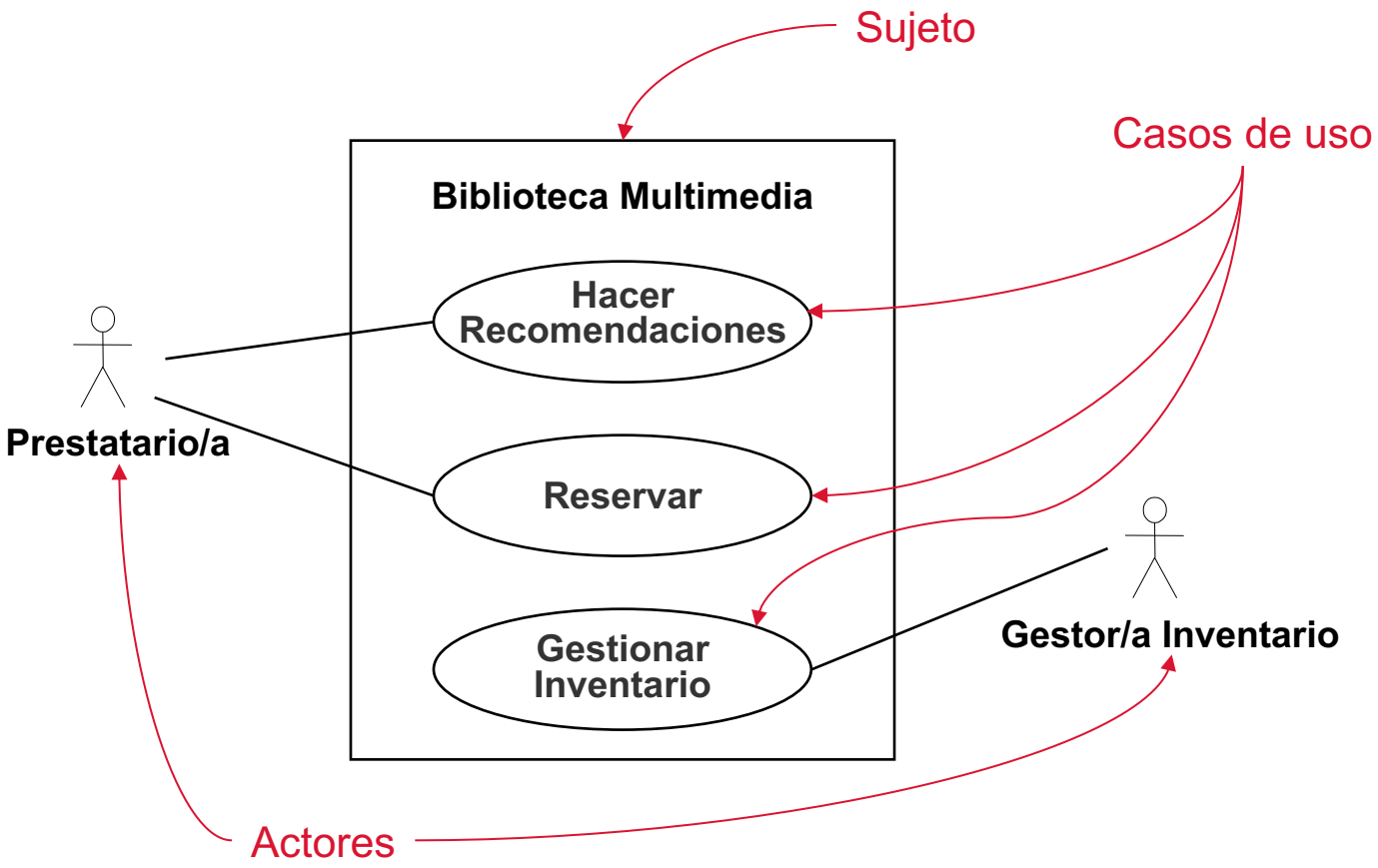
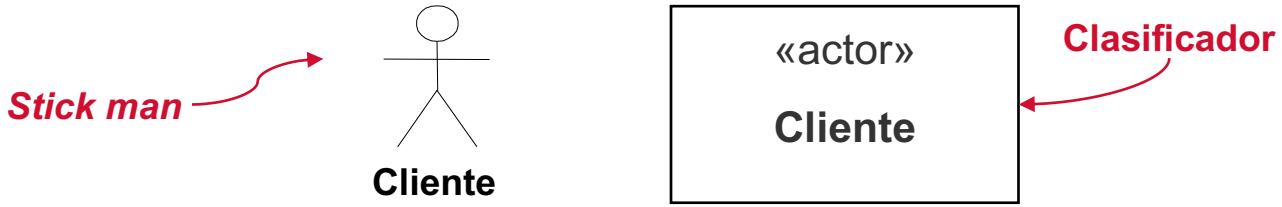


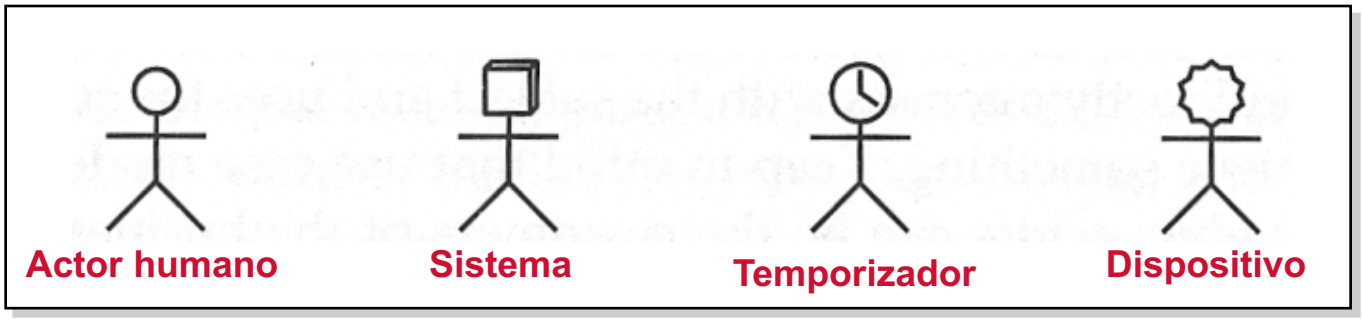
Diagrama de casos de uso

Actores

- Un actor es un clasificador que modela un tipo de rol que juega una entidad que interacciona con el sujeto pero que es externa a él
  - Un actor puede tener múltiples instancias físicas
  - Una instancia física de un actor puede jugar diferentes papeles
- Los actores se comunican con el sujeto intercambiando mensajes (señales, llamadas o datos)
- Notación:
  - Se representan con el icono estándar de "*stick man*" o "monigote" con el nombre del actor (obligatorio) cerca del símbolo, normalmente se pone encima o debajo
  - También se puede representar mediante un símbolo de clasificador con el estereotipo **«actor»**
  - Los nombres de los actores suelen empezar por mayúscula
  - Se pueden usar otros símbolos para representar tipos de actores, por ejemplo para representar actores no humanos

Actores



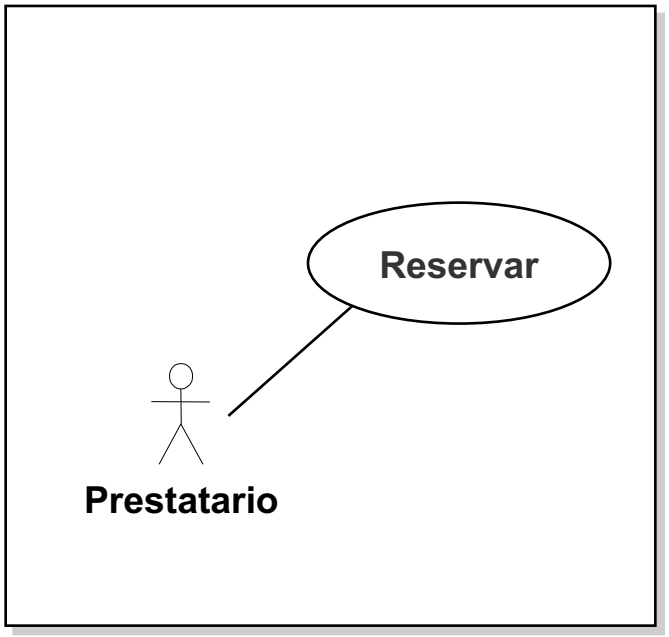


Símbolos utilizados para representar tipos de actores

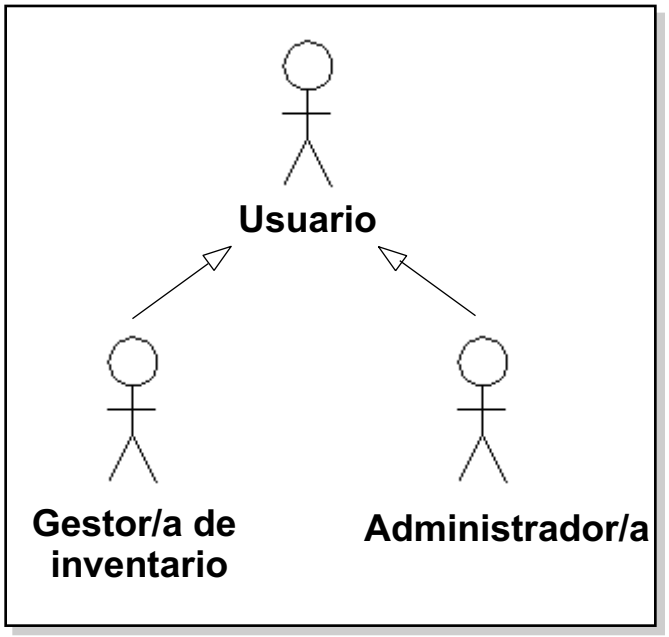
Relaciones entre actores

- Los actores sólo pueden tener asociaciones con casos de uso, subsistemas, componentes y clases y dichas asociaciones deben ser binarias
- Se pueden establecer relaciones de generalización entre actores
  - El actor general describirá el comportamiento de un rol más general
  - Los actores especializados heredan el comportamiento del actor general y lo extienden de alguna forma
  - Una instancia de un actor descendiente siempre se puede utilizar en aquellos casos en los que se espera una instancia del actor antecesor
  - Los actores pueden ser abstractos, en ese caso se representan con el nombre en cursiva

Relaciones entre actores



Asociación entre un actor y un caso de uso



Relaciones de generalización entre actores

Casos de uso

- Un caso de uso se define como un conjunto de acciones realizadas por el sistema que dan lugar a un resultado observable
- El caso de uso especifica un comportamiento que el sujeto puede realizar en colaboración con uno o más actores, pero sin hacer referencia a su estructura interna
- El caso de uso puede contener posibles variaciones de su comportamiento básico incluyendo manejo de errores y excepciones
- Una instanciación de un caso de uso es un **escenario** que representa un uso particular del sistema (un camino)
- Características de los casos de uso
  - Un caso de uso se inicia por un actor
  - Los casos de uso proporcionan valores a los actores
  - La funcionalidad de un caso de uso debe ser completa
- El comportamiento de un caso de uso se puede describir mediante interacciones, actividades, máquinas de estado...

Casos de uso

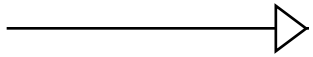
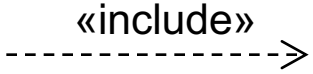
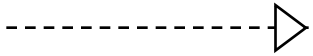
- Notación
  - Elipse con el nombre del caso de uso dentro o debajo de ella. Se puede colocar algún estereotipo encima del nombre y una lista de propiedades debajo
  - La representación alternativa es la del símbolo del clasificador con una elipse pequeña en la esquina superior derecha

Notaciones usadas para la representación de casos de uso

Relaciones de los casos de uso

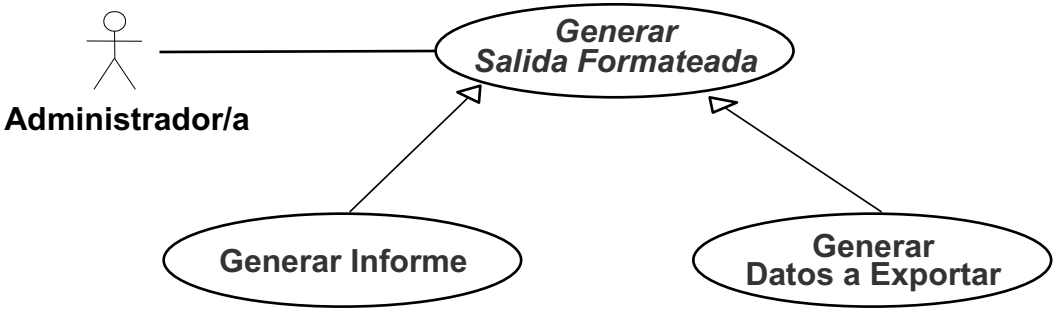
- Los casos de uso pueden tener asociaciones y dependencias con otros clasificadores
- Relación **entre actores y casos de uso**
  - **Asociación**
- Relaciones **entre casos de uso**
  - **Generalización**: Un caso de uso también se puede especializar en uno o más casos de uso hijos
  - **Inclusión**: Un caso de uso puede incorporar el comportamiento de otros casos de uso como fragmentos de su propio comportamiento
  - **Extensión**: Un caso de uso también se puede definir como una extensión incremental de un caso de uso base
- Relación **entre un caso de uso y una colaboración**
  - **Realización**

Relaciones de los casos de uso

Relación	Notación
Asociación	
Generalización	
Inclusión	
Extensión	
Realización	

Relaciones de los casos de uso

- **Generalización de casos de uso**
  - Una relación de generalización relaciona un caso de uso especializado con otro caso de uso más general
  - El hijo hereda las relaciones y comportamiento del padre y puede agregar atributos y operaciones propios
  - El caso de uso hijo añade comportamiento al caso de uso padre **insertando secuencias de acción** adicionales en la secuencia del padre en puntos arbitrarios
  - También puede **modificar algunas operaciones y secuencias heredadas**, pero debe hacerse de forma que la intención del padre se mantenga
  - El caso de uso padre puede ser abstracto



Relaciones de generalización entre casos de uso

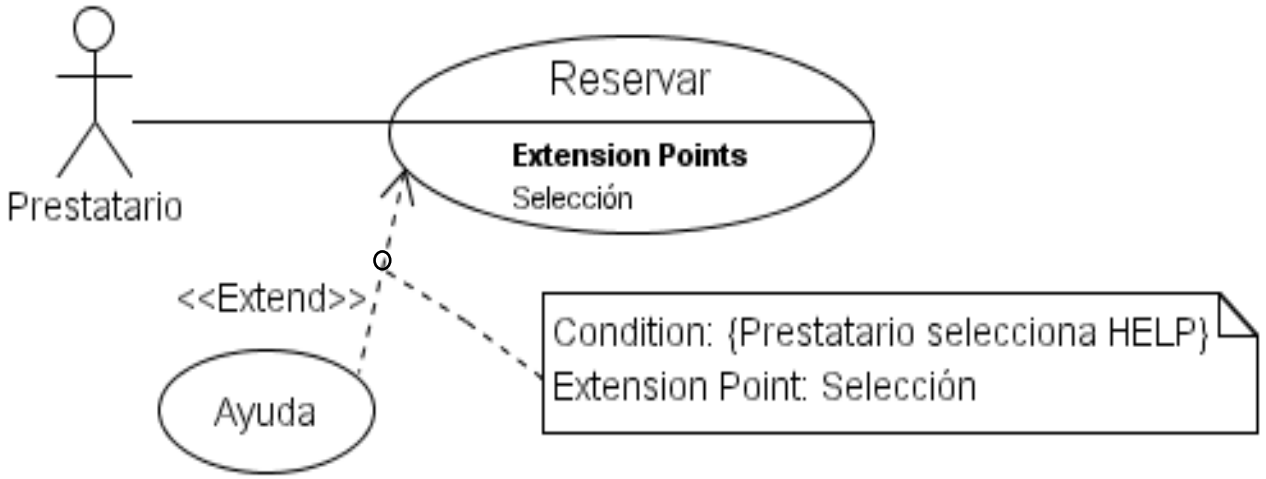
Relaciones de los casos de uso

- **Relación de extensión (I)**
  - Dependencia entre dos casos de uso que especifica que el comportamiento de un caso de uso base (**extendido**) puede ser extendido con comportamiento adicional definido en otro caso de uso (**extensor**)
  - El caso de uso extendido define un comportamiento que tiene significado con independencia del caso de uso extensor
  - El comportamiento del caso de uso extensor incrementa el del caso de uso base sólo en determinadas condiciones
  - Un caso de uso extensor puede extender varios casos de uso base y puede, a su vez, ser extendido por otro caso de uso
  - La extensión tiene lugar en **puntos de extensión**
    - pertenecen al caso de uso extendido
    - Indican el lugar donde se insertan los **fragmentos de comportamiento** del caso de uso extensor

Relaciones de los casos de uso

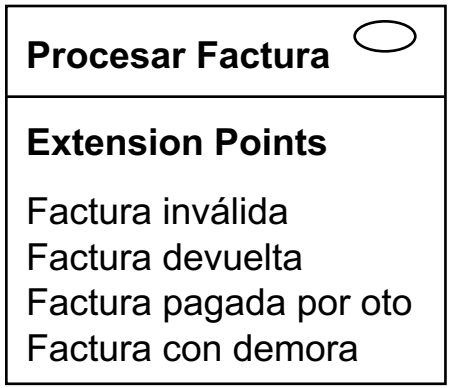
- **Relación de extensión (II)**
  - Notación de la relación de extensión: símbolo de dependencia con el estereotipo **«extend»** y opcionalmente una nota con las condiciones y las referencias a los puntos de extensión
  - Notación de los puntos de extensión: se representan como una cadena de texto dentro del caso de uso conforme a la sintaxis:

< nombre > [: <explicación>]



Relaciones de los casos de uso

- **Relación de extensión (III)**
  - Si hay varios puntos de extensión en un caso de uso es mejor representar el caso de uso con el símbolo del clasificador
  - **Condición de la extensión**
    - Es única para todos los puntos de extensión de una relación de extensión



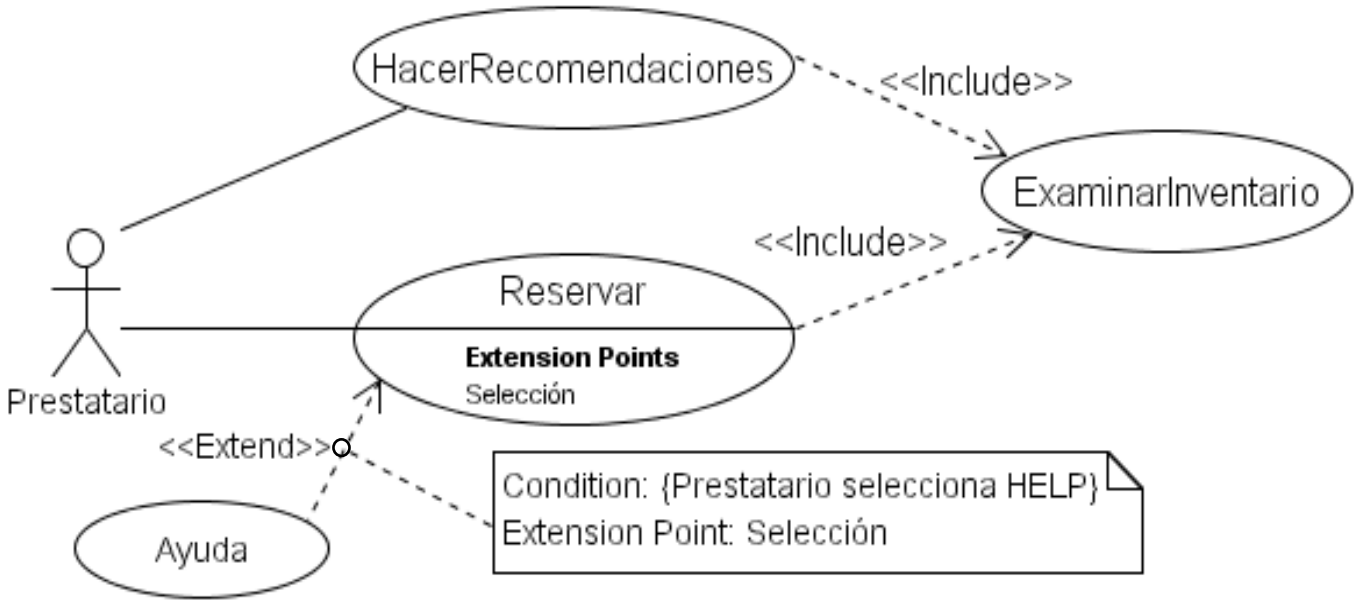
      - Si es verdadera cuando se alcanza el primer punto de extensión al ejecutar el caso de uso base, serán ejecutados todos los fragmentos del caso de uso extensor correspondientes a todos los puntos de extensión. Terminada la ejecución de un fragmento dado el control retorna al caso de uso base y continua su ejecución hasta el siguiente punto de extensión
      - Si la condición es falsa no se produce la extensión
      - Si no hay condición la extensión es incondicional

Relaciones de los casos de uso

- **Relación de inclusión (I)**
  - Relación entre dos casos de uso que indica que el comportamiento de un caso de uso (**incluido**) se inserta en el comportamiento de otro caso de uso (**base** o **inclusor**) en la **localización** especificada en este último
  - La inclusión no es condicional
  - El propósito de la inclusión es la reutilización de porciones de comportamiento comunes a varios casos de uso
    - Un caso de uso incluido puede insertarse en varios casos de uso base y puede, a su vez, incluir otros casos de uso
    - Un caso de uso base puede tener relaciones de inclusión con varios casos de uso incluidos
  - La ejecución es análoga a las llamadas a procedimientos
  - Notación de la relación de inclusión: símbolo de dependencia con el estereotipo **«include»**

Relaciones de los casos de uso

- **Relación de inclusión (II)**



Organización de los casos de uso

- Los casos de uso se pueden agrupar en paquetes
  - Los paquetes se pueden organizar jerárquicamente
  - Un caso de uso puede estar en más de un paquete
  - Pueden existir relaciones entre casos de uso de diferentes paquetes
  - Se pueden agrupar actores en un paquete
- Los clasificadores pueden poseer casos de uso
  - Forma de organización alternativa permitida en UML 2
  - El conjunto completo de casos de uso de un clasificador especifica todas las distintas formas que hay de utilizar ese clasificador

Organización de los casos de uso

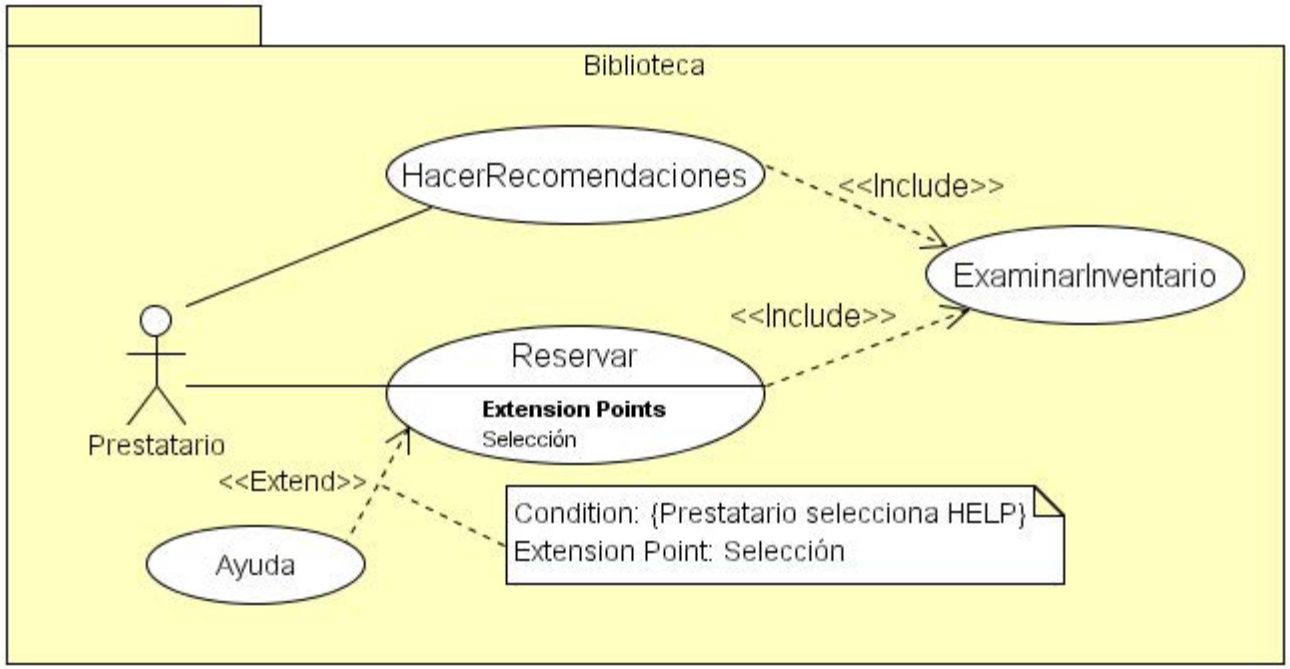
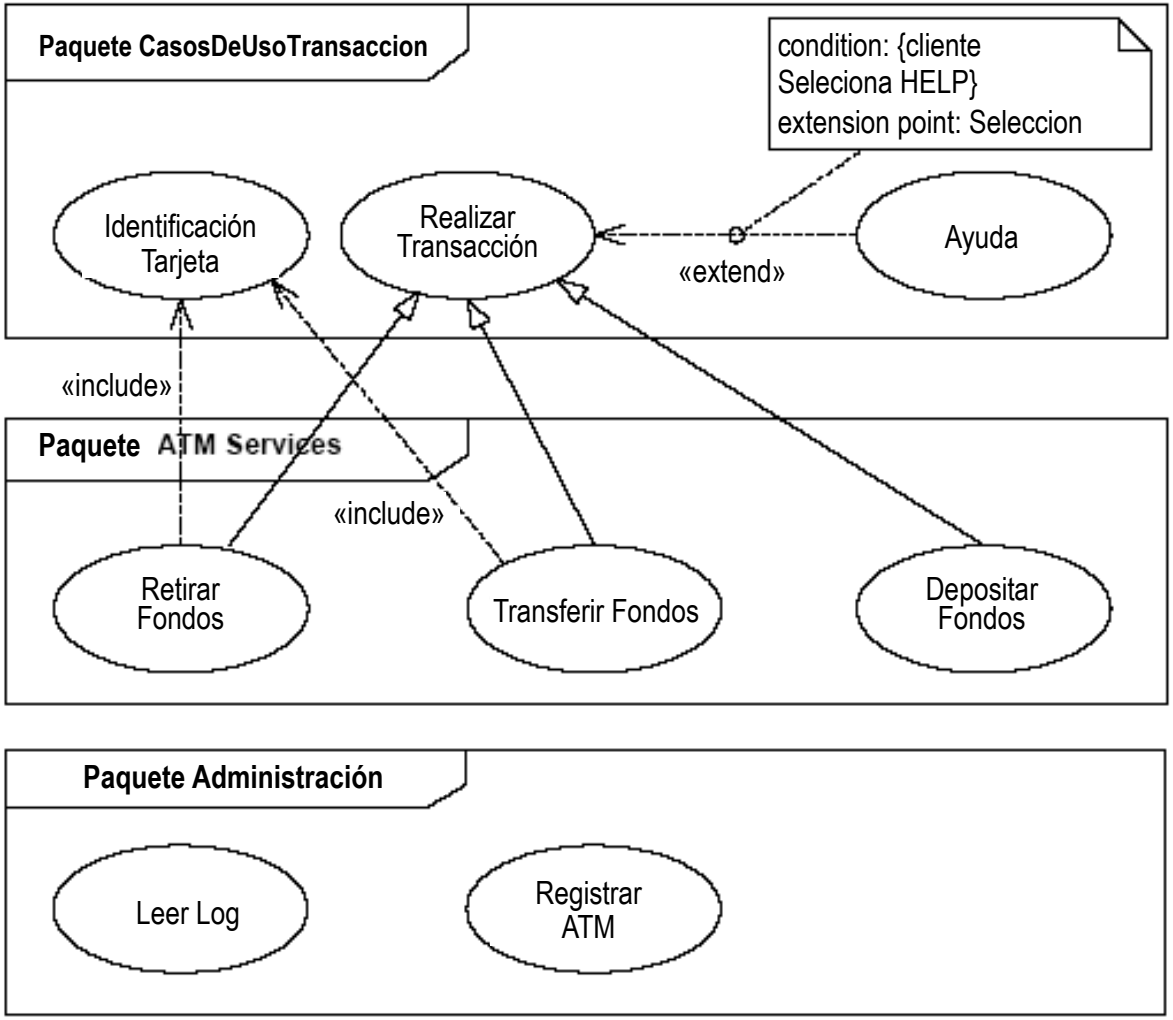
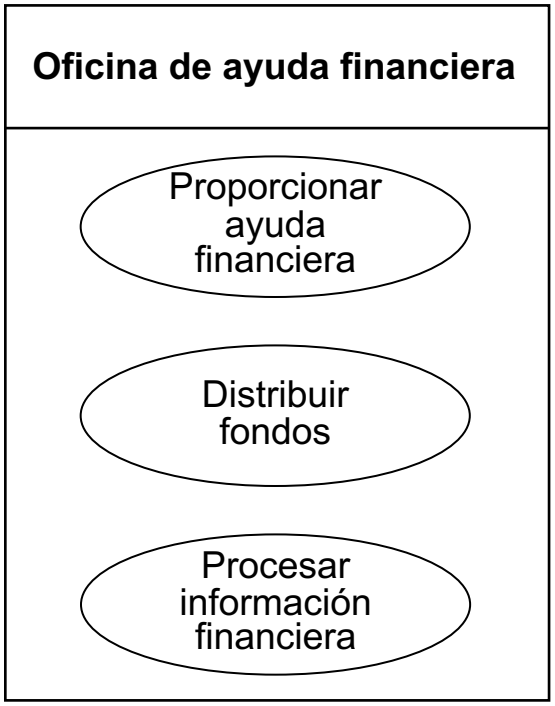


Diagrama de casos de uso del paquete Biblioteca

Organización de los casos de uso

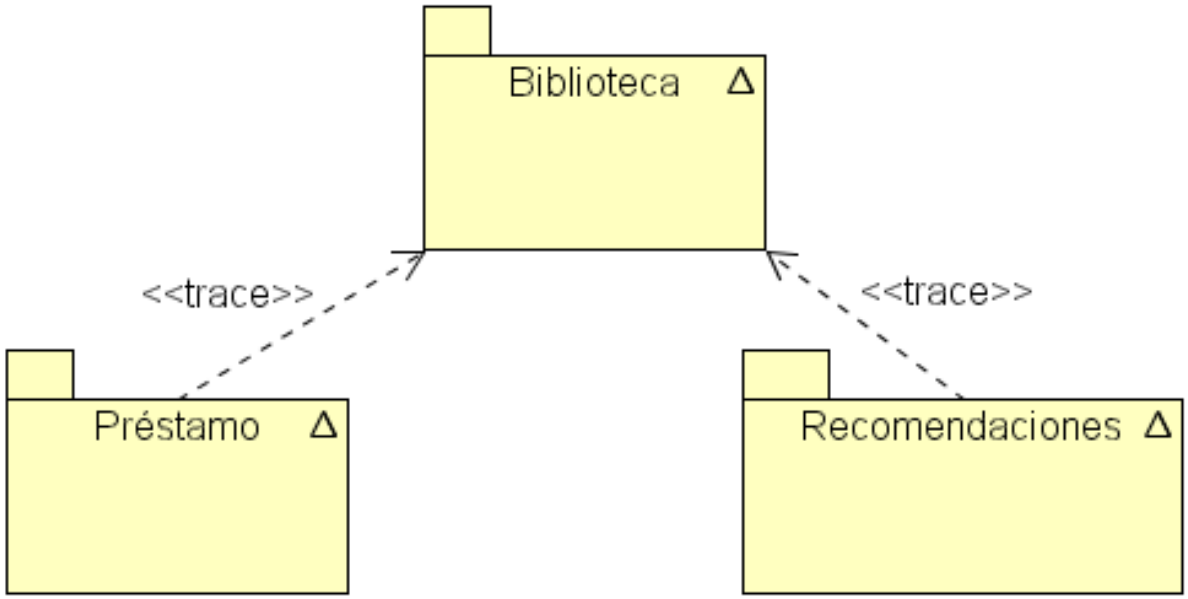


Relaciones entre casos de uso de diferentes paquetes



Casos de uso contenidos en un clasificador

Organización de los casos de uso



Paquetes de casos de uso en diferentes niveles de abstracción

Organización de los casos de uso

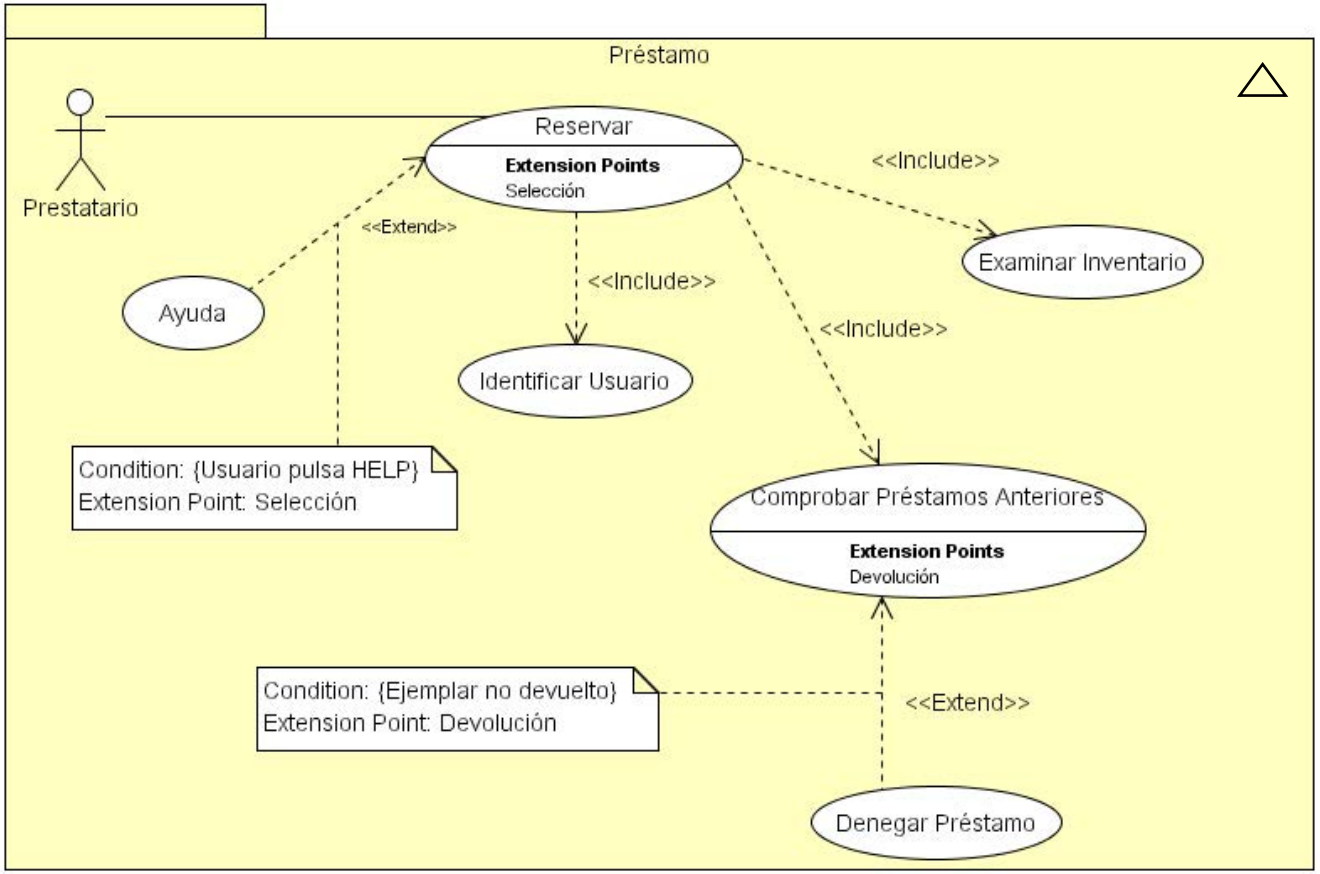


Diagrama de casos de uso del paquete Préstamo

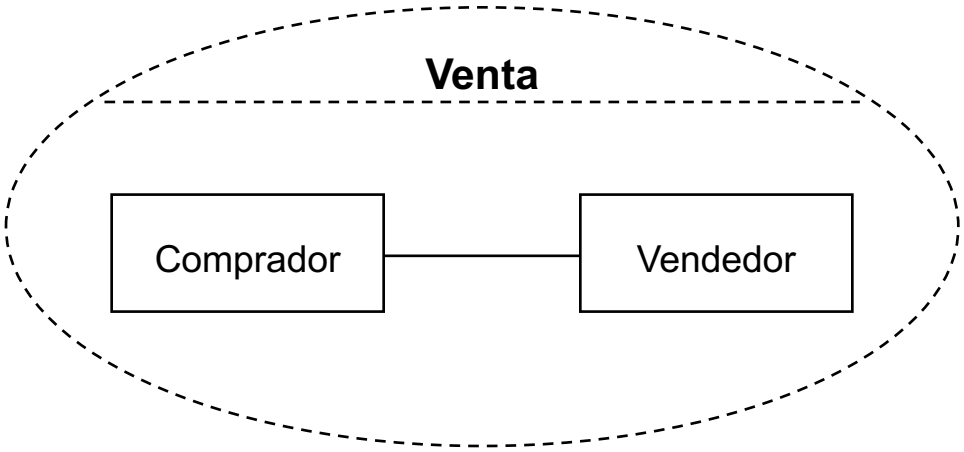
Realización de los casos de uso

- Las responsabilidades de realización de las acciones descritas en los casos de uso se asignan a objetos que colaboran e implementan la funcionalidad del caso de uso
- **Principios para la realización de los casos de uso (I)**
  - Una **colaboración** realiza un caso de uso: solución dependiente de la implementación
    - **Contexto** de la colaboración: relaciones entre clases y objetos
    - **Interacción** de la colaboración: interacciones entre ellos para alcanzar la funcionalidad deseada

    El símbolo de la colaboración es una elipse con la línea discontinua y con el nombre en su interior
- Para explicar una colaboración se requieren diagramas que muestren el contexto y la interacción entre los elementos que colaboran: diagramas de comunicación, de secuencia, de visión global de la interacción, de actividad y de máquina de estados

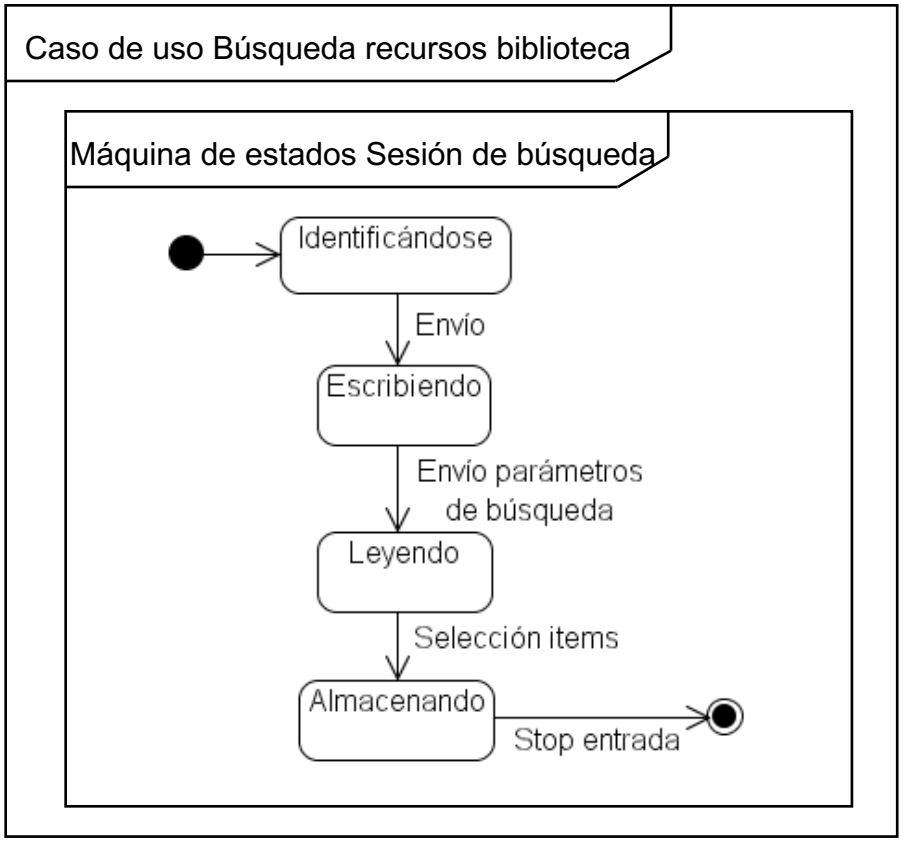
Realización de los casos de uso

- **Principios para la realización de los casos de uso (II)**
  - Un **escenario** es una instancia de un caso de uso
    - Un escenario es un camino de ejecución específico que representa una instanciación específica de un caso de uso
    - Un escenario visto como una ocurrencia de una colaboración incluye la interacción entre las partes dentro del sistema
  - Un caso de uso puede poseer diagramas que detallen su estructura interna: pueden enfatizar su estructura de tiempo de ejecución u otros elementos que surgen en la implementación del caso de uso (por ejemplo un diagrama de máquina de estados)



Colaboración que realiza un caso de uso

Realización de los casos de uso



Caso de uso con diagramas que detallan su estructura interna

Vista de interacción

Características

- **Interacción**: unidad de comportamiento que se centra en el intercambio de información observable entre elementos que pueden conectarse
  - La comunicación se realiza mediante mensajes
  - Viene dada por un par de conjuntos de trazas (secuencias de eventos): trazas válidas e inválidas. La unión de esos conjuntos no cubre necesariamente el universo entero de trazas
  - Las interacciones se pueden especializar añadiendo más trazas a la interacción original
- **Diagramas**
  - **Diagrama de secuencia**: hacen hincapié en la secuencia de intercambio de mensajes entre objetos
  - **Diagrama de comunicación (colaboración)**: se centran en las interacciones y enlaces entre objetos que colaboran
  - **Diagrama de visión global de la interacción**: variante del diagrama de actividad que muestra el flujo de control de la interacción a alto nivel
  - **Diagrama de tiempo**: diagrama de interacción que muestra sobre un eje de tiempo los cambios de estado o condición de una instancia o papel de clasificador

Características

- Los **diagramas de secuencia** muestran la interacción entre los objetos centrándose en la secuencia de mensajes que envían y reciben
- Tiene dos usos diferentes
  - Forma de **instancia**: describe un escenario específico, una posible interacción
  - Forma **genérica**: describe todas las posibles alternativas en un escenario. Puede incluir ramas, condiciones y bucles
- Se representan dentro de un marco con el nombre del diagrama precedido del prefijo **sd** dentro del símbolo que aparece en la esquina superior izquierda del marco
- Un diagrama de secuencia representa una interacción como un diagrama bidimensional
  - La dimensión vertical es el eje de tiempos
  - La dimensión horizontal muestra la **línea de vida** (***lifeline***) de los objetos implicados en la interacción:
    - Una línea de vida muestra una participación individual en la interacción. Representa la existencia de un objeto
    - **Notación**: rectángulo con una línea discontinua debajo. Una cruz al final indica la destrucción del objeto (*evento de destrucción*)

Características

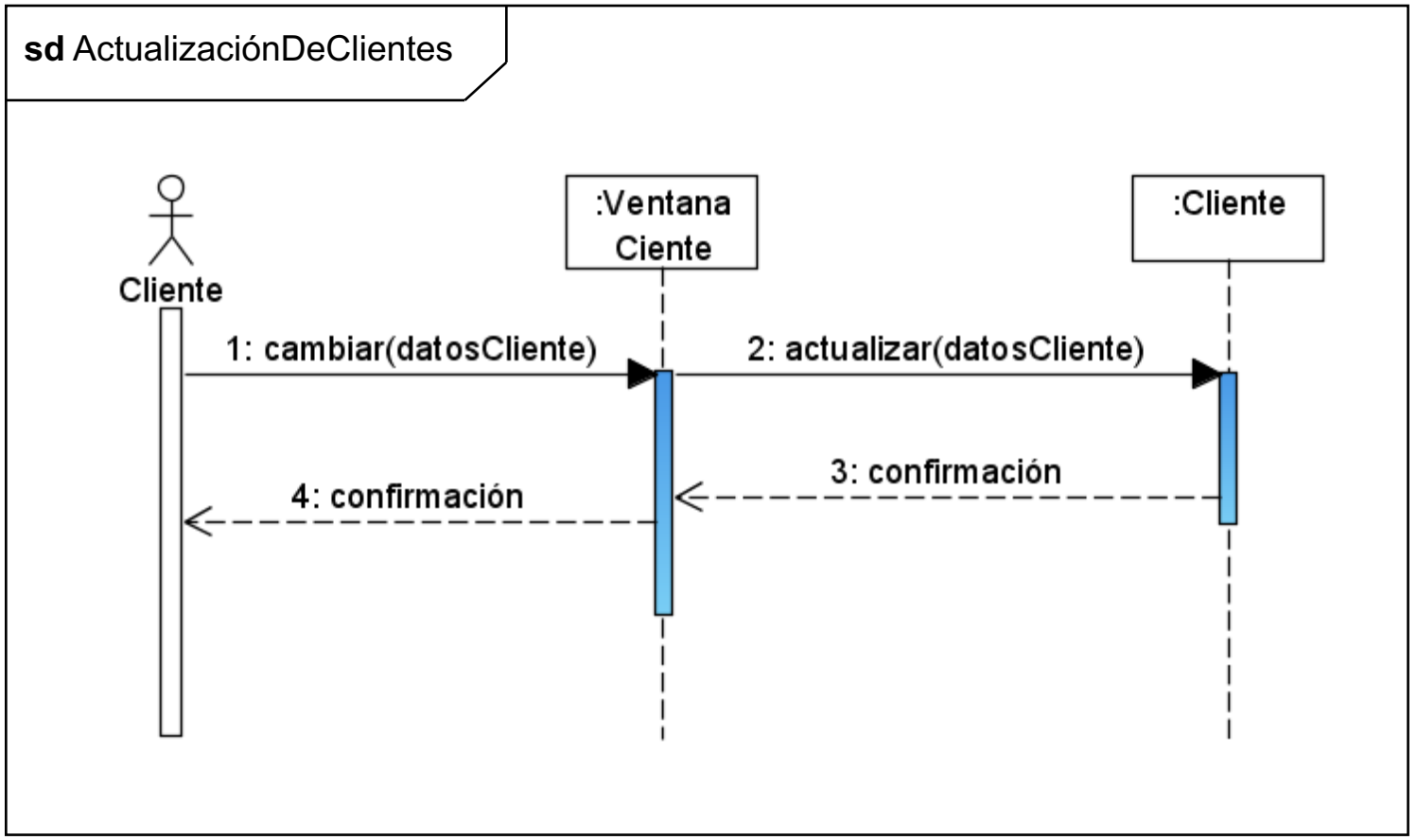
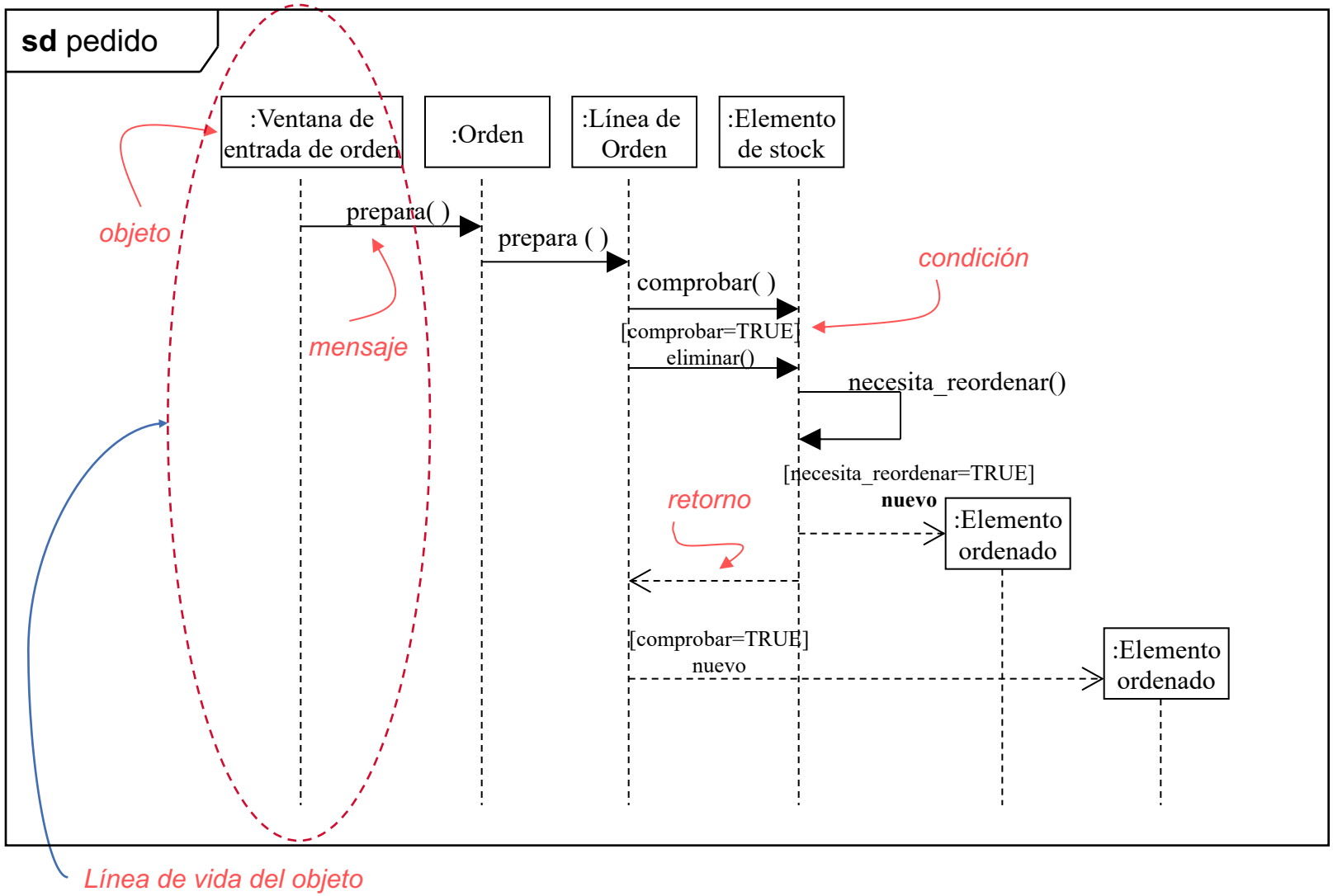


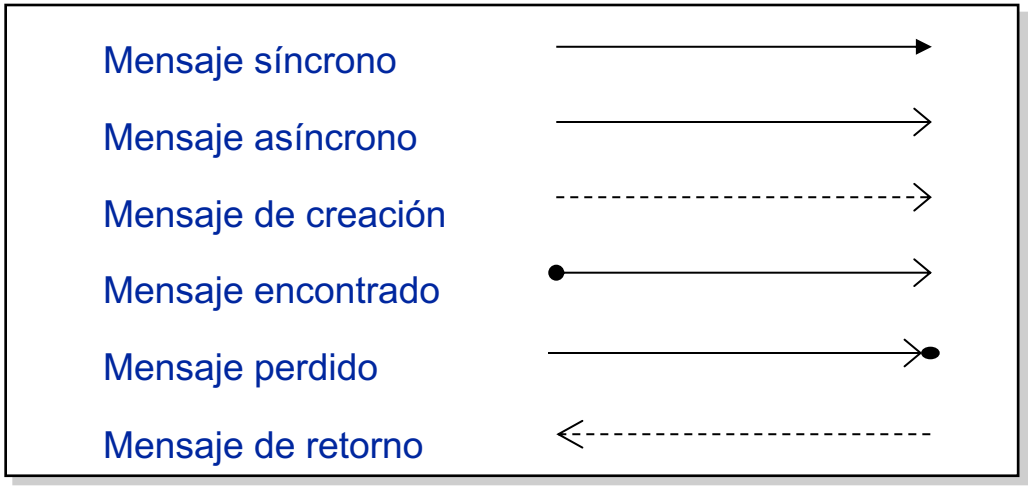
Diagrama de secuencia

Diagramas de secuencia



Diagramas de secuencia

- Un **mensaje** representa una comunicación entre objetos
  - Transporta información para la realización de una acción. Cuando un objeto recibe un mensaje realiza una actividad: ***ocurrencia de ejecución***
  - Los mensajes pueden ser señales, invocaciones a operaciones, llamadas a procedimientos remotos...
  - **Notación**:
    - Se muestran como flechas entre las líneas de vida de los objetos
    - Existen símbolos específicos para representar diferentes tipos de mensajes
    - Pueden tener una signatura: nombre, parámetros y valor de retorno



Notación utilizada para representar diferentes tipos de mensaje

Diagramas de secuencia

- **Ocurrencia de ejecución**

  Muestra el foco del control que ejecutan los objetos activados en algún momento
  - Un **objeto activado** está ejecutando su propio código o esperando el retorno de otro objeto al que ha enviado un mensaje
  - Su representación es opcional. El símbolo utilizado es un rectángulo estrecho sobre la línea de vida del objeto

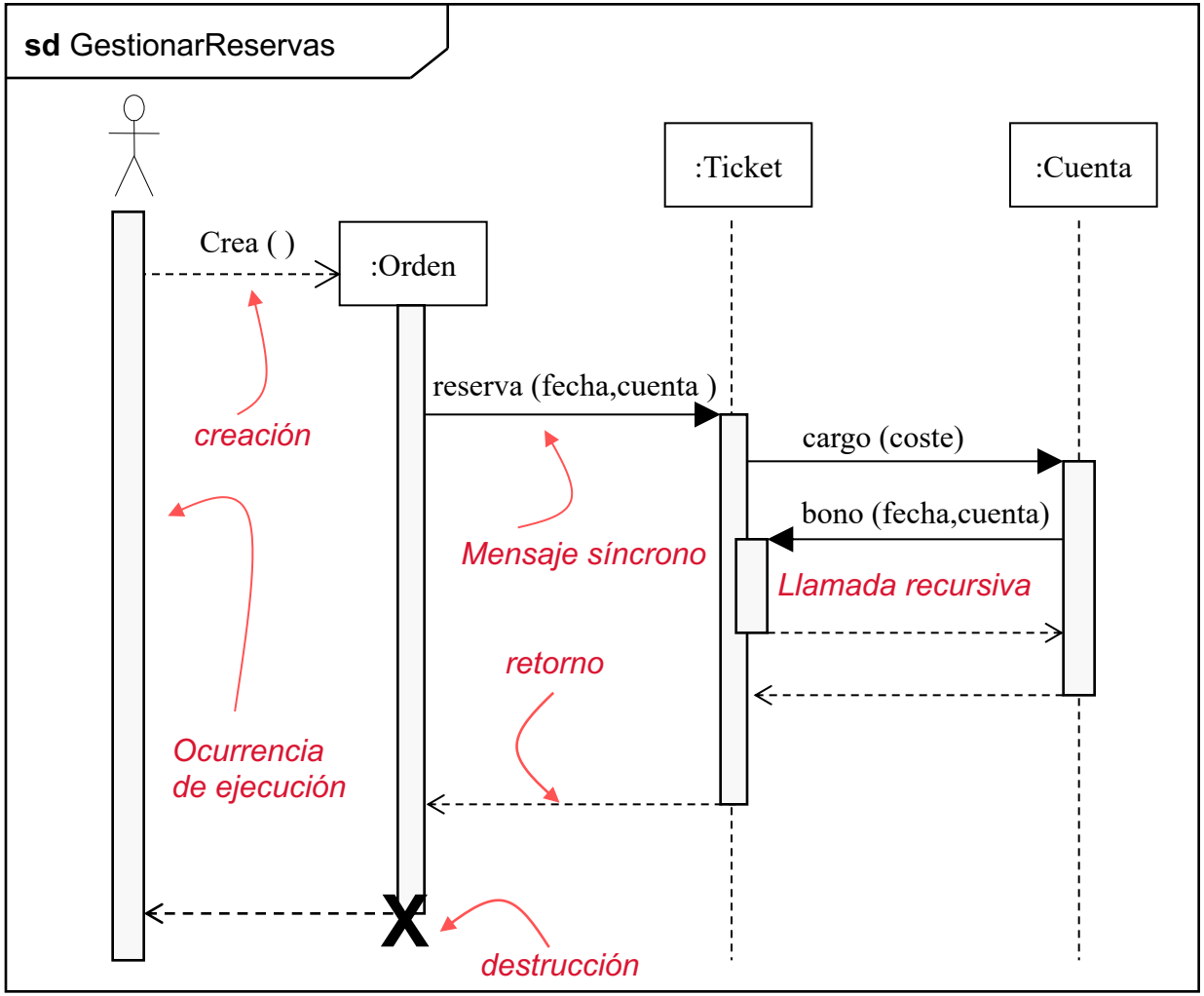


Diagrama de secuencia con representación de ocurrencias de ejecución

Diagramas de secuencia

- **Fragmentos combinados**
  - Encapsulan porciones del diagrama de secuencia
  - Tienen un operador de interacción que indica como se maneja el fragmento
    - **alt**: estructura alternativa
    - **opt**: comportamiento opcional
    - **loop:** bucle, comportamiento repetitivo. Ej. **loop** (1,5)
    - **par**: comportamientos paralelos
    - **critical**: región crítica (no se pueden intercalar eventos)
    - …
  - **Notación**: el conjunto de trazas del fragmento se colocan dentro de un marco (*frame*). Dentro del símbolo de la parte superior izquierda se pone la palabra clave correspondiente al operador

Diagramas de secuencia

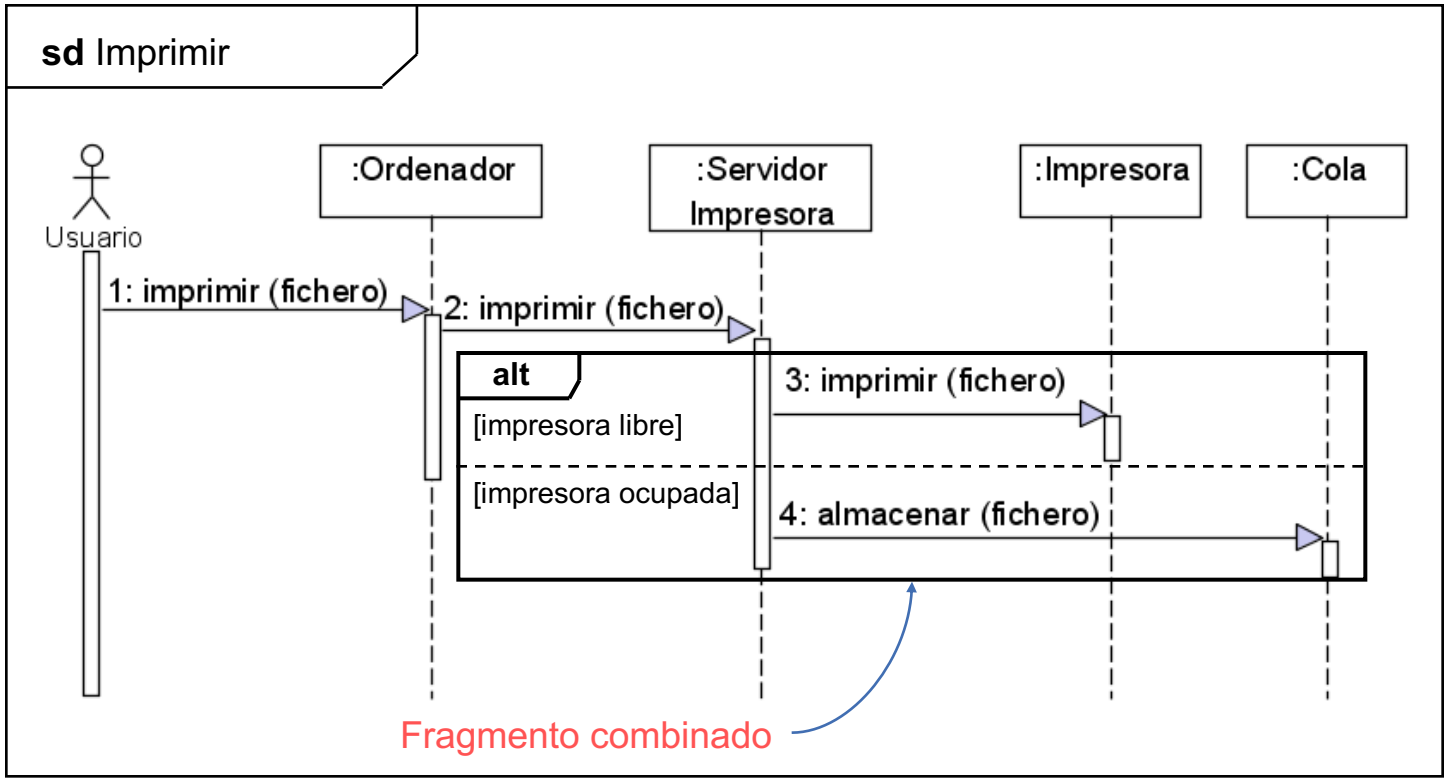


Diagrama de secuencia con un fragmento combinado

Diagramas de secuencia

- **Ocurrencia o uso de interacción**
  - Permite a múltiples interacciones referirse a una interacción que representa una porción común de sus especificaciones
  - **Notación**:
    - Se representa como un fragmento combinado con el operador **ref**
    - Dentro del marco se coloca el nombre de la interacción con la siguiente sintaxis:

nombre [(argumentos)] [:valorRetorno]

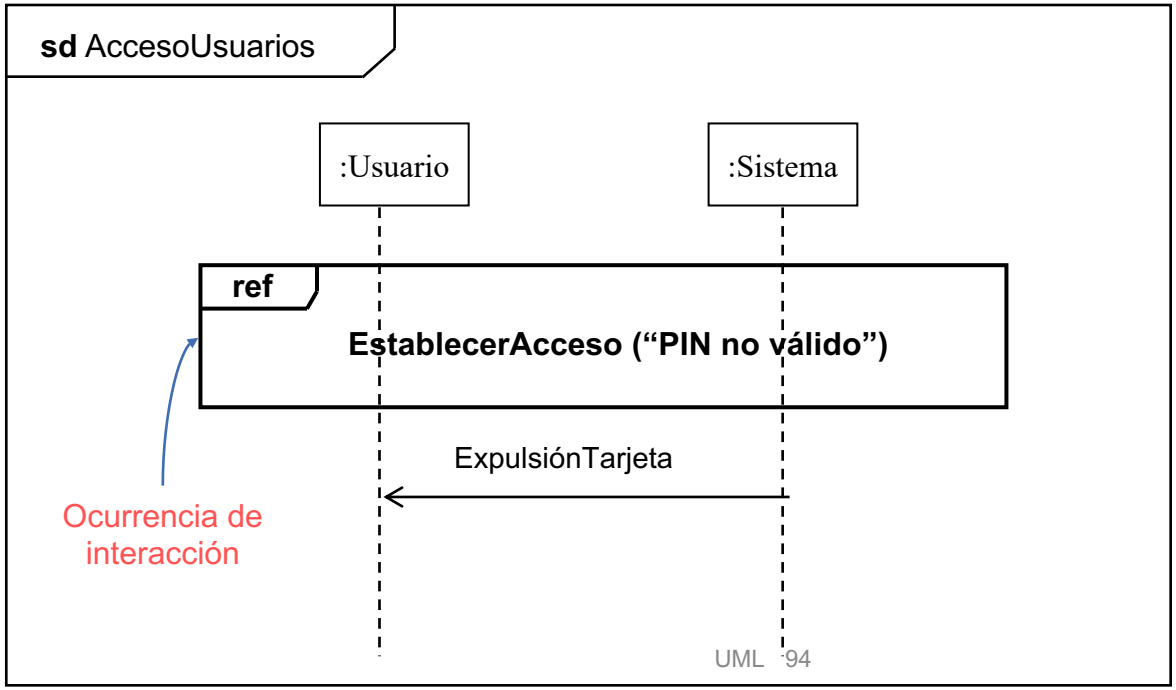


Diagrama de secuencia con una ocurrencia de interacción

Diagramas de comunicación

- Los **diagramas de comunicación** se centran en las interacciones y en los enlaces entre los objetos que colaboran, siendo secundario el orden de envío y recepción de mensajes
- Se representan dentro de un marco con el nombre del diagrama precedido del prefijo **sd** dentro del símbolo que aparece en la esquina superior izquierda del marco
  - Sólo se representa el rectángulo de la línea de vida
  - Los mensajes se colocan cerca de los enlaces. Se representan con una flecha y una etiqueta que contiene el nombre del mensaje y otra información adicional

Diagramas de comunicación

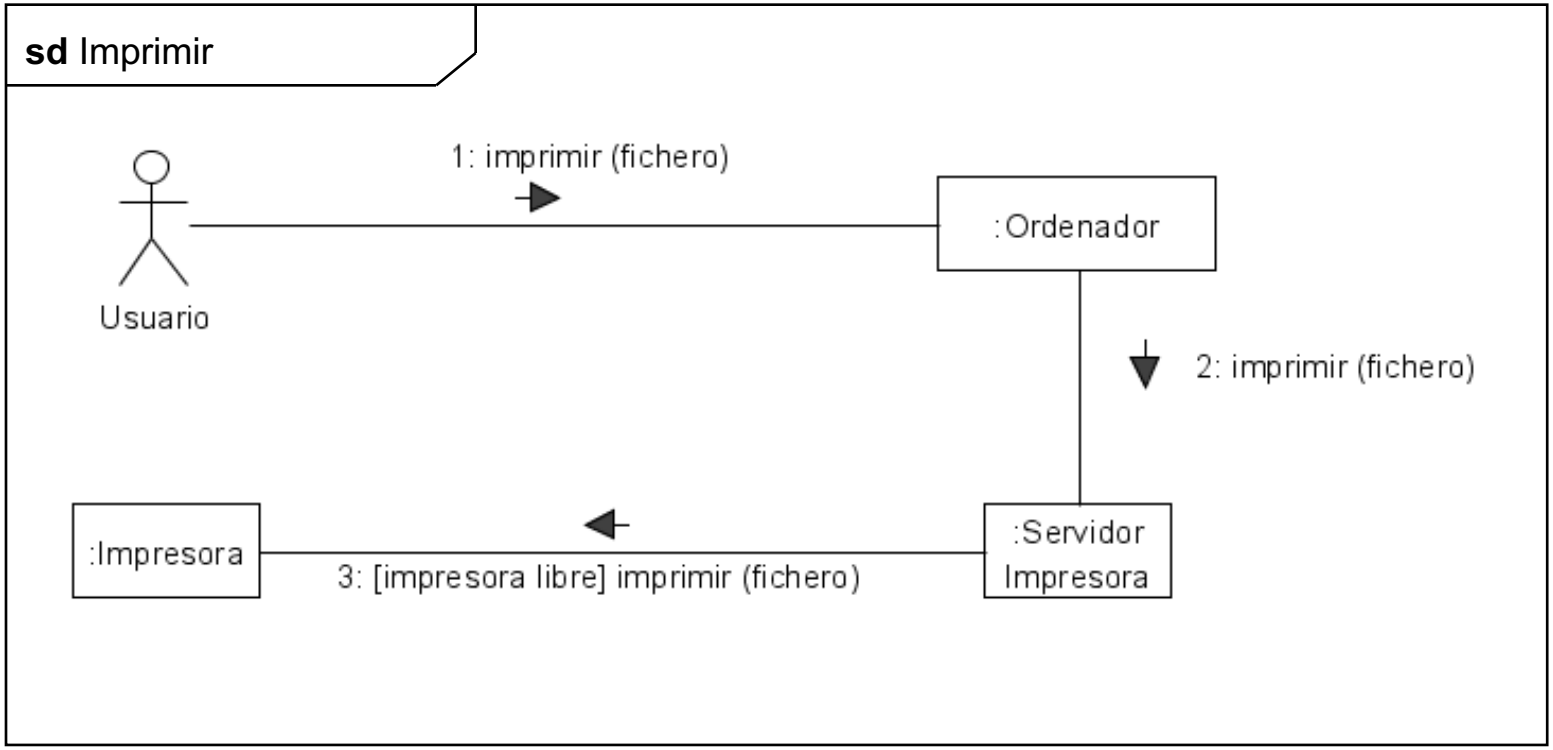


Diagrama de comunicación

Diagramas de comunicación

- Sintaxis de la etiqueta del mensaje

[predecesor] [condición-guarda] [expresión-secuencia] [valor-retorno :=]
nombre-mensaje (argumentos)

  - **Predecesor**: lista de números de secuencia separados por comas y seguidos por '/'
    - El flujo de mensajes no queda habilitado mientras no se hayan producido todos los flujos de mensaje cuyos números de secuencia estén habilitados
  - **Condición de guarda**: expresión booleana encerrada entre corchetes
  - La **expresión de secuencia** es una lista de términos de secuencia separada por puntos y seguida de ':'
    - Cada término representa un nivel de anidamiento procedural dentro de la interacción global
    - Si es concurrente todo el control, entonces no se produce anidamiento
    - Todos los **términos de secuencia** poseen la sintaxis etiqueta [recurrencia]
      La **etiqueta** es un entero o un nombre
        - El **nombre** representa un hilo de control concurrente (los mensajes que difieren en el nombre final son concurrentes en ese nivel de anidamiento)

      La **recurrencia** representa la ejecución condicional o iterativa
        - * '['cláusula de iteración']'
        - '['cláusula de condición']'

Diagramas de comunicación

- Ejemplos de mensajes de control
  - 2: visualizar (x, y) — Mensaje simple
  - 1.3.1: p:=buscar (espec) — Llamada anidada que proporciona un valor
  - [x<0] 4: invertir (x, color) — Mensaje condicional
  - 3.1 *: update () — Iteración
  - 1.1a, 1.1b/1.2: copiar (x,y) — Sincronización con otros hilos

Diagramas de comunicación

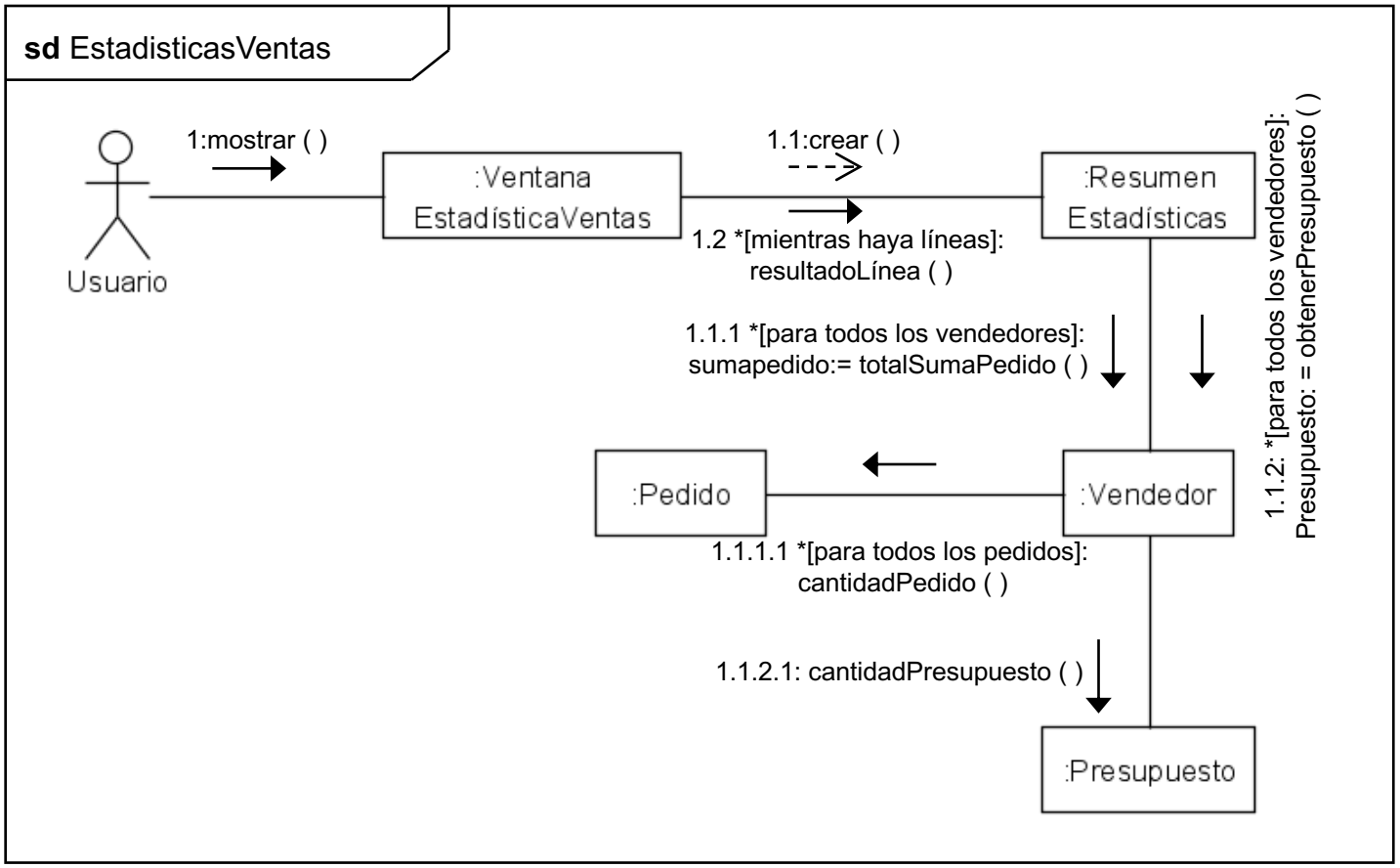


Diagrama de comunicación con mensajes anidados

Diagramas de visión global de la interacción

- Los **diagramas de visión global de la interacción** (*Interaction Overview Diagrams*) son una variante de los diagramas de actividad en los que los nodos son interacciones u ocurrencias de interacciones
- Características de los diagramas de visión global de la interacción:
  - Para representarlos se utiliza un marco igual que el de cualquier diagrama de interacción pero el texto de cabecera puede incluir una lista de las líneas de vida que contiene precedida de la palabra reservada **lifelines**
  - Las interacciones u ocurrencias de interacción que contienen sustituyen a los nodos objeto y se consideran formas especiales de invocación a actividades
  - Los **fragmentos combinados alternativos** se representan con un nodo de decisión y su correspondiente nodo fusión
  - Los **fragmentos combinados paralelos** se representan con un nodo *fork* y su correspondientes nodo *join*
  - Los **bucles de fragmentos combinados** se representan por ciclos simples
  - Se pueden anidar estructuras

Diagramas de visión global de la interacción

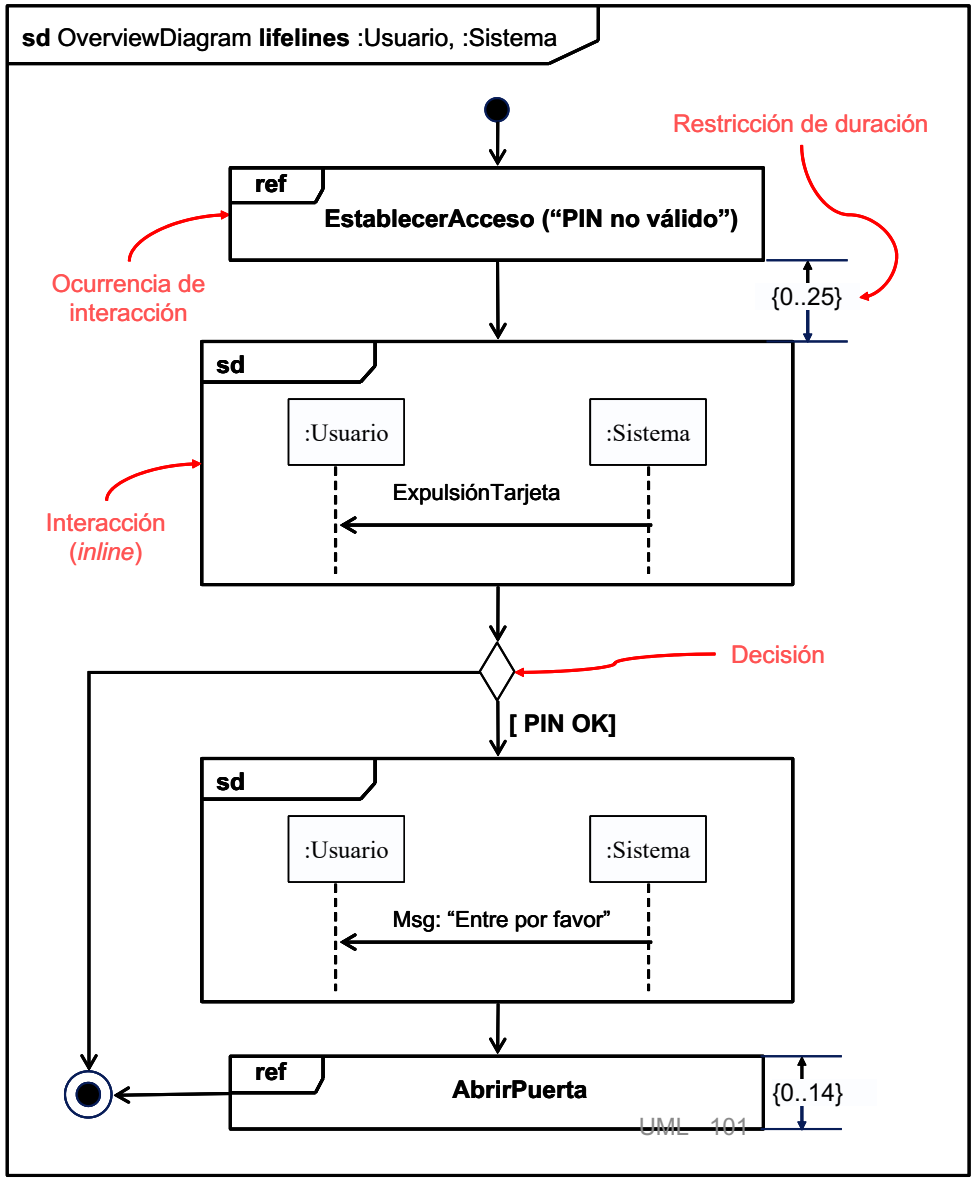


Diagrama de visión global de la interacción

Diagramas de tiempo

- Los **diagramas de tiempo** (*timing diagrams*) proporcionan una forma de mostrar los objetos activos y sus cambios de estado durante sus interacciones con otros objetos activos y con otros recursos del sistema
- Para representarlos se utiliza el marco de los diagramas de interacción
  - El **eje X** muestra las **unidades de tiempo** y el **eje Y** muestra los **objetos y sus estados**
  - Se puede representar el cambio en el estado de un objeto a lo largo del tiempo en respuesta a eventos o estímulos
  - Permite la representación de diferentes tipos de mensajes. Los mensajes se pueden dividir mediante etiquetas para mejorar la legibilidad de los diagramas

Diagramas de tiempo

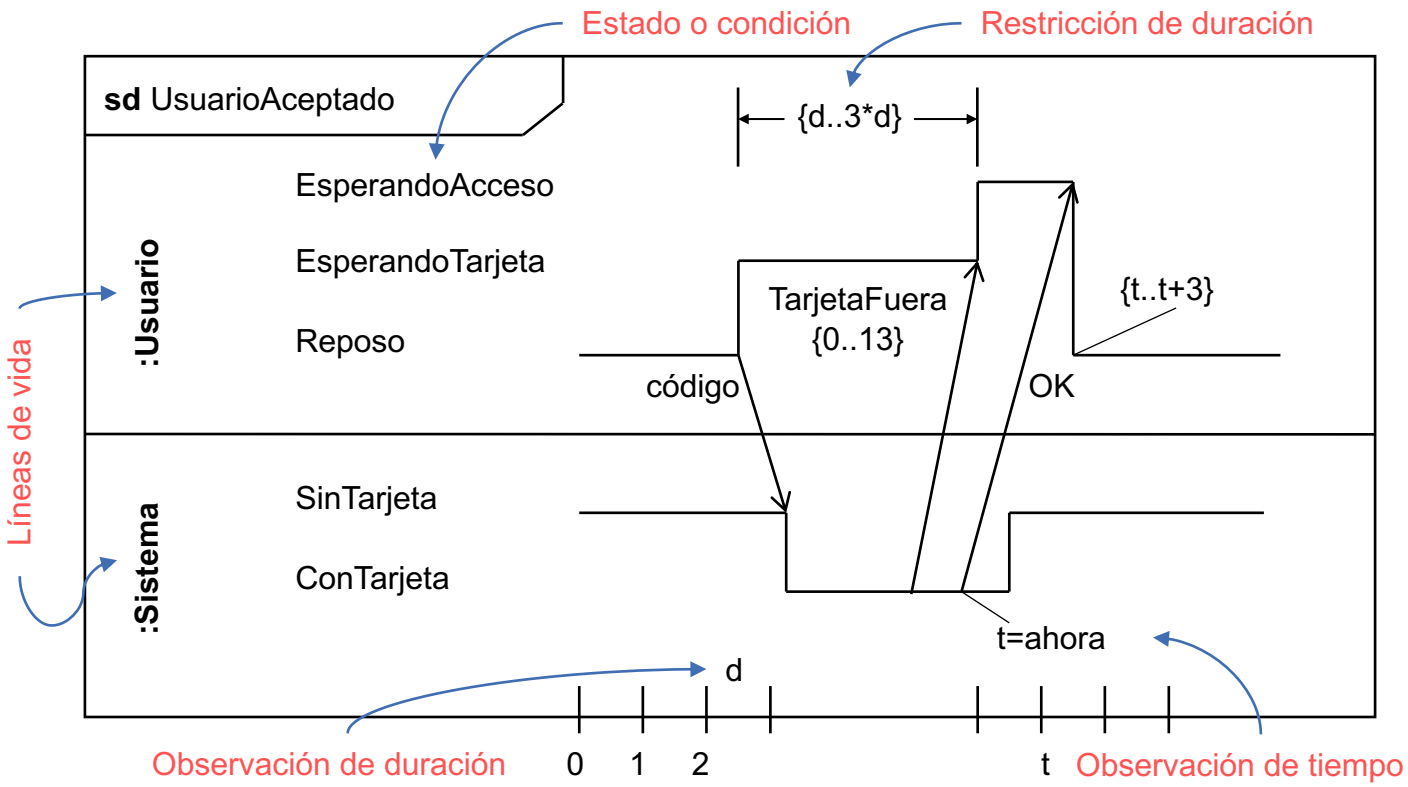
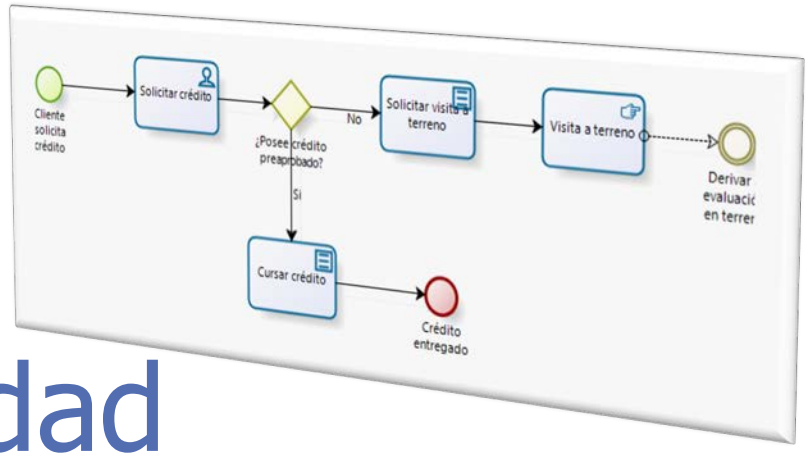


Diagrama de tiempo

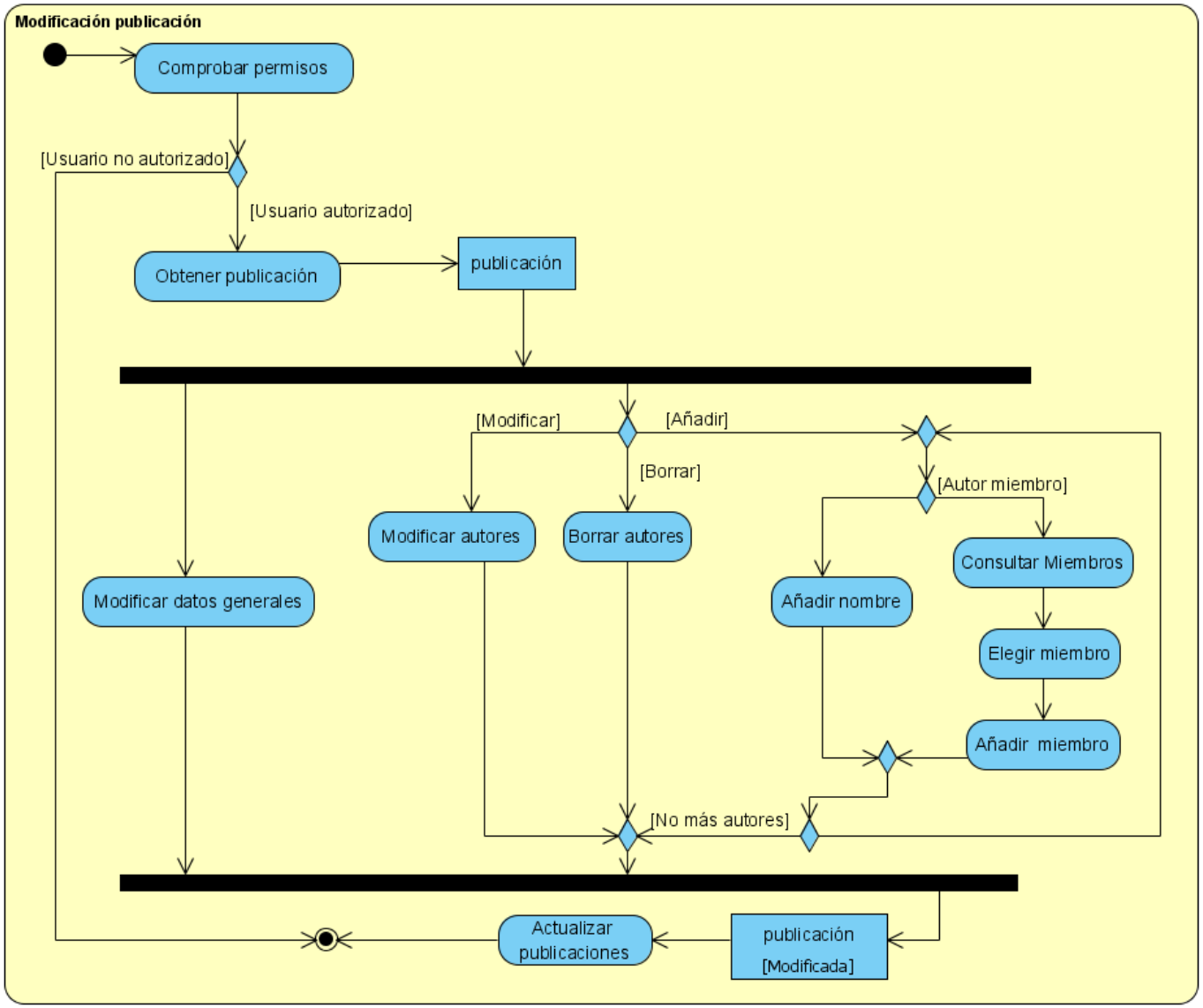


Vista de actividad

Características

- Representación del comportamiento dinámico del sistema mediante actividades
- Un **diagrama de actividad** representa el comportamiento mediante un modelo de flujo de datos y flujo de control
  - **Actividad**: especificación de un comportamiento parametrizado que se expresa como un flujo de ejecución por medio de una secuencia de unidades subordinadas
  - **Acción**: especificación de una unidad fundamental de comportamiento que representa una transformación o procesamiento
    - Las acciones están contenidas en actividades que le proporcionan su contexto
    - Los diagramas de actividad capturan las acciones y sus resultados
- Los pasos de ejecución dentro de una actividad pueden ser concurrentes o secuenciales
- Una actividad involucra constructores de sincronización y de bifurcación

Características



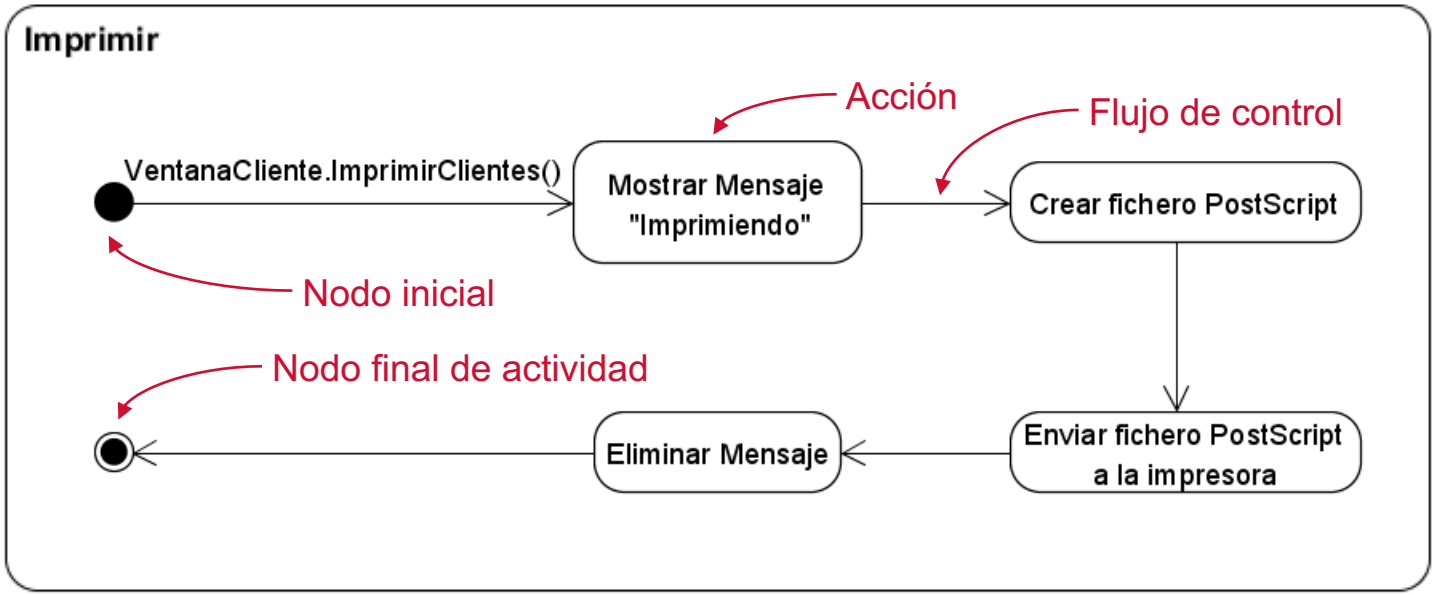
Características

- Usos de los diagramas de actividad
  - Capturar las acciones que se realizan cuando se ejecuta una operación
  - Capturar el trabajo interno de un objeto
  - Mostrar como se pueden realizar un conjunto de acciones relacionadas y como afectan a los objetos
  - Mostrar como se puede realizar una instancia de un caso de uso en términos de acciones y cambios de estado de los objetos
  - Mostrar como trabaja un negocio en términos de trabajadores (actores), flujos de trabajo, organización y objetos (factores intelectuales y físicos usados en un negocio)

Acciones

- En el diagrama de actividad una acción es un nodo de actividad ejecutable que representa una unidad fundamental de funcionalidad ejecutable en una actividad
- Representan pasos simples (no pueden descomponerse) dentro de una actividad
  - Una acción puede tener arcos de entrada y salida que especifican flujo de control y datos desde y hacia otros nodos
  - Una acción no puede comenzar su ejecución hasta que se cumplan todas las condiciones de entrada
  - La terminación de la ejecución de una acción puede permitir la ejecución de nodos que tienen como arcos de entrada arcos de salida de la acción
  - Pueden tener pre y post-condiciones locales: se especifican dentro de una nota unida a la acción
- Notación: rectángulo con las esquinas redondeadas. Dentro del símbolo puede aparecer el nombre de la acción o cualquier otra descripción

Acciones



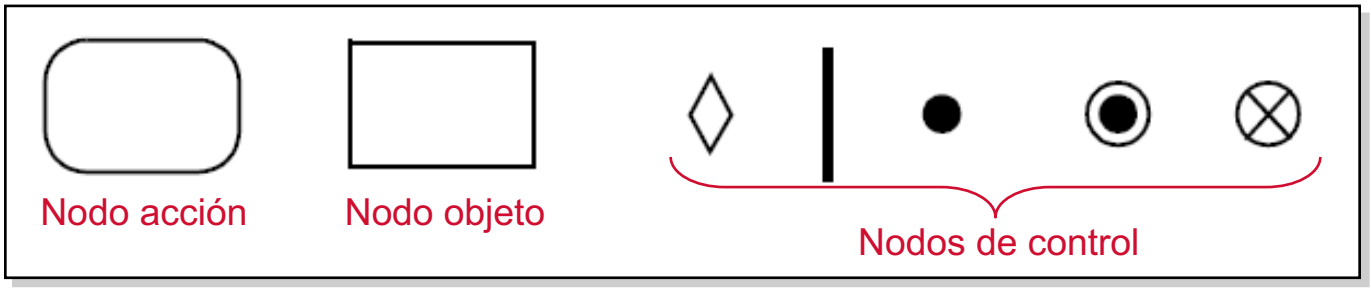
Representación de acciones de un diagrama de actividad

Actividades

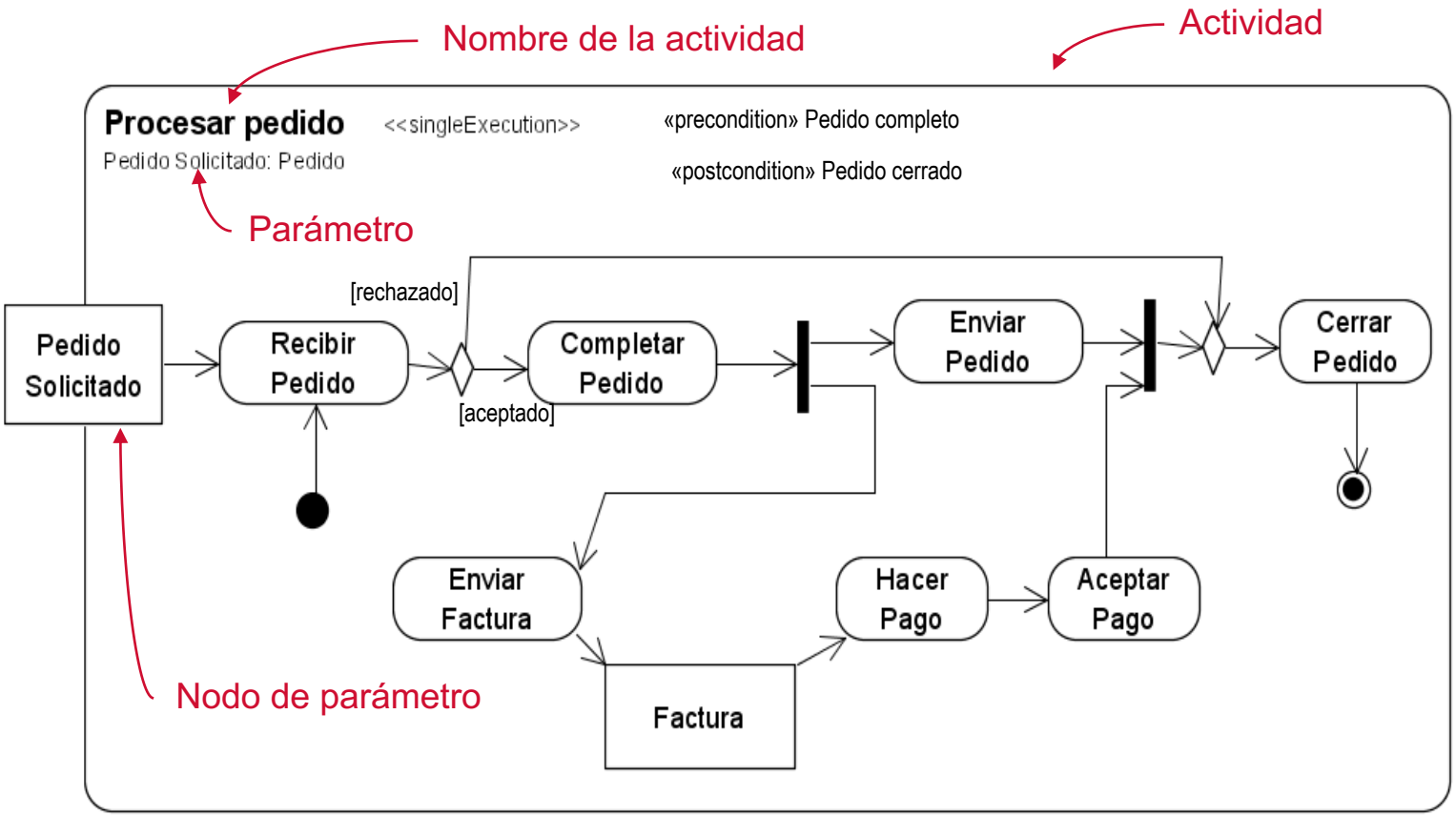
- Una actividad representa un comportamiento compuesto de elementos individuales que son las acciones
- Coordina los comportamientos subordinados (acciones) mediante modelos de flujo de datos y de control. Dichos comportamientos se inician:
  - por la finalización de otros comportamientos del modelo
  - por la disponibilidad de objetos y datos
  - por eventos externos al flujo
- El flujo de ejecución se modela con **nodos de actividad** conectados por **arcos de actividad**. Los nodos pueden ser de tres tipos:
  - **Nodos ejecutables (acciones)** de varios tipos: funciones primitivas, invocación de comportamiento, acciones de comunicación (señales) y manipulación de objetos
  - **Nodos objeto**: nodos que proporcionan y aceptan objetos y datos
  - **Nodos de control**: gestionan el flujo (sincronización, decisión y control de concurrencia)
- Las actividades pueden ser invocadas por las acciones
- Un clasificador puede ser el ***contexto*** de una actividad
- Puede haber actividades parametrizadas

Actividades

- **Notación**: combinación de las notaciones de nodos y arcos que contiene dentro de un borde con el nombre de la actividad en la parte superior izquierda
  - Los **nodos de parámetro** son rectángulos colocados sobre el borde. El nombre y tipo de parámetro se coloca bajo el nombre de la actividad. Sintaxis:
    nombre parámetro: tipo
  - Pueden contener en la parte superior precondiciones y postcondiciones. Sintaxis:
    «precondition» restricción
    «postcondition» restricción
- Notación alternativa: símbolo de clase con el estereotipo «activity»



Actividades



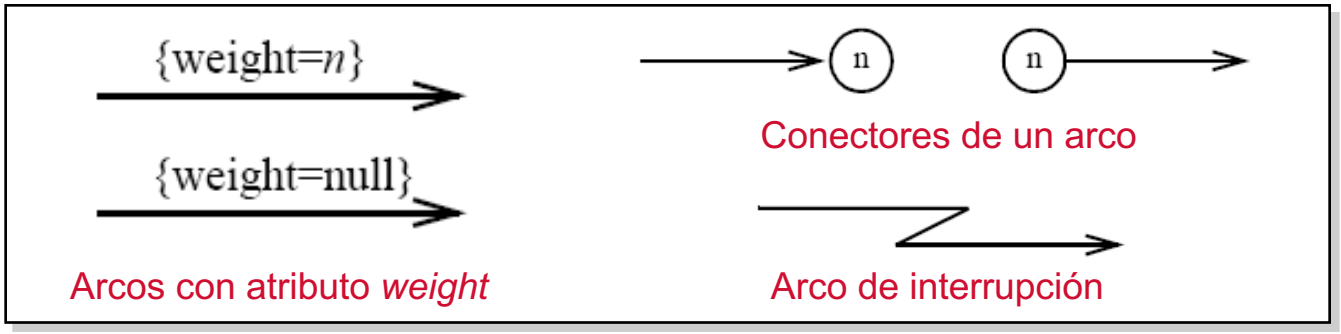
Representación de una actividad y sus acciones

Diagramas de actividad

- **Tokens**
  - Vehículos para mover información y eventos a través del sistema. Pueden transportar objetos o valores
  - Los elementos de los diagramas de actividad proporcionan reglas para manejar los tokens
  - Las terminación de una acción proporciona tokens y el comienzo de la ejecución de una acción consume tokens
- **Arcos de actividad (I)**
  - Conexiones dirigidas entre nodos de actividad a través de las cuales fluyen los tokens
  - Pueden tener condiciones de guarda para permitir o no el paso de los tokens
  - Otras reglas de paso de tokens dependen del tipo de arco y de las características de los nodos fuente y destino

Diagramas de actividad

- **Arcos de actividad (II)**
  - **Notación**: flecha con un nombre (opcional) cerca. El nombre no tiene que ser único dentro de la actividad
    - Se pueden representar usando un conector
    - Puede representarse el valor del atributo *weight* entre llaves para indicar el número mínimo de tokens que deben atravesar el arco a la vez
    - Existe una notación especial para arcos que representan interrupciones



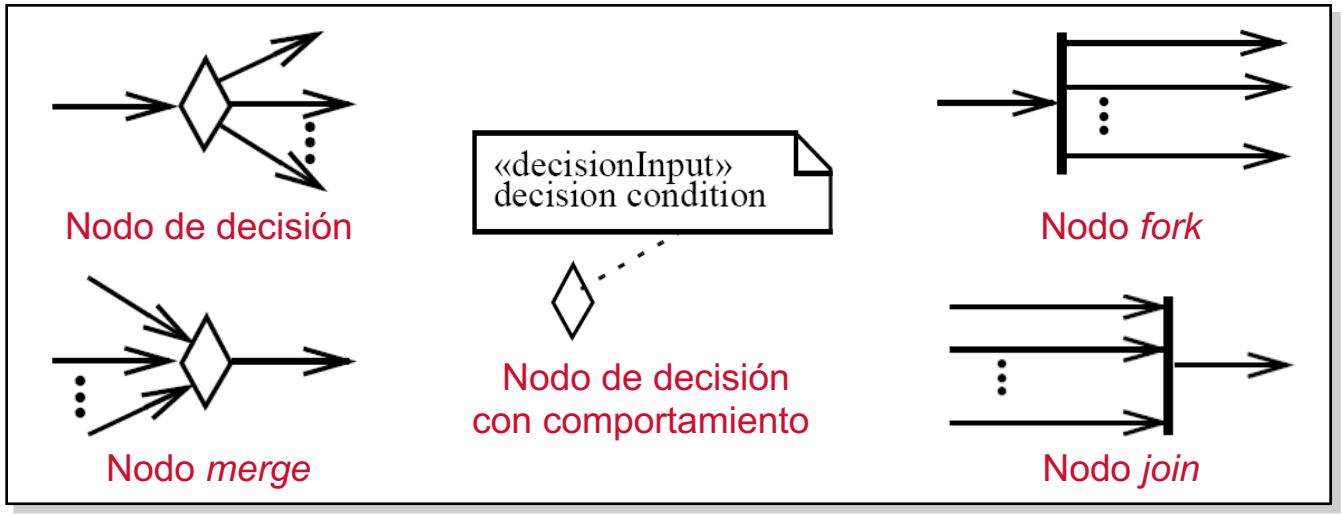
Notación de distintos arcos de actividad

Diagramas de actividad

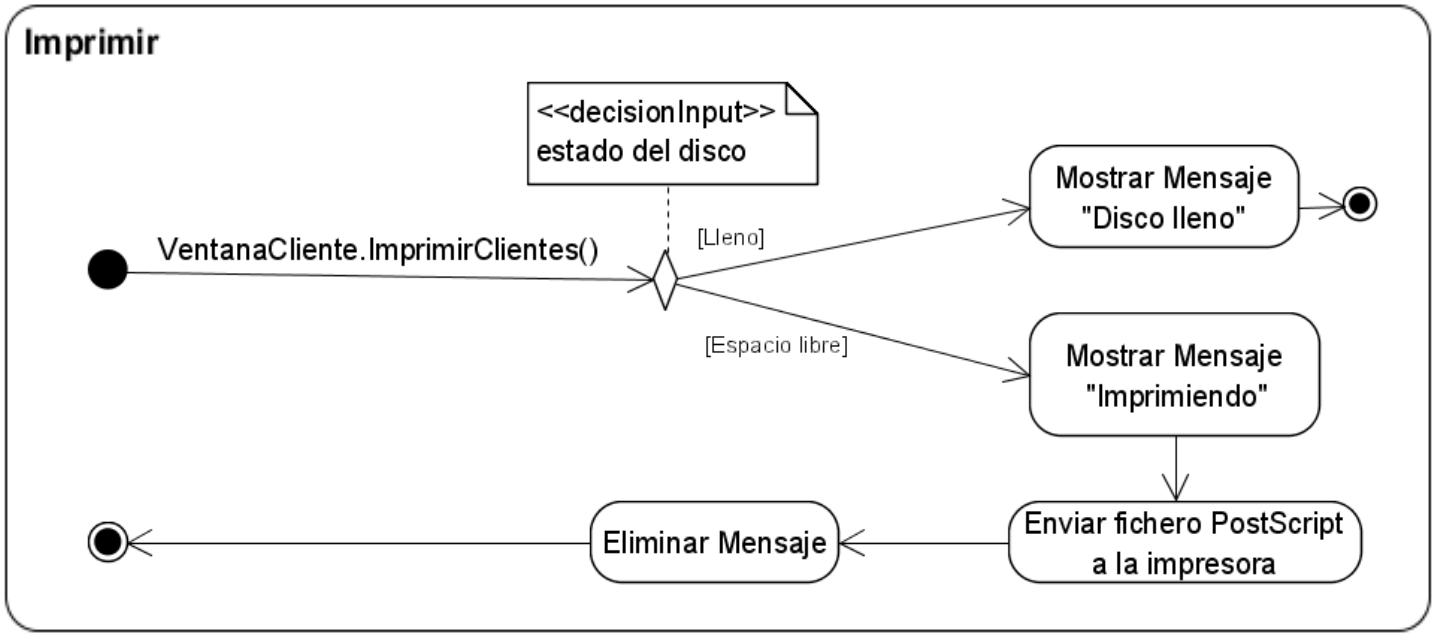
- **Nodo de decisión**
  - Nodo con un arco de entrada y varios arcos de salida
  - Cada token que llega a un nodo de decisión puede atravesar sólo un arco de salida
  - Las condiciones de guarda de los arcos de salida determinan qué arcos deben ser atravesados
- **Nodo fusión (*merge*)**
  - Nodo de control al que llegan varios flujos alternativos
- **Nodo horca *(fork)***
  - Nodo de control que divide un flujo en múltiples flujos concurrentes
  - Para cada token que llegan al nodo se transmiten copias del mismo a todos los arcos que salen del nodo
- **Nodo unión (*join*)**
  - Nodo que tiene como arcos de entrada múltiples flujos concurrentes y un solo arco de salida
  - Puede tener una condición que indique en qué circunstancias debe emitir un token (por defecto "and")

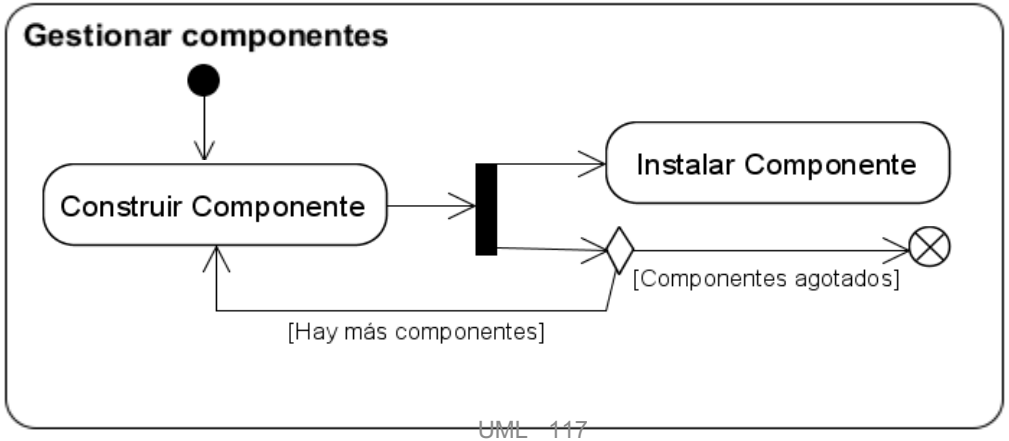
Diagramas de actividad

- **Nodo inicial**
  - Nodo de control en el que comienza el flujo cuando se invoca una actividad
- **Nodo final**
  - De **actividad**: nodo que detiene todos los flujos de una actividad
  - De **flujo**: nodo que detiene un flujo destruyendo todos los tokens que llegan a él



Diagramas de actividad



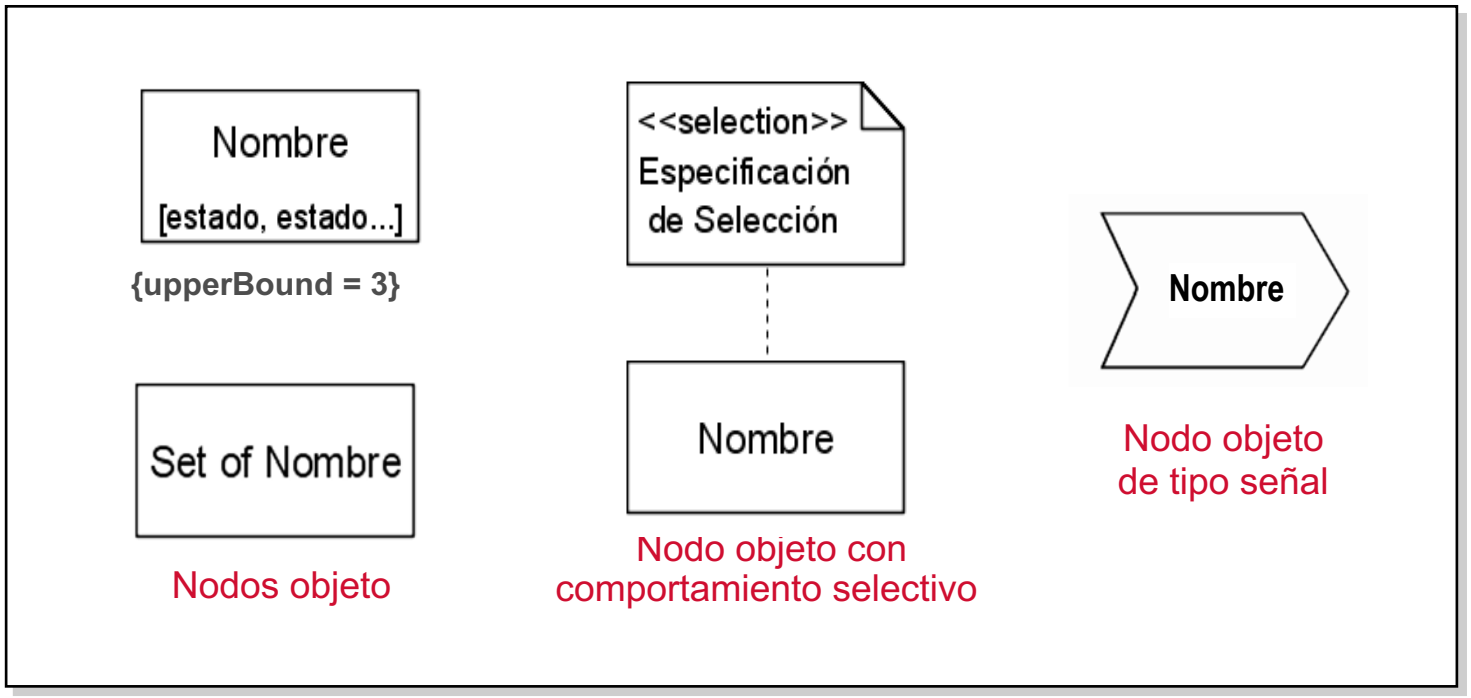


Diagramas de actividad

- **Nodos objeto (I)**
  - Nodos que representan una instancia de un clasificador particular, posiblemente en un determinado estado, disponible en un punto determinado de la actividad
  - Sólo pueden contener valores en tiempo de ejecución acordes con el tipo de nodo objeto en el estado o estados especificados
  - En un nodo objeto pueden residir a la vez múltiples tokens con el mismo valor, incluyendo tipos de datos, pero no puede contener más tokens que el valor del atributo *upperBound*
  - **Notación**: se representan como rectángulos con el nombre en su interior
    - El nombre indica el tipo de nodo. Si las instancias son conjuntos el nombre estará precedido por "set of"
    - Los nodos objeto de tipo señal tienen un símbolo específico
    - El estado se representa debajo del nombre entre corchetes
    - Se puede especificar el atributo *upperBound* entre llaves
    - El comportamiento selectivo se especifica con la palabra clave «selection» dentro del símbolo de una nota

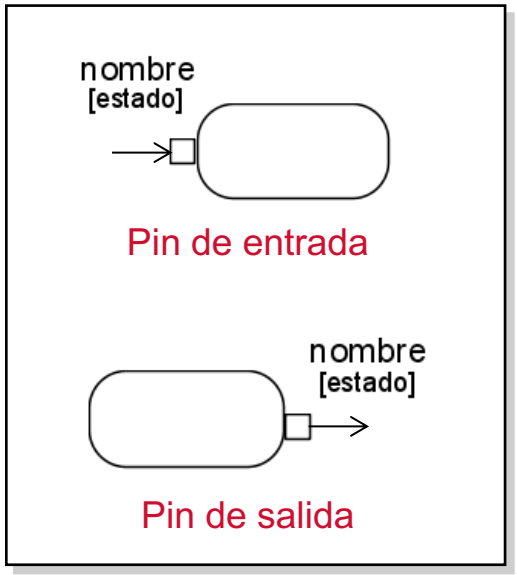
Diagramas de actividad

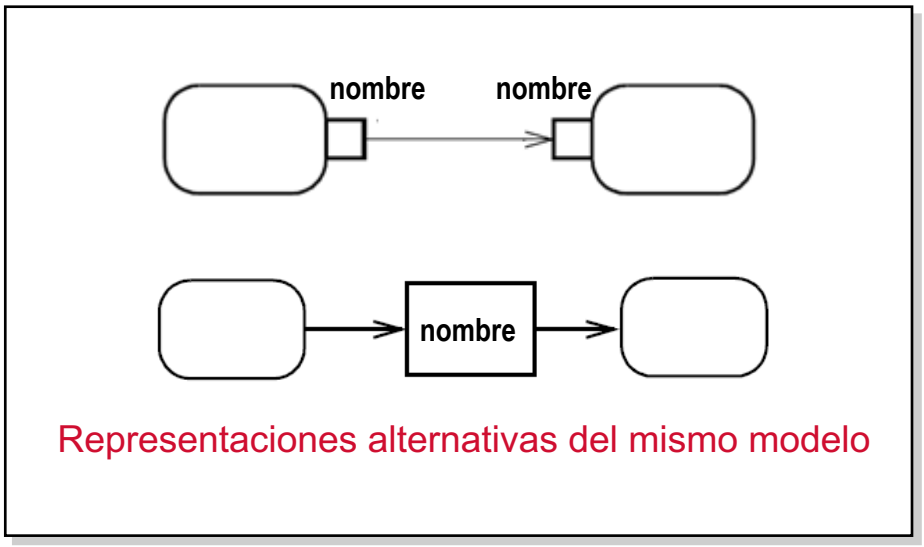
- **Nodos objeto (II)**



Diagramas de actividad

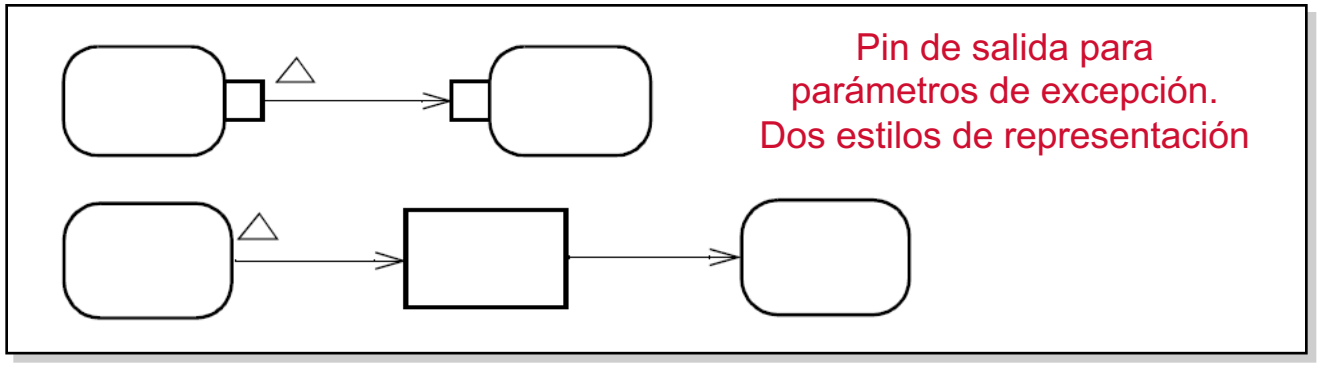
- **Pines (I)**
  - Nodos objeto que representan una entrada o una salida a una acción
  - Notación:
    - Se pueden representar como pequeños rectángulos unidos al símbolo de la acción o de forma independiente conectados por flujos a las acciones
    - Pueden llevar asociadas las restricciones {stream} o {nonStream}
    - Los pines de parámetros de excepción se representan colocando un pequeño triángulo cerca del origen del arco que sale del pin

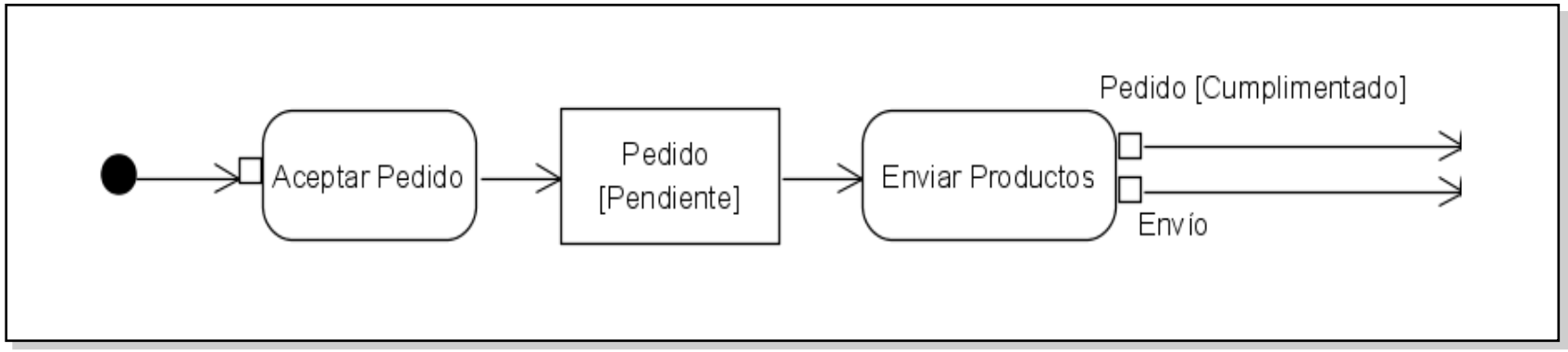




Diagramas de actividad

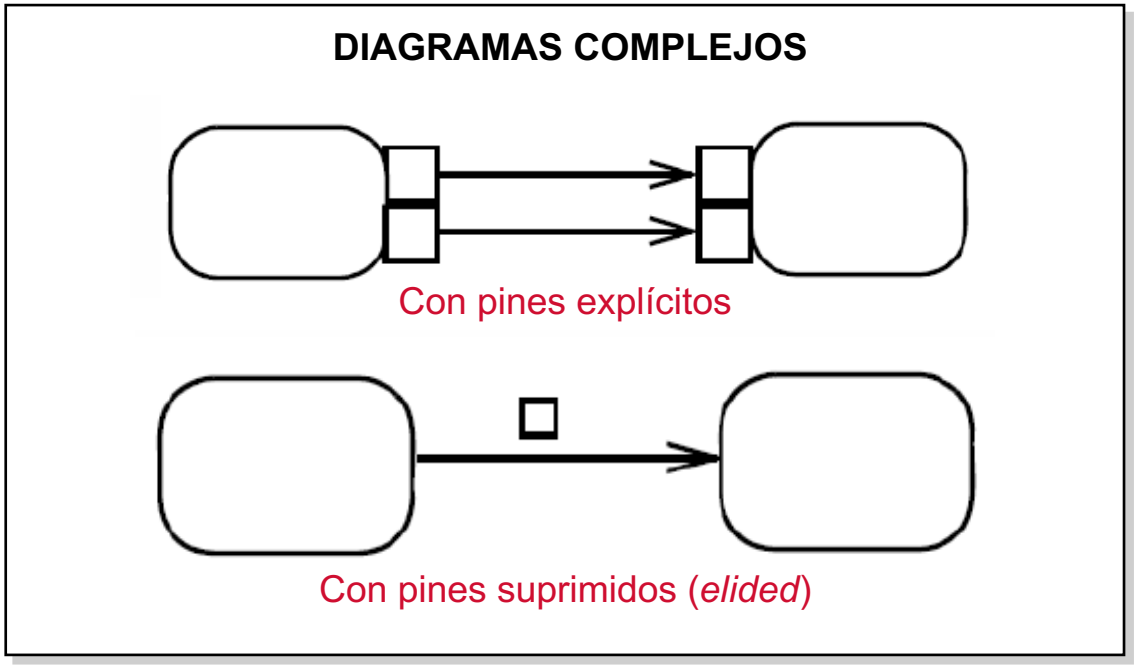
- **Pines (II)**





Diagramas de actividad

- **Pines (III)**

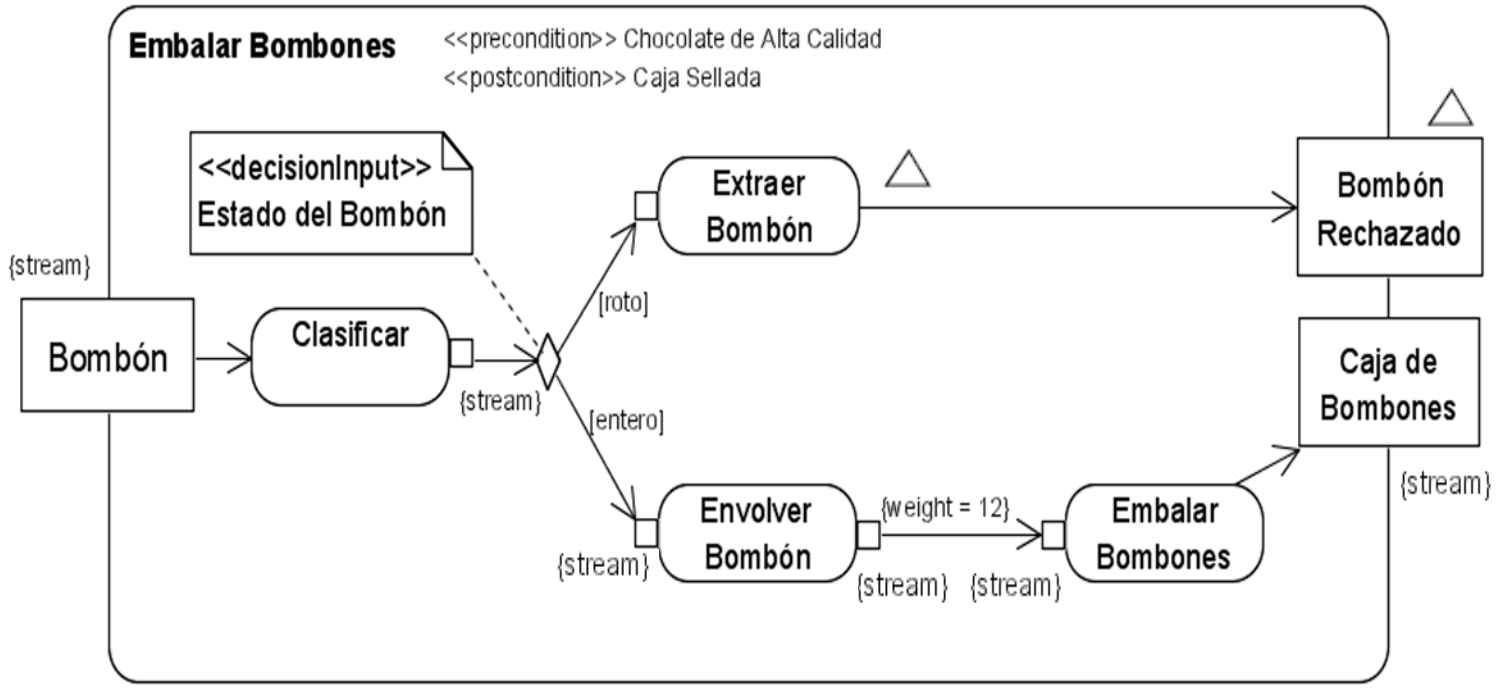


Diagramas de actividad

- **Nodos de parámetro (I)**
  - Nodos objeto que sirven para proporcionar entradas y salidas a una actividad
  - Cuando se invoca una actividad los valores de entrada se colocan como tokens en los nodos de parámetro de entrada
  - Las salidas de una actividad deben fluir a los nodos de parámetro de salida de la actividad
  - **Notación**: rectángulos colocados en el borde del símbolo de actividad
    - Los nodos de parámetro de entrada sólo tienen arcos salientes
    - Los nodos de parámetro de salida sólo tienen arcos entrantes
    - Se pueden representar asociados a los nodos las mismas restricciones que en los pines y en los nodos objeto:
      - {upperBound = valor}
      - {stream}
      - Parámetros de excepción
      - …

Diagramas de actividad

- **Nodos de parámetro (II)**

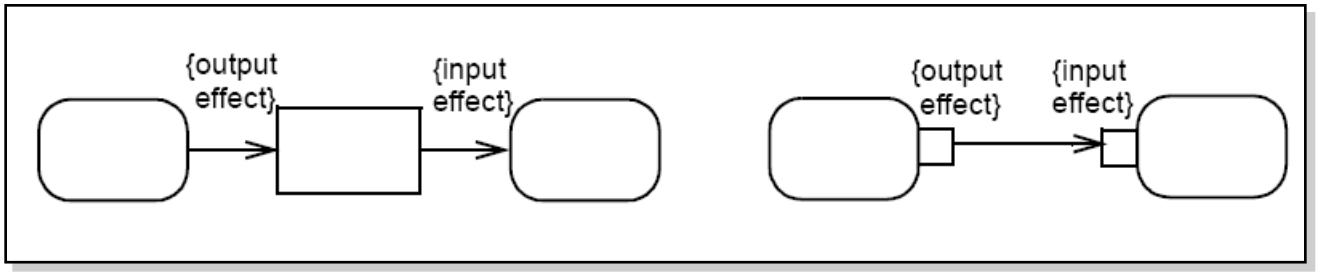


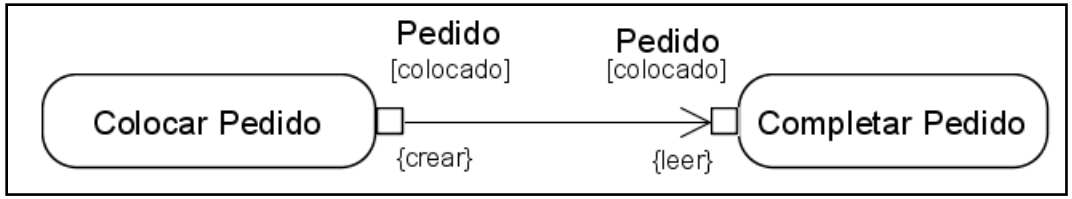
Diagramas de actividad

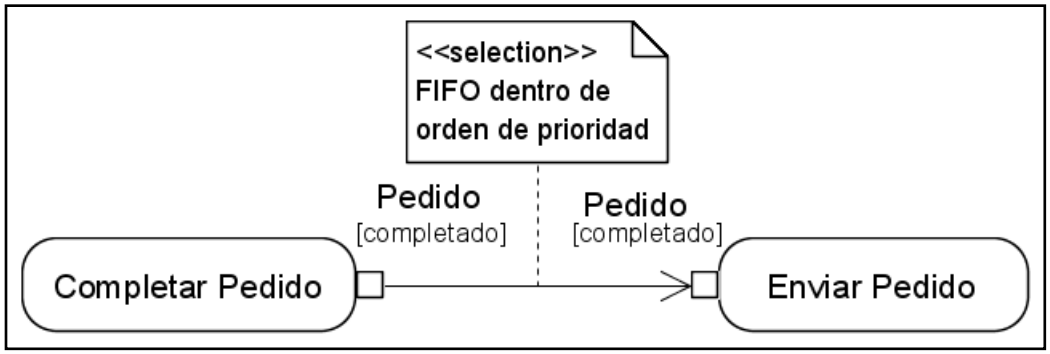
- **Flujo de objetos (I)**
  - Arco de actividad por el que pueden pasar datos u objetos
  - Pueden tener más de una acción al final
  - Los nodos objeto que conecta deben ser compatibles y tener el mismo valor del atributo *upperBound*
  - Varios flujos de objetos pueden tener el mismo nodo objeto en el origen, pero si uno de ellos toma el objeto, los otros arcos no tienen acceso a él
  - Se pueden especificar:
    - Comportamientos de transformación y selectivos utilizando las palabras clave «transformation» y «selection»
    - El efecto que las acciones tienen sobre los objetos que fluyen por el arco colocándolos entre llaves

Diagramas de actividad

- **Flujo de objetos (II)**







Diagramas de actividad

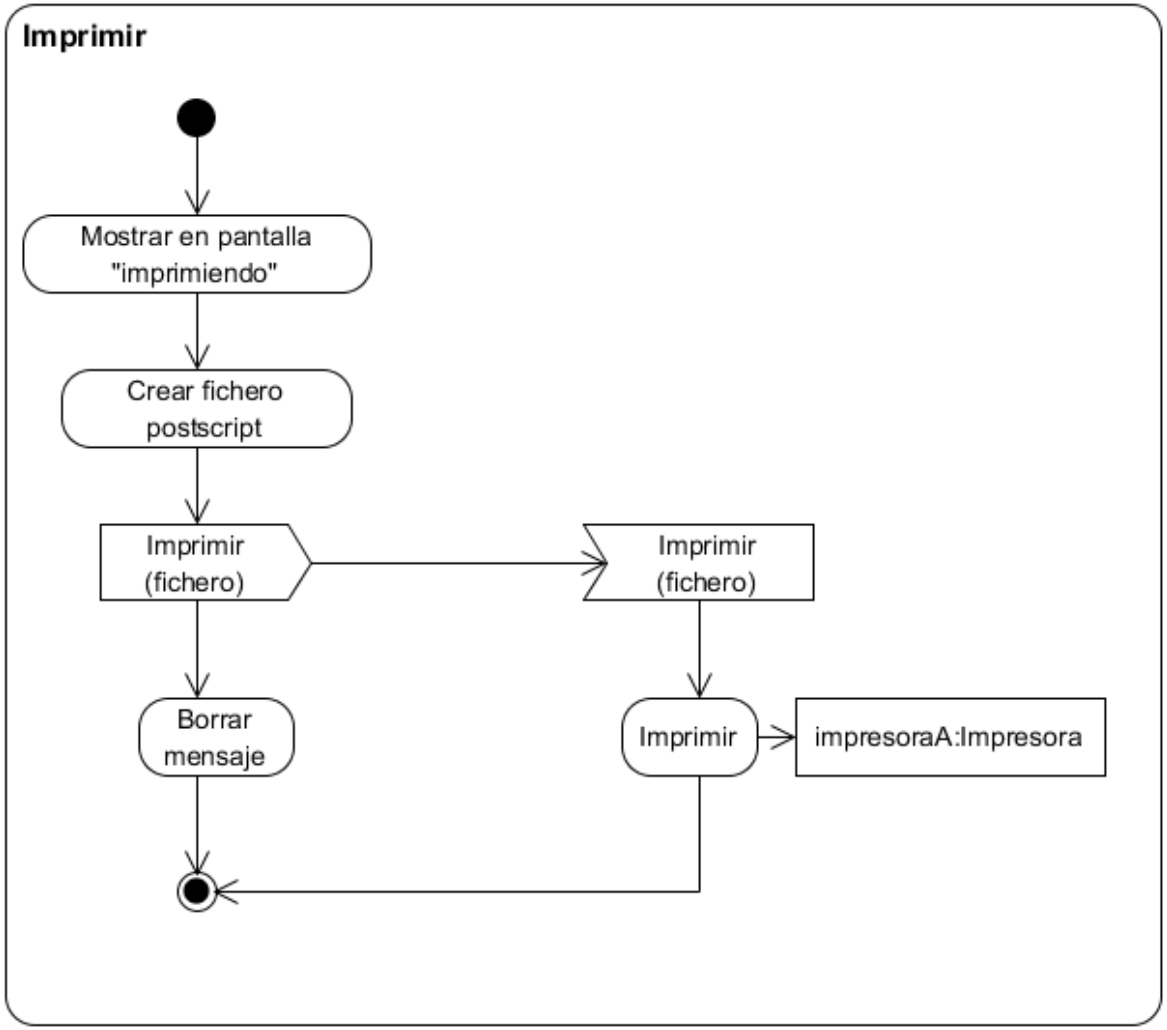
- **Señales (I)**
  - En los diagramas de actividad se puede representar el envío o la recepción de señales o eventos mediante acciones especiales
    - **Acción de envío de evento**
      - Su ejecución crea una instancia de una señal especificada utilizando los valores de entrada de la acción para asignar valores a los atributos de la señal
      - La ejecución termina cuando se hallan transmitido todas las instancias de la señal
      - La transmisión de la señal y el comportamiento del objeto de destino se realizan de forma concurrente
    - **Acción de recepción o aceptación de evento**
      - La ejecución de la acción bloquea la ejecución del hilo actual hasta que el objeto que lo posee detecta un tipo especificado de evento
      - Los parámetros del evento se incluyen en la salida de la acción
      - Se utiliza para invocaciones asíncronas
      - Se pueden utilizar para la recepción de **eventos de tiempo**

Diagramas de actividad

- **Señales (II)**
  - Los símbolos pueden ligarse a los objetos emisores y receptores de las señales
  - Cuando un receptor de señales tiene flujo de entrada representa que cuando el flujo le llega se habilita para aceptar una única señal
  - Cuando no tiene flujo de entrada, representa que puede aceptar una o muchas señales

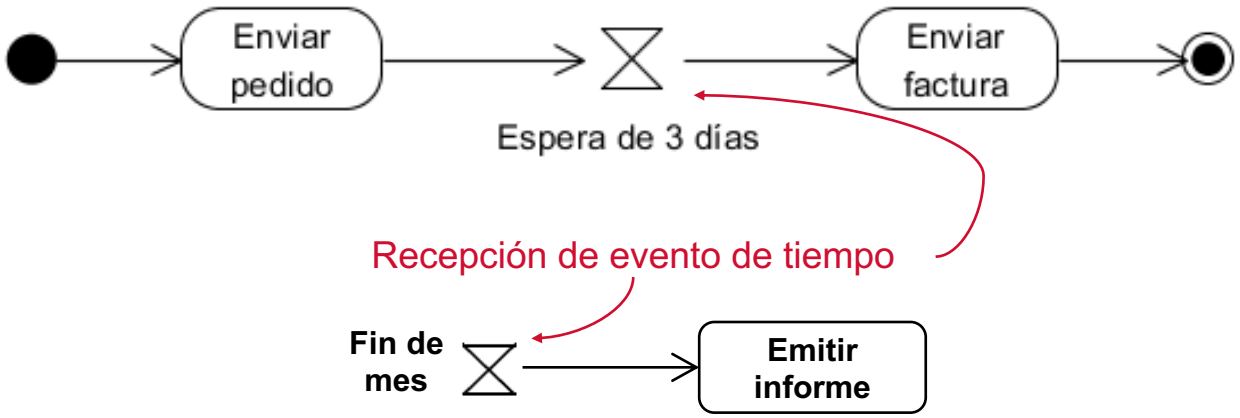
Diagramas de actividad

- **Señales (III)**



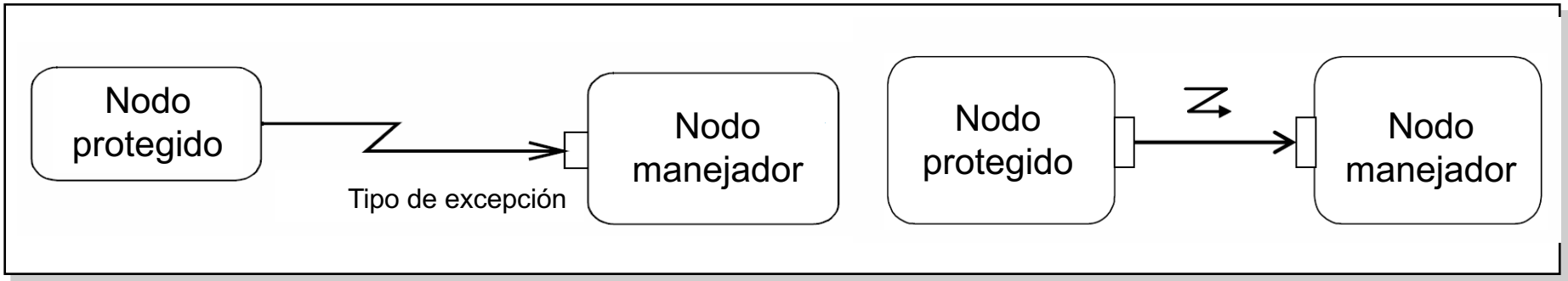
Diagramas de actividad

- **Recepción de evento de tiempo**
  - Es un tipo especial de acción de recepción de eventos que acepta un evento de tiempo
  - Se utiliza para modelar activaciones temporizadas, esperas, retrasos, etc.
  - Existen dos tipos
    - Con flujo de entrada: Se activa una vez tras recibir el flujo
    - Sin flujo de entrada: La activación puede ser repetida en el tiempo



Diagramas de actividad

- **Manejo de excepciones (I)**
  - Un **manejador de excepción** es un elemento que especifica un cuerpo que se ejecuta en caso de que ocurra una acción especificada durante la ejecución de un **nodo protegido**
  - La entrada al manejador de la excepción es la propia excepción
  - El nodo manejador de la excepción tiene un nodo objeto de entrada que no puede recibir más de un arco
  - **Notación**:
    - Arco de interrupción que conecta el nodo protegido y el nodo manejador de la excepción
    - El arco de interrupción está etiquetado con el nombre del tipo de excepción



Diagramas de actividad

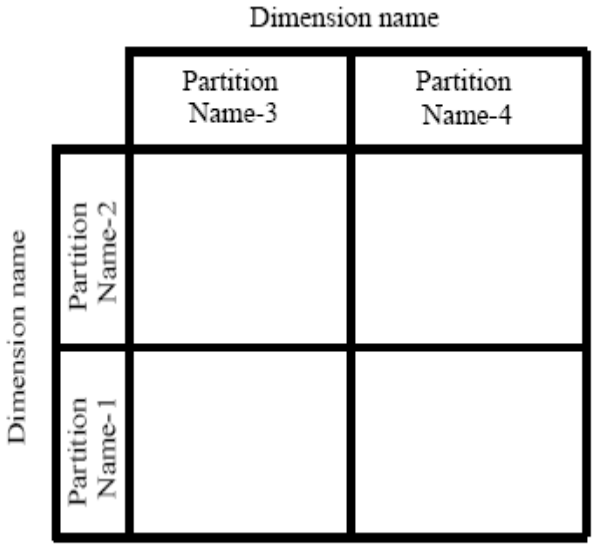
- **Particiones de actividad (*swinlanes*) (I)**
  - Particiones utilizadas para identificar acciones que comparten características comunes.
  - Pueden representar:
    - Unidades organizativas en el modelado de negocio
    - Clasificadores
    - Instancias
    - Partes
    - Atributos y valores de los atributos
  - Las particiones pueden compartir contenidos
  - **Notación**:
    - Líneas paralelas horizontales o verticales con el nombre en una caja colocada en un extremo
    - Se pueden representar particiones jerárquicas y multidimensionales

Diagramas de actividad

- **Particiones de actividad (*swinlanes*) (II)**

Representación de una partición con notación *swinlane*

Representación particiones jerárquicas



Representación de particiones Multidimensionales

Diagramas de actividad

- **Particiones de actividad (*swinlanes*) (III)**

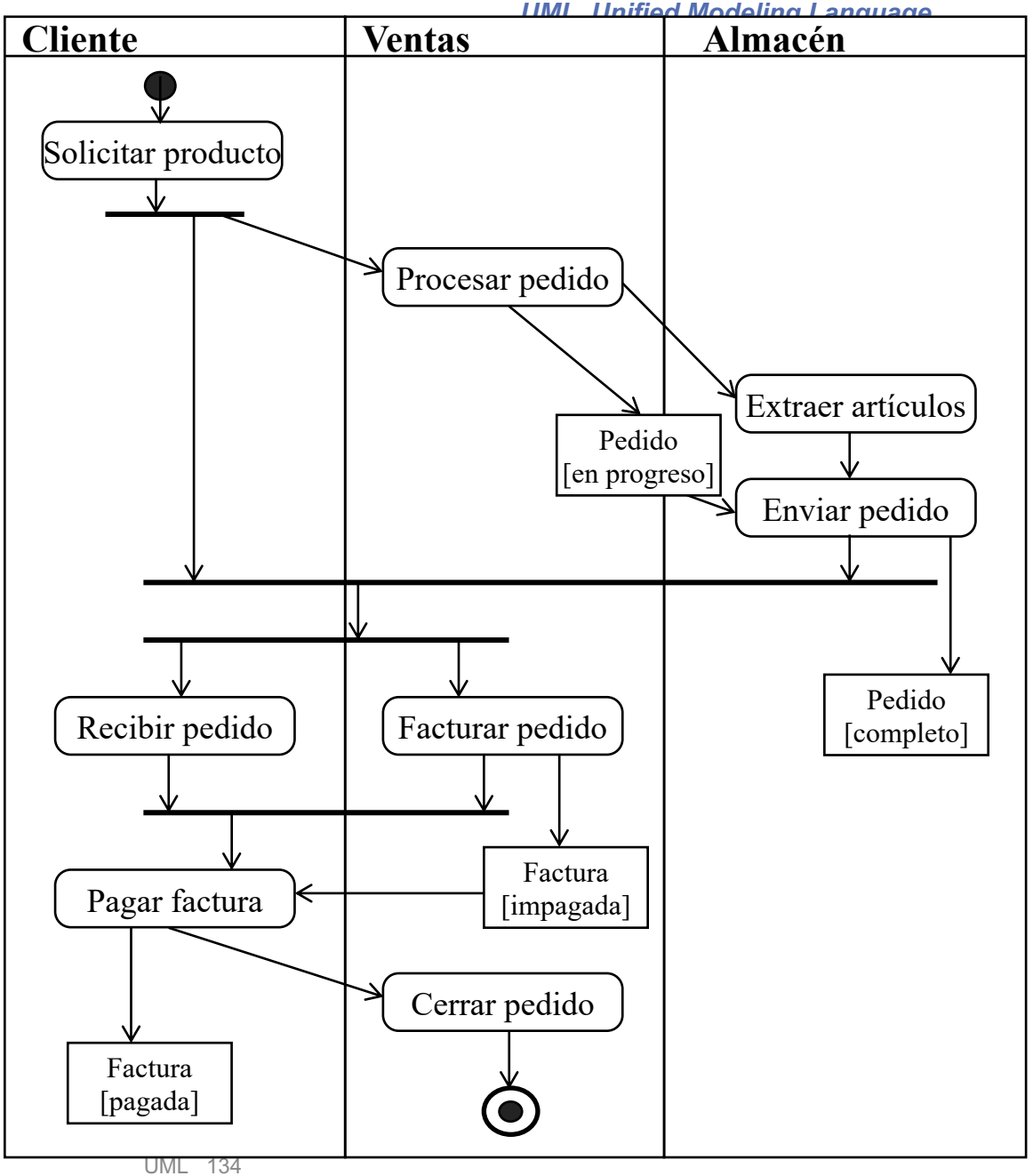
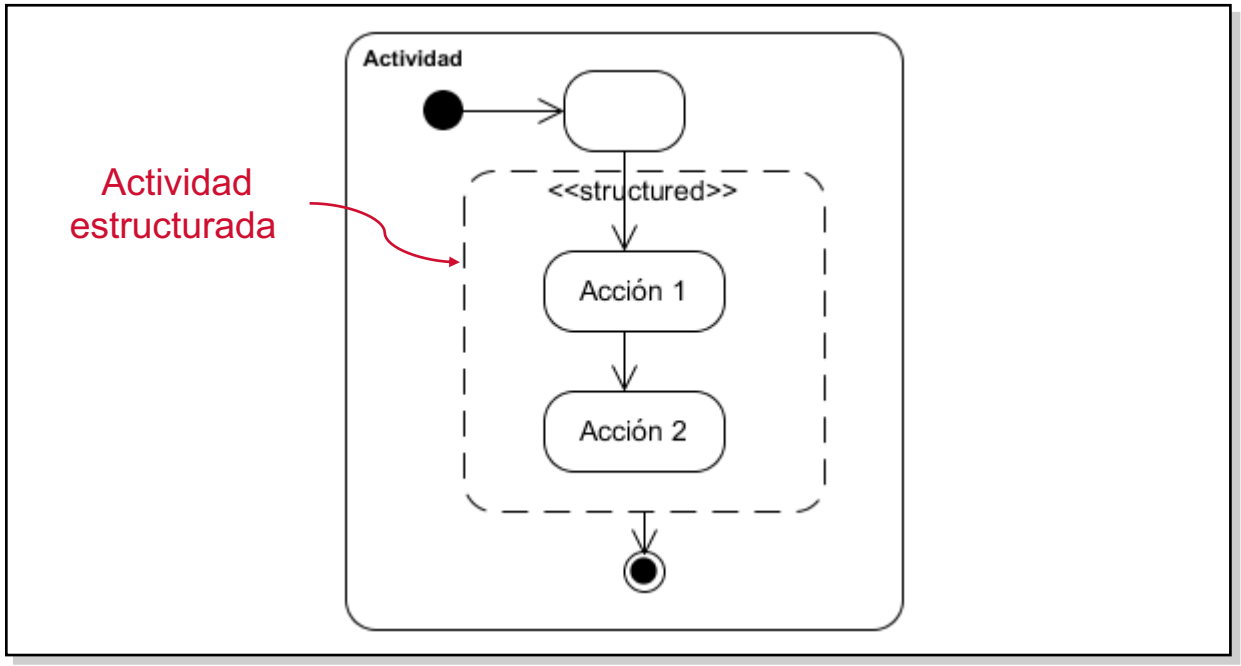


Diagrama de actividad con tres particiones

Diagramas de actividad

- **Actividad estructurada**
  - Nodo de actividad ejecutable que puede expandirse en nodos subordinados
    - Los nodos solo pueden pertenecer a una actividad estructurada
    - Los nodos pueden estar anidados
  - **Notación**: rectángulo con las esquinas redondeadas y línea discontinua que contiene la palabra clave «structured» en la parte superior

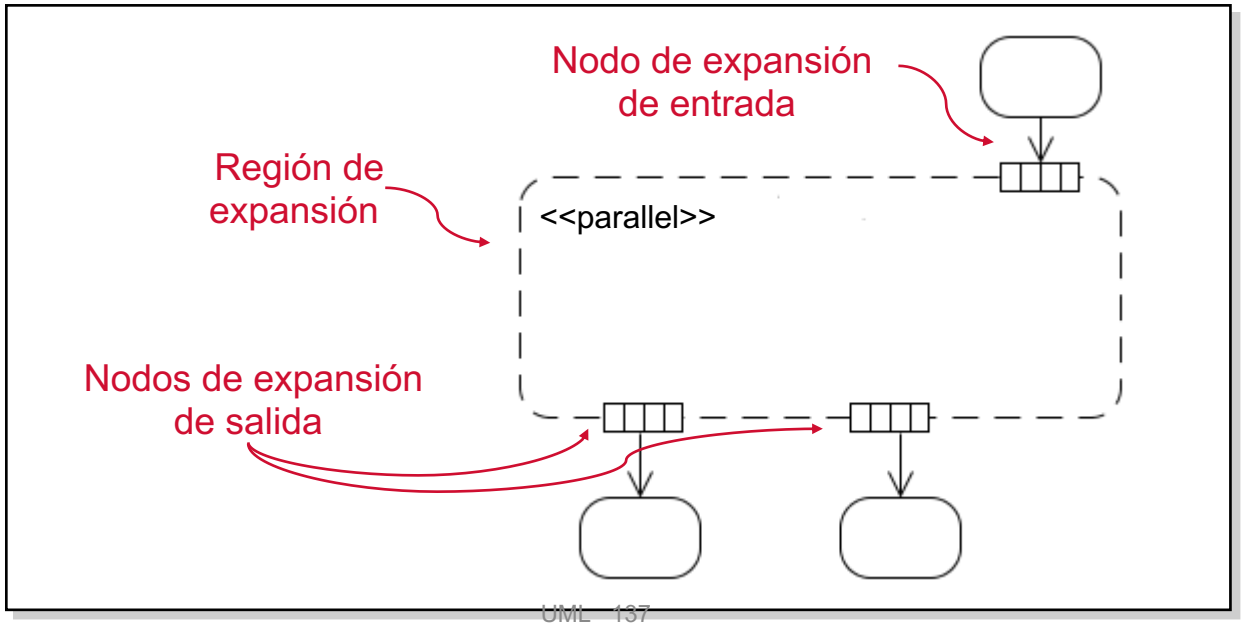


Diagramas de actividad

- **Región de expansión**
  - Región de actividad estructurada que se ejecuta múltiples veces, una vez por cada elemento de una colección de elementos de entrada
  - Tipos
    - **Paralela (*parallel*)**: La ejecución de los elementos de la colección de entrada se puede solapar en el tiempo
    - **Iterativa (*iterative*)**: La ejecución tiene lugar secuencialmente por cada elemento de la colección de entrada
    - **De flujo (*stream*)**: Tiene lugar una única ejecución en la que se van procesando los elementos de la colección de entrada como un flujo (no hace falta que termine completamente el procesamiento de uno para que comience el del siguiente)
  - **Notación**: rectángulo con las esquinas redondeadas y línea discontinua que contiene las palabras clave *parallel*, *iterative* o *stream* en la parte superior

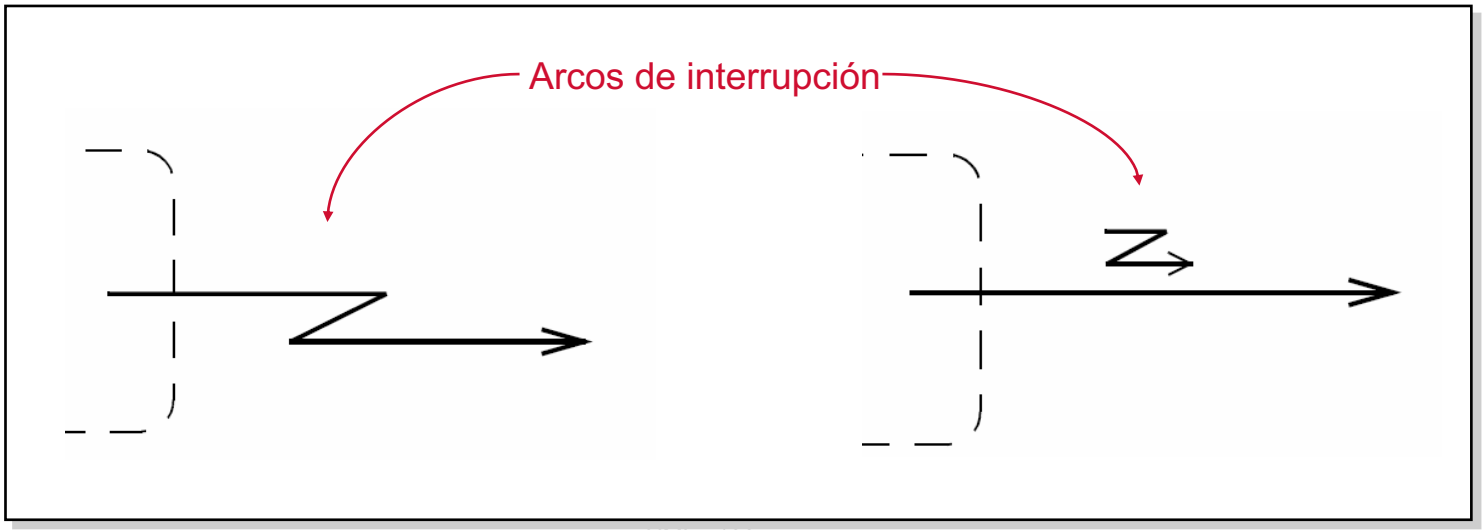
Diagramas de actividad

- **Nodos de expansión**
  - Nodos objeto utilizados para indicar un flujo que atraviesa los límites de una región de expansión
    - **De entrada**: flujo de entrada a la región de expansión que contiene una colección que se divide en elementos individuales dentro de la región
    - **De salida**: flujo de salida de la región de expansión que combina los elementos individuales en una colección que se usa fuera de la región



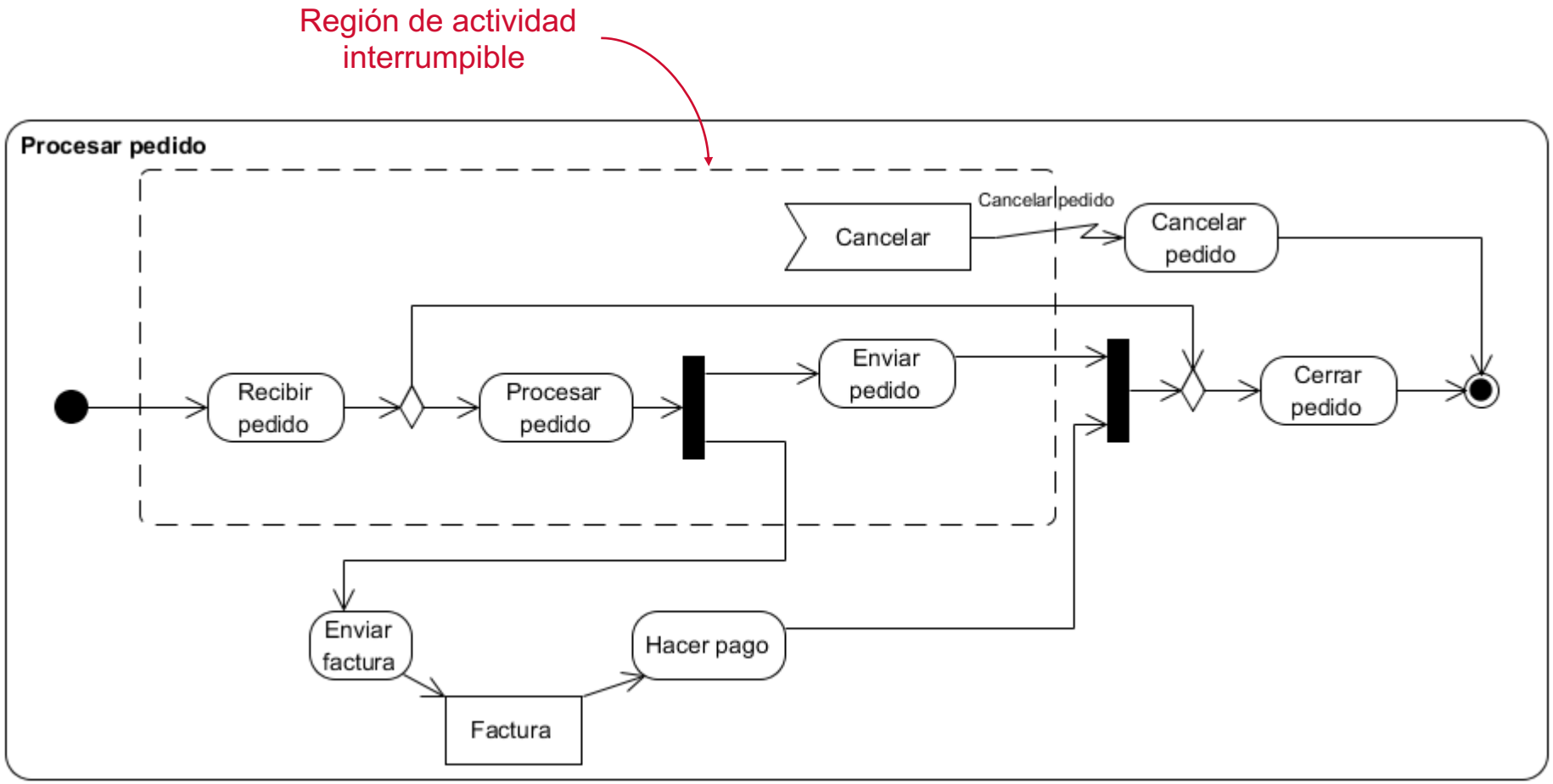
Diagramas de actividad

- **Región de actividad interrumpible (I)**
  - Actividad estructurada dentro de la cual se puede interrumpir una actividad si se produce el evento especificado
  - Al producirse el evento de interrupción se termina toda la actividad de la región y se transfiere el control al nodo que maneja la interrupción
  - **Notación**: rectángulo con las esquinas redondeadas y línea discontinua de cuyo interior parten uno o más arcos de interrupción (dos notaciones alternativas)



Diagramas de actividad

- **Región de actividad interrumpible (II)**



Diagramas de actividad

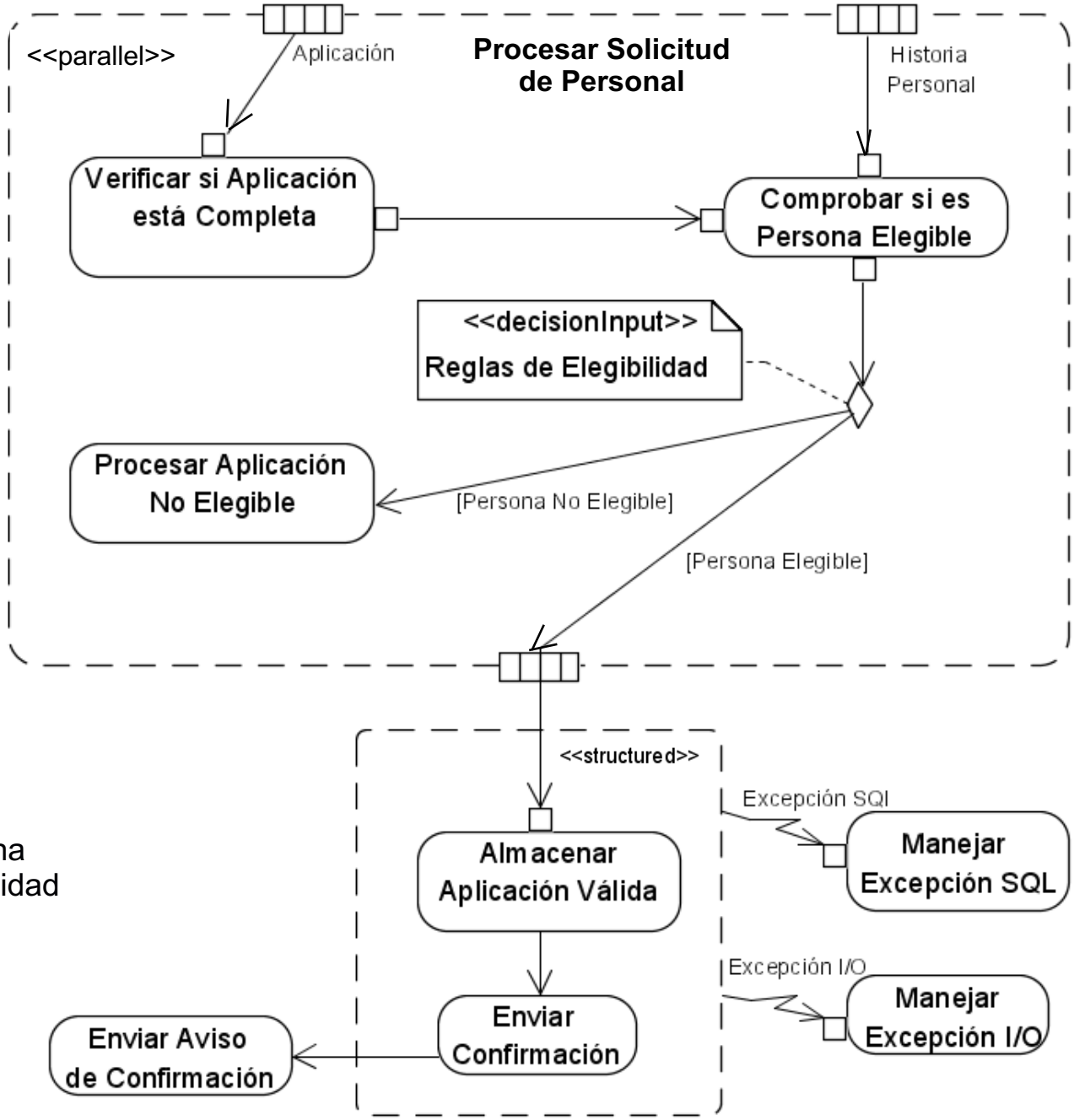
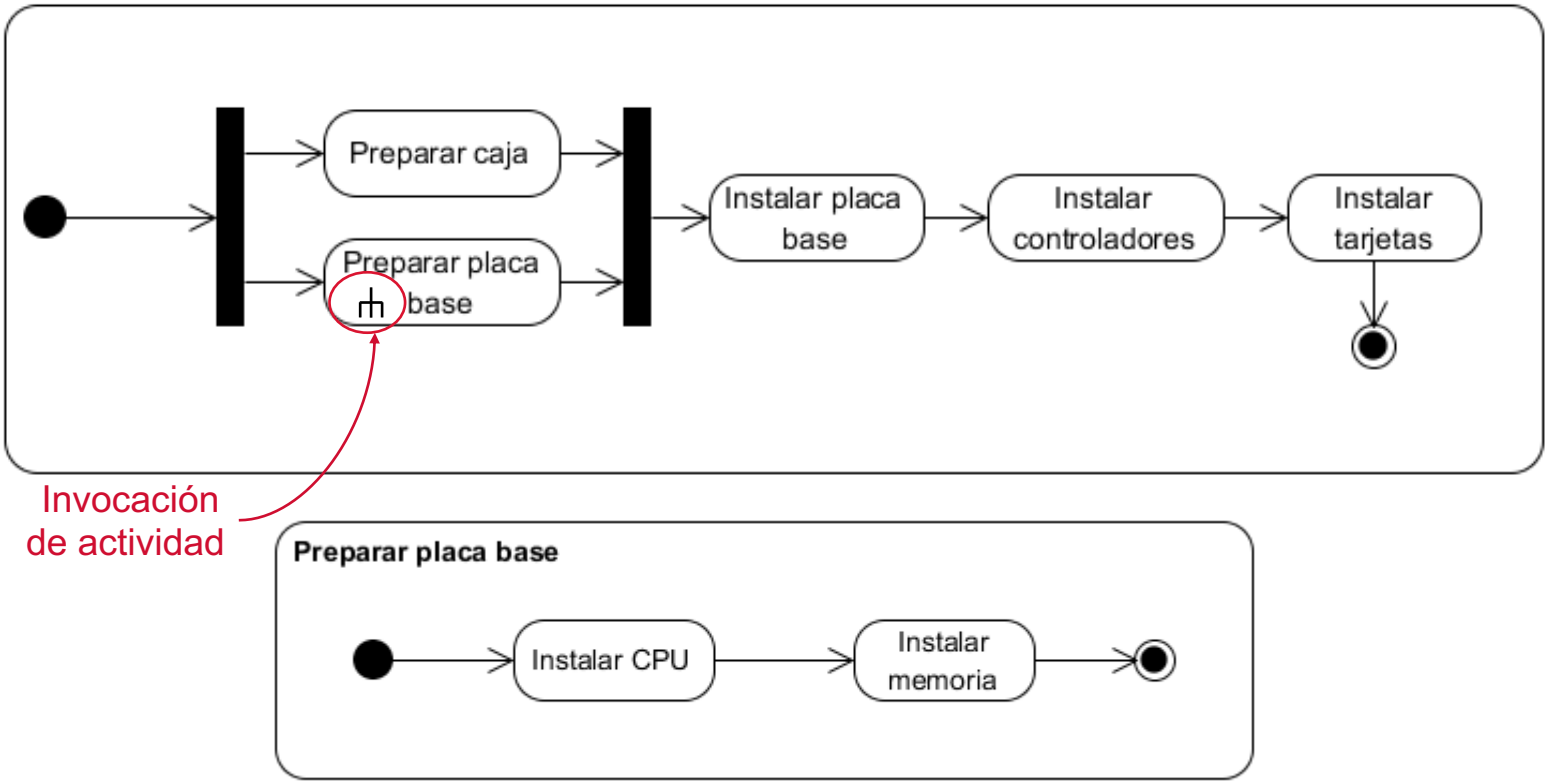


Diagrama de actividad con una región de expansión y una actividad estructurada

Diagramas de actividad

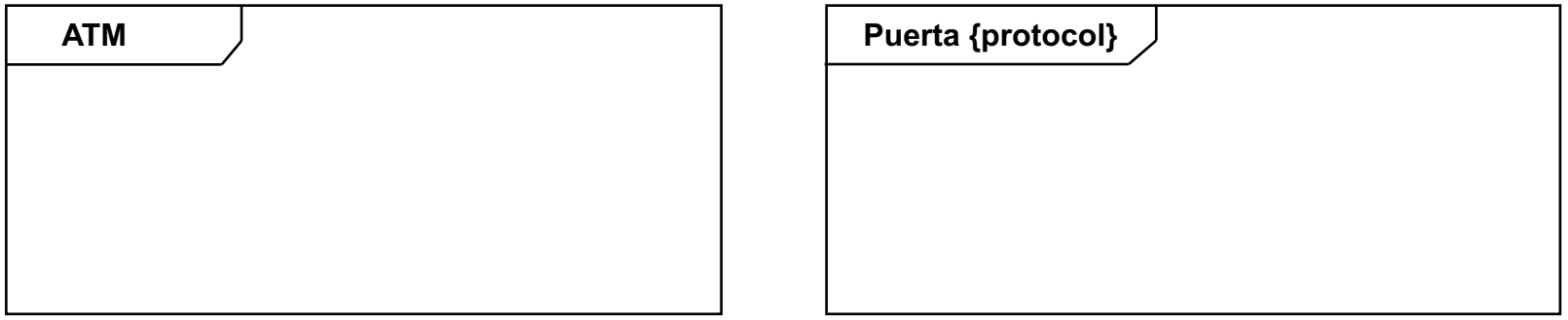
- **Invocación de actividad**
  - Una acción puede representar la invocación a una actividad que se modela en otro diagrama
    - La actividad referenciada se ejecuta utilizando los valores de entrada a la acción



Vista de máquina de estados

Características

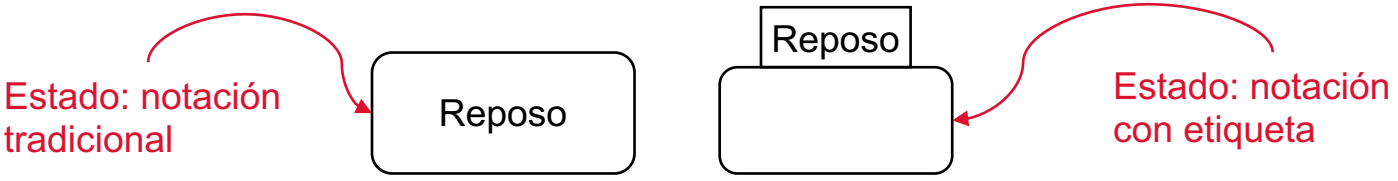
- Las máquinas de estados describen los estados que un objeto puede tener durante su ciclo de vida, el comportamiento en esos estados y los eventos que causan los cambios de estado
- UML define dos tipos de máquinas de estados:
  - **De comportamiento**: capturan los ciclos de vida de los objetos, subsistemas y sistemas
  - **De protocolo**: se usan para especificar las transformaciones legales que pueden ocurrir en un clasificador abstracto como una interfaz o un puerto (protocolos de uso)



Marcos utilizados para representar máquinas de estados de comportamiento y de protocolo

Estados y transiciones

- El estado de un objeto es consecuencia de las actividades que ha realizado previamente y está determinado por los valores de sus atributos y los enlaces con otros objetos
- Las clases pueden tener un atributo específico que indique su estado
- Las transiciones de un estado a otro se producen cuando ocurre un evento
- **Notación**
  - Los estados se representan con el símbolo del rectángulo con las esquinas redondeadas con el nombre dentro o como una etiqueta adosada al símbolo
  - Las transiciones se representan con flechas etiquetadas con el evento que causa la transición



Eventos

- Un evento es algo que sucede y que puede causar alguna acción
- Las conexiones bien definidas entre eventos y acciones se denominan ***causalidad***
- Los eventos son disparadores que activan las transiciones de estado
- Una clase puede recibir o enviar eventos
- Cuando un objeto detecta un evento responde de diferente forma dependiendo de su estado. La respuesta puede incluir la ejecución de una acción o un cambio de estado
- Los eventos pueden ser de cuatro **tipos**
  - Condiciones que se hacen ciertas
  - Recepción de señales explícitas desde otros objetos
  - Recepción de una llamada a una operación por otro objeto o por el mismo
  - Transcurso de un periodo de tiempo determinado

Eventos

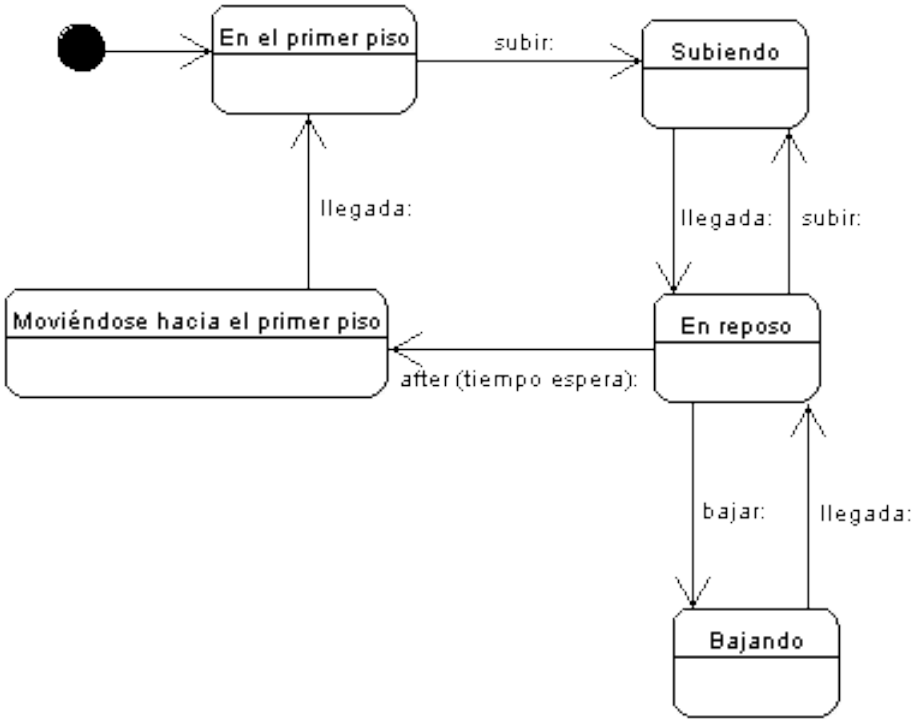
- Se representan mediante una signatura que se compone del nombre del evento y una lista de parámetros separados por comas con la sintaxis

  NombreEvento (NombreParámetro: expresiónTipo, NombreParámetro: expresiónTipo ...)

- Un disparador de tiempo relativo se expresa

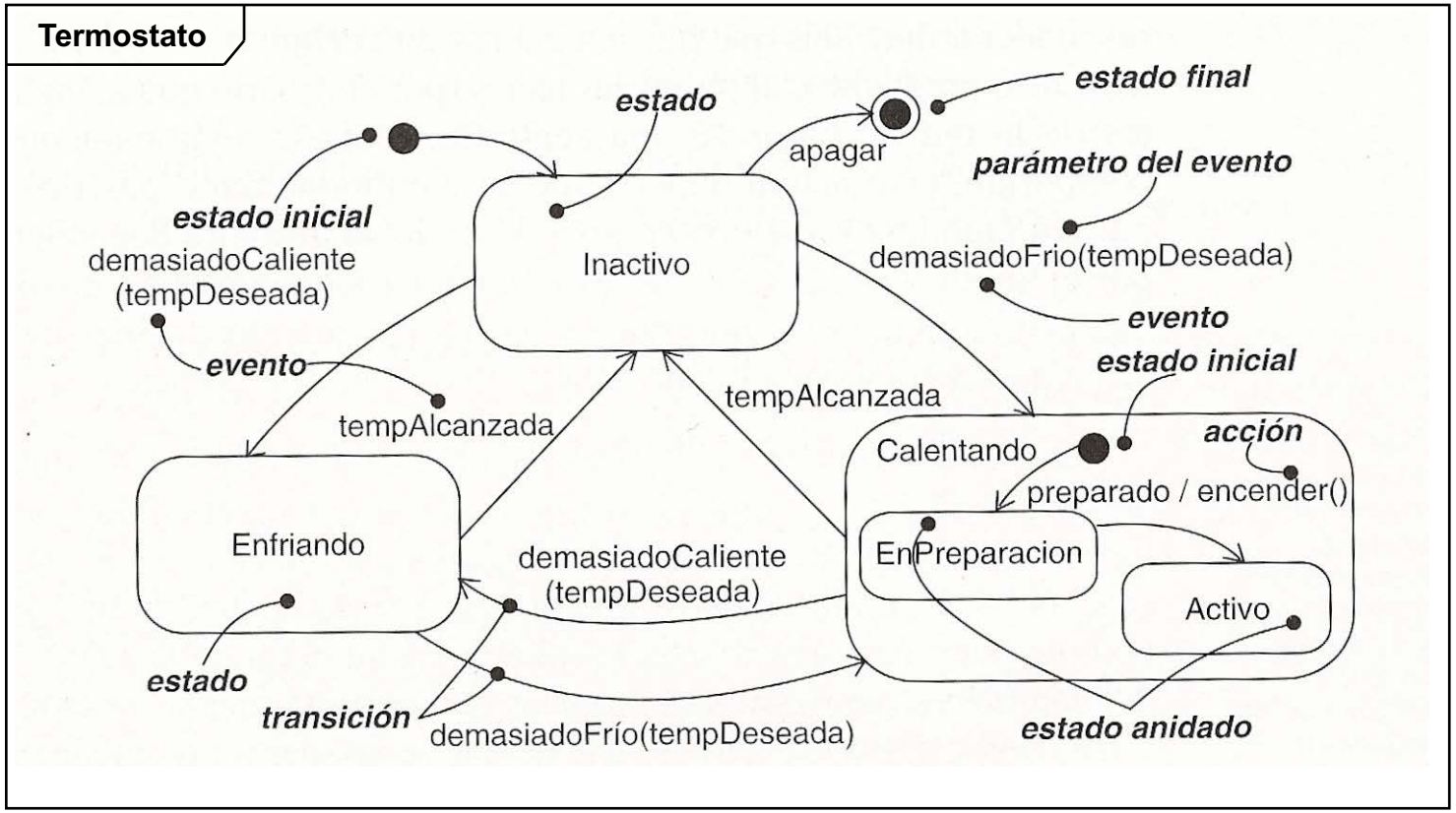
  after (tiempo)

  Ejemplos
  - dibujar (f: Figura, c: Color)
  - redibujar ( )
  - redibujar
  - imprimir (factura)
  - after (5 segundos)



Máquina de estados de un ascensor

Eventos



Máquina de estados de un termostato

Transiciones

- Una transición es una relación dirigida de un nodo origen a un nodo destino que representa la respuesta de la máquina de estados a un evento particular
- **Tipos de transiciones**
  - **Externas**: Causan un cambio de estado. Abandonan el estado fuente cuando se disparan
  - **Internas**: No causan cambio de estado. Cuando se disparan no entran ni salen del estado origen, por lo que no se ejecutan sus acciones de entrada y salida
  - **Locales**: No salen del estado compuesto pero pueden entrar y salir en los subestados del estado compuesto
- **Condición de guarda**: expresión booleana que se asocia al evento que produce la transición. Si el evento ocurre y la condición de guarda es verdadera la transición se dispara

  [t = 15 seg.]
  reembolso (cantidad) [saldo >= cantidad]
- Una transición está habilitada si
  - Todos sus estados fuente están activos
  - Ocurre el evento que la dispara
  - Las condiciones de guarda son verdaderas

Transiciones

- **Notación**: Se representan con una flecha etiquetada con una expresión con la siguiente sintaxis

  SignaturaEvento `[´Guarda`]´ / expresiónAcción ^ClausulaSend

  - **Expresión de acción**: Expresión que se ejecuta cuando se dispara la transición
    - Una transición puede tener asociadas varias acciones separadas por el carácter "/"
    - Las acciones se ejecutan en el orden en que están especificadas

      Ejemplo: Increase ( ) / n:= n + 1 / m:= m + 1
  - **Cláusula *send*** : es un caso especial de acción que representa el envío de un mensaje durante la transición entre dos estados
    - Sintaxis:

      ExpresiónDestino. NombreEvento (listaArgumentos)
      - **ExpresiónDestino**: Evalúa un objeto o conjunto de objetos
      - **NombreEvento**: Evento con significado para el objeto u objetos destino

Ejemplos

[timer = time-out] ^ self.bajar(primerPiso)
BotonIzqRatonPulsado (posicion) /color:=selec(posicion) ^lapiz.set(color)

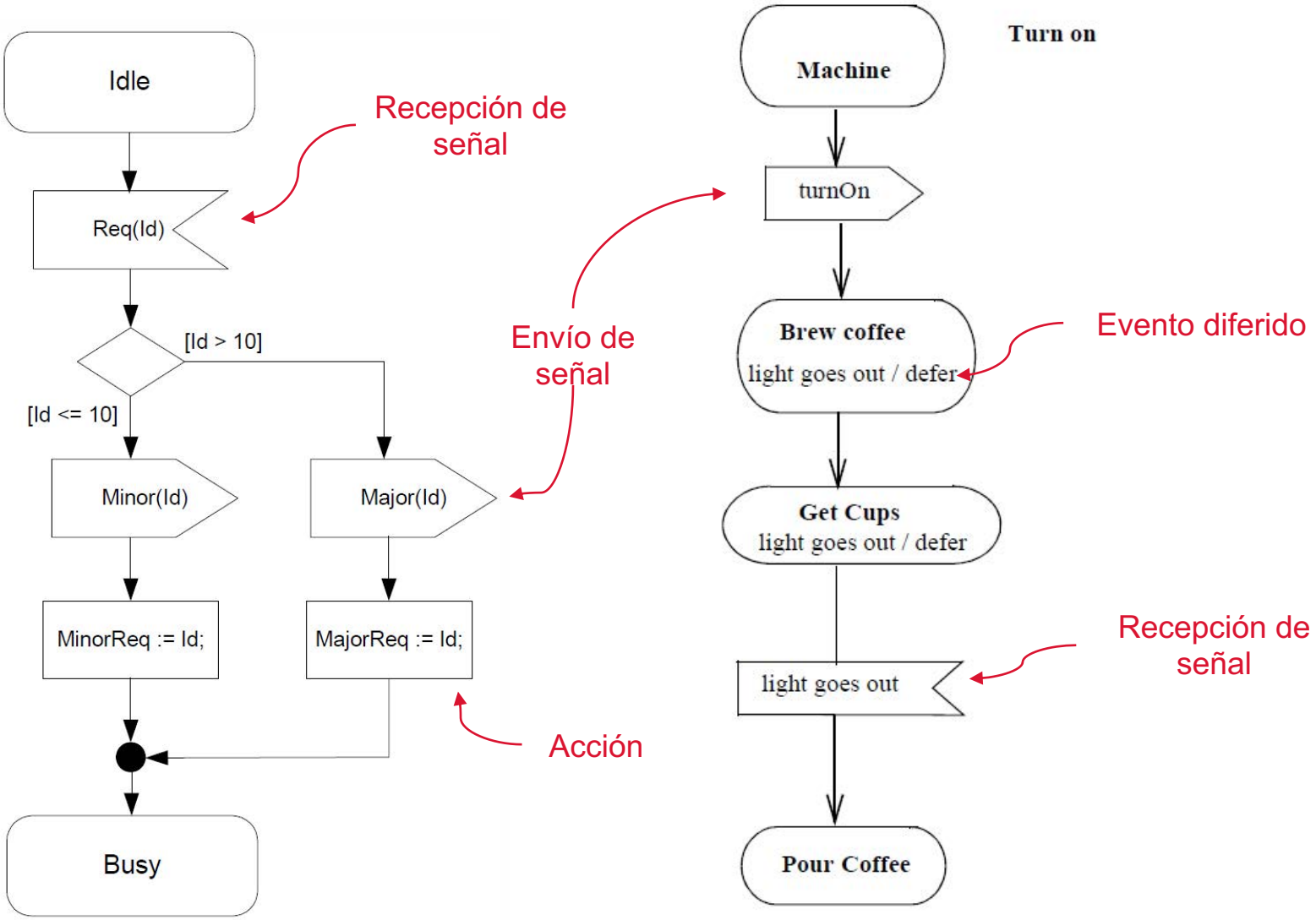
Transiciones

- **Señales y acciones en una transición**
  - **Recepción de señal**
    - Representa el disparador de una transición
    - La especificación textual del evento disparador se coloca dentro del símbolo y si la transición tiene una condición de guarda también se especifica

      evento '['guarda ']'
  - **Envío de señal**
    - Representa una acción
    - Los parámetros de la señal se muestran dentro del símbolo
  - **Otras acciones**
    - Se representan con un rectángulo que contiene la representación textual de la acción

Transiciones

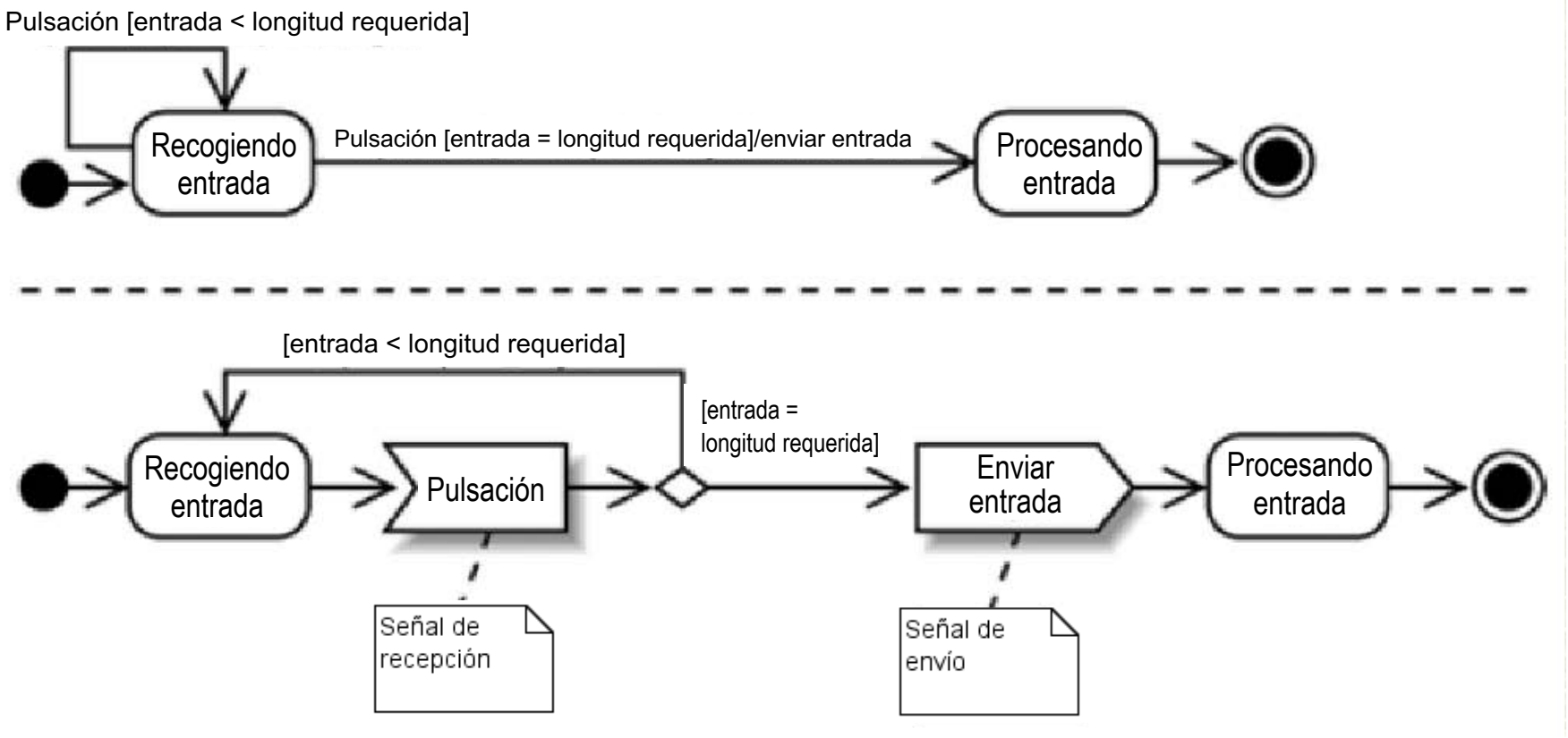
- **Señales y acciones en una transición (II)**



Ejemplos de representación de envío y recepción de señales y acciones en las transiciones

Transiciones

- **Señales y acciones en una transición (III)**



Representaciones alternativas de las transiciones entre estados

Estados

- **Estructura de un estado (I)**
  - **Compartimento del nombre**
    - **Nombre**: Cadena de texto que distingue al estado de otros estados
  - **Compartimento de actividades internas**: es opcional y describe comportamientos que se realizan cuando el elemento está en ese estado

    Sintaxis: etiquetaActividad / expresiónComportamiento

    La **etiqueta de actividad** identifica las circunstancias bajo las que se invoca el comportamiento especificado en la expresión de comportamiento

    Etiquetas estándar definidas en UML
    - **Etiqueta de entrada (*entry*)**: usada para especificar acciones ejecutadas al entrar al estado
    - **Etiqueta de salida (*exit*)**: usada para especificar acciones ejecutadas al salir del estado
    - **Etiqueta hacer (*do*)**: usada para especificar una actividad que se realiza mientras se está en ese estado o hasta que el procesamiento especificado por la expresión se ha completado

Estados

- **Estructura de un estado (II)**
  - **Compartimento de transiciones internas**: es opcional y contiene una lista de transiciones internas
    - **Transiciones internas**: Transiciones que se manejan sin causar un cambio en el estado

      Sintaxis:

      SignaturaEvento `[´Guarda`]´ / expresiónAcción^ClausulaSend

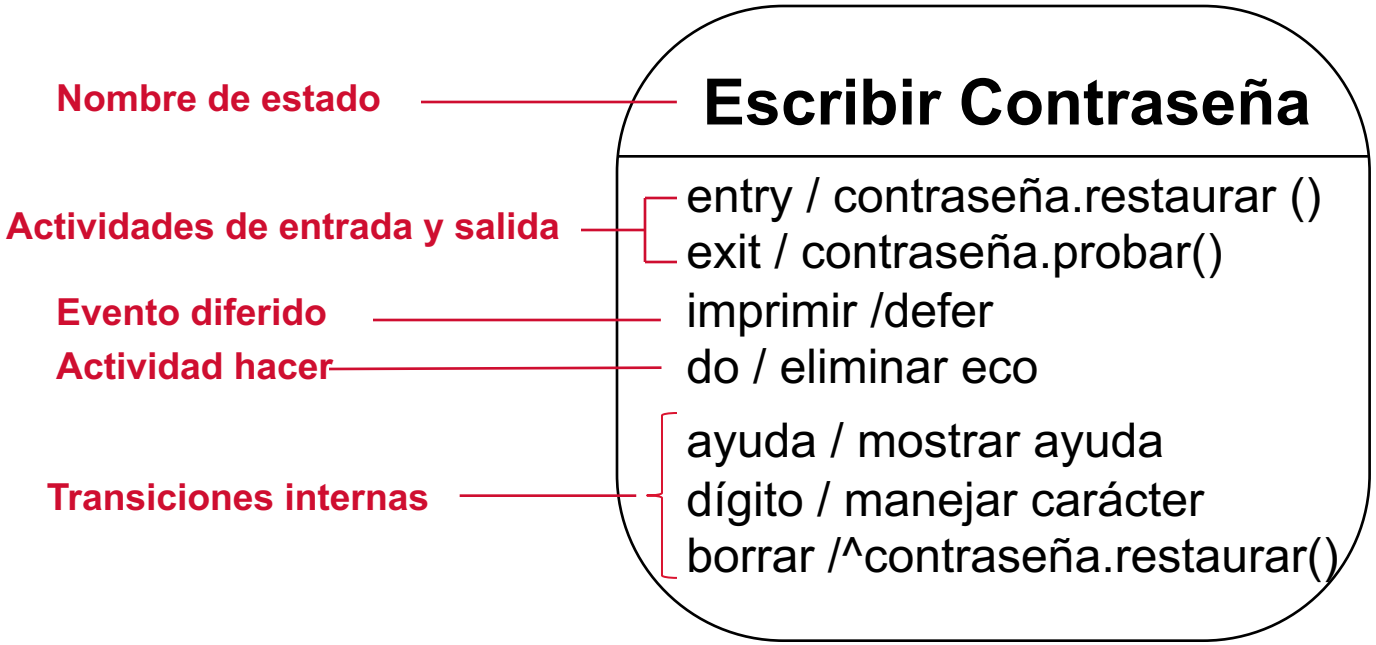
      Cualquier tipo de evento se puede diferir:

      - **Evento diferido**: se puede retrasar en un cierto estado y en sus subestados

        Sintaxis: signaturaEvento / defer
  - **Compartimento de descomposición**: sólo se usa en **máquinas de estados compuestas**
    - **Subestados**: Estructura anidada de un estado que engloba subestados disjuntos (activos secuencialmente) o concurrentes (activos concurrentemente)

Estados

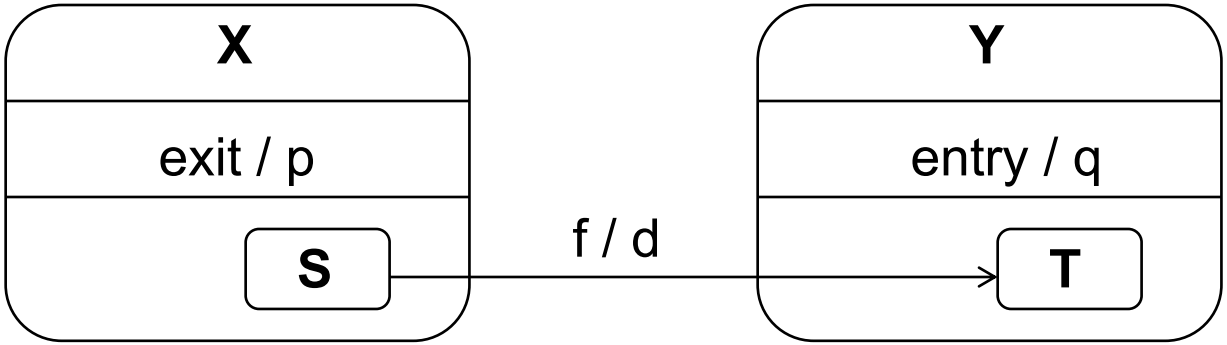
- **Estructura de un estado (III)**



Estructura interna de un estado

Estados

- **Estructura de un estado (IV)**

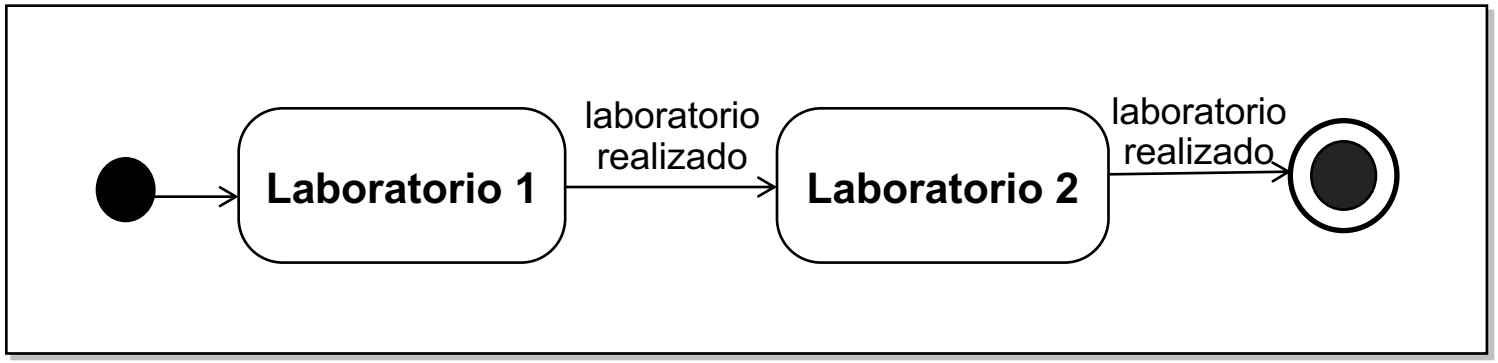


Resultado efectivo -> f / p; d; q

Transición entre estados compuestos

Estados

- **Estados inicial y final**: Estados especiales que se pueden definir en la máquina de estados de un objeto. Son *pseudoestados*, cuyo propósito es ayudar a estructurar la máquina de estados
  - El **estado inicial** indica el punto de comienzo por defecto para la máquina de estados o el subestado. Se representa por un círculo negro
  - El **estado final** indica la finalización de la ejecución de la máquina de estados o del estado que lo contiene. Se representa por un círculo negro dentro de un círculo blanco



Máquina de estados con estado inicial y final

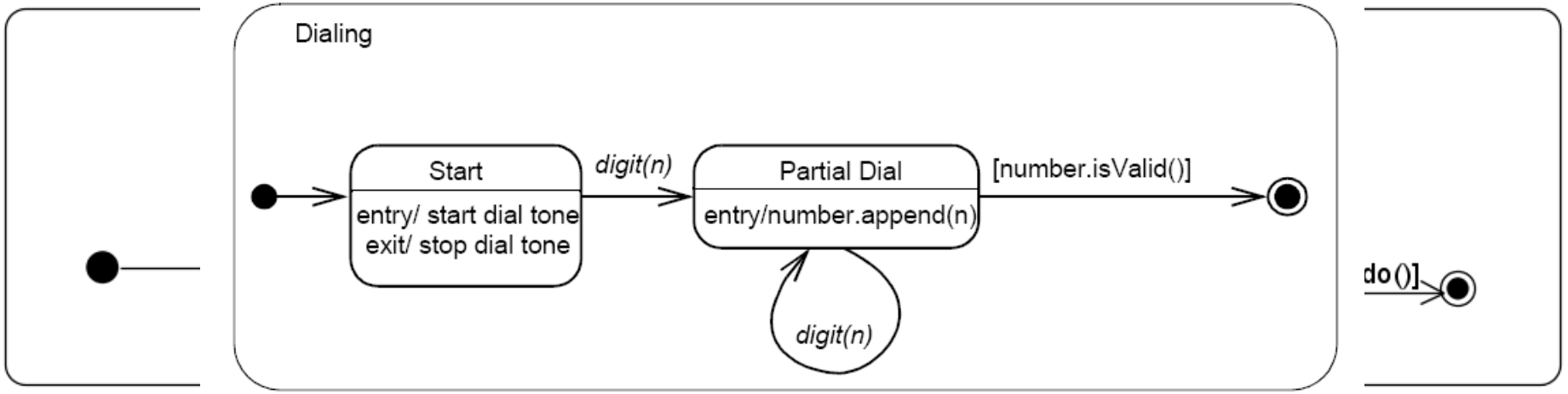
Estados

- **Estados compuestos (I)**

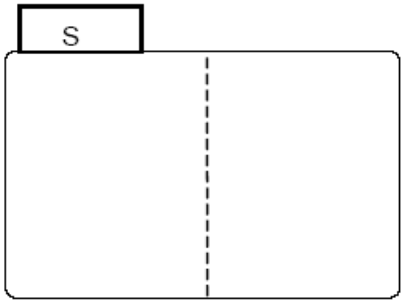
  Un estado compuesto es aquél que se ha descompuesto en subestados secuenciales (disjuntos) o concurrentes (ortogonales)
  - La descomposición en subestados disjuntos es un tipo de especialización de un estado
  - Un sistema puede estar en múltiples estados simultáneamente
    - El conjunto de estados activos se llama configuración del estado activo
    - Si el objeto permite concurrencia, entonces puede estar activo más de un subestado concurrente
  - **Notación**
    - Estado con un compartimento adicional: compartimento de descomposición
    - La expansión de un estado en subestados disjuntos se muestran mediante un diagrama de estados anidado dentro de la región gráfica
    - Una expansión de un estado concurrente compuesto en subestados concurrentes se muestra dividiendo la zona gráfica del estado usando líneas discontinuas para separarlo en subregiones
    - Las zonas de texto del estado completo se separan de los subestados concurrentes por una línea continua
    - Se puede mostrar la estructura de un estado compuesto en un diagrama independiente, entonces el estado puede representase como un estado simple con un icono especial en la esquina inferior derecha

Estados

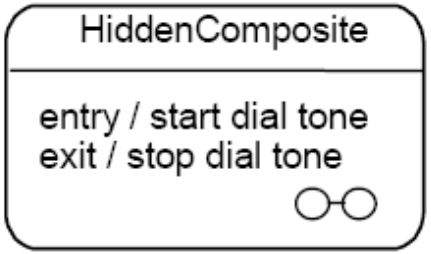
- **Estados compuestos (II)**



Estructura interna de un estado compuesto



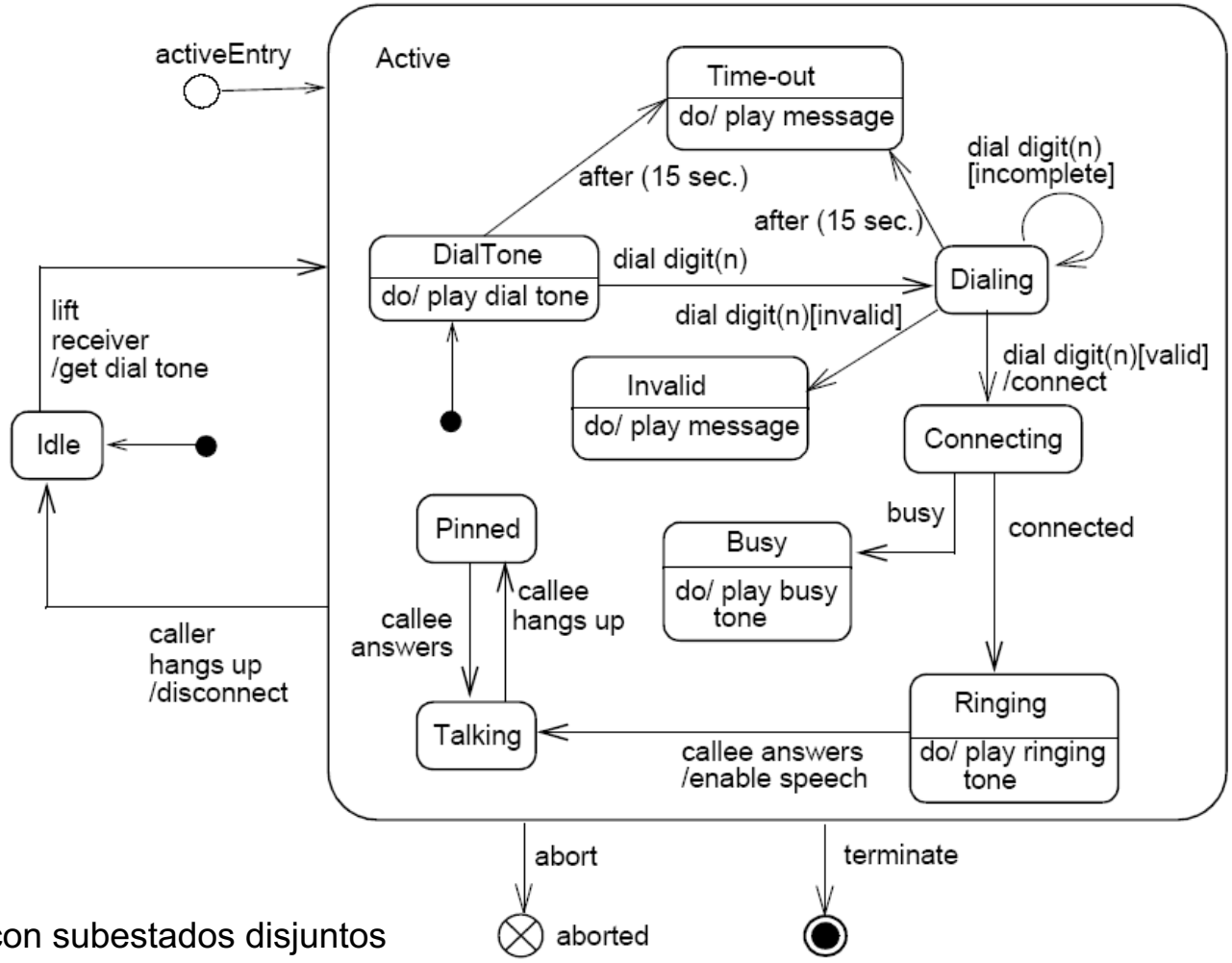
Estado compuesto con regiones



Estado compuesto con descomposición oculta

Estados

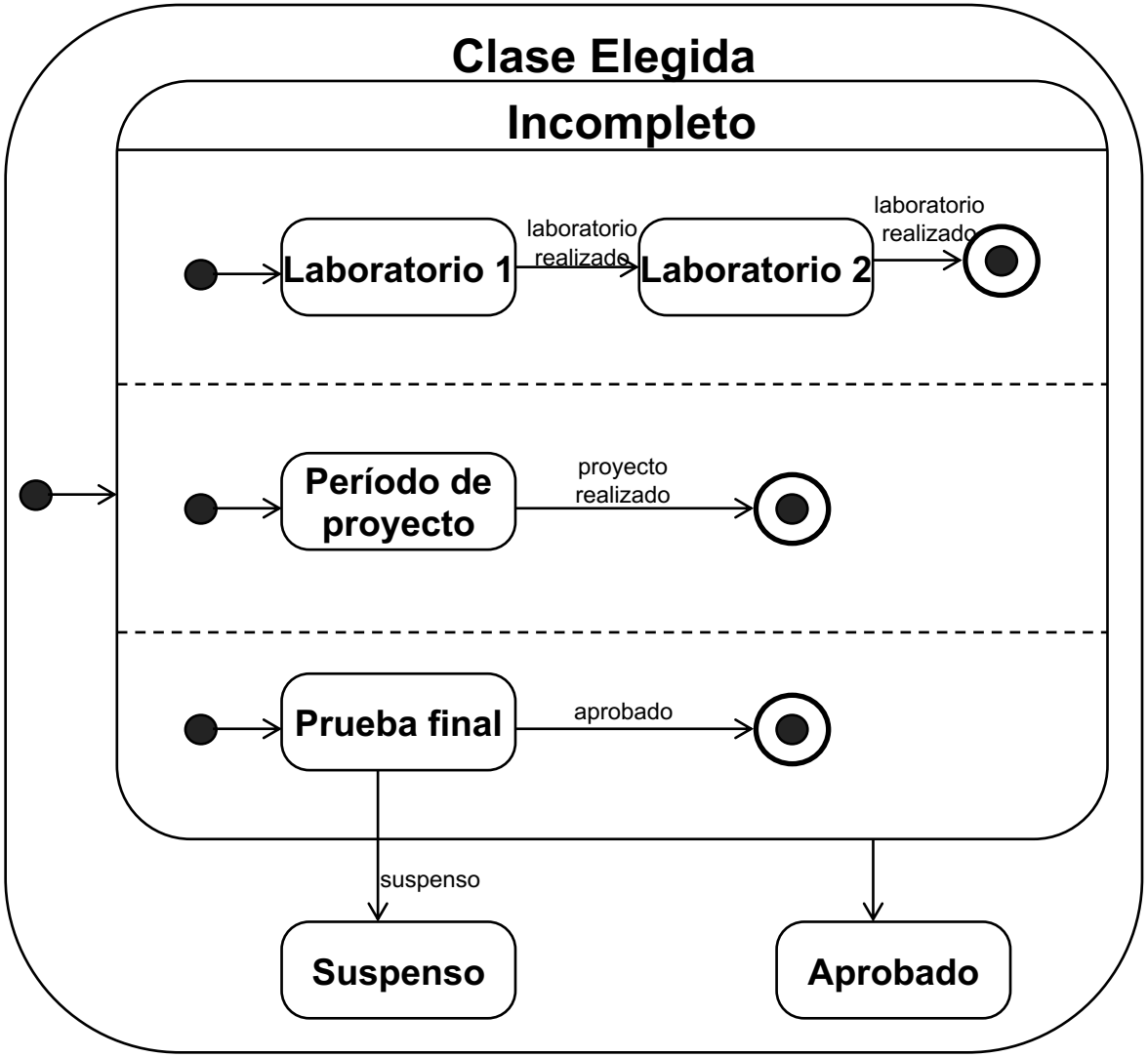
- **Estados compu**



Estado compuesto con subestados disjuntos

Estados

- **Estados compuestos (IV)**



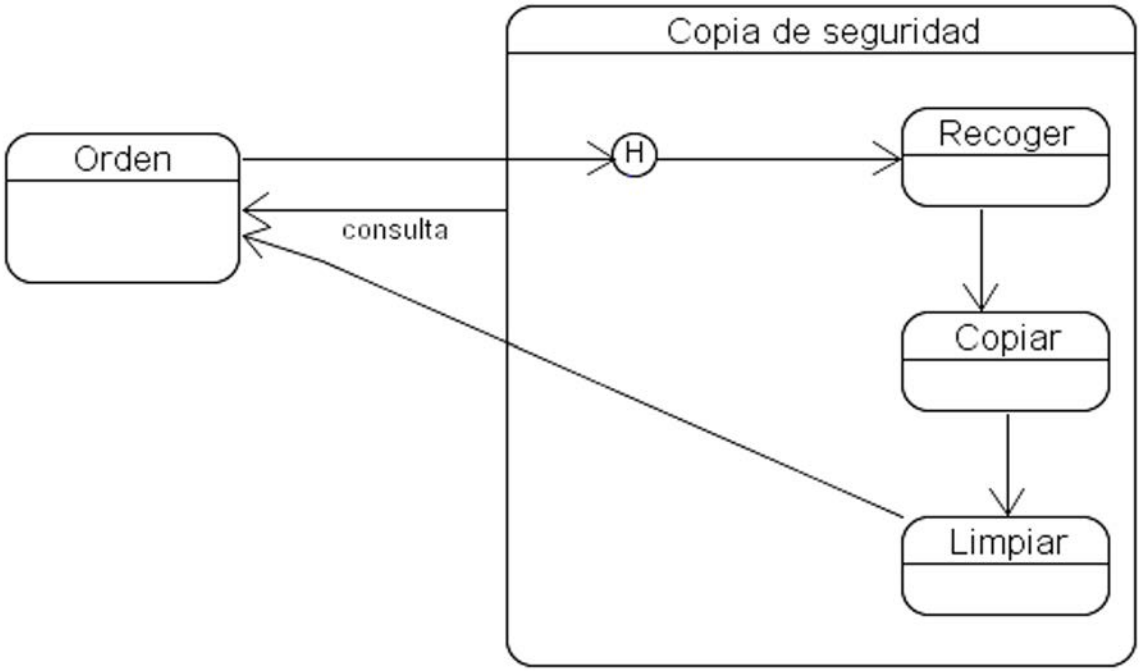
Estado compuesto con regiones ortogonales

Estados

- **Estados o indicadores de historia (I)**
  - Se utilizan para memorizar el último subestado que estaba activo antes de abandonar el estado compuesto
  - Cuando se desea que una transición active el último subestado, se representa una transición desde un origen externo al estado compuesto dirigida directamente hacia el estado de historia
  - Hay dos tipos:
    - **Superficiales**: recuerdan sólo la historia de la máquina de estados anidada inmediata. Se representan como un pequeño círculo que contiene el símbolo H
    - **Profundos**: recuerdan el estado anidado más interno a cualquier profundidad. Se representan como un pequeño círculo que contiene el símbolo H*
  - Si sólo hay un nivel de anidamiento, los estados de historia superficial y profunda son semánticamente equivalentes

Estados

- **Estados o indicadores de historia (II)**

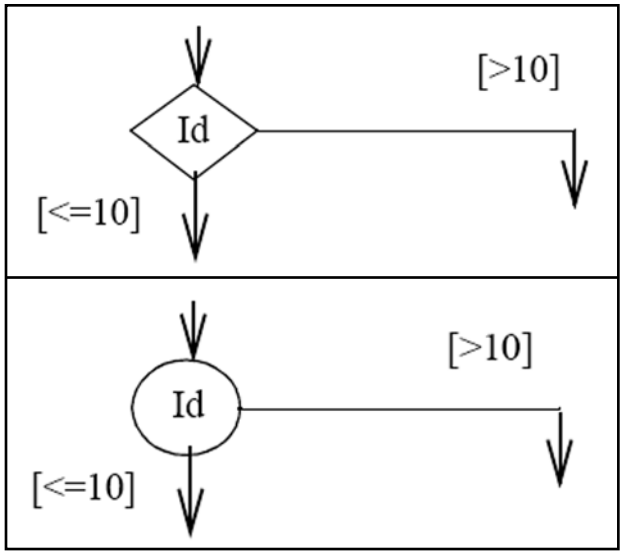


Estado compuesto con indicador de historia

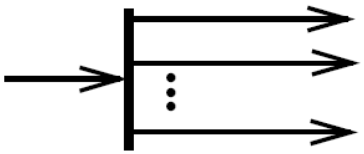
Máquinas de estado complejas

- Para modelar complejos caminos de transiciones de estado se suele recurrir a seudoestados de los siguientes tipos:
  - ***Fork***: vértices que sirven para dividir una transición en varias transiciones que terminan en vértices de diferentes regiones ortogonales
  - ***Join***: vértices utilizados para unir transiciones procedentes de diferentes regiones ortogonales
  - ***Choice***: vértice que divide una transición en múltiples caminos de salida con diferentes condiciones de guarda
  - ***Junction***: vértices utilizados para encadenar múltiples transiciones
  - ***Entry point***: seudoestado que representa una entrada a una máquina de estados
  - ***Exit point***: seudoestado que representa una salida de una máquina de estados
  - ***Terminate***: finalización de la ejecución de la máquina de estados por medio de su objeto contexto

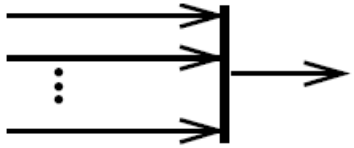
Máquinas de estado complejas



choice: notaciones alternativas



fork



join

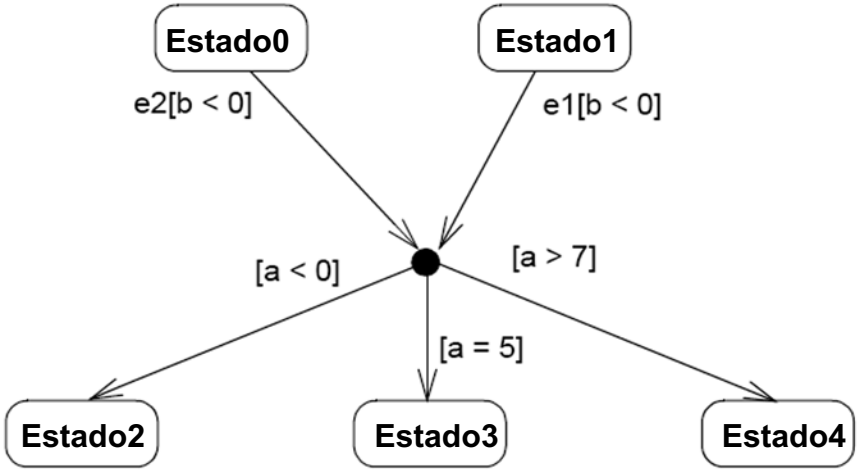
junction

entry point

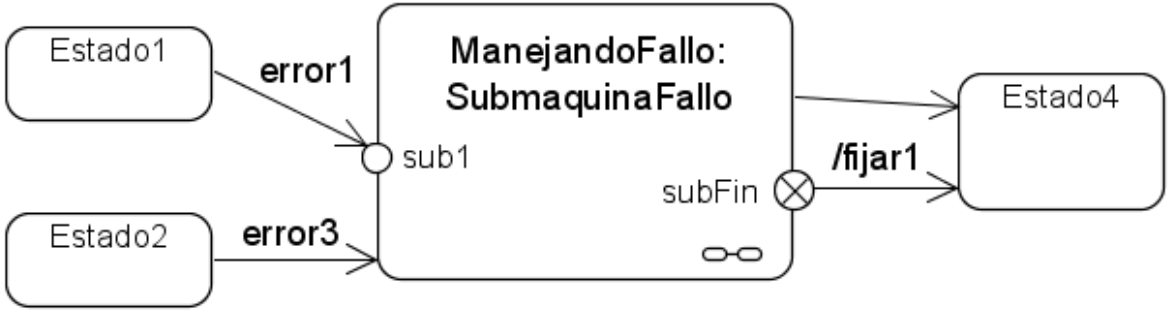
exit point

terminate

Máquinas de estado complejas

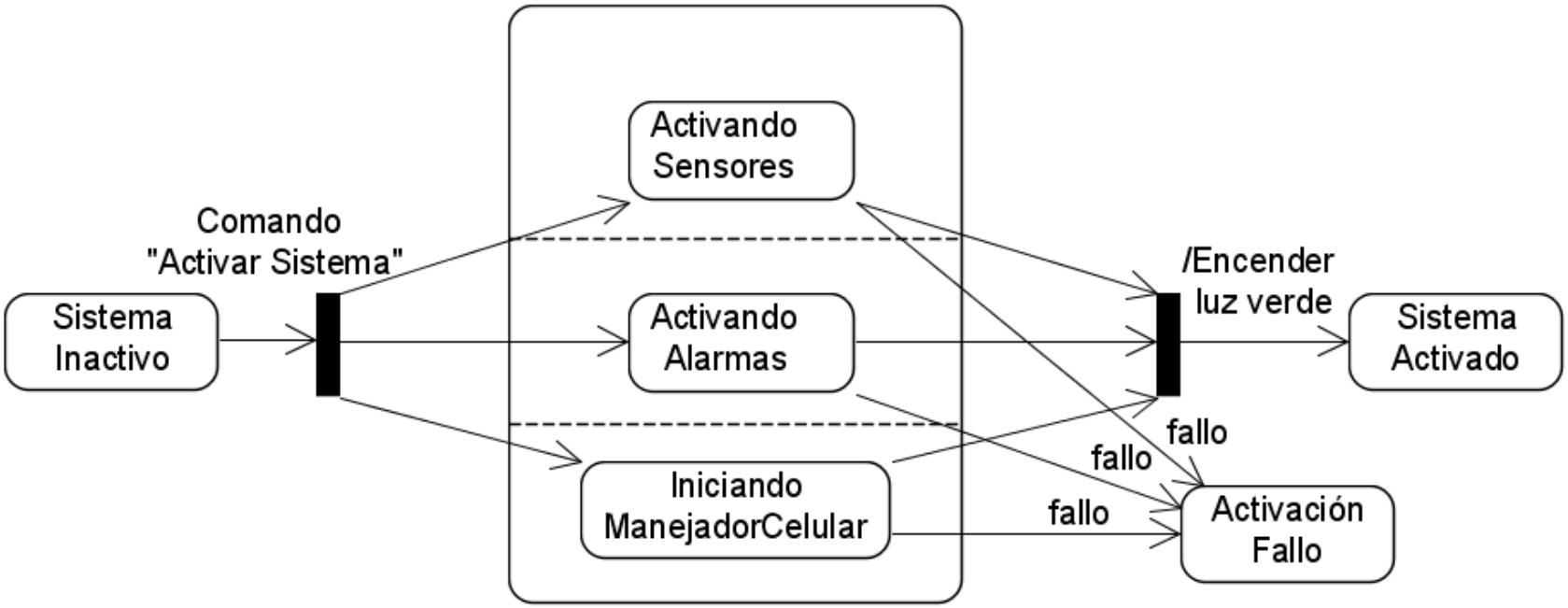


Ejemplo de seudoestado *junction*



Estado de submáquina con puntos *entry* y *exit*

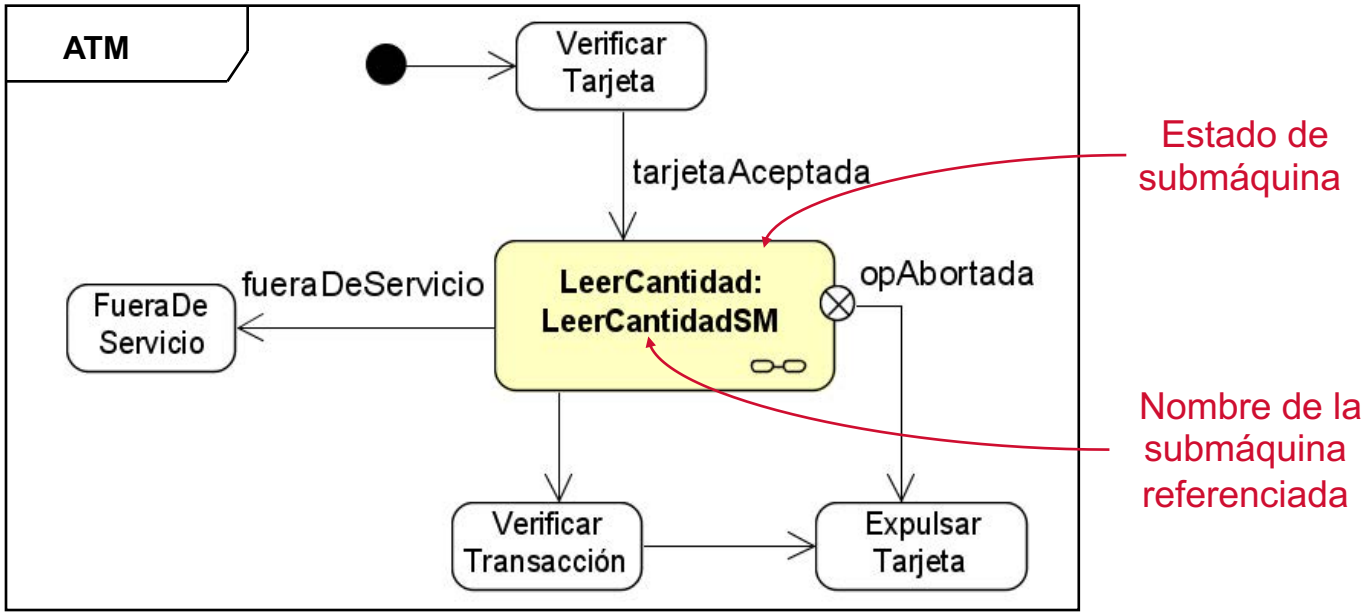
Máquinas de estado complejas



División y unión de transiciones con seudoestados *fork* y *joint*

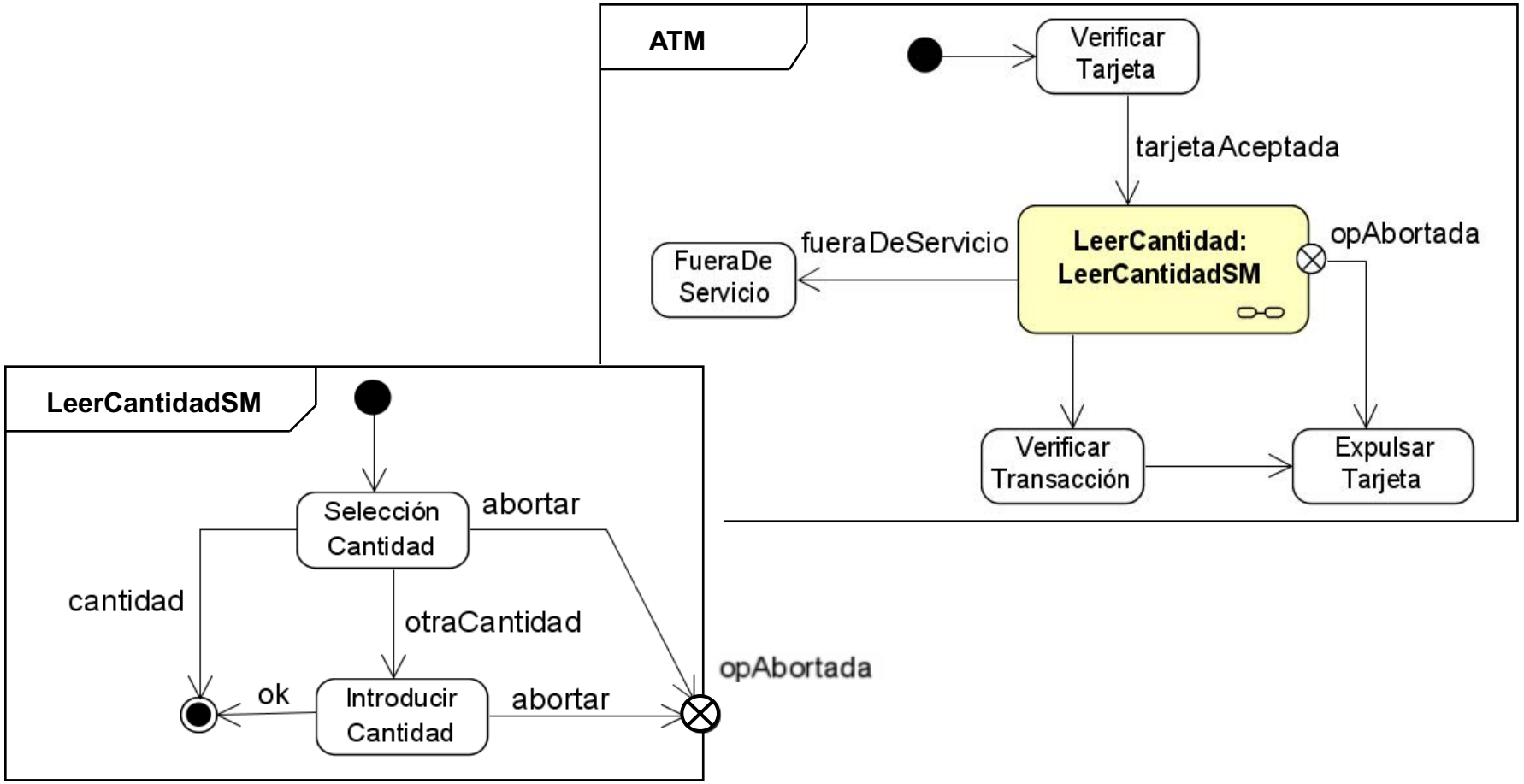
Estados de submáquina

- Mecanismo de descomposición que permite factorizar y reutilizar comportamientos comunes mediante la referencia desde una máquina de estados a otra máquina de estados
- Es semánticamente equivalente a un estado compuesto definido por la máquina de estados referenciada



Máquina de estado con el estado de submáquina "LeerCantidad"

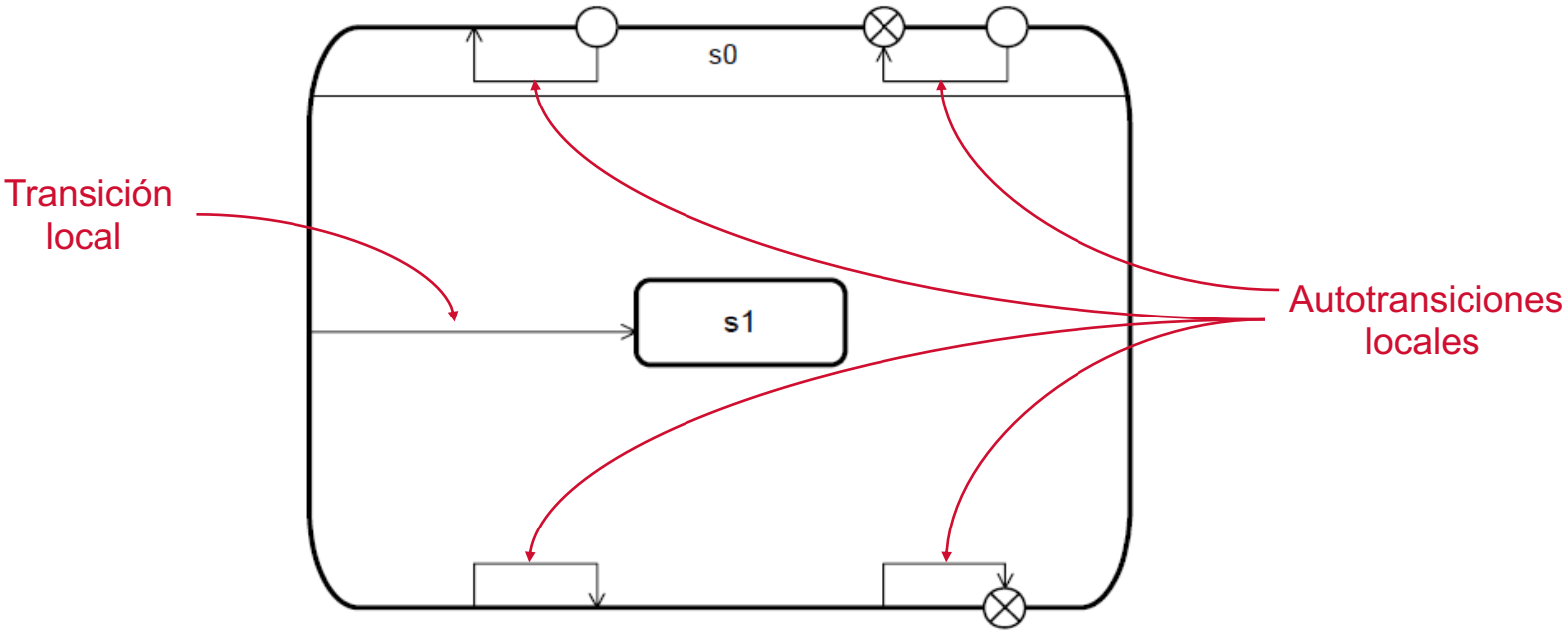
Estados de submáquina



Máquina de estados correspondiente al estado de submáquina "LeerCantidad"

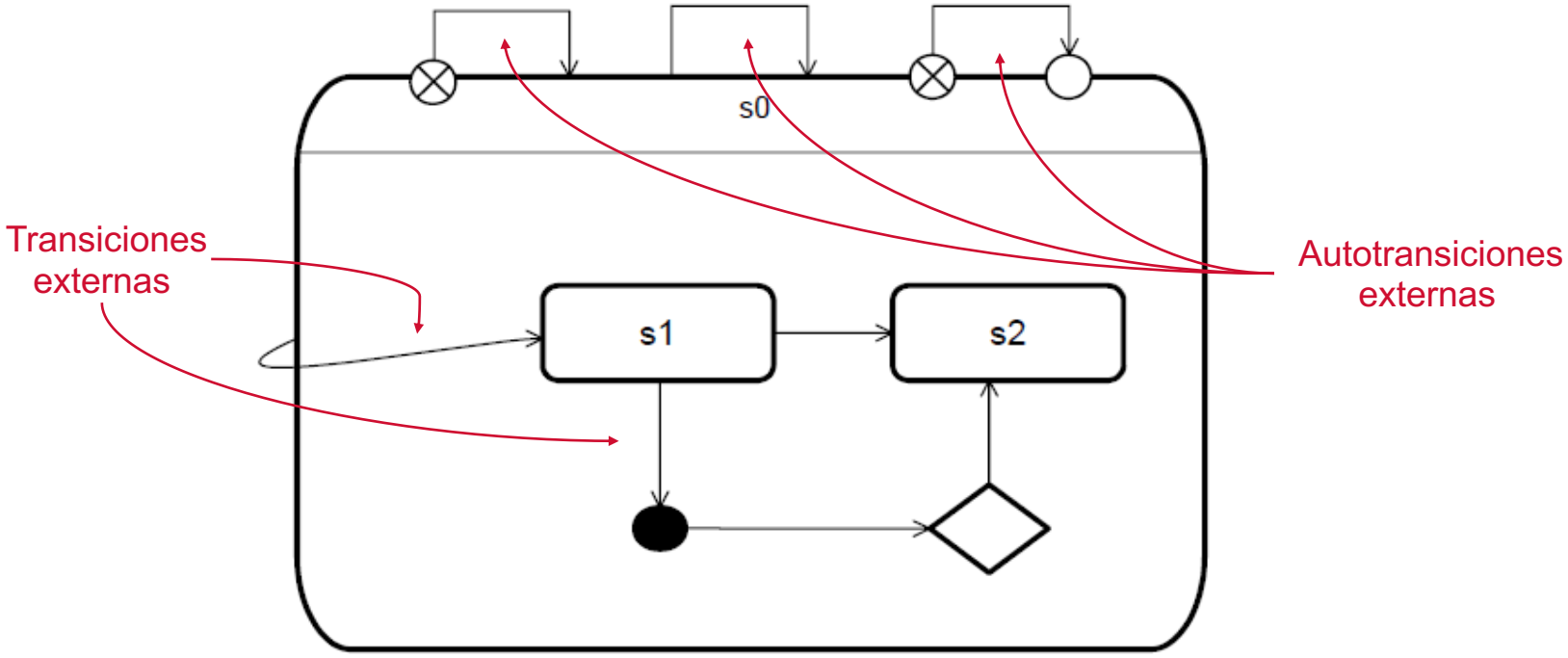
Transiciones locales

- Se encuentran en el marco de un estado compuesto, parten del borde de el estado compuesto o uno de sus puntos de entrada y terminan en uno de los nodos del estado compuesto
- En las autotransiciones locales el estado destino puede ser el estado fuente o uno de sus puntos de salida



Transiciones externas

- Tienen como destino cualquier nodo que esté dentro o fuera del estado fuente pero deben salir del estado fuente
- En las autotransiciones externas el estado fuente puede ser el propio estado fuente o uno de sus puntos de salida y el estado destino puede ser el estado fuente o uno de sus puntos de entrada

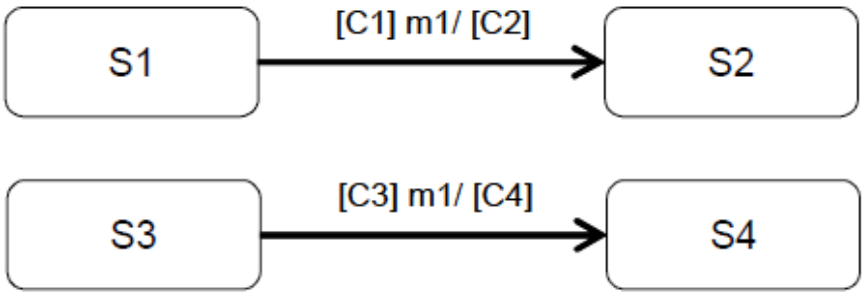


Máquinas de estados de protocolo

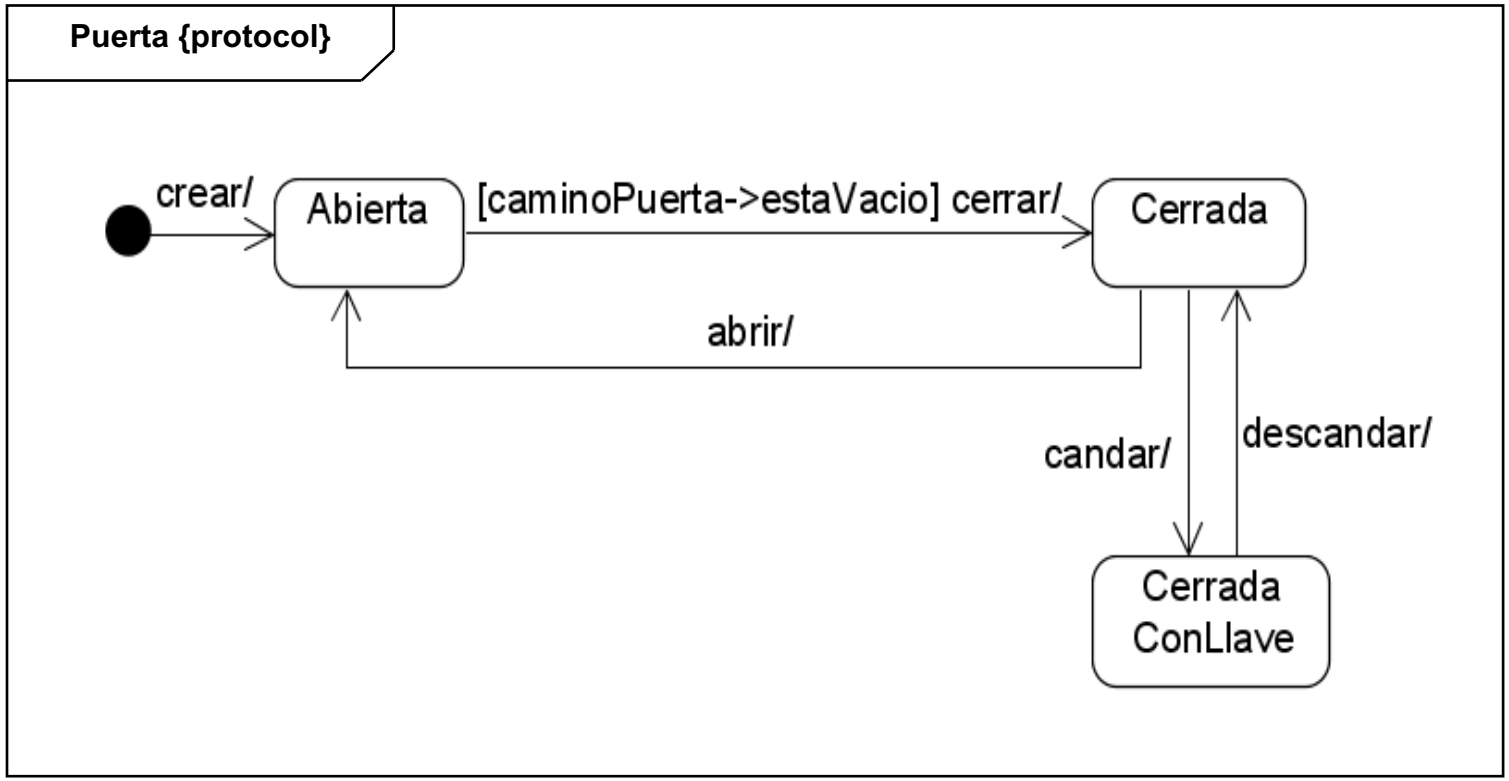
- Una máquina de estados de protocolo
  - Se define en el contexto de un clasificador
  - Ayuda a definir el modo de uso las operaciones y entradas de un clasificador especificando
    - En qué estado y bajo qué precondiciones pueden usarse
    - Si hay un orden entre ellas
    - Qué resultados se esperan de su uso
  - Constituyen un medio para formalizar las interfaces de clases expresando reglas de consistencia para la implementación o para el comportamiento dinámico
  - Se usan para especificar las transformaciones legales que pueden ocurrir en un clasificador abstracto como una interfaz o un puerto
    - Si una sucesión de eventos lleva a un camino válido a través de la máquina de estados esa sucesión es legal
    - Si una sucesión de eventos no lleva a un camino válido no puede ocurrir

Máquinas de estados de protocolo

- Transiciones de las máquinas de estados de protocolo
  - Se refieren a operaciones
  - No tienen efectos
  - Pueden tener una precondición que tiene que ser verdad si el evento ocurre
  - Pueden tener una postcondición que indica el estado del objeto después de que la transición esté completa. Indican restricciones en los resultados de la operación
- **Notación**
  - Marco con la restricción {protocol} detrás del nombre
  - Transición: `[´precondición`]´evento/`[´postcondición`]´

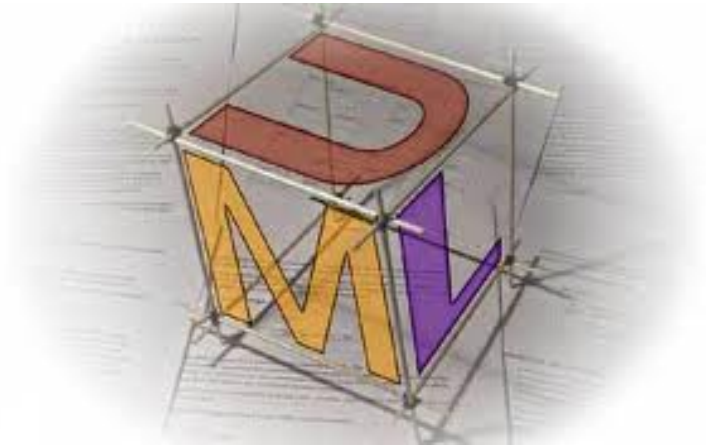


Máquinas de estados de protocolo



Máquina de estados de protocolo

Vista de diseño



Características

- La vista de diseño modela la estructura de diseño de la aplicación
  - Expansión en clasificadores estructurados
  - Colaboraciones que proporcionan la funcionalidad
  - Ensamblado a partir de componentes con interfaces bien definidas
- Diagramas
  - Estructuras compuestas
  - Colaboración
  - Componentes

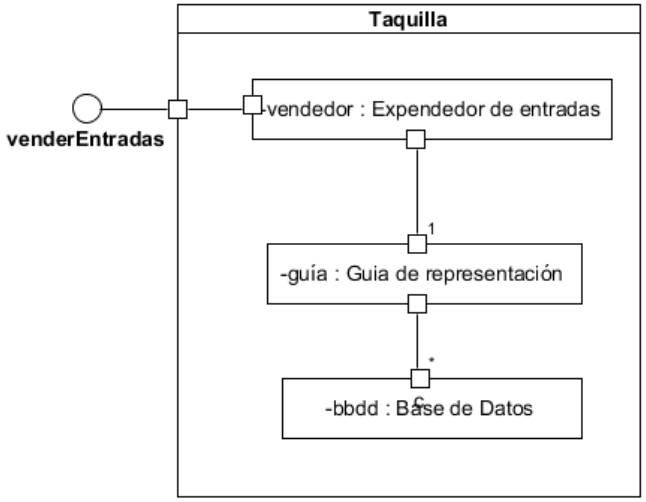


Diagrama de estructuras compuestas

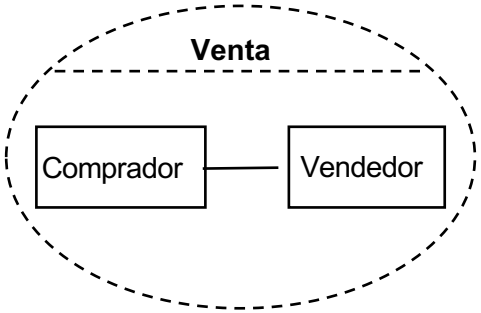


Diagrama de colaboración

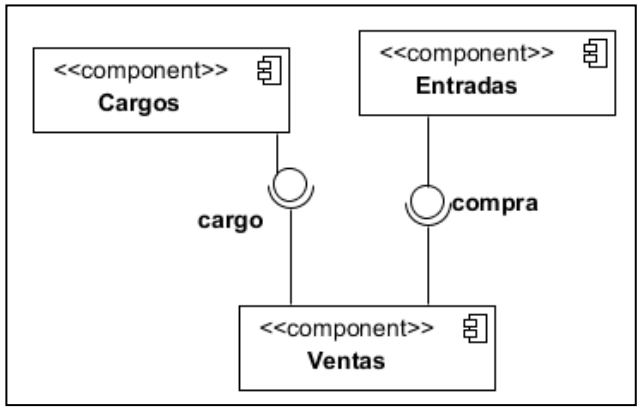
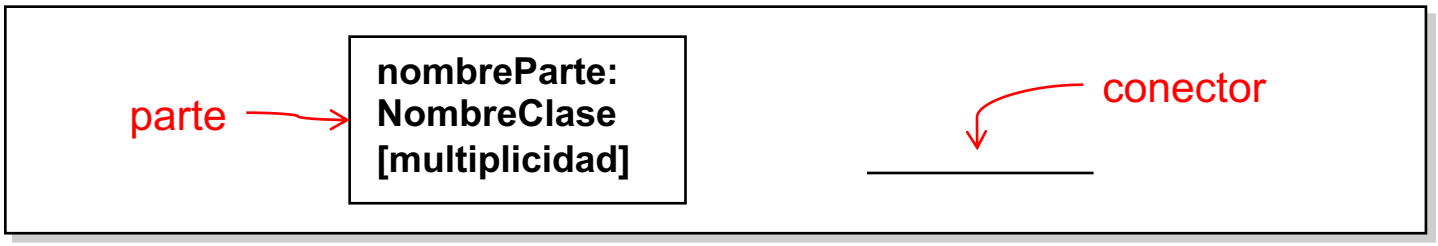


Diagrama de componentes

Estructuras compuestas

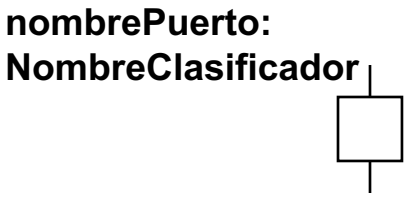
- **Estructura**: Composición de elementos interconectados que representan instancias que colaboran en tiempo de ejecución a través de enlaces de comunicación para alcanzar objetivos comunes
- **Estructura interna**: mecanismo para especificar estructuras de elementos interconectados que se crean dentro de una instancia de un clasificador
- **Diagrama de estructuras compuestas**: representación de la estructura interna de un clasificador. Describe las conexiones entre elementos que trabajan juntos en un clasificador (caso de uso, objeto, colaboración, clase o actividad...). Están formados por:
  - **Partes**: roles de los clasificadores
  - **Conectores**: enlaces entre instancias de las partes conectadas.
    - Sus extremos pueden tener los mismos adornos que los extremos de una asociación
    - El extremo de un conector puede ser un **puerto** (mecanismo utilizado para aislar un clasificador de su entorno)



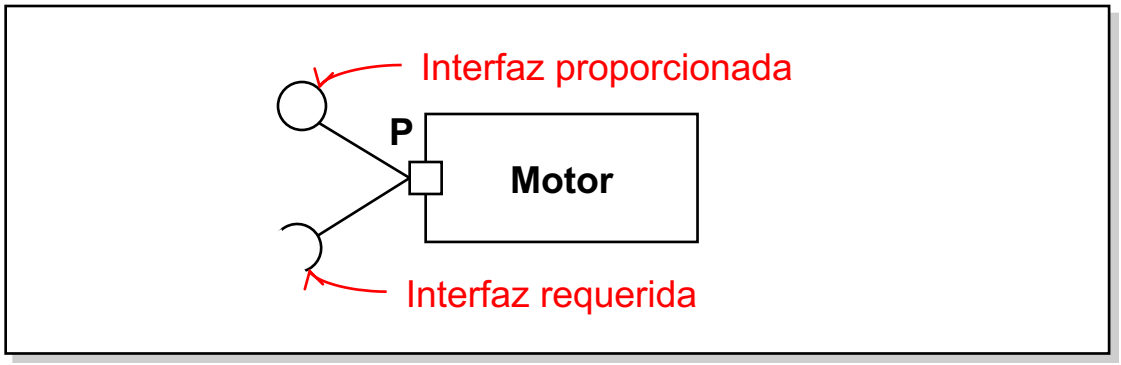
Notación de partes y conectores de un diagrama de estructuras compuestas

Estructuras compuestas

- **Puerto**: punto de interacción entre un clasificador y su entorno o entre el clasificador y sus partes internas
  - Pueden especificar los servicios que el clasificador proporciona y los servicios que requiere el clasificador
  - Las interfaces asociadas a un puerto especifican la naturaleza de las interacciones que ocurren a través de él
  - Se representan con el símbolo de un pequeño cuadrado solapándose con el límite del símbolo del clasificador (puerto público) o dentro del símbolo del clasificador (puerto protegido, por defecto, o visibilidad especificada). Si el puerto está en el extremo de un conector se coloca cerca de su símbolo el nombre del puerto seguido del nombre del clasificador (opcional), separados por ":"



Estructuras compuestas



Representación de puertos e interfaces

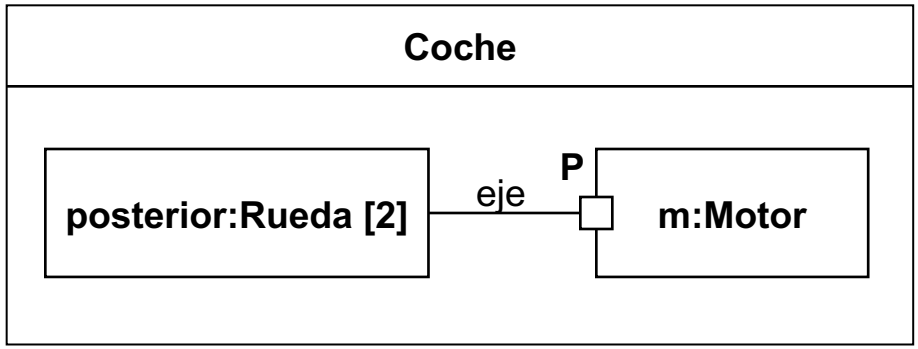


Diagrama de estructuras compuestas

Diagramas de colaboración

- Una colaboración representa un conjunto de instancias que cooperan para realizar una tarea o conjunto de tareas
- Componentes
  - Roles: papel que desempeñan los participantes en la tarea (propiedades de un tipo determinado)
  - Conectores: relaciones relevantes para la tarea entre los roles
- **Notación**
  - Elipse con la línea discontinua que contiene el nombre de la colaboración y un compartimento para la representación de su estructura interna mediante roles y conectores
  - Notación del rol: nombre: Tipo [multiplicidad]

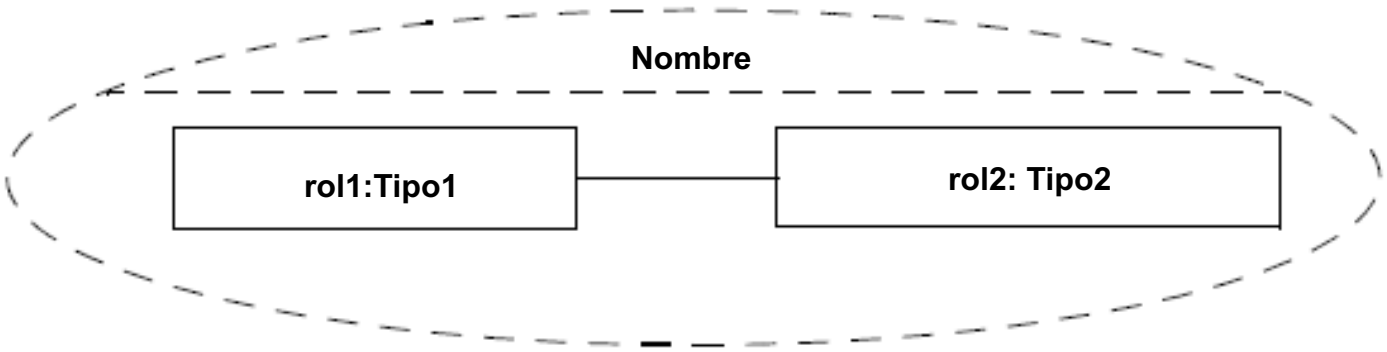
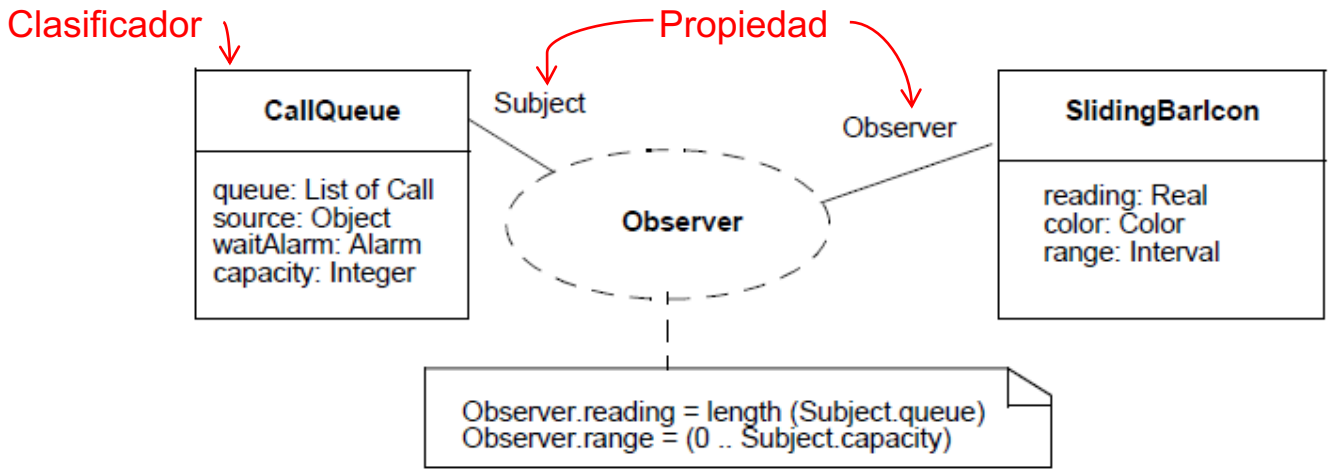


Diagrama de Colaboración

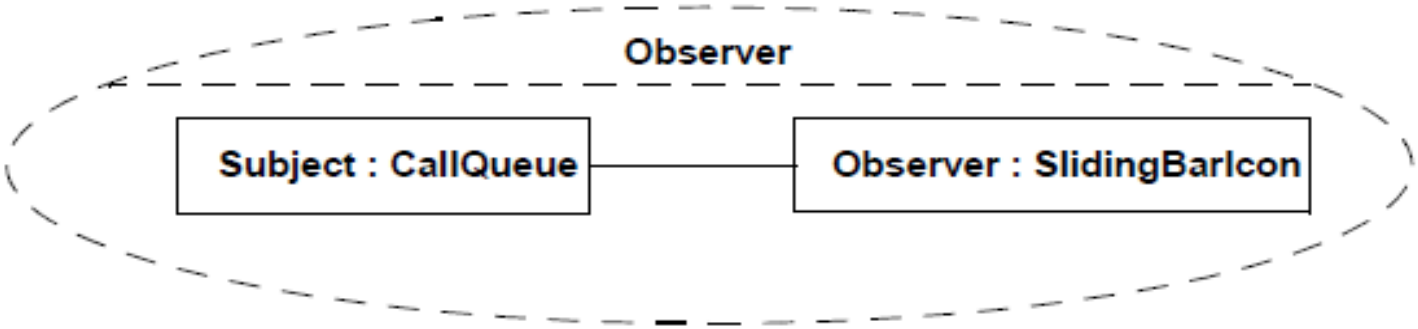
Diagramas de colaboración

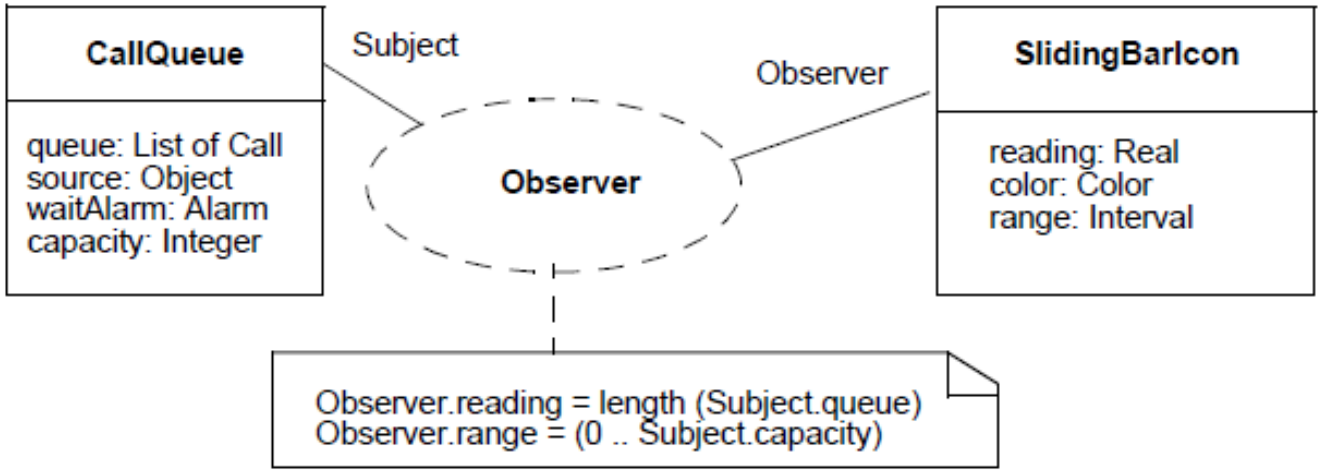
- **Uso de la colaboración**
  - Representa la aplicación de un patrón descrito por una colaboración para una situación específica que involucra clases o instancias específicas que juegan los roles de la colaboración
  - Se utiliza una notación alternativa a la de la colaboración para representar a los clasificadores que son tipos de propiedades de la colaboración, uniéndolos a ella mediante una línea etiquetada con el nombre de la propiedad (rol)



Uso de una colaboración

Diagramas de colaboración

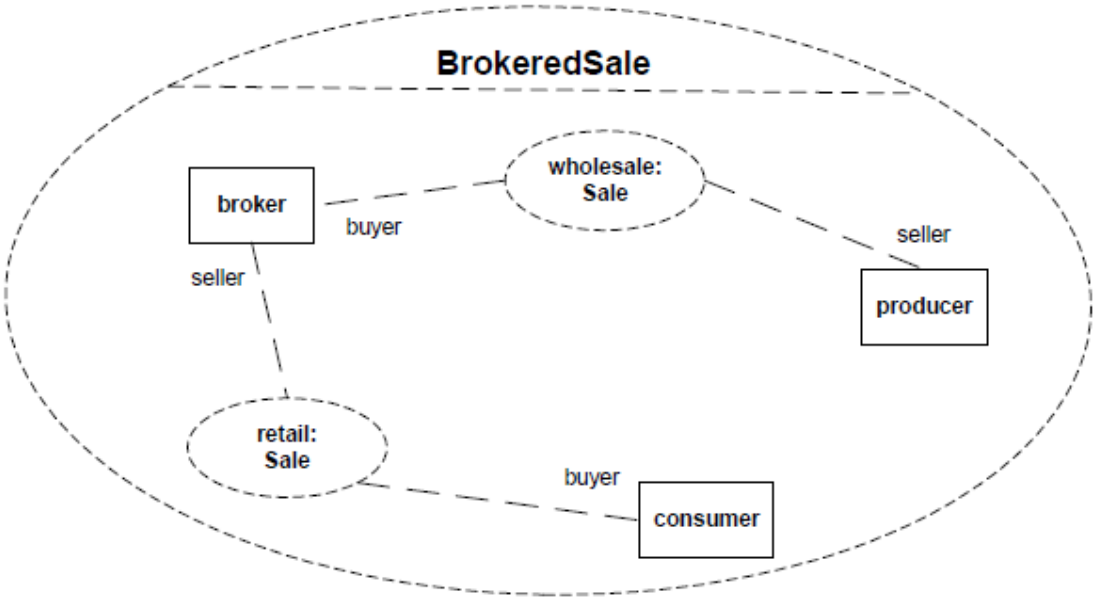


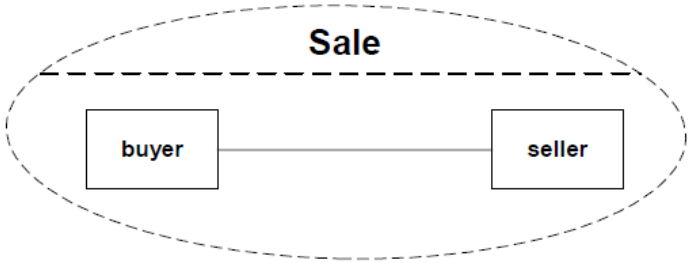


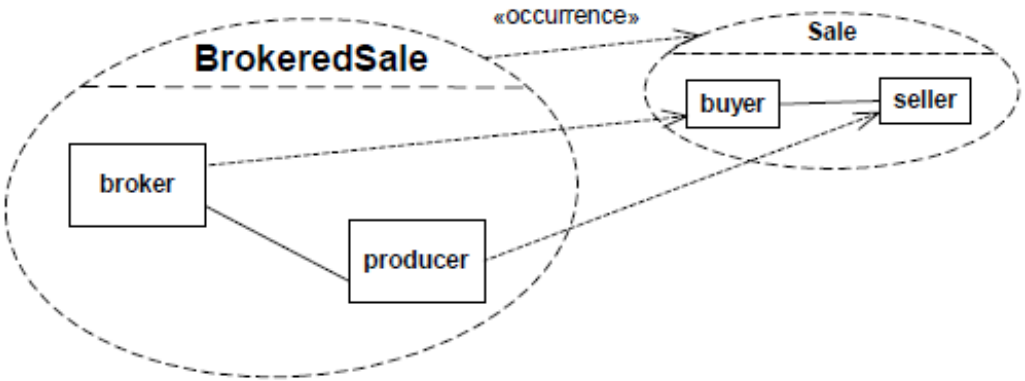
Representación de una colaboración y un uso de esa colaboración

Diagramas de colaboración

- Descripción de una colaboración en función de otras colaboraciones

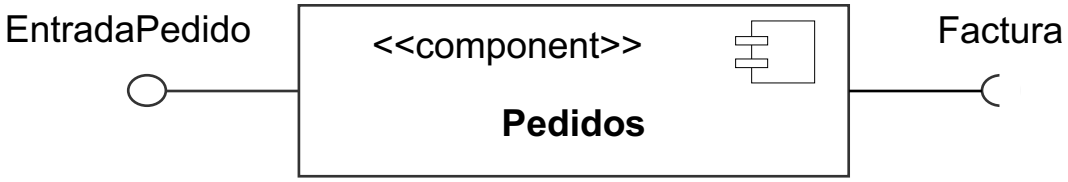






Diagramas de componentes

- **Componente**: unidad autocontenida que encapsula el estado y el comportamiento de varios clasificadores
  - Especifica un contrato de los servicios que proporciona a sus clientes y de los servicios que requiere de otros componentes
  - Es una unidad que puede sustituirse en tiempo de diseño o de ejecución por otro componente que ofrezca funcionalidad equivalente basada en la compatibilidad de sus interfaces
  - Las interfaces proporcionadas y requeridas por un componente pueden organizarse en puertos
  - Se implementan normalmente como ficheros
    - **Artefactos fuente**: ficheros de código fuente que contienen una o más clases
    - **Artefactos ejecutables**: ficheros ejecutables de un programa
  - Notación: rectángulo de clasificador con el estereotipo <<component>> y opcionalmente el símbolo de componente en la esquina superior derecha



Componente con una interfaz requerida y otra proporcionada

Diagramas de componentes

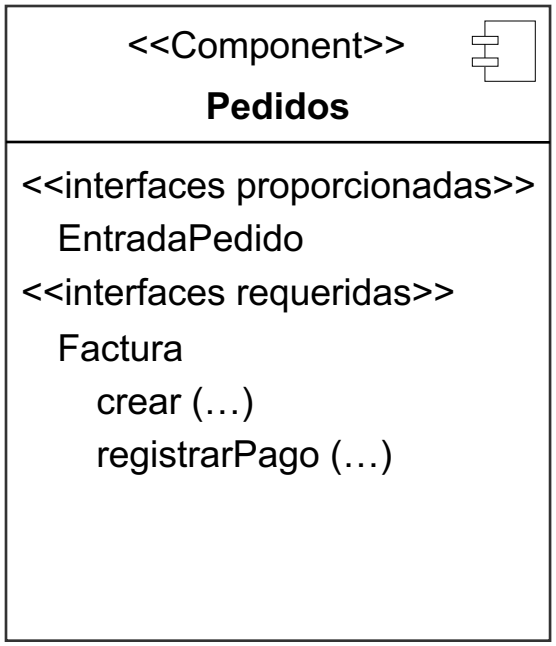
- **Vistas de un componente (I)**
  - **Externa** (caja negra). dos opciones de representación
    - Símbolos de las interfaces fuera de la caja del componente
    - Interfaces, operaciones y atributos en los compartimentos de la caja del componente
  - **Interna** (caja blanca): se listan los clasificadores que realizan el componente en un compartimento adicional. Se pueden incluir compartimentos para listar partes, conectores y artefactos de implementación

    Los clasificadores internos que realizan un componente se pueden mostrar mediante:
    - Dependencias generales
    - Anidados dentro del componente

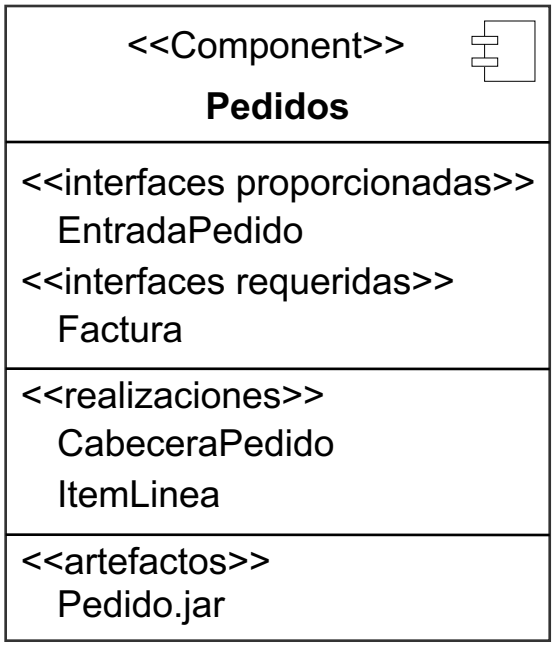
    Se puede representar la estructura interna de partes y conectores a nivel de instancias (estructuras compuestas)

Diagramas de componentes

- **Vistas de un componente (II)**

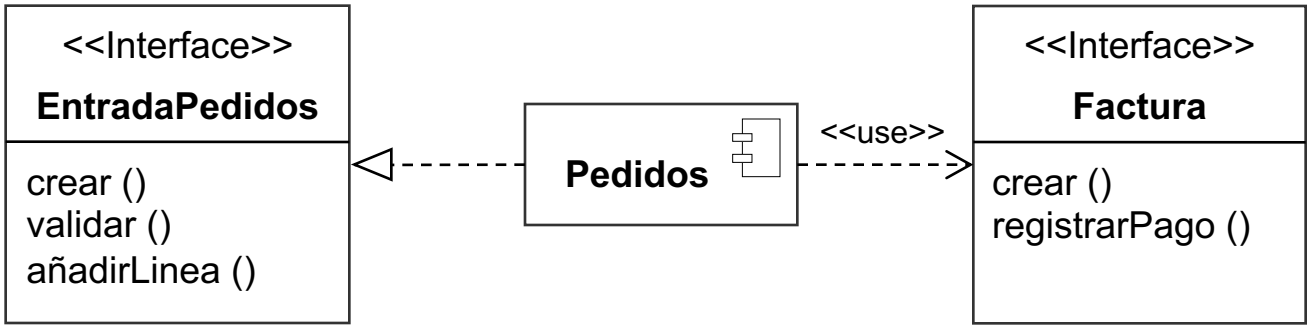


Vista externa de un componente

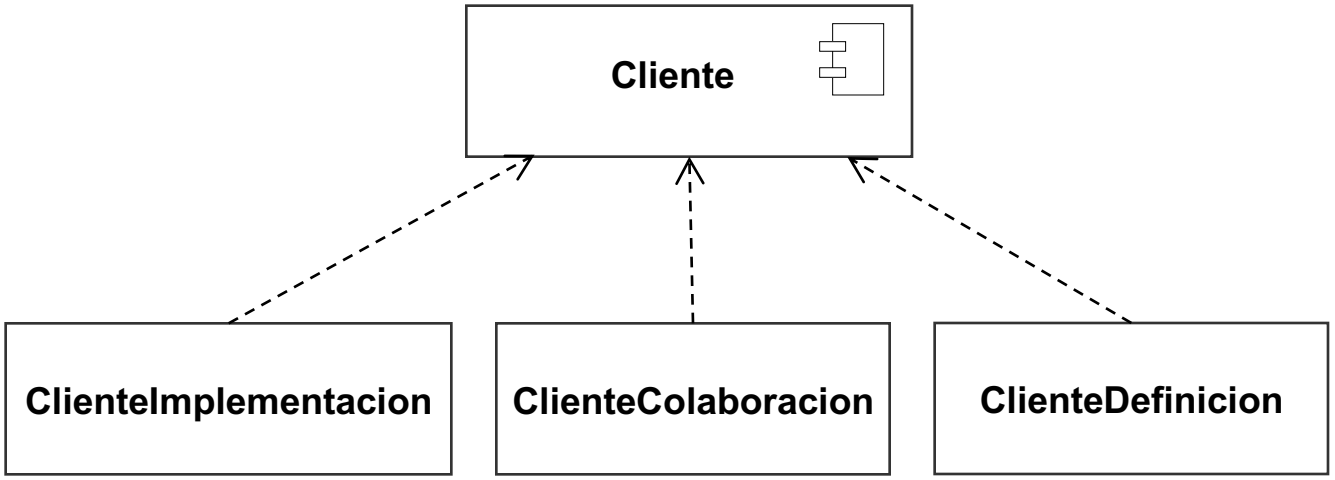


Vista interna de un componente

Diagramas de componentes

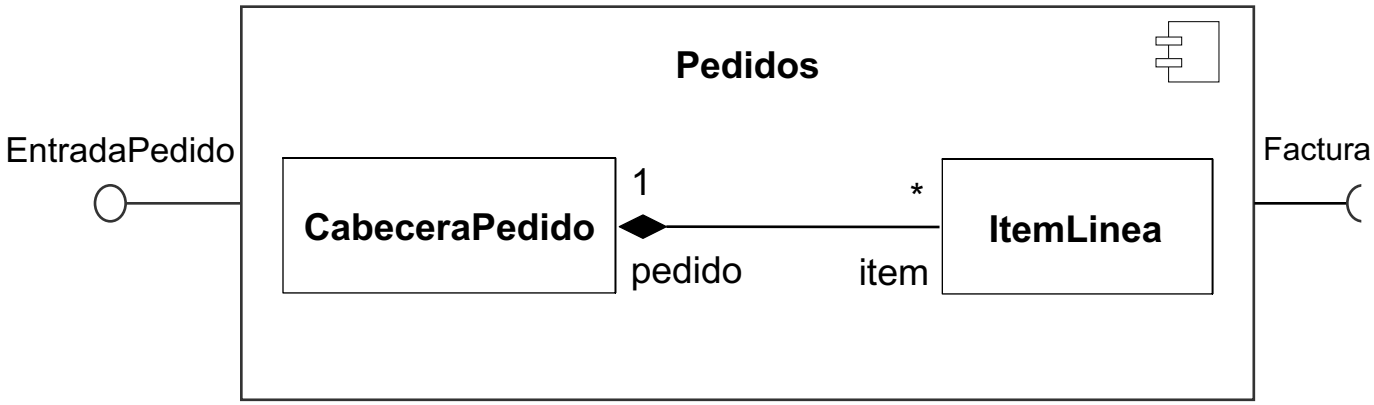


Representación explícita de las interfaces proporcionadas y requeridas por un componente



Representación de la realización de un componente mediante dependencias

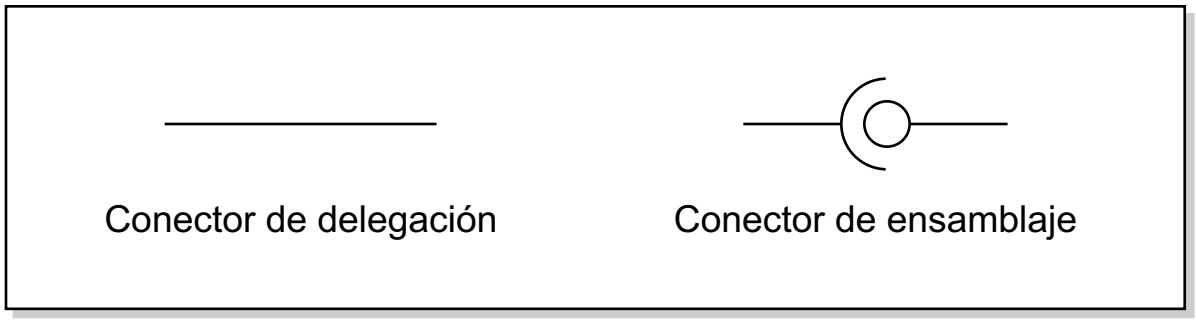
Diagramas de componentes



Representación anidada de los clasificadores que realizan un componente

Diagramas de componentes

- **Conectores (I)**
  - De **delegación**: enlaza el contrato externo (puerto) de un componente con su realización interna (partes)
  - De **ensamblaje**: enlaza dos o más partes o puertos de manera que unas partes proporcionan los servicios que usan otras partes



Diagramas de componentes

- **Conectores (II)**

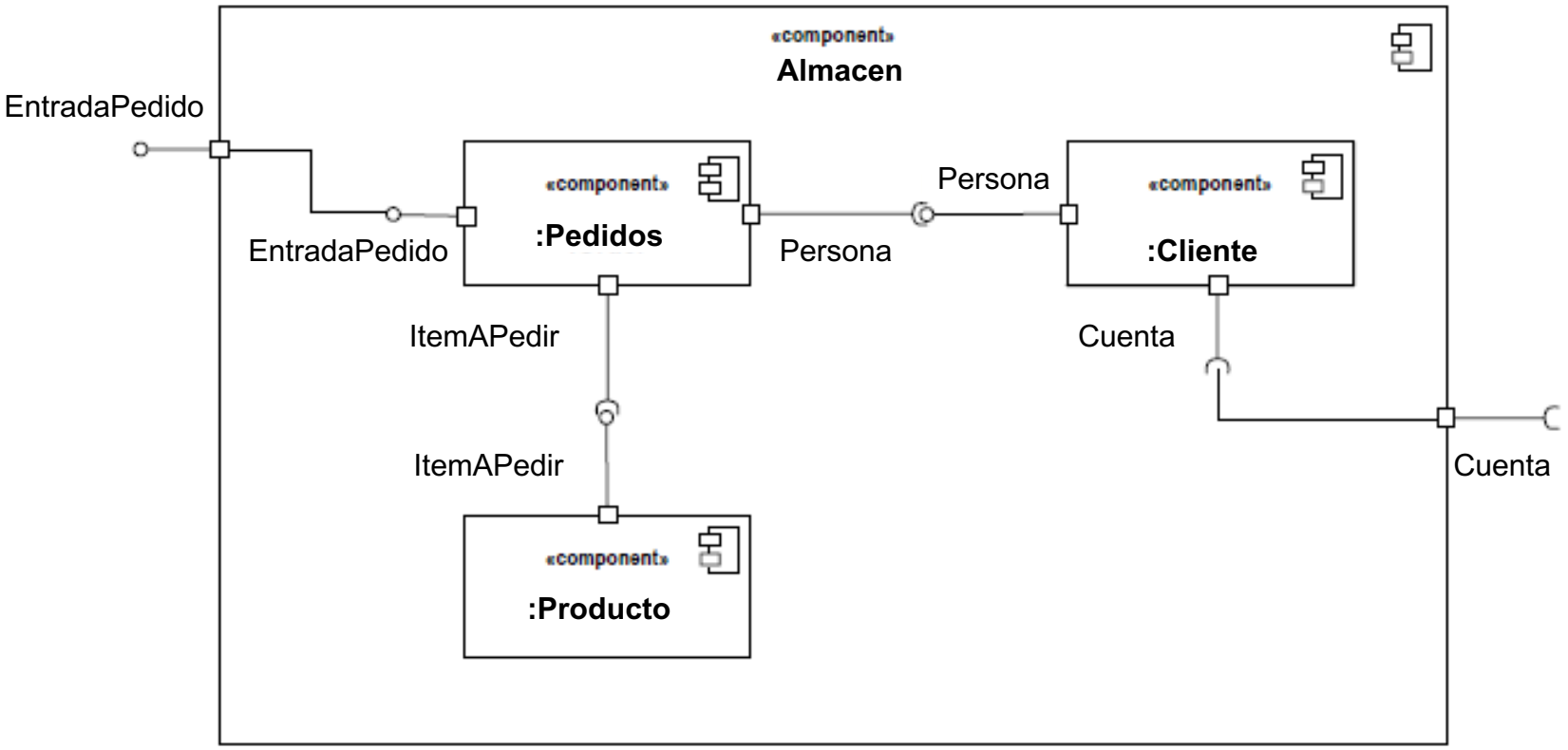
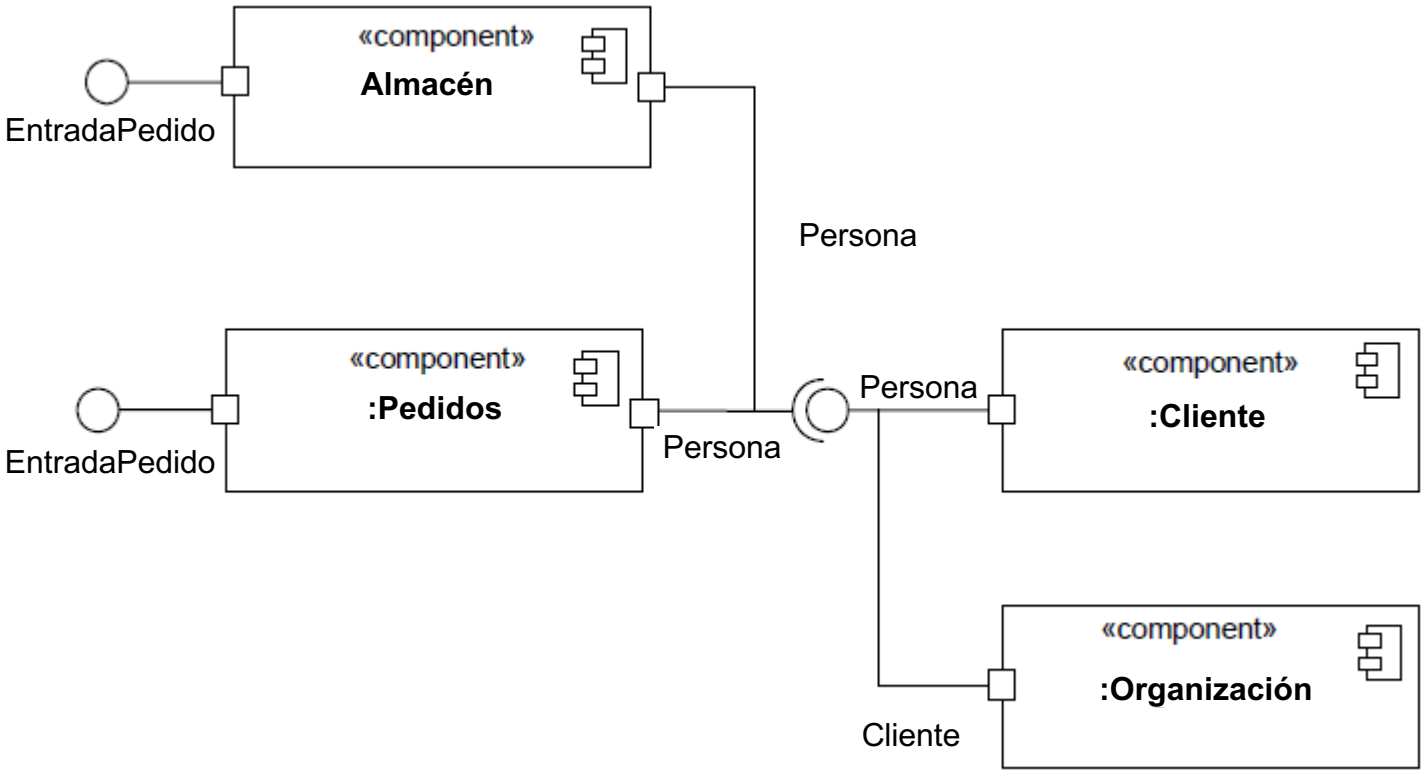


Diagrama de estructura compuesta: vista interna de la estructura de un componente que contiene otros componentes como partes

Diagramas de componentes

- **Conectores (III)**



Conexión de componentes con puertos que proporcionan o requieren la misma interfaz

Diagramas de componentes

- Los componentes que se conectan se pueden representar en diagramas de estructura que muestran los clasificadores y las dependencias entre ellos

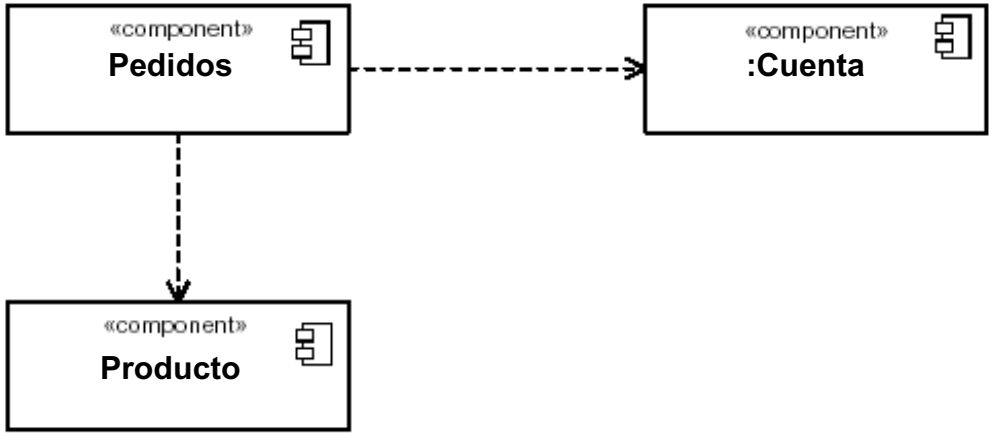


Diagrama de componentes y sus dependencias generales

Diagramas de componentes

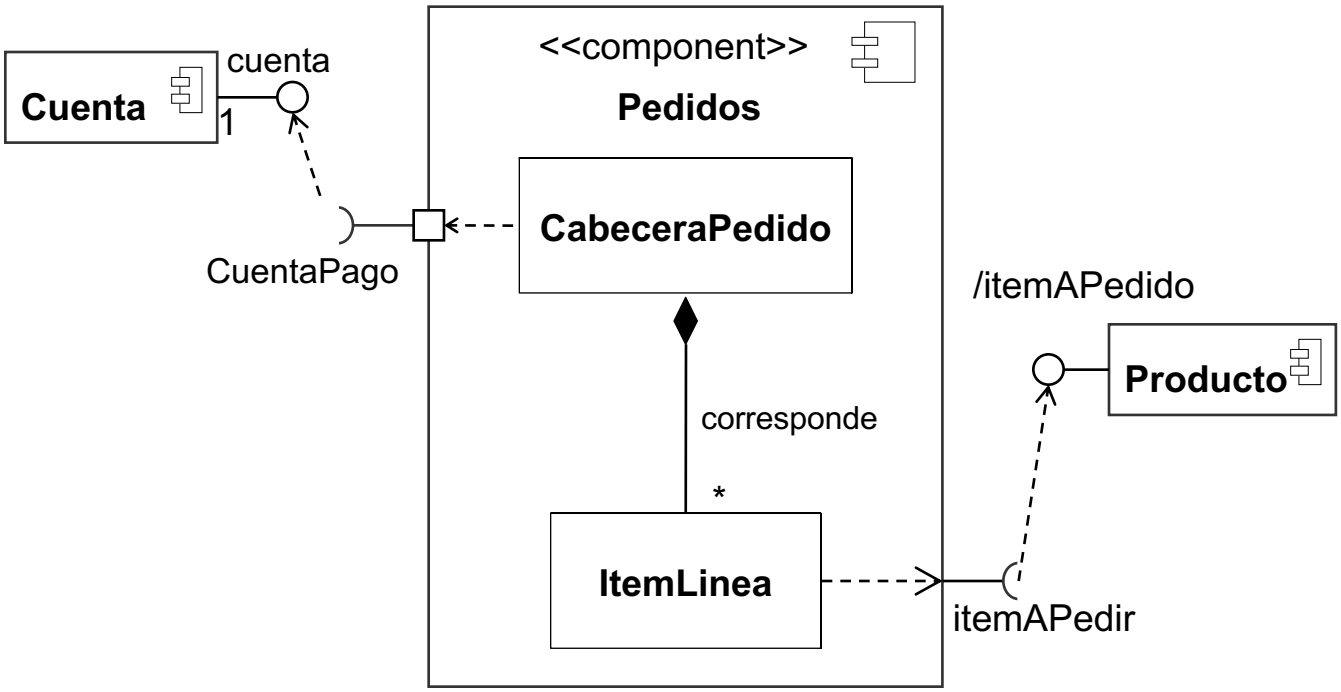
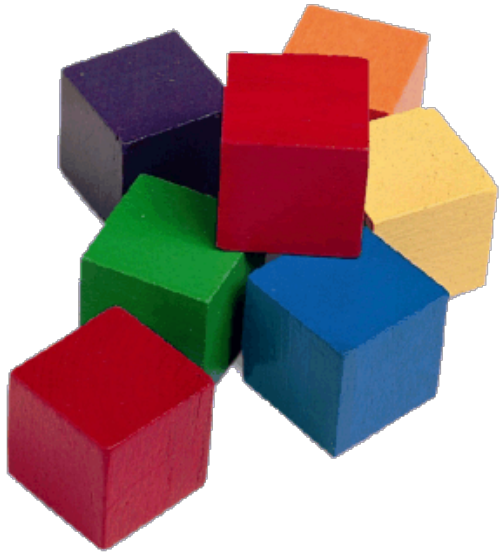


Diagrama de estructura de componentes con interfaces

Vista de despliegue



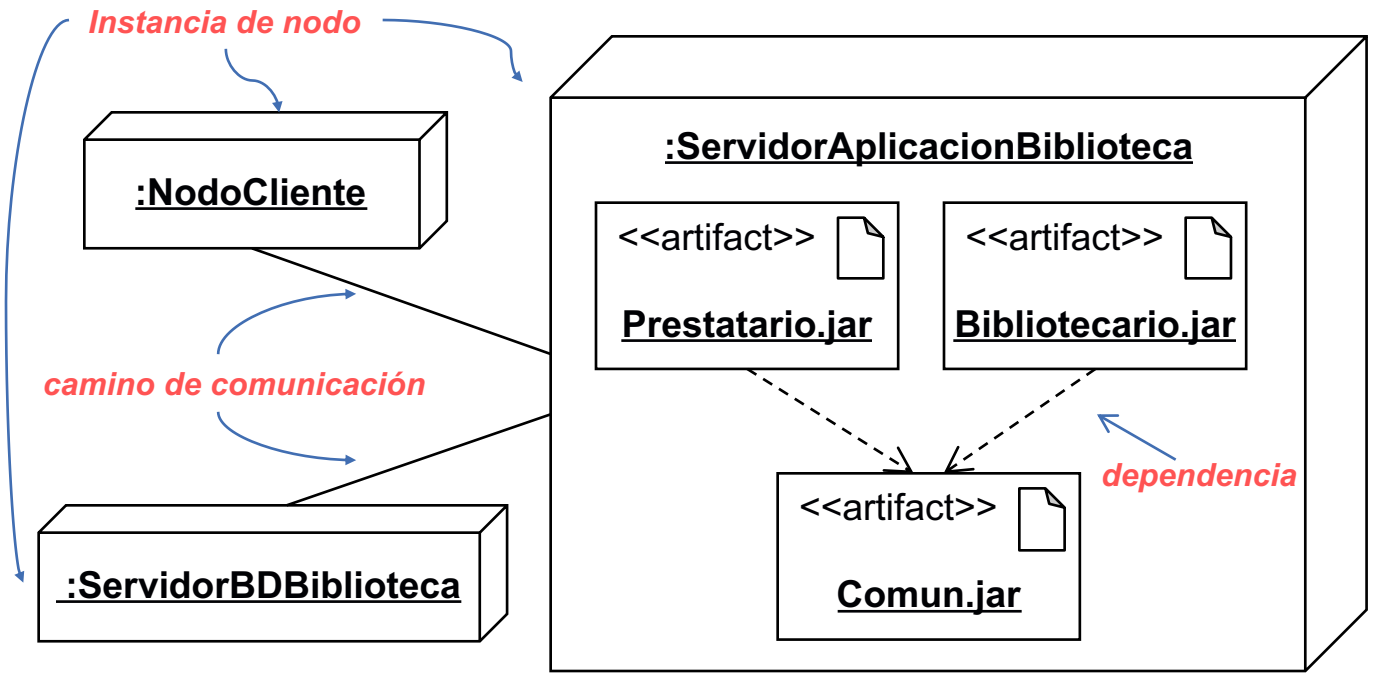
Características

- La vista de despliegue representa el despliegue de artefactos de tiempo de ejecución sobre nodos
- Los diagramas de despliegue, junto con los diagramas de componentes, forman parte de la arquitectura física
- La **arquitectura física** es una descripción detallada del sistema que muestra la asignación de artefactos de *software* a nodos físicos
  - **Estructura física del software**: distribución y dependencias, en tiempo de ejecución, de los artefactos software definidos en la arquitectura lógica
    - **Componentes**: empaquetamiento físico de una colección de elementos de modelado
    - **Artefactos**: porción física de información que usa o produce el proceso de desarrollo de software (manifestaciones físicas de los componentes)
    - **Especificadores de despliegue**: conjunto de propiedades que determinan los parámetros de ejecución de un componente desplegado en un nodo
  - **Estructura del hardware**: nodos y forma de conexión entre ellos
    - **Dispositivos**: recursos computacionales (nodos)
    - **Caminos de comunicación**: mecanismos de comunicación entre nodos
    - **Entornos de ejecución**: subnodos de los dispositivos

Diagramas de despliegue

- Los diagramas de despliegue describen la topología física del sistema: la estructura de las unidades *hardware* y el *software* que se ejecuta en cada unidad
- Elementos
  - **Artefacto de despliegue**: entidad física concreta cuyas instancias se despliegan en instancias de nodos
  - **Nodo**: Recurso computacional sobre el que se pueden desplegar los artefactos para su ejecución
  - **Entorno de ejecución**: nodo que ofrece un entorno de ejecución para tipos específicos de componentes que se despliegan sobre él en forma de artefactos ejecutables
  - **Camino de comunicación**: asociación entre nodos que indica que existe algún tipo de comunicación entre ellos, que intercambian objetos y envían mensajes a través del camino

Diagramas de despliegue

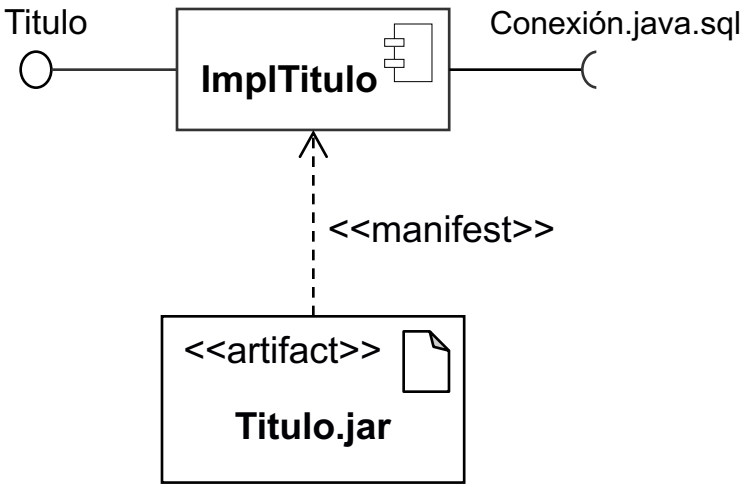


Diagramas de despliegue

- **Artefactos de despliegue (I)**
  - Clasificador que representa una entidad física: fichero de modelos, fichero fuente, ejecutable, tabla de BBDD…
  - Representan la manifestación física de un componente o de cualquier elemento empaquetable. La manifestación es una relación de dependencia con el estereotipo <<manifest>>
  - Pueden tener propiedades que representan sus características y las operaciones que se pueden realizar sobre sus instancias
  - Participan en asociaciones con otros artefactos
  - Pueden instanciarse (artefacto desplegado): una instancia particular o copia se despliega en una instancia de un nodo
  - Notación: rectángulo con el estereotipo <<artifact>> y alternativamente un icono. El nombre subrayado de la instancia del artefacto puede ser omitido
  - Estereotipos predefinidos para perfiles de artefactos
    <<source>> <<executable>> <<document>> <<library>>

Diagramas de despliegue

- **Artefactos de despliegue (II)**



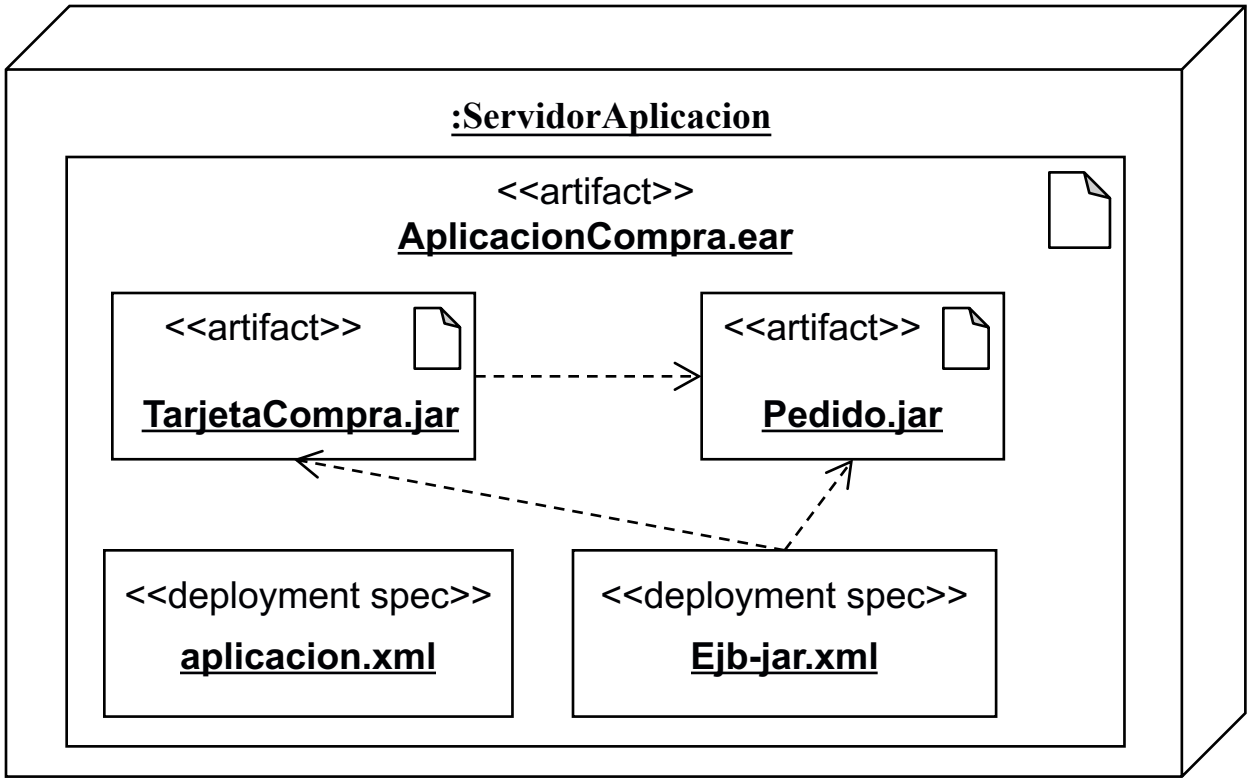
Diagramas de despliegue

- **Artefactos desplegados (I)**
  - Un artefacto desplegado es una instancia de un artefacto que se ha desplegado en un nodo
  - Puede representarse con un conjunto de propiedades que describen los parámetros de ejecución del artefacto en un nodo particular. Dos opciones de modelado:
    - Directamente dentro del artefacto desplegado
    - Como una **especificación de despliegue**:

      Especificación separada que se representa como un clasificador con el estereotipo <<deployment spec>>
      - Pueden existir dentro de un artefacto
      - Puede estar conectada a un artefacto con una asociación unidireccional
  - Se pueden modelar relaciones de dependencia entre los artefactos desplegados

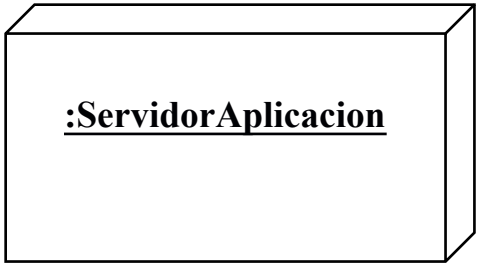
Diagramas de despliegue

- **Artefactos desplegados (II)**



Diagramas de despliegue

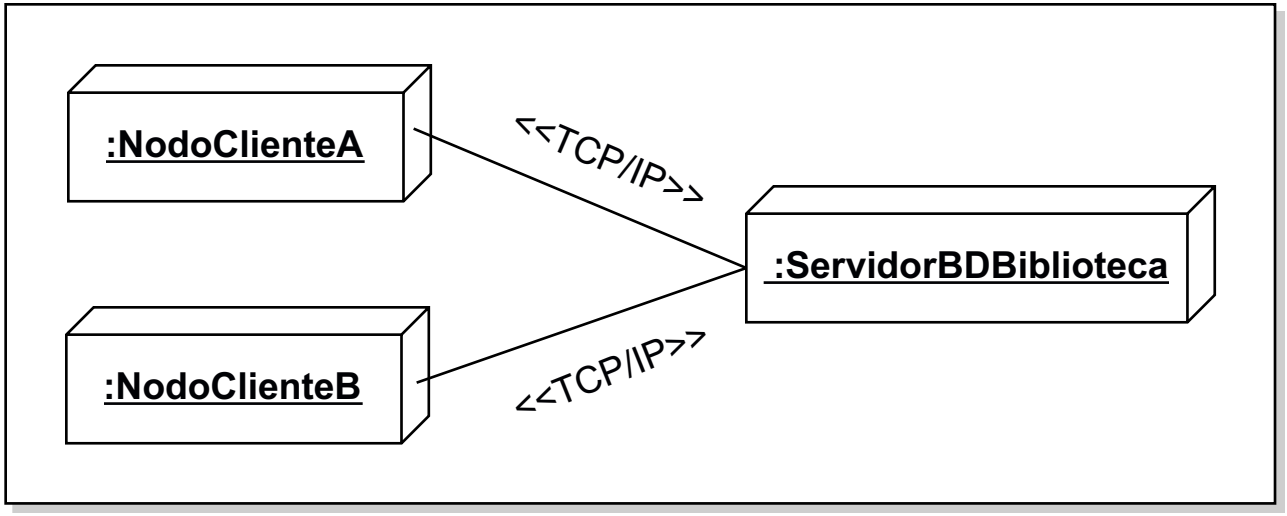
- **Nodos y caminos de comunicación (I)**
  - Un nodo es un clasificador que representa recursos computacionales: ordenadores con procesadores, dispositivos móviles, dispositivos de comunicación, lectores de tarjetas…
  - Pueden conectarse mediante caminos de comunicación para definir topologías de red
  - Un **camino de comunicación** es una asociación entre dos nodos que pueden intercambiar señales y mensajes
  - Las topologías de red específicas se definen mediante enlaces entre instancias de nodos



Instancia de un nodo

Diagramas de despliegue

- **Nodos y caminos de comunicación (II)**



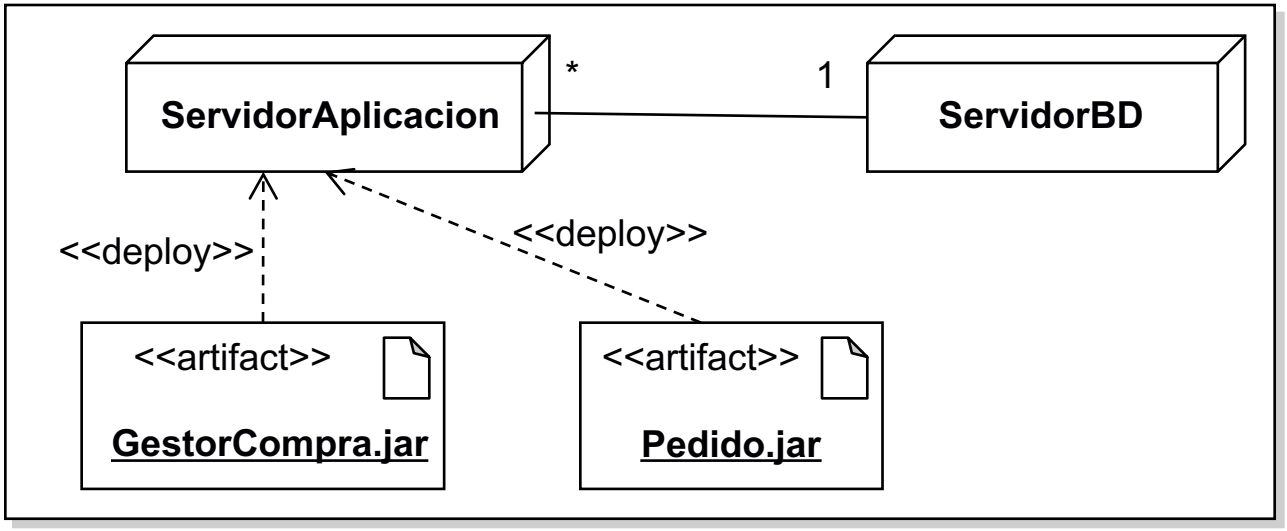
Enlaces entre instancias de nodos: Topología de red específica

Diagramas de despliegue

- **Nodos y caminos de comunicación (III)**
  - Las estructuras jerárquicas de nodos se modelan mediante asociaciones de composición o definiendo una estructura interna
  - El símbolo utilizado para representar un nodo es un cubo
  - La capacidad de un tipo de nodo para desplegar un tipo de componente se representa con la flecha de dependencia estereotipada con la palabra clave <<deploy>>. También se pueden anidar los componentes dentro del símbolo del nodo
  - Un nodo con el estereotipo <<device>> representa un **dispositivo**: recurso computacional con capacidad de procesamiento en el que los artefactos pueden desplegarse para su ejecución. Pueden anidarse si una máquina física se descompone en sus elementos
  - Otros estereotipos usados en los nodos: «application server», «client workstation», «mobile device», «embedded device…

Diagramas de despliegue

- **Nodos y caminos de comunicación (IV)**



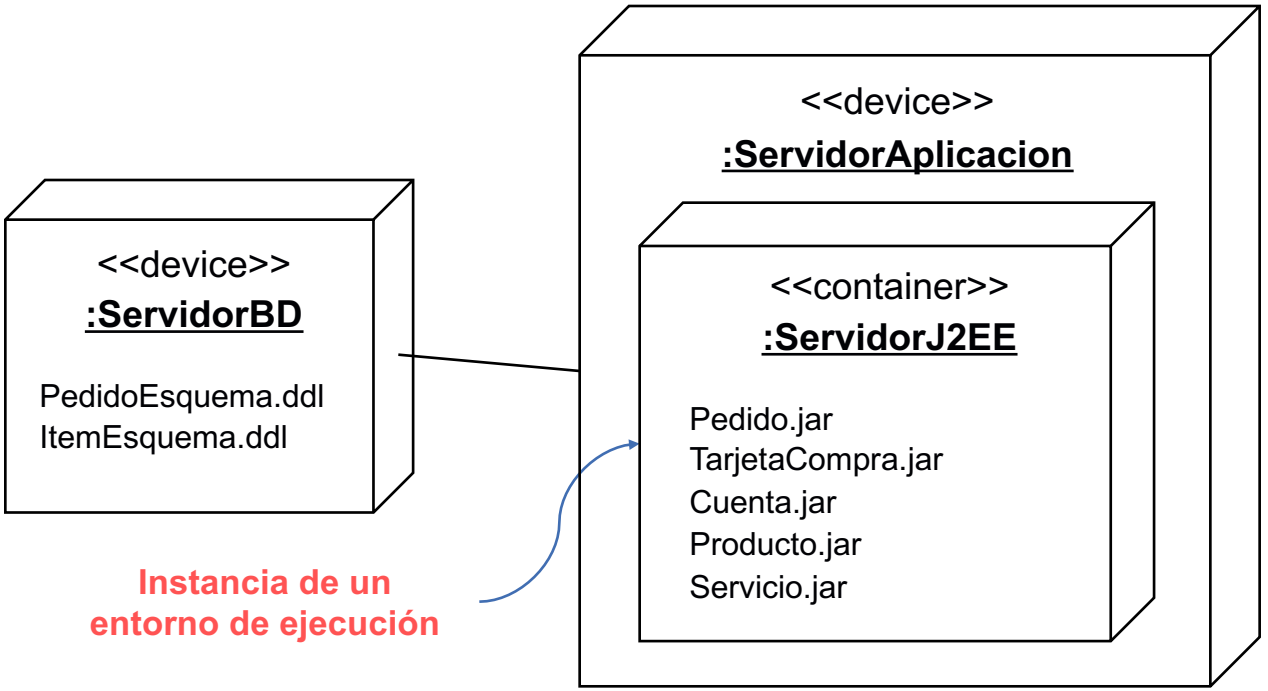
Asociación entre dos tipos de nodos y despliegue de uno de ellos

Diagramas de despliegue

- **Entornos de ejecución (I)**
  - Un entorno de ejecución es un tipo especial de nodo que implementa un conjunto de servicios que los componentes requieren en tiempo de ejecución
  - Generalmente forman parte de un nodo que representa el hardware en el que reside el entorno de ejecución
  - Las instancias de entornos de ejecución se asignan a instancias de nodos usando asociaciones de estructuras compuestas donde los entornos juegan el papel de "parte"
  - Pueden anidarse entornos de ejecución
  - Pueden tener interfaces de servicios de nivel del sistema que pueden usar los elementos desplegados
  - Notación: nodo con el estereotipo «ExecutionEnvironment»
  - Otros estereotipos usados para definir perfiles son: «OS», «database system», «J2EE container» …

Diagramas de despliegue

- **Entornos de ejecución (II)**



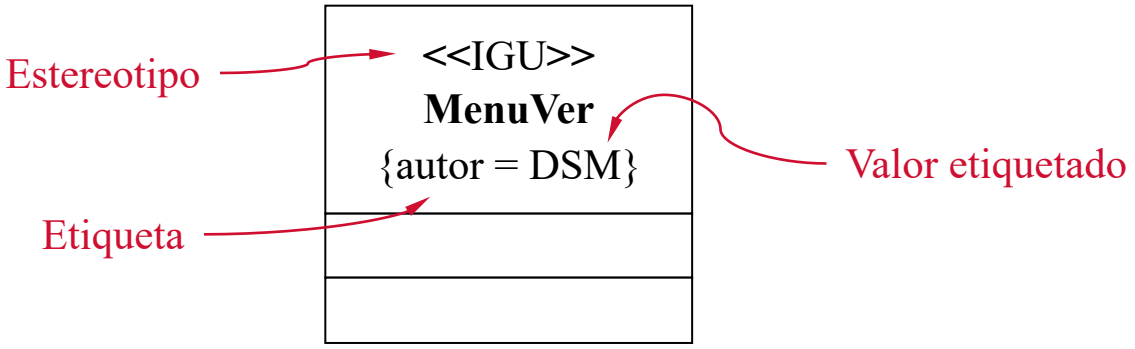
Perfiles

Introducción

- 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 manteniendo la interoperabilidad entre herramientas
- Los perfiles se construyen haciendo uso de los principales mecanismos de extensibilidad de UML: restricciones, estereotipos y valores etiquetados
  - Permiten muchos tipos de extensiones sobre UML sin necesidad de cambiar el metamodelo básico
  - Se utilizan para crear versiones adaptadas de UML
  - Un perfil se modela como un conjunto coherente de estereotipos con su definición de etiquetas y restricciones
  - Un perfil es un paquete que también puede contener elementos del modelo o dependencias con un paquete que contenga elementos del modelo

Mecanismos de extensibilidad

- **Mecanismos de extensibilidad**: permiten al usuario adaptar y extender los elementos del modelo UML definiendo características adicionales mediante:
  - **Estereotipos**: extienden las clases del metamodelo (metaclases) creando un nuevo elemento del metamodelo (subclases de las metaclases de UML con nuevos atributos y semántica adicional)
  - **Definiciones de etiquetas**: especifican nuevas clases de propiedades que pueden asociarse a los elementos del modelo. Se pueden considerar definiciones de metaatributos
  - **Valores etiquetados**: definen una propiedad como un par nombre-valor (el nombre es la etiqueta): {etiqueta = valor}
  - **Restricciones**: reglas que restringen la semántica de uno o más elementos UML. Existen restricciones predefinidas en UML



Ejemplos de uso de los mecanismos de extensión

Mecanismos de extensibilidad

- **Extensiones estándar (I)**

Mecanismos de extensión que proporciona UML para definir características adicionales. Los tres elementos de extensión principales son ***valores etiquetados***, ***restricciones*** y ***estereotipos***

- **Valores etiquetados**: propiedades ligadas a los elementos UML. Se representan como pares nombre-valor

  {etiqueta1 = valor1, etiqueta2 = valor2 …} o {etiqueta}

  Valores predefinidos

  - **Unión:** valor derivado de la unión de subconjuntos

    {union}
  - **Subsets:** El atributo o fin de asociación que lleva el valor etiquetado representa un subconjunto de la propiedad

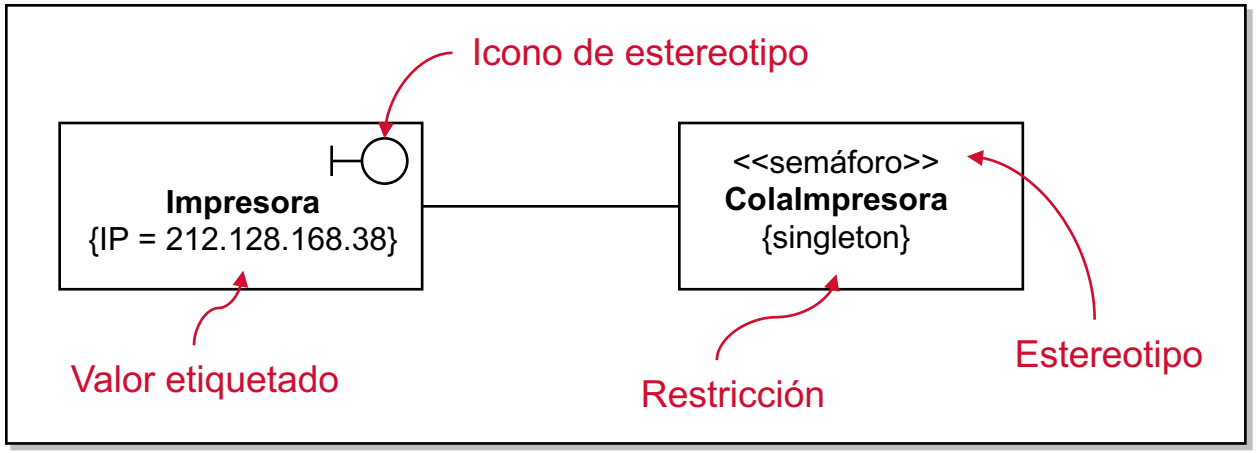
    {subsets <nombre-propiedad>}
  - **Redefinición:** redefinición de un atributo o fin de asociación

    {redefines <nombre-propiedad>}

Mecanismos de extensibilidad

- **Extensiones estándar (II)**
  - **Restricciones**: reglas que restringen la semántica de uno o más elementos UML. Se representa como una cadena entre llaves

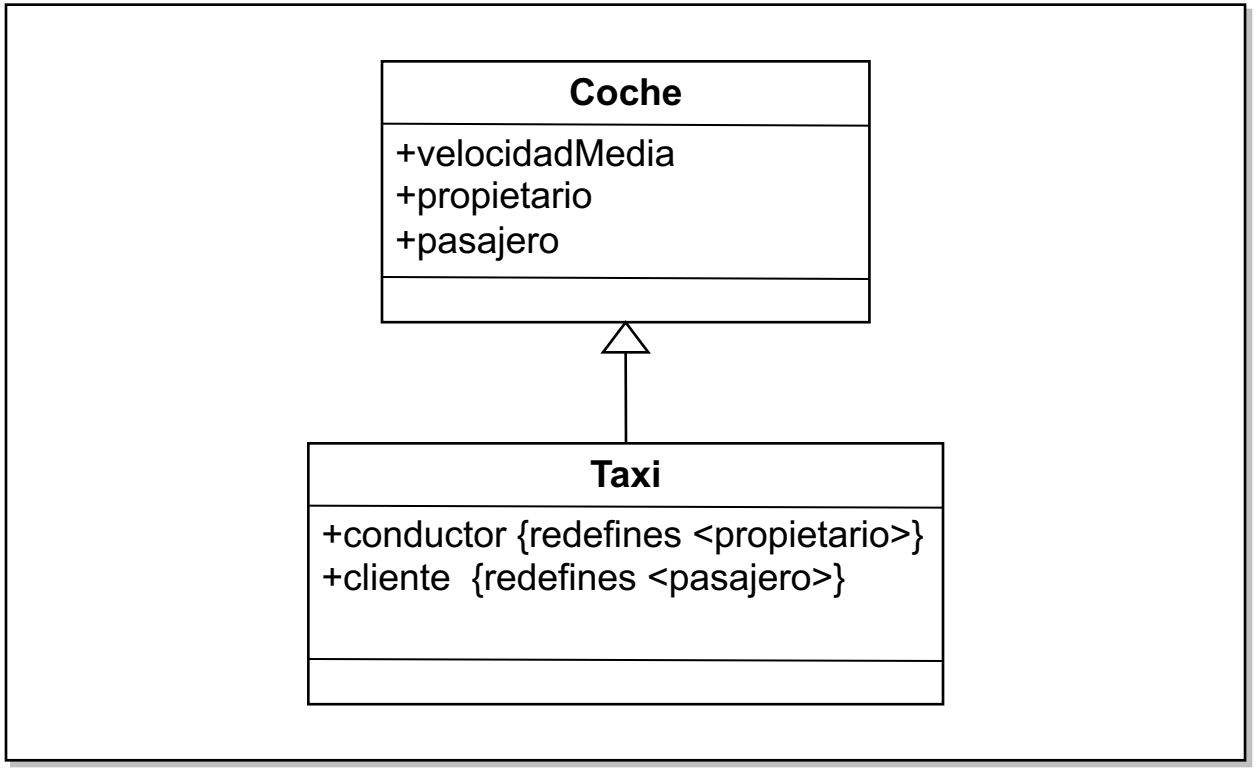
    {restricción}
  - **Estereotipos**: mecanismo para extender las metaclases, creando un nuevo elemento del metamodelo. Se representan como una cadena entre los símbolos `<<´ y `>>´. También se puede asociar un icono gráfico a un estereotipo



Ejemplos de uso de los mecanismos de extensión

Mecanismos de extensibilidad

- **Extensiones estándar (III)**

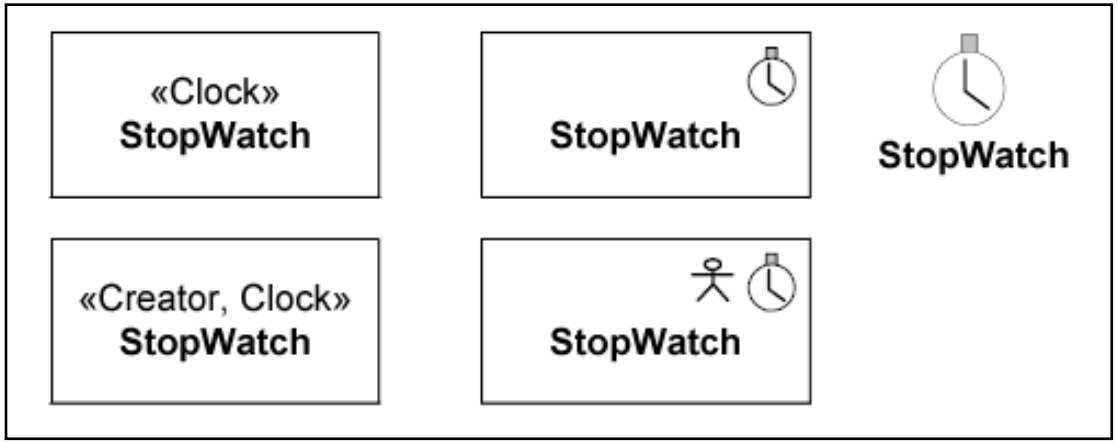


Ejemplos de redefinición de propiedades

Mecanismos de extensibilidad

- **Extensiones estándar (IV)**

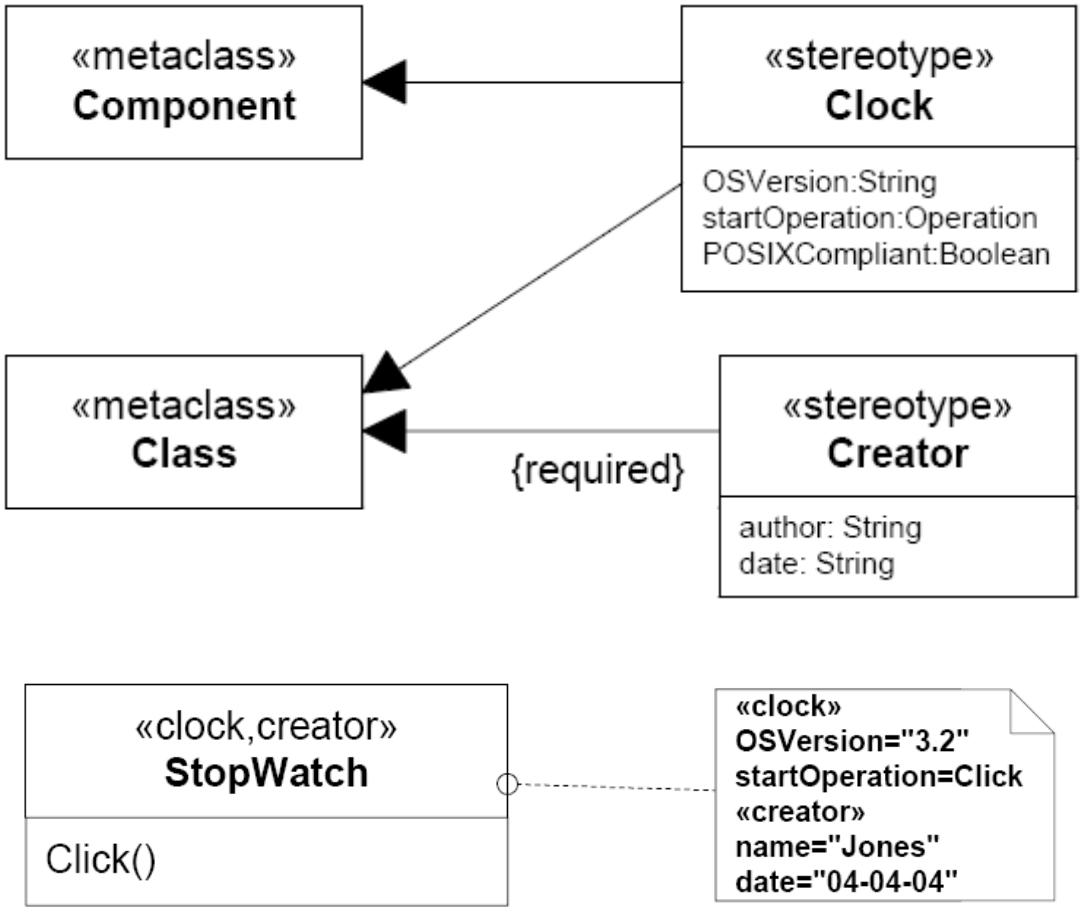
Definición de un estereotipo



Opciones de representación de una clase extendida con un estereotipo

Mecanismos de extensibilidad

- **Extensiones estándar (V)**



Ejemplos de definición y uso de estereotipos

Definición y aplicación de perfiles

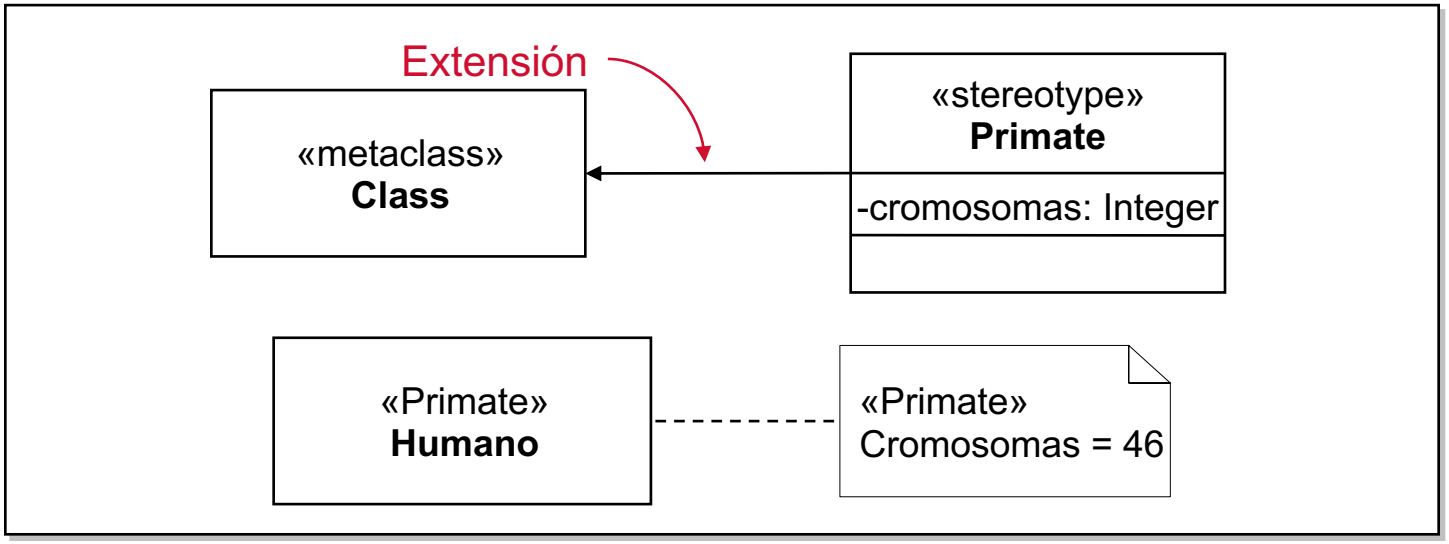
- **Perfiles**

Mecanismos para extender un metamodelo de referencia con objeto de adaptarlo a las características específicas de una plataforma (.NET, J2EE …) o de un dominio de aplicación (tiempo real, modelado de procesos de negocio …)

- El paquete "perfiles" (*profiles*) de UML contiene características que permiten modelar las extensiones
- Al crear un estereotipo que extiende una metaclase el usuario puede definir un conjunto de propiedades o restricciones de ese elemento
- **Notación**
  - **Extensión**: flecha con un triángulo sólido negro en el extremo de la metaclase que extiende el estereotipo
  - **Perfil**: se define en un paquete con el estereotipo «profile». El paquete contiene todas las extensiones que forman el perfil
  - **Aplicación del perfil**: relación de dependencia con el estereotipo «apply»

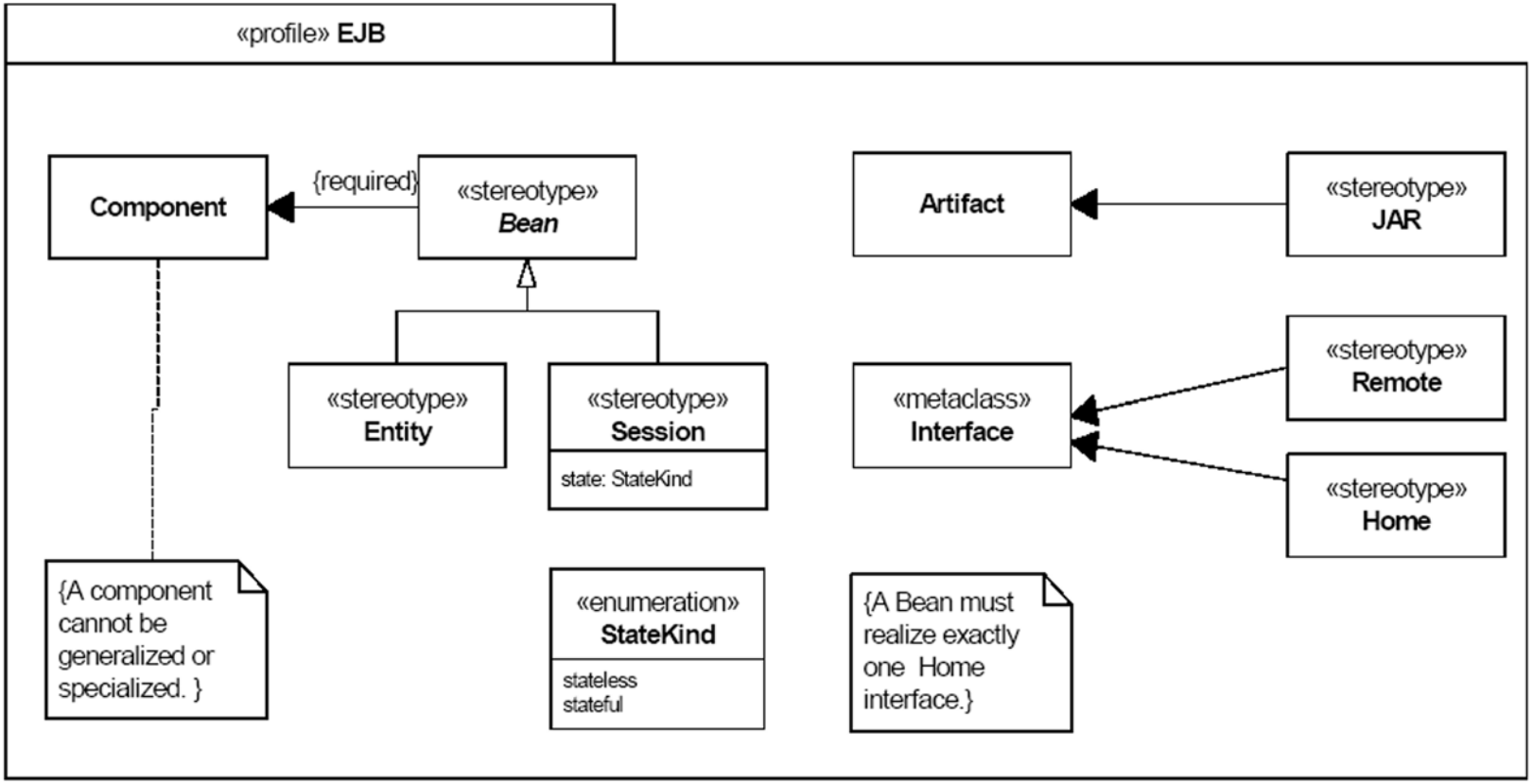
Definición y aplicación de perfiles

- Una extensión es una relación entre un estereotipo y una metaclase que indica que las propiedades definidas en el estereotipo pueden ser aplicadas a instancias de la metaclase que lleven ese estereotipo
- Una extensión puede ser opcional u obligatoria
  - Se utiliza la palabra clave {required} para indicar que es obligatoria, en caso contrario es opcional



Definición de un estereotipo (Primate) y clase (Humano) construida a partir de él

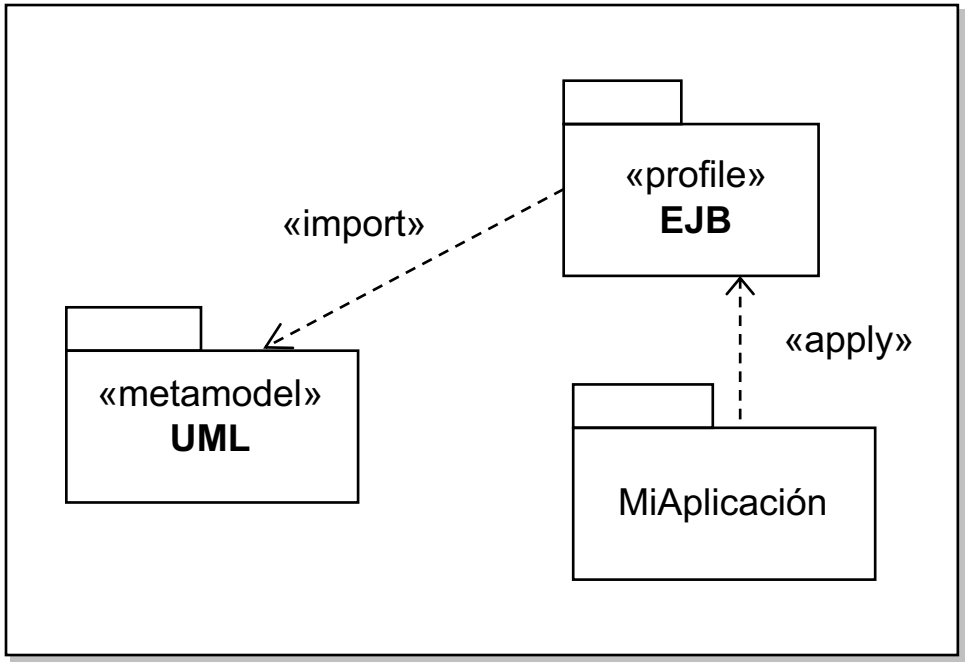
Definición y aplicación de perfiles



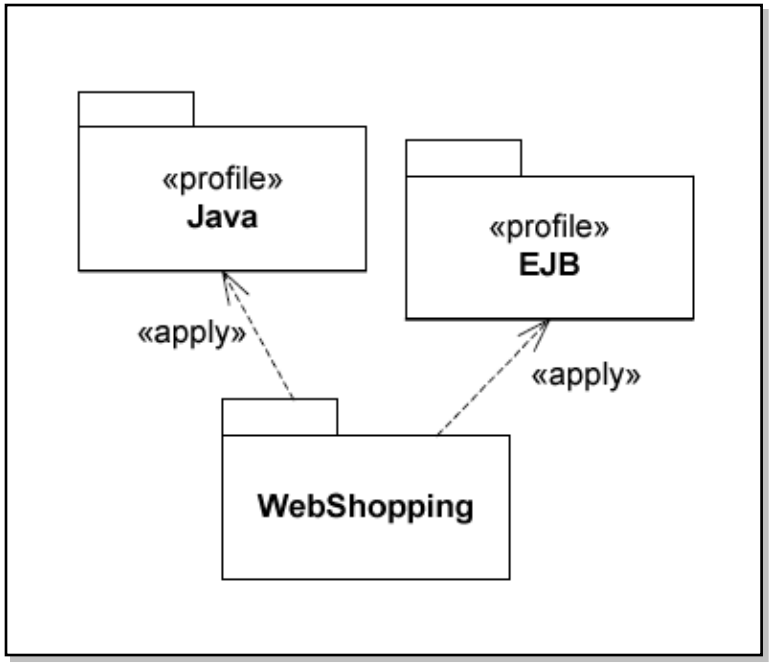
Definición del perfil EJB

Definición y aplicación de perfiles

- La especificación de los estereotipos y restricciones declarados en un perfil puede aplicarse a elementos del modelo dentro de un paquete
  - Pueden aplicarse uno o más perfiles a un paquete dado
  - Si un perfil depende de otro perfil ambos se aplican al modelo

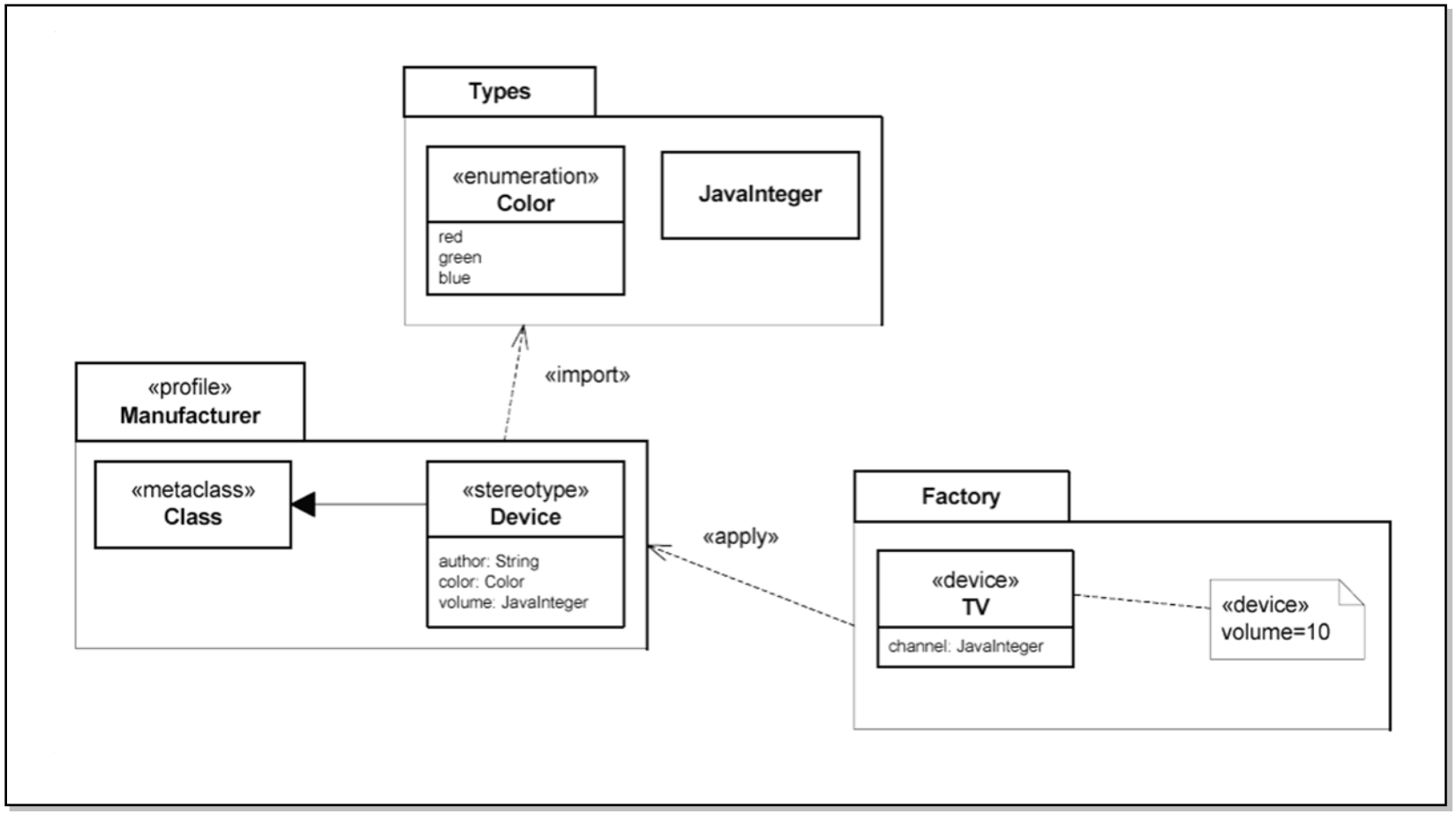


Aplicación del perfil EJB que usa el metamodelo de referencia UML



Uso de los perfiles *Java* y *EJB* en la aplicación *WebShopping*

Definición y aplicación de perfiles



Ejemplos de uso de perfiles

Bibliografía

Booch, G. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

Booch, G.; Rumbaugh, J. "*Unified Method for Object-Oriented Development*". Documentation set, version 0.8. Rational Software Corporation, 1995.

Booch, G.; Jacobson, I.; Rumbaugh, J. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 1.0. Rational Software Corporation, 13 January 1997.

Booch, G.; Rumbaught, J.; Jacobson, I. "*The Unified Modeling Language. User Guide*". Addison-Wesley Object Technology Series, 1999.

Cockburn, A. "*Writing Effective Use Cases*". Addison-Wesley, 2000.

Eriksson, H. E.; Penker, M. "*UML Toolkit*". John Wiley and Sons, 1998.

Eriksson, H. E.; Penker, M., Lyons, B.; Fado, D. "*UML 2 Toolkit*". OMG Press, 2004.

Fowler, M., Scott, K. "*UML Distilled. A Brief Guide to the Standard Object Modeling Language*". 2nd edition. Addison Wesley, 2000.

Jacobson, I. "*Object Oriented Development in an Industrial Environment*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). Pages 183-191. ACM, 1987.

Jacobson, I.; Christerson, M.; Jonsson, P.; Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

Jacobson, I.; Booch, G. Rumbaugh, J. "*The Unified Software Development Process*". Addison-Wesley Object Technology Series, 1999.

Bibliografía

OMG**.** "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. June 1999.

OMG. "*Common Warehouse Metamodel (CWA) Specification. Version 1.1*". Object Management Group Inc., March, 2003.

OMG. "*Meta Object Facility (MOF) Specification. Version 1.4*". Object Management Group Inc., April 2002.

OMG. "*Meta Object Facility (MOF) Specification. Version 2.0*". Object Management Group Inc., October, 2003.

OMG. "*MDA Guide Version 1.0.1*". Object Management Group Inc., June 2001.

OMG. "*OMG Unified Modeling Language Specification. Version 1.4*". Object Management Group Inc. September 2001.

OMG. "*OMG Unified Modeling Language Specification. Version 1.5*". Object Management Group Inc., March 2003.

OMG. "*UML 2.0 Infrastructure Specification*". Object Management Group Inc., December, 2003.

OMG. "*UML 2.0 Superstructure Specification*". Object Management Group Inc.,, October, 2004.

OMG. "Unified Modeling Language specification version 2.5.1". Object Management Group,, Needham, MA, USA, formal/17-12-05, 2017. Disponible en: https://goo.gl/kaE82a

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F. Lorensen. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

Rumbaugh, J.; Jacobson, I.; Booch, G. "*El Lenguaje Unificado de Modelado. Manual de Referencia*". 2ª ed. Pearson Education, 2007.

INGENIERÍA DE SOFTWARE I

Tema 8 – UML. Unified Modeling Language

2º G.I.I.
Fecha de última modificación: 26-2-2020

Dr. Francisco José García Peñalvo / fgarcia@usal.es
Dra. María N. Moreno García / mmg@usal.es
Dra. Alicia García Holgado / aliciagh@usal.es
Andrea Vázquez Ingelmo / andreavazquez@usal.es

Departamento de Informática y Automática
Universidad de Salamanca