



INGENIERÍA DEL SOFTWARE I

Tema 2

Lenguaje Unificado de Modelado - UML

Universidad Cantabria – Facultad de Ciencias

Patricia López, Francisco Ruiz



Objetivos y Bibliografía

- Objetivos
 - Presentar el estándar UML 2.
 - Conocer los principales constructores del lenguaje, así como los diversos tipos de diagramas.
 - Conocer algunos conceptos útiles acerca de modelado.

- Bibliografía
 - Básica
 - Booch, Rumbaugh y Jacobson (2006): El Lenguaje Unificado de Modelado
 - Caps. 2 y 7.
 - Complementaria
 - Booch, Rumbaugh y Jacobson (2006): El Lenguaje Unificado de Modelado
 - Caps. 4, 5 y 6.
 - Rumbaugh, Jacobson y Booch (2007): El Lenguaje Unificado de Modelado Manual de Referencia.
 - Cap. 3.



Bibliografía (cont.)

- Estándares UML y OCL
 - www.uml.org
 - Especificaciones, tutoriales, etc.
 - Última versión de la especificación UML: 2.3
 - <http://www.omg.org/spec/UML/2.3/Superstructure/PDF/>
- Webs
 - Múltiples enlaces e informaciones sobre UML:
 - http://www.cetus-links.org/oo_uml.html
 - Nuevas características del UML 2
 - http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=31
 - Diagramas de UML 2 (con Visual-Paradigm)
 - <http://www.visual-paradigm.com/VPGallery/diagrams/index.html>



Contenido

- Introducción
 - Importancia del modelado
 - UML
 - Características principales
 - Ventajas e inconvenientes
- Objetivos de UML
- Conceptos de Modelado
 - Modelos
 - Vistas Arquitecturales
- Modelo UML de un sistema
- Modelo Conceptual
 - Elementos
 - Estructurales
 - De Comportamiento
 - De Agrupación
 - De Anotación
 - Relaciones
 - Diagramas
 - Estructurales
 - De Comportamiento
 - Reglas
 - Mecanismos Comunes
 - Especificaciones
 - Adornos
 - Divisiones comunes
 - Extensibilidad
- OCL



¿Qué es UML?

¿Para qué sirve UML?

¿Qué es un modelo?

¿Para qué sirve un modelo?

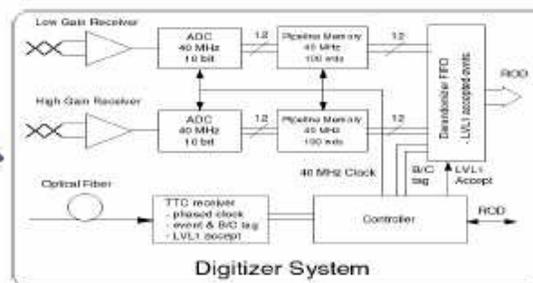


Introducción - Modelado

- Un **modelo** es una **abstracción** de un sistema o entidad del mundo real.
- Una abstracción es una **simplificación**, que incluye sólo aquellos detalles relevantes para algún determinado propósito
- El modelado permite **abordar la complejidad** de los sistemas



Modeled system



Functional model



Introducción – Modelado de software

```
package codemodel;
public class Guitarist extends Person implements MusicPlayer {
    Guitar favoriteGuitar;
    public Guitarist (String name) {super(name);}
    // A couple of local methods for accessing the class's properties
    public void setInstrument(Instrument instrument) {
        if (instrument instanceof Guitar) {
            this.favoriteGuitar = (Guitar) instrument;
        } else {
            System.out.println("I'm not playing that thing!");
        }
    }
    public Instrument getInstrument( ) {return this.favoriteGuitar;}
}
```

- Representa sólo la lógica e ignora el resto
- El ser humano lo interpreta muy lentamente
- No facilita la reutilización ni la comunicación



Introducción – Modelado de software

Guitarist is a class that contains six members: one static and five non-static. Guitarist uses, and so needs an instance of, Guitar; however, since this might be shared with other classes in its package, the Guitar instance variable, called favoriteGuitar, is declared as default.

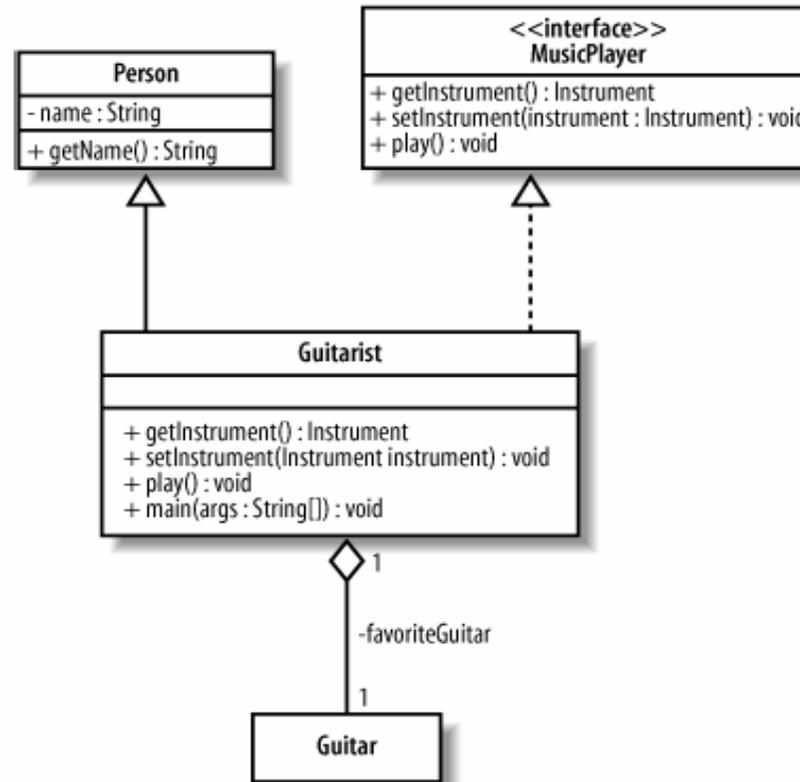
Five of the members within Guitarist are methods. Four are not static. One of these methods is a constructor that takes one argument, and instances of String are called name, which removes the default constructor.

Three regular methods are then provided. The first is called setInstrument, and it takes one parameter, an instance of Instrument called instrument, and has no return type. The second is called getInstrument and it has no parameters, but its return type is Instrument. The final method is called play. The play method is actually enforced by the MusicPlayer interface that the Guitarist class implements. The play method takes no parameters, and its return type is void.

- Es ambigua y confusa
- Es lenta de interpretar
- Difícil de procesar



Introducción – Modelado de software



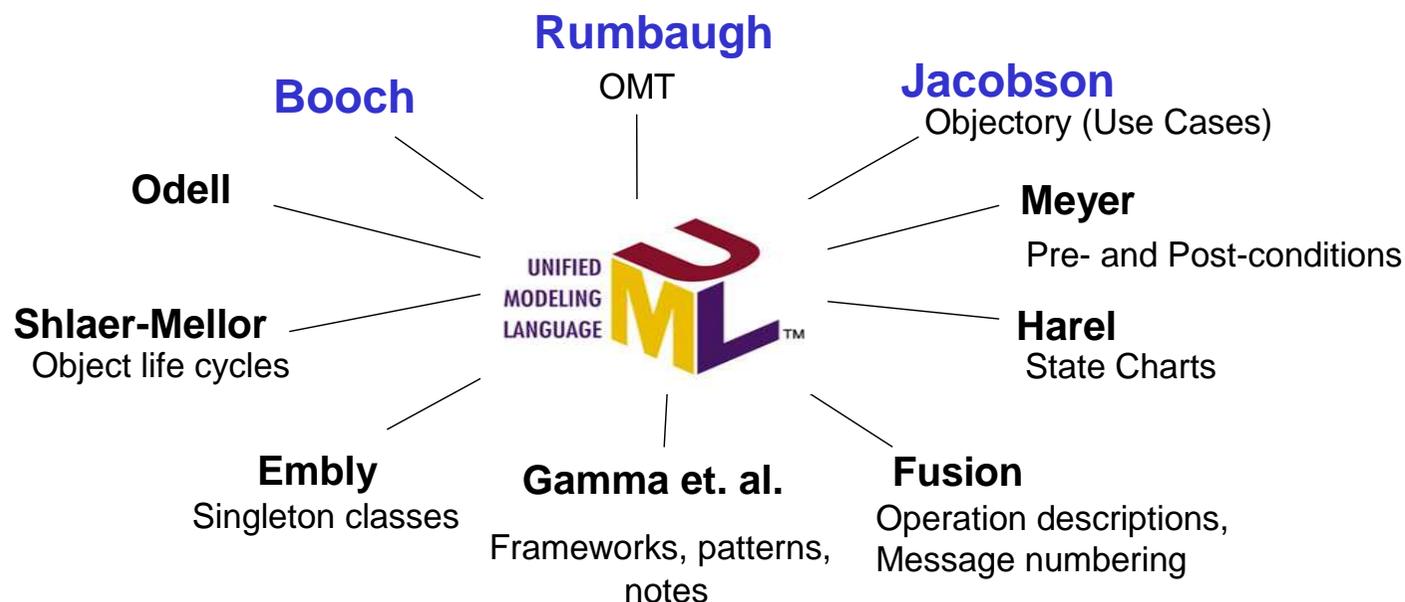
- No es ambigua ni confusa (una vez conocemos la semántica de cada elemento de modelado)
- Es fácil y rápida de interpretar
- Es fácil de procesar por herramientas



Introducción - ¿Qué es UML?

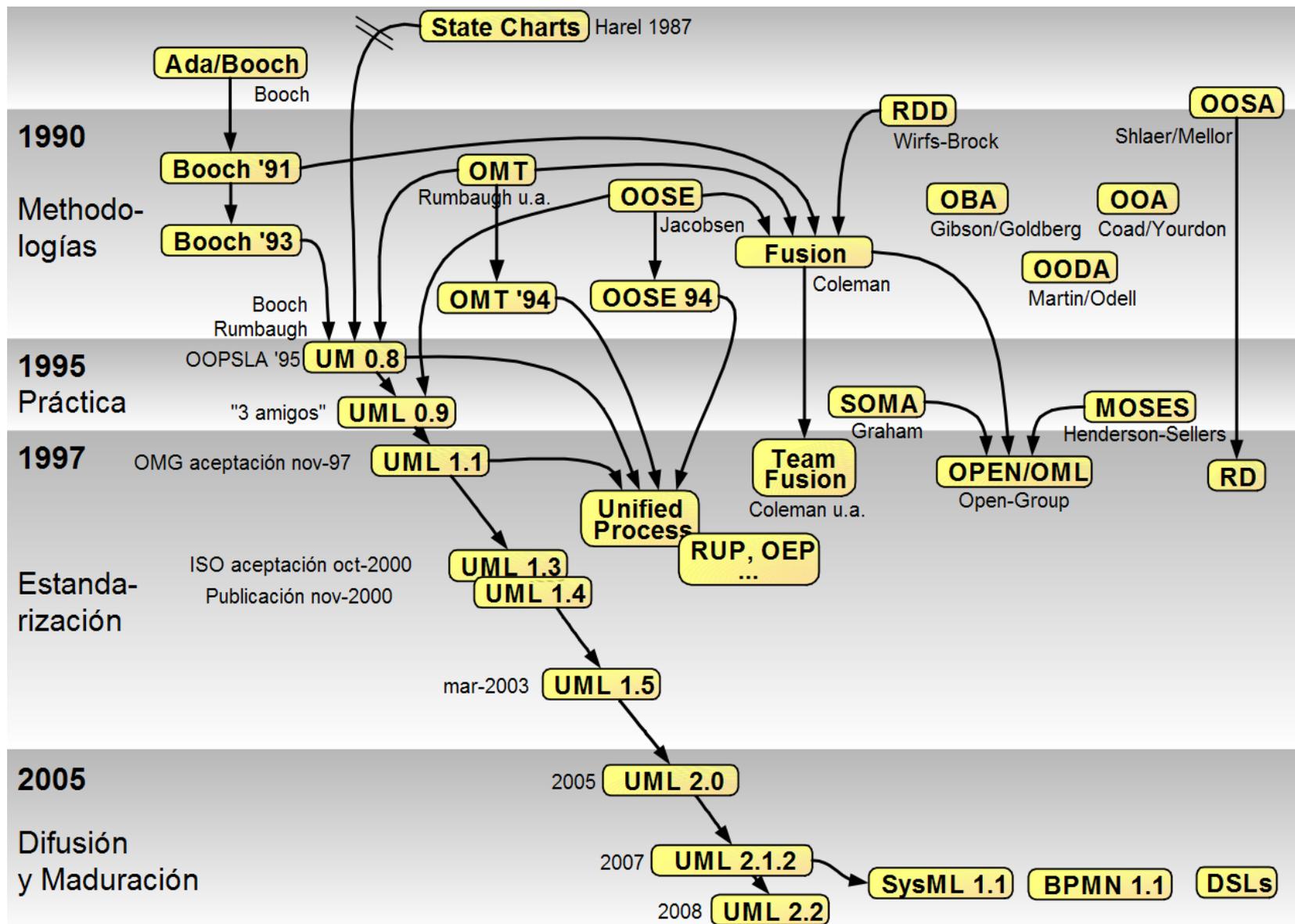
UML = Unified Modeling Language

- Es un lenguaje de **modelado visual** de propósito general **orientado a objetos**.
 - Impulsado por el **Object Management Group** www.omg.org
- **Estándar**: Independiente de cualquier fabricante comercial
- Agrupa notaciones y conceptos provenientes de distintos tipos de métodos orientados a objetos





Introducción - Evolución de UML

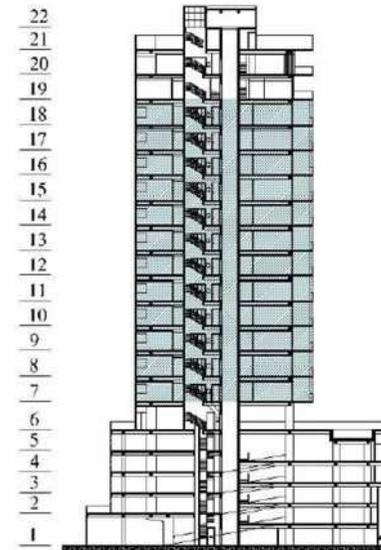
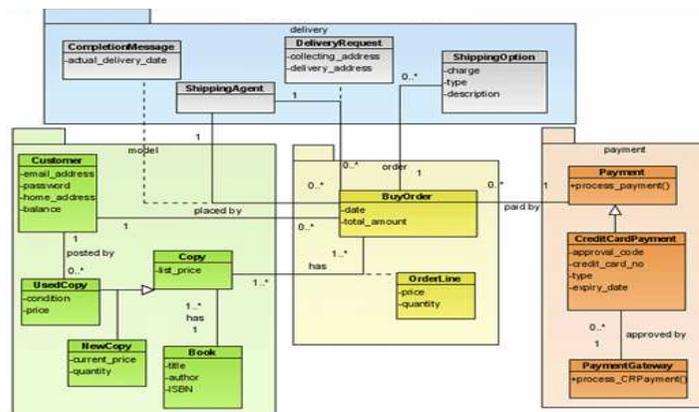




Introducción – Características de UML

- **Lenguaje de Modelado UML**

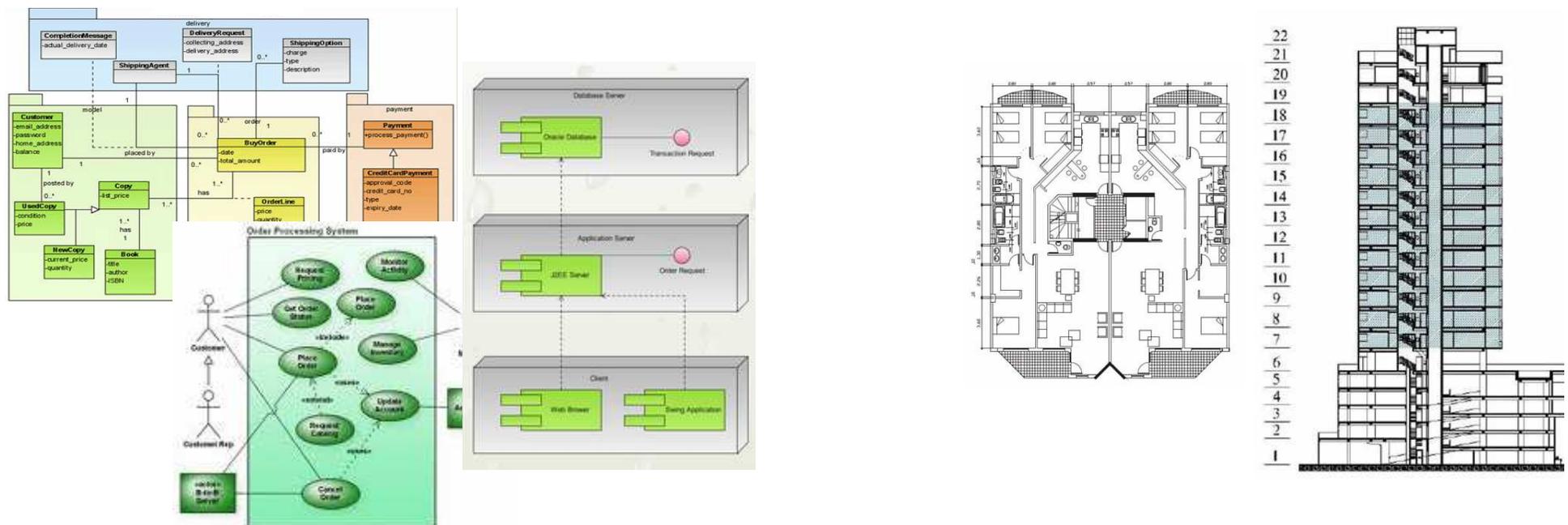
- “Lenguaje cuyo vocabulario y reglas se centran en la representación conceptual y física de un sistema” (Booch, Jacobson y Rumbaugh)
- Lenguaje = Notación + Reglas (Sintácticas, Semánticas)
- **UML** ofrece vocabulario y reglas:
 - para crear y leer modelos bien formados
 - que constituyen los **planos de un sistema software.**





Introducción – Características de UML

- **UML** es **independiente** del Proceso de desarrollo
 - Un **uso óptimo** se consigue en procesos dirigidos por casos de uso, centrados en la arquitectura, iterativos e incrementales
 - → Proceso Unificado de Desarrollo (RUP)
- **UML** cubre las diferentes **vistas** de la arquitectura de un sistema mientras evoluciona a través del ciclo de vida del desarrollo de software
 - **Vistas Software** (estáticas, dinámicas, etc..)





Introducción – Ventajas e inconvenientes de UML

- **Ventajas** de UML
 - Es **estándar** => Facilita la **comunicación**
 - Está basado en **metamodelo** con una semántica bien definida
 - Se basa en una **notación gráfica** concisa y fácil de aprender y utilizar
 - Se puede utilizar para modelar **sistemas software** en diversos dominios:
 - Sistemas de información empresariales, Sistemas WEB, sistemas críticos y de tiempo real, etc.
 - Incluso en **sistemas que no son software**
 - Es fácilmente **extensible**
- **Inconvenientes** de UML
 - **No es una metodología.** Además de UML, hace falta una metodología OO
 - **No cubre todas** las necesidades de especificación de un proyecto software
 - No define los documentos textuales o el diseño de interfaces de usuario
 - Faltan **ejemplos** elaborados en la documentación
 - Puede resultar **complejo** alcanzar un conocimiento completo del lenguaje
 - Sin embargo => Regla del 80 - 20



Introducción – Futuro de UML

- UML 2.4
- Extensiones de UML
 - SysML - Systems Modeling Language (SysML, www.sysml.org)
- Generación automática de código a partir de modelos
 - Model-Driven Engineering (MDE)
- Extendiendo UML mediante **perfiles**
 - MARTE – Modeling and Analysis of Real-Time Embedded Systems
- Entornos avanzados basados en **metamodelado**
 - Eclipse Modeling Framework (EMF)
- Modelado y generación de código en **dominios específicos**
 - Domain-Specific Modeling (DSM, www.dsmforum.org)



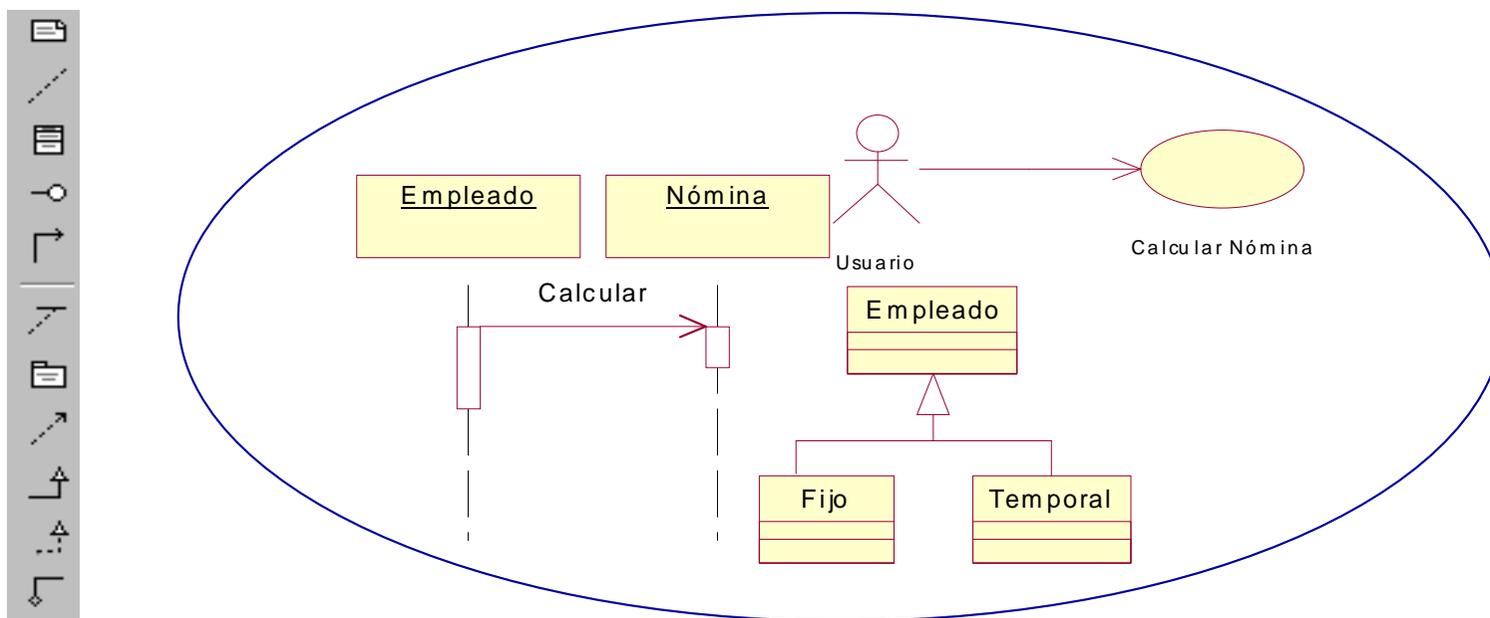
Objetivos

- **UML** es un **lenguaje de modelado visual** que sirve para
 - **visualizar,**
 - **especificar,**
 - **construir** y
 - **documentar**
- Sistemas
- independientemente de la metodología de análisis y diseño
- pero siempre con una perspectiva **orientada a objetos**



Objetivos - Visualizar

- Detrás de cada símbolo en **UML** hay una semántica bien definida.
 - Basada en un metamodelo estándar MOF-*compliant*.
- Es más que un montón de símbolos gráficos
- Trasciende lo que puede ser representado en un lenguaje de programación
- Modelo explícito, que facilita la comunicación.





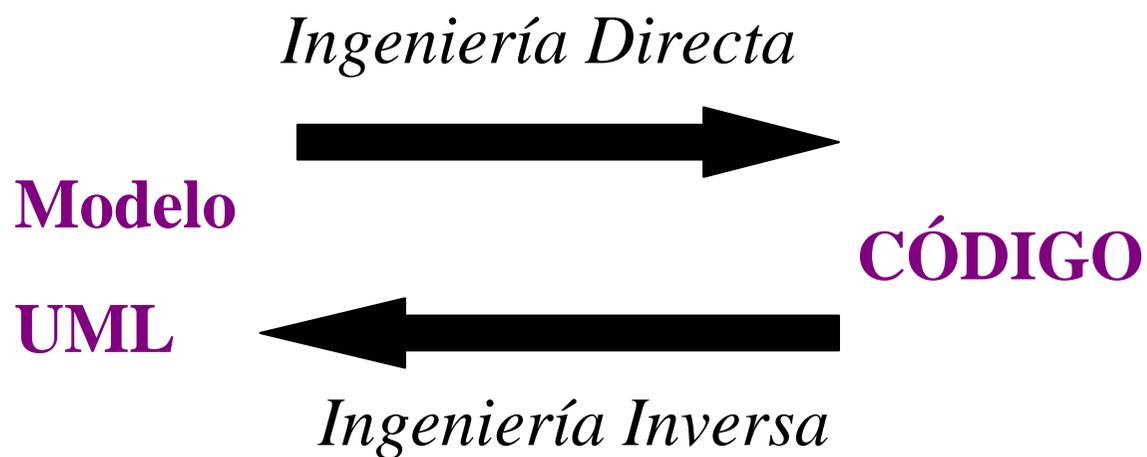
Objetivos - Especificar

- Especificar es equivalente a **construir modelos** precisos, no ambiguos y completos.
- **UML** cubre la especificación del análisis, diseño e implementación de un sistema intensivo en software.



Objetivos - Construir

- **UML** no es un lenguaje de programación visual, pero es posible establecer **correspondencias** entre un modelo UML y lenguajes de programación (Java, C++) y bases de datos (relacionales, OO).





Objetivos - Documentar

- **UML** cubre toda la documentación de un sistema:
 - Arquitectura del sistema y sus detalles (diseño)
 - Expresar requisitos y pruebas
 - Modelar las actividades de planificación y gestión de versiones.

Importancia en el mantenimiento



Conceptos de Modelado

- **Sistema**
 - Colección de elementos, posiblemente divididos en subsistemas, organizados para lograr un propósito. Está descrito por un conjunto de modelos.
- **Modelo**
 - Simplificación completa y autoconsistente de la realidad, creado para comprender mejor un sistema.
- **Vista (Arquitectural)**
 - Proyección de la organización y estructura de un modelo de un sistema, centrada en un aspecto.
 - Incluye un subconjunto de los elementos incluidos en el modelo
- **Diagrama**
 - Representación gráfica de un conjunto de elementos del modelo y sus relaciones. En UML generalmente corresponde a un grafo conexo de nodos (elementos) y arcos (relaciones).



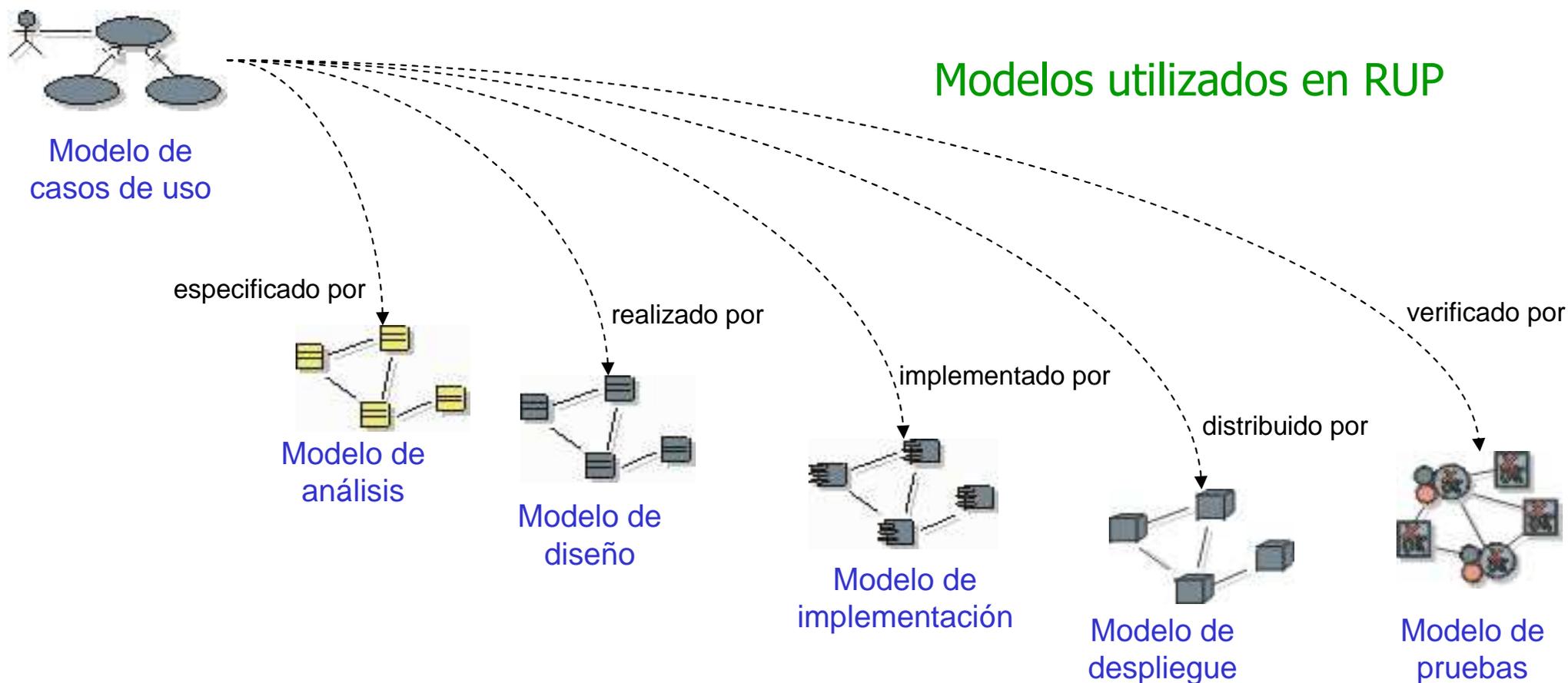
Conceptos de Modelado - Modelos

- Un **modelo** captura las propiedades estructurales (estática) y de comportamiento (dinámicas) de un sistema.
- Es una abstracción de dicho sistema, considerando un cierto propósito.
- El modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo, y a un apropiado nivel de detalle.
 - El **código fuente** del sistema es el modelo más detallado del sistema (y además, es ejecutable). Sin embargo, se requieren otros modelos.
- Cada modelo es completo desde un punto de vista del sistema. Sin embargo, existen relaciones de **trazabilidad** entre los diferentes modelos.



Conceptos de Modelado - Modelos

- Los modelos a utilizar los define la metodología que se aplique en el proceso de desarrollo





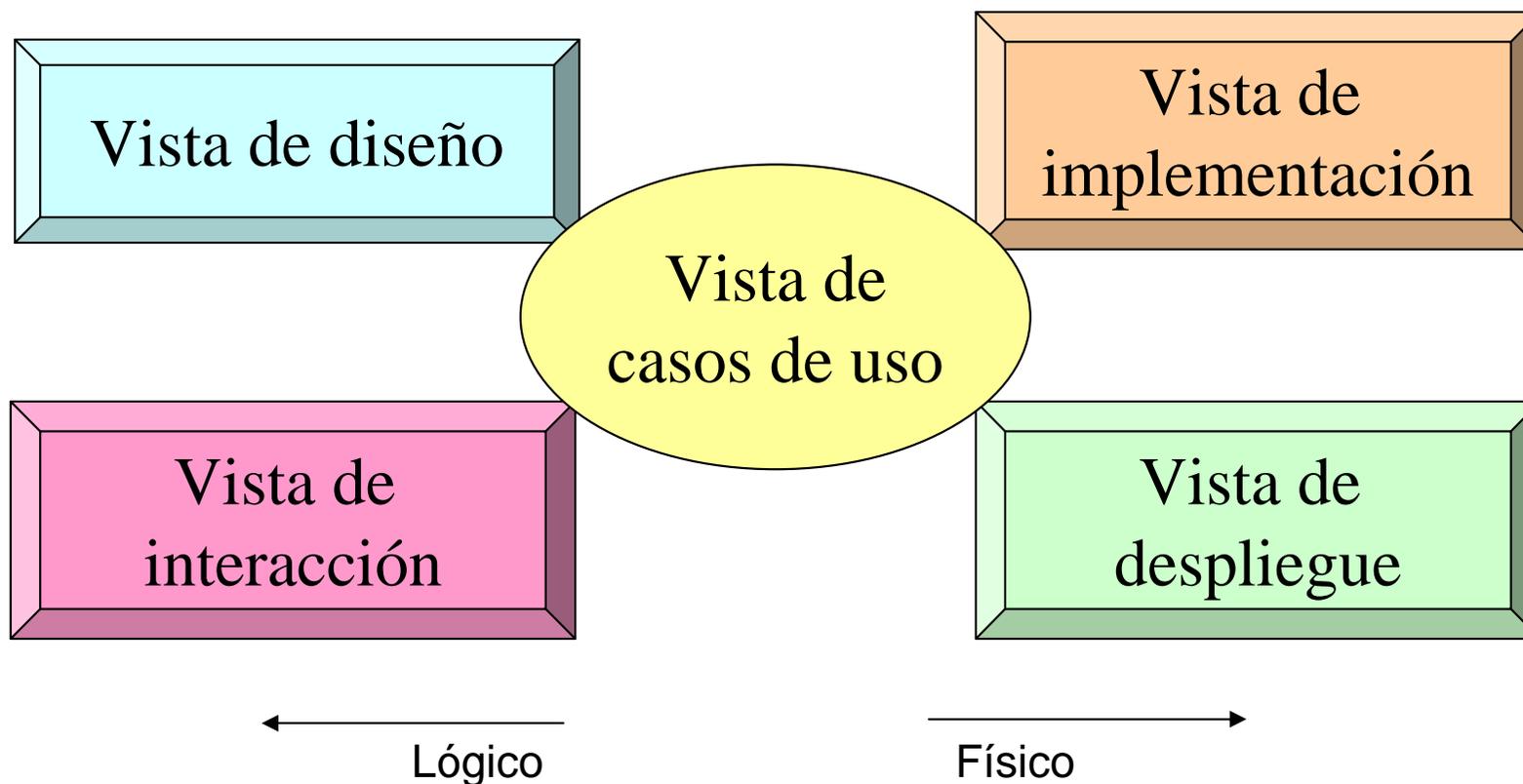
Conceptos de Modelado - Vistas Arquitecturales

- Durante el desarrollo de un sistema software se requiere que éste sea visto desde **varias perspectivas**.
- **Diferentes usuarios** miran el sistema de formas diferentes en momentos diferentes.
- La **arquitectura** del sistema es clave para poder manejar estos puntos de vista diferentes:
 - Se organiza mejor a través de **vistas arquitecturales** interrelacionadas.
 - Proyecciones del modelo del sistema centradas en un aspecto particular.
- No se requiere una vista que contenga la **semántica completa** de la aplicación. **La semántica reside en el modelo.**



Conceptos de Modelado - Vistas Arquitecturales

Las 4 +1 Vistas Arquitecturales (Philippe Krutchen)





Conceptos de Modelado - Vistas Arquitecturales

- **Vista de Casos de Uso (Use Case View)**

- Captura la **funcionalidad** del sistema tal y como es percibido por los usuarios finales, analistas y encargados de pruebas.
- Describe la funcionalidad en base a **casos de uso**.
- En esta vista no se especifica la organización real del sistema software.
- Los **diagramas** que le corresponden son:
 - Aspectos **estáticos**: diagramas de **casos de uso**.
 - Aspectos **dinámicos**: diagramas de **interacción**, de **estados** y de **actividades**.



Conceptos de Modelado - Vistas Arquitecturales

- **Vista de Diseño (Logical View)**

- Captura las clases, interfaces y colaboraciones que describen el sistema:
 - En el dominio del **problema**
 - En el dominio de la **solución**
- Las elementos que la forman dan soporte a los **requisitos funcionales** del sistema.
- Los **diagramas** que le corresponden son:
 - Aspectos **estáticos**: diagramas de **clases** y de **objetos**. También son útiles los diagramas de **estructura compuesta** de clases.
 - Aspectos **dinámicos**: diagramas de **interacción**, de **estados** y de **actividades**.



Conceptos de Modelado - Vistas Arquitecturales

- **Vista de Interacción (Process view)**

- Captura el **flujo de control** entre las diversas **partes del sistema**, incluyendo los posibles mecanismos de concurrencia y sincronización.
- Abarca en especial **requisitos no funcionales** como el rendimiento, escalabilidad y capacidad de procesamiento.
- Los **diagramas** que le corresponden son los mismos que la vista de diseño:
 - Aspectos **estáticos**: diagramas de **clases** y de **objetos**.
 - Aspectos **dinámicos**: diagramas de **interacción**, de **estados** y de **actividades**.
- Pero atendiendo más las **clases activas** que controlan el sistema y los **mensajes** entre ellas.



Conceptos de Modelado - Vistas Arquitecturales

- **Vista de Implementación (Development View)**

- Captura los **artefactos** que se utilizan para ensamblar y poner en producción el sistema **software real**.
- Define la arquitectura física del sistema
- Se centra en:
 - La configuración de las distintas versiones de los archivos físicos,
 - Correspondencia entre clases y ficheros de código fuente,
 - Correspondencia entre componentes lógicos y artefactos físicos.
- Los **diagramas** que le corresponden son:
 - Aspectos **estáticos**: diagramas de **componentes** (especialmente la versión de **artefactos**) y de **estructura compuesta**.
 - Aspectos **dinámicos**: diagramas de **interacción**, de **estados** y de **actividades**.



Conceptos de Modelado - Vistas Arquitecturales

- **Vista de Despliegue (Physical View)**

- Captura las características de **instalación y ejecución** del sistema.
- Contiene los **nodos y enlaces** que forman la topología **hardware** sobre la que se ejecuta el sistema software.
- Se ocupa principalmente de la distribución de las partes que forman el sistema software real.
- Los **diagramas** que le corresponden son:
 - Aspectos **estáticos**: diagramas de **despliegue**.
 - Aspectos **dinámicos**: diagramas de **interacción**, de **estados** y de **actividades**.

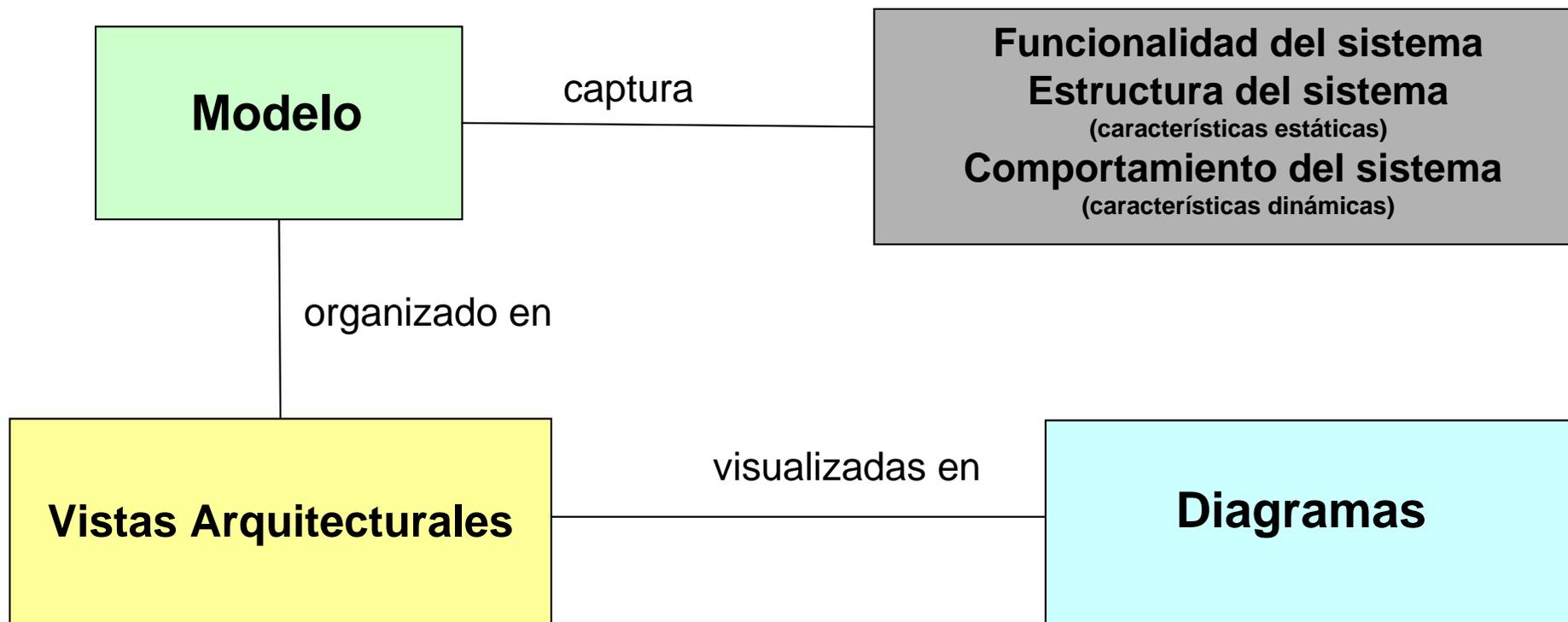


Conceptos de Modelado - Vistas Arquitecturales

- Cada vista puede existir de forma **independiente**.
- Pero **interactúan** entre sí:
 - Los nodos (vista de despliegue) contienen componentes (vista de implementación).
 - Dichos componentes representan la realización (software real) de las clases, interfaces, colaboraciones y clases activas (vistas de diseño y de interacción).
 - Dichos elementos de las vistas de diseño e interacción representan el sistema solución a los casos de uso (vista de casos de uso) que expresan los requisitos.



Modelo UML de un sistema



- Vista de Casos de Uso (Funcionalidad)
- Vista de Diseño (Arquitectura interna)
- Vista de Interacción (Concurrencia y sincronización)
- Vista de Implementación (Arquitectura externa)
- Vista de Despliegue (Plataforma y distribución)

- Diagramas de Casos de Uso
- Diagramas de Clases
- Diagramas de Objetos
- Diagramas de Secuencia
- Diagramas de Colaboración
- Diagramas de Estado
- Diagramas de Actividad
- Diagramas de Componentes
-



En resumen

- Por tanto, el modelo UML de un sistema consiste en:
 - Un conjunto de **elementos de modelado** que definen la estructura, el comportamiento y la funcionalidad del sistema y que se agrupan en una **base de datos única**.
 - La presentación de esos conceptos a través de múltiples **diagramas** con el fin de introducirlos, editarlos, y hacerlos comprensibles.
 - Los diagramas pueden agruparse en **vistas**, cada una enfocada a un aspecto particular del sistema.
- La gestión de un modelo UML requiere una **herramienta específica** que mantenga la consistencia del modelo.



Modelo Conceptual

- Aplicación eficaz de UML => Conocer y comprender su **metamodelo**
- Metamodelo = **Modelo conceptual del lenguaje**
 - ¿Qué elementos nos ofrece UML para modelar un sistema?
 - ¿Qué representa cada uno? ¿Para qué se usa?
- El metamodelo de UML incluye tres tipos de elementos principales:
 - **Bloques de construcción** de UML
 - Elementos (estructurales, de comportamiento, de agrupación, de anotación)
 - Relaciones
 - Diagramas
 - **Reglas** que dictan cómo pueden combinarse estos bloques
 - **Mecanismos comunes** que se aplican a lo largo del lenguaje
 - Especificaciones
 - Adornos
 - Divisiones comunes
 - Mecanismos de extensibilidad.



Elementos

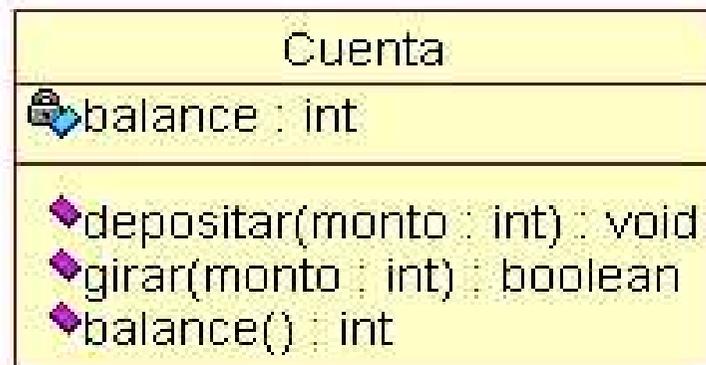
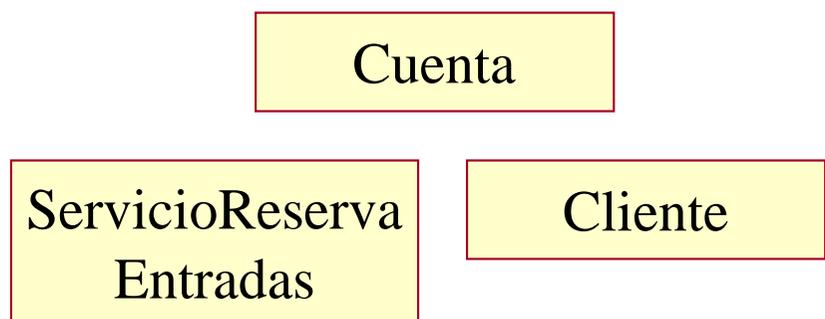
- Los **elementos de UML** son los bloques básicos de construcción de un sistema orientado a objetos.
 - Son abstracciones que constituyen los ciudadanos de primera clase en un modelo.
 - Se utilizan para construir **modelos bien formados**.
- Cuatro **tipos de elementos**:
 - Estructurales
 - De Comportamiento
 - De Agrupamiento
 - De Anotación



Elementos - Estructurales

• Clase

- Una clase es una descripción de un **conjunto de objetos** que comparten los **mismos atributos, operaciones, relaciones y semántica**.
- Es un concepto de **diseño**
 - En ejecución, el sistema está formado por instancias de clases (Objetos)
- Las clases se pueden utilizar para:
 - capturar el **vocabulario del sistema** que se está desarrollando (fase de análisis)
 - representar la **solución** software (fase de diseño)
- Las clases pueden representar cosas hardware, software, conceptos genéricos, personas...

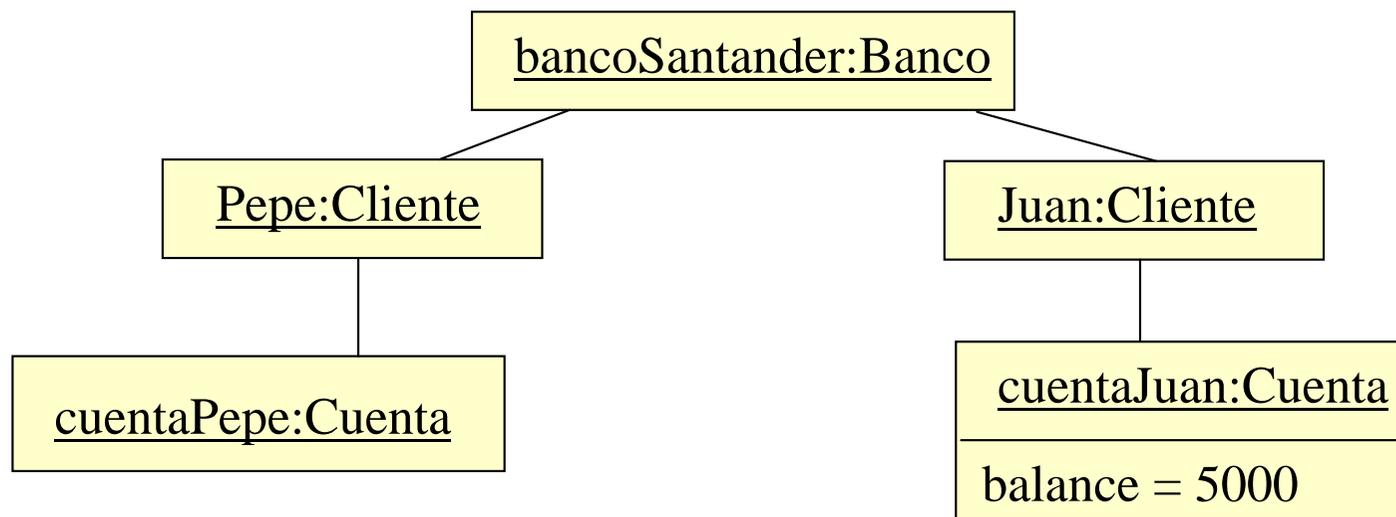




Elementos - Estructurales

● Objeto

- Un objeto representa una instancia de una clase en un determinado contexto
- Es un concepto de **ejecución**
 - El sistema estará formado por un conjunto de objetos interaccionando entre sí

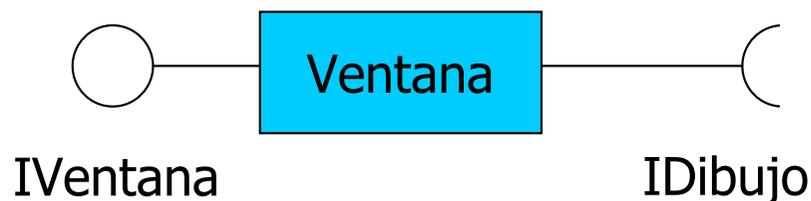
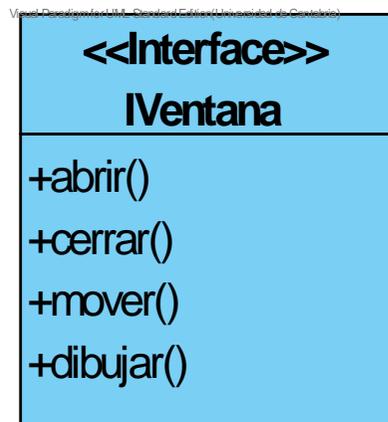




Elementos - Estructurales

● Interfaz

- **Colección de operaciones** que especifican un **servicio** que puede ser ofrecido por una clase o componente.
- Describen el comportamiento visible externamente de dichos elementos.
 - Pueden representar el comportamiento completo de la clase/componente o solo una parte.
- Una interfaz define un conjunto de especificaciones de operaciones (**signaturas**), pero no las implementaciones de dichas operaciones.





Elementos - Estructurales

- **Caso de Uso**

- Describe un **comportamiento** de un sistema, clase o componente, desde el punto de vista del **usuario**.
- Describe un conjunto de secuencias de acciones que ejecuta un sistema y que produce un **resultado observable** que es de interés para un usuario particular.
- Se emplea para estructurar los aspectos de comportamiento.
- Un caso de uso es realizado por una colaboración.

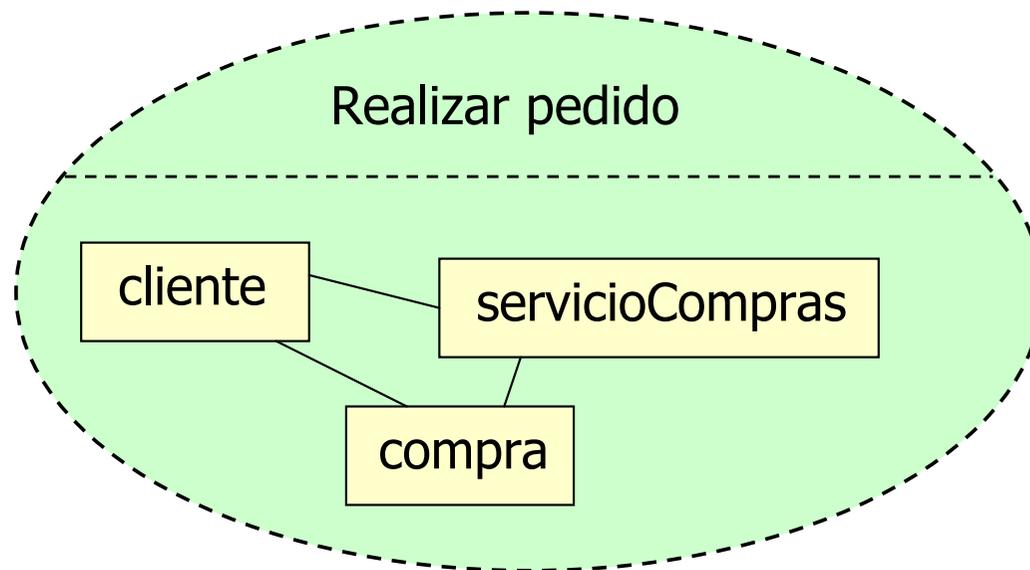
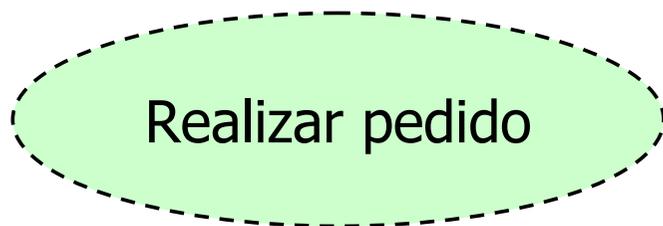
Realizar pedido



Elementos - Estructurales

● Colaboración

- Define una interacción entre una sociedad de roles y otros elementos que colaboran para proporcionar un comportamiento mayor que la suma de sus comportamientos aislados.
- Tienen dimensión estructural y de comportamiento.





Elementos - Estructurales

- **Clase Activa**

- Tipo especial de clase cuyos objetos tienen uno o más procesos o hilos de ejecución =>
 - Pueden dar origen a actividades de control.
- Son iguales que las clases salvo que sus **objetos** pueden ser **concurrentes** con otros objetos de clases activas.



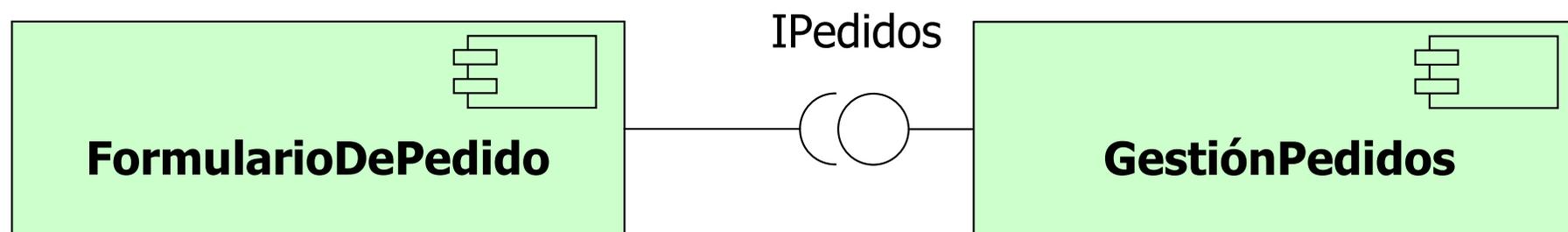
- **Especializaciones** de clase activa: Procesos, Hilos.



Elementos - Estructurales

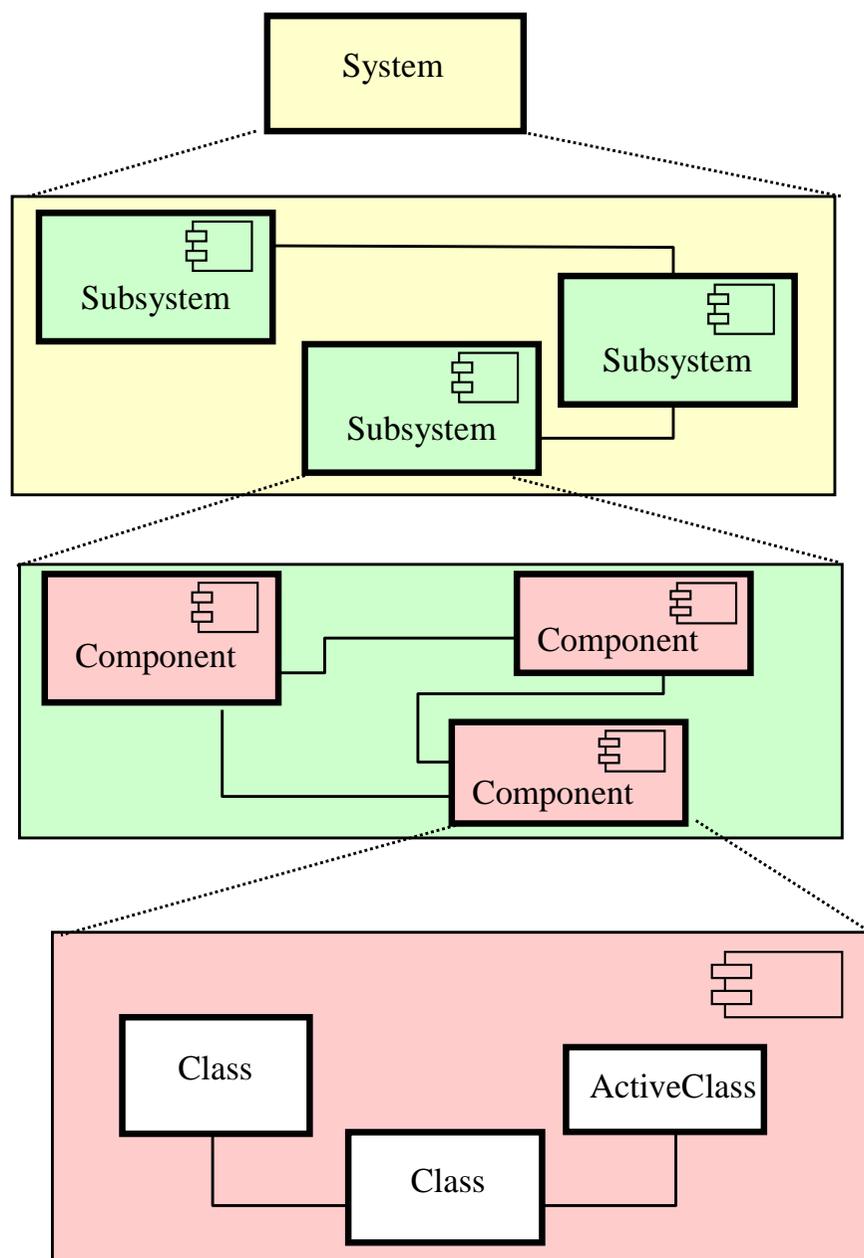
● **Componente**

- Parte modular de la **arquitectura física** de un sistema que oculta su implementación tras un conjunto de interfaces externas.
- Define su comportamiento en base a **interfaces** requeridas y ofertadas
 - Reemplazables
- El sistema se define en base a componentes conectados entre sí.
- Los componentes pueden ser de granularidad variable





Elementos - Jerarquía de elementos estructurales





Elementos - Estructurales

- **Artefacto**

- Parte **física** y reemplazable de un sistema que contiene información física (bits).
- Es utilizada o generada en el proceso de desarrollo
- Hay diferentes artefactos de despliegue:
 - código fuente, ejecutables, scripts, etc.



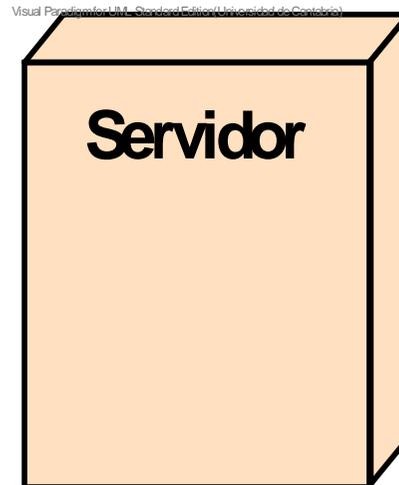
- **Especializaciones** de artefacto: Aplicaciones, Documentos, Archivos, Bibliotecas, Páginas, Tablas.



Elementos - Estructurales

- **Nodo**

- Elemento físico que existe en tiempo de ejecución y representa un **recurso computacional**.
- En un nodo pueden residir un conjunto de artefactos.
- Sirven para describir las **plataformas** en las que se ejecutan las aplicaciones





Elementos – De Comportamiento

- Son las partes **dinámicas** de los modelos UML.
- Equivalen a los **verbos** de un modelo.
- Representan comportamiento en el tiempo y el espacio.
- Suelen estar conectados semánticamente a elementos estructurales.
- Hay tres tipos:

- Interacción
- Máquina de Estados
- Actividad

Énfasis en

conjunto de objetos que interactúan

ciclo de vida de un objeto

flujo entre pasos, sin mirar qué

objeto ejecuta cada paso



Elementos - De Comportamiento

● Interacción

- Comportamiento que comprende un **conjunto de mensajes** intercambiados entre un **conjunto de objetos**, dentro de un contexto particular, para un propósito específico.
- Sirven para modelar el comportamiento de una sociedad de objetos, o una operación individual.
- Además de a los objetos implicados, involucran:
 - **Mensajes**,
 - Acciones y
 - Enlaces (conexiones entre objetos).

—————→
dibujar



Elementos - De Comportamiento

● Máquina de Estados

- Comportamiento que especifica las **secuencias de estados** por las que pasa un objeto o una interacción durante su vida, en respuesta a **eventos**, junto con sus **reacciones** a dichos eventos.
- Sirven para especificar el comportamiento de una clase individual o una colaboración de clases.
- Involucran a:
 - **Estados**,
 - Transiciones (flujo de un estado a otro),
 - Eventos (que disparan una transición) y
 - Actividades (respuesta a una transición)



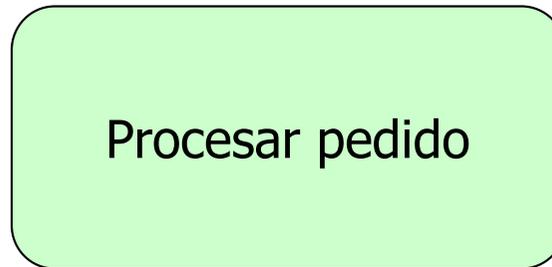
Esperando



Elementos - De Comportamiento

- **Actividad**

- Comportamiento que especifica la **secuencia de pasos** que ejecuta un proceso computacional.
- Una **acción** es un paso de una actividad.

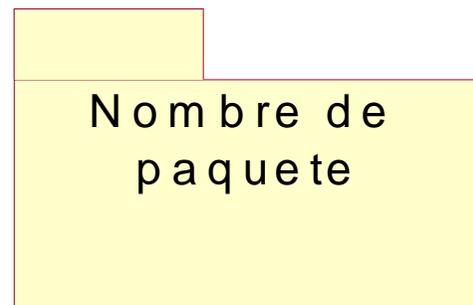


- El símbolo es el mismo que para estado. Se distinguen por el contexto.



Elementos – De Agrupación

- Son las partes **organizativas** de los modelos UML.
- Son las “cajas” en las que puede dividirse un modelo.
- Hay un tipo principal: **Paquete**
 - Mecanismo de propósito general para **organizar el propio diseño**
 - Un paquete puede incluir elementos estructurales, de comportamiento y otros paquetes.
 - Un paquete es puramente conceptual (sólo existe en tiempo de desarrollo)





Elementos – De Agrupación

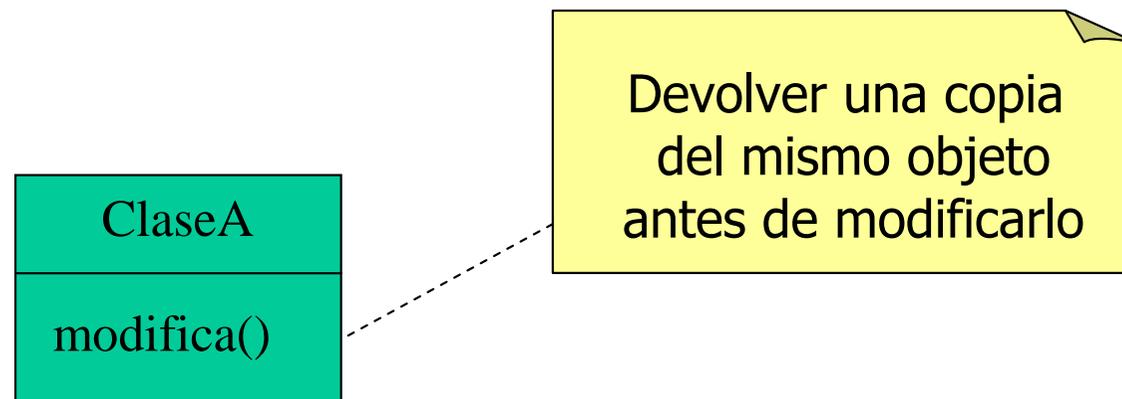
- **Paquete**

- Es recomendable que el contenido sea una colección de elementos UML relacionados de forma lógica.
- Se pueden utilizar en cualquier tipo de diagrama UML.
- Un paquete puede contener otros paquetes, sin límite de anidamiento, pero cada elemento pertenece a (está definido en) sólo un paquete.
- La visibilidad de los elementos incluidos en un paquete puede controlarse para que algunos sean visibles fuera del paquete mientras que otros permanezcan ocultos.



Elementos – De Anotación

- Son las partes **explicativas** de los modelos UML.
- Son comentarios que se añaden para describir, clarificar y hacer observaciones.
- Hay un tipo principal:
 - **Nota**
 - Símbolo para mostrar restricciones y comentarios asociados a un elemento o colección de elementos.
 - Se usan para aquello que se muestra mejor en forma textual (comentario) o formal (restricción).





Relaciones

- Una **relación** es una conexión entre elementos estructurales.
- Existen cuatro tipos de **relaciones** en UML2:
 - **Dependencia**
 - **Asociación**
 - **Generalización**
 - **Realización**
- Hay dos tipos especiales de asociación:
 - **Agregación**
 - **Composición**



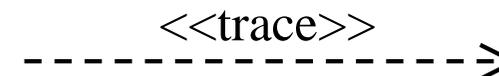
Relaciones - Dependencia

- **Dependencia**

- Relación semántica en la cual un cambio a un elemento (independiente) puede afectar a la semántica del otro elemento (dependiente).



- Caso frecuente: **uso entre clases**
 - Una clase utiliza a otra como argumento en la signatura de una operación.
- **Especializaciones:** Refinamiento (refine), Traza (trace), extensión (extend), inclusión (include),...

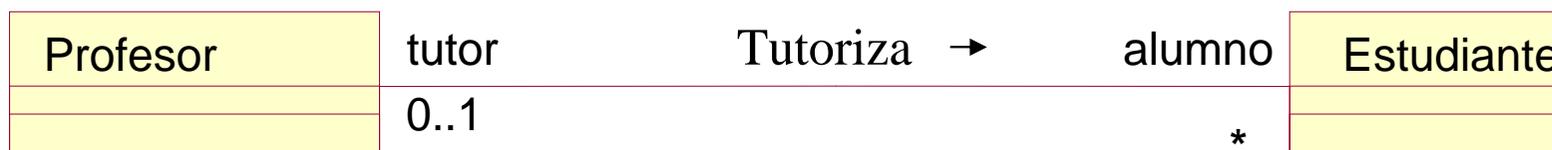




Relaciones – Asociación

● Asociación

- Relación estructural entre clases que describe un conjunto de enlaces (conexiones entre objetos que son instancias de las clases).
- Indica que los objetos están relacionados durante un periodo de tiempo continuado
- Puede tener un nombre que la describe (verbo, con dirección de lectura).
- Puede tener otros adornos:
 - rol que describe el papel específico que una clase juega en una asociación,
 - multiplicidad para clase participante.

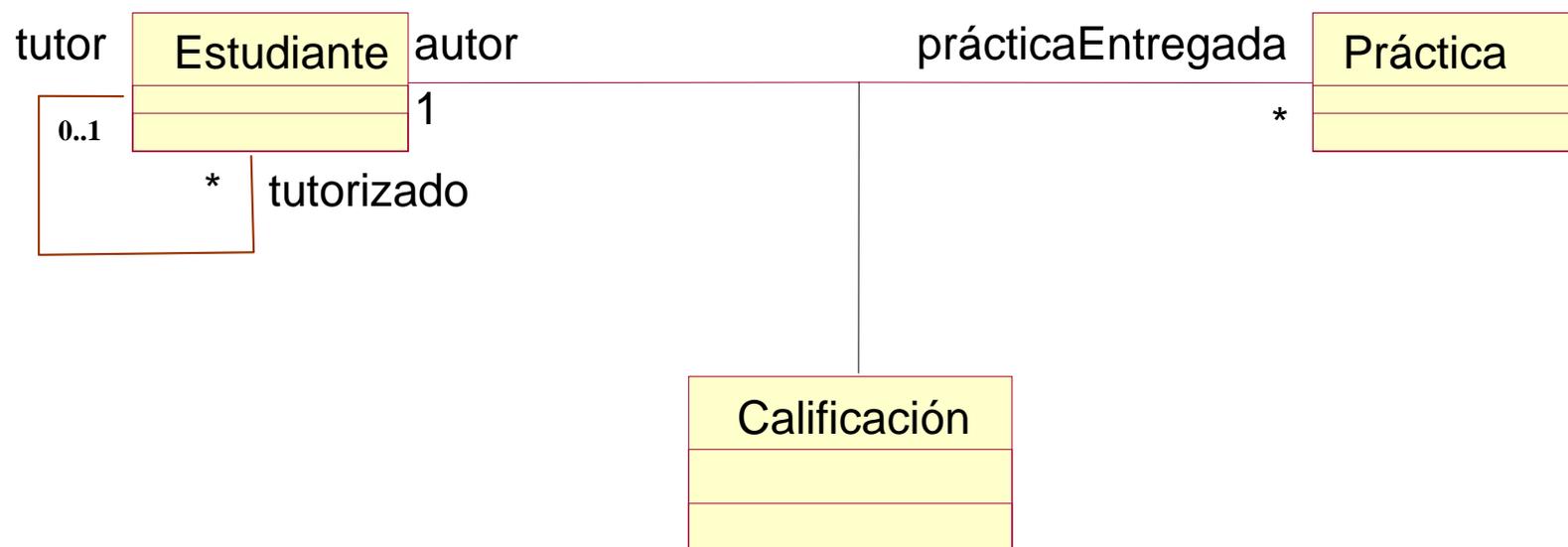




Relaciones - Asociación

• Asociación

- Ejemplo de **clase de asociación** y asociación **reflexiva**.





Relaciones – Agregación y Composición

- **Agregación y Composición**

- Modelar objetos complejos en base a **relaciones todo – parte**.

- **Agregación**

- Relación dinámica: el tiempo de vida del objeto que se agrega es independiente del objeto agregador.
- El objeto agregador utiliza al agregado para su funcionamiento.
 - SIMILAR parámetro pasado “por referencia”.

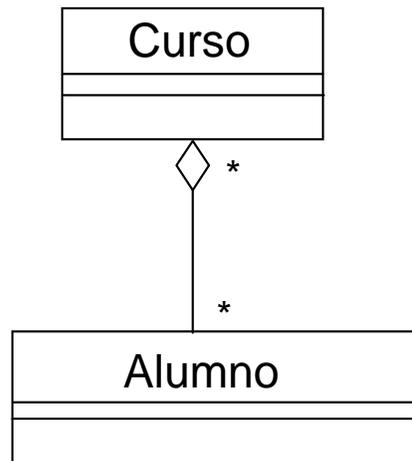
- **Composición**

- Relación estática: el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del objeto compuesto.
- El objeto compuesto se construye a partir del objeto incluido.
 - SIMILAR parámetro pasado “por valor”.

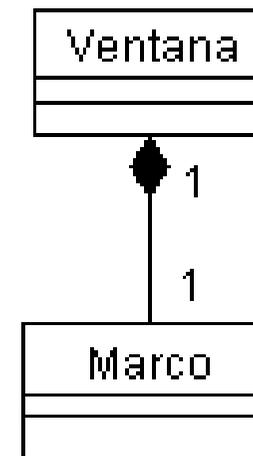


Relaciones – Agregación y Composición

- Ejemplos



Agregación
(por referencia)



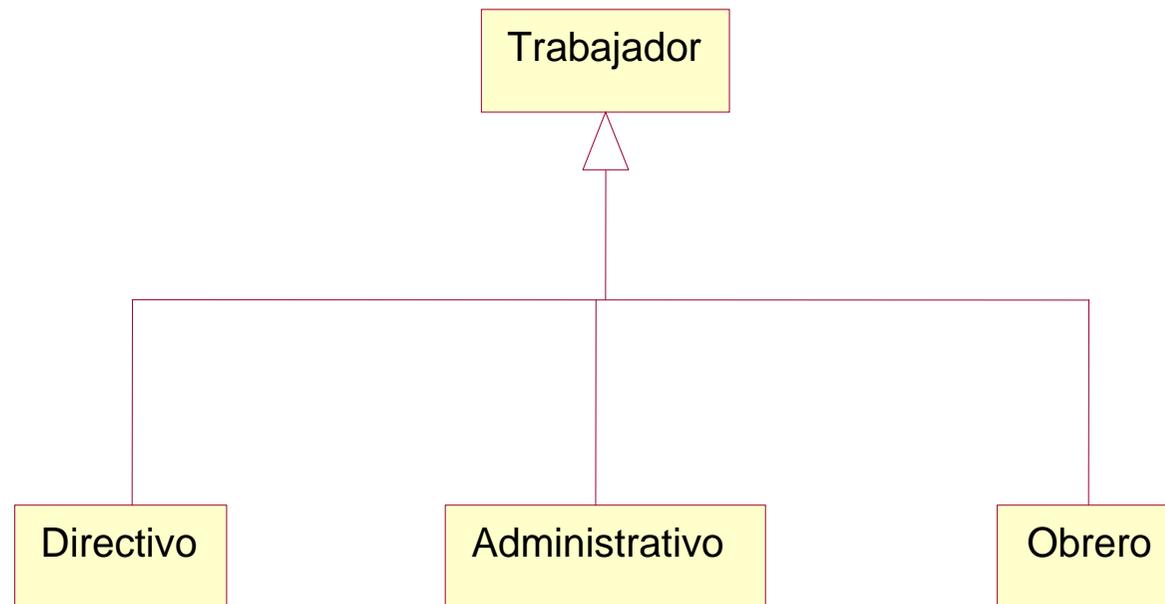
Composición
(por valor)



Relaciones - Generalización

- **Generalización**

- Relación de especialización/generalización en la que el elemento especializado (hijo) extiende a la especificación del elemento generalizado (padre).
- Las subclases (hijos) comparten la estructura y el comportamiento de la superclase (padre).

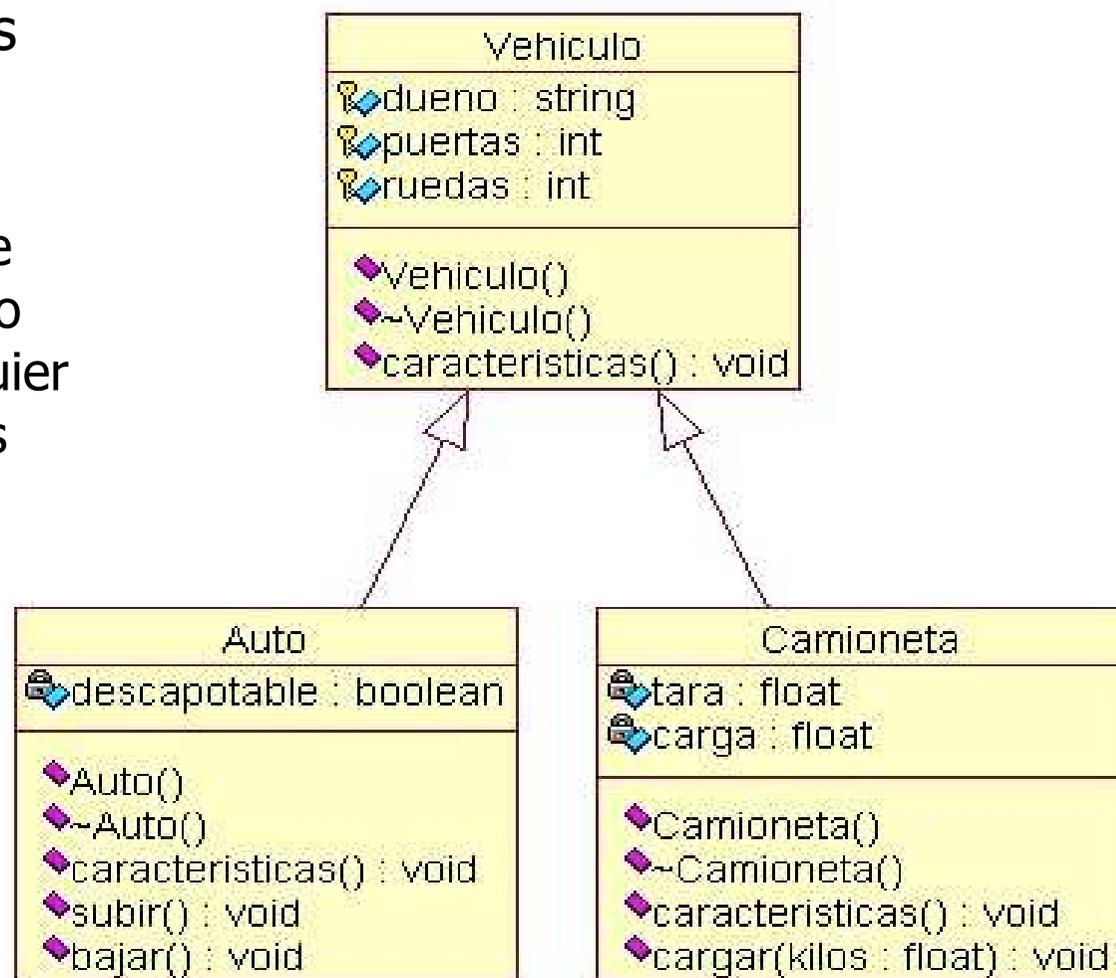




Relaciones - Generalización

• Generalización

- Una generalización da lugar al **polimorfismo** entre clases de una jerarquía de generalizaciones:
 - Un objeto de una subclase puede sustituir a un objeto de la superclase en cualquier contexto. Lo inverso no es cierto.

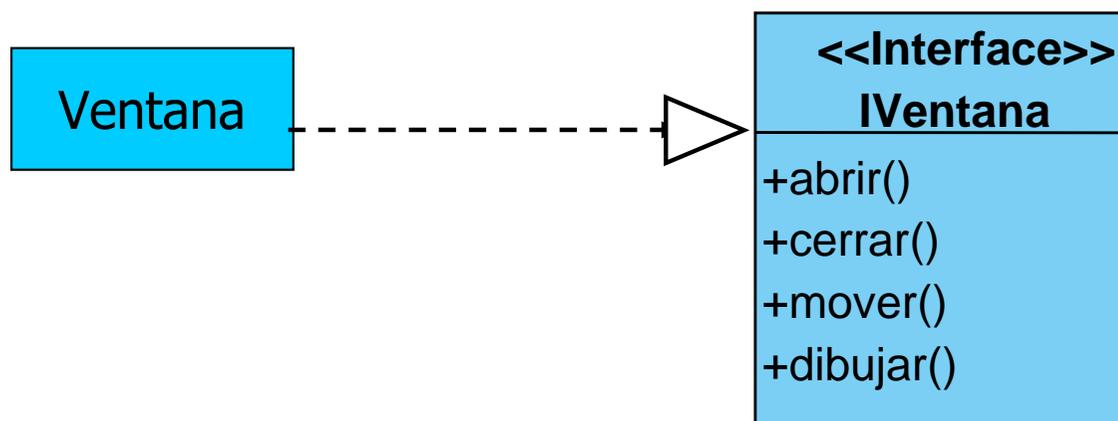




Relaciones - Realización

● Realización

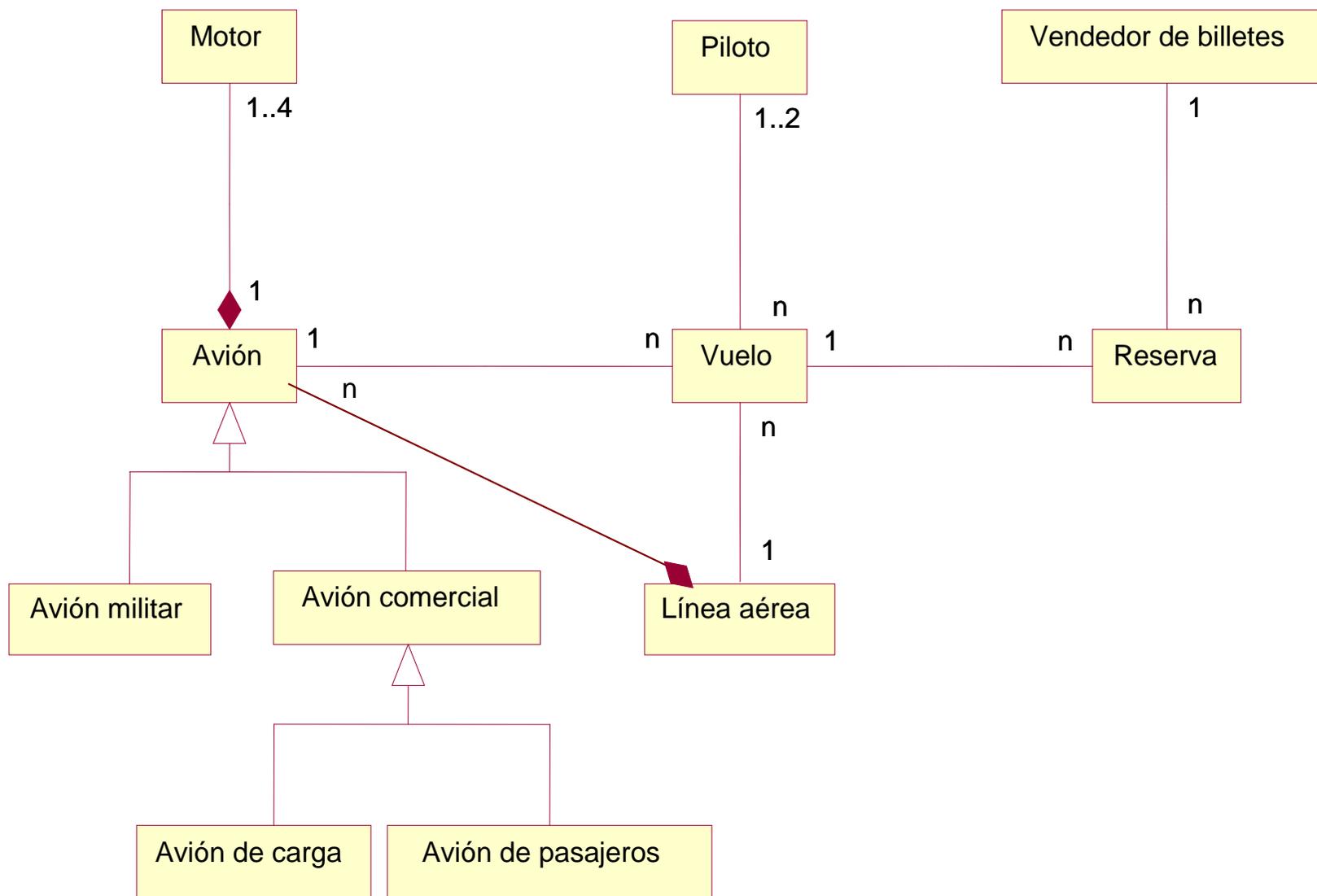
- Relación semántica entre clasificadores, donde un clasificador especifica un **contrato** que otro clasificador garantiza que cumplirá.
- Se pueden encontrar en dos casos:
 - Clases o componentes que realizan interfaces.
 - Colaboraciones que realizan casos de uso.





Relaciones - Jerarquías

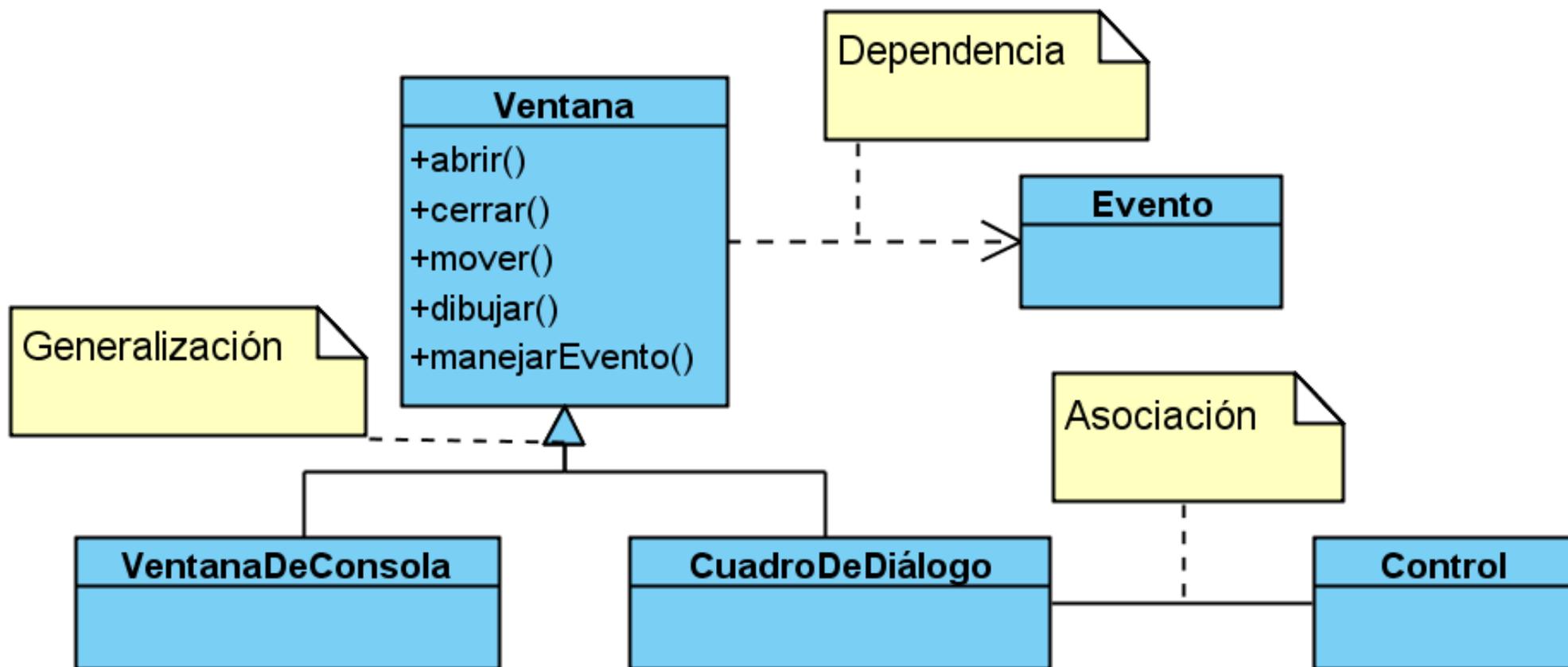
- **Jerarquías** de Relaciones





Relaciones - Jerarquías

- **Ejemplo** con generalizaciones, asociación y dependencia.





Diagramas

- Sirven para visualizar un sistema desde **diferentes perspectivas**.
- Un diagrama es una **proyección gráfica de un sistema**.
 - Vista resumida de los elementos que constituyen el sistema.
- El mismo elemento puede aparecer en varios diagramas.
- En teoría, un diagrama puede contener cualquier combinación de elementos y relaciones, sin embargo en la práctica solo un pequeño número de combinaciones tiene sentido:
 - Surgen así los tipos de diagramas de UML 2.



Diagramas

- **UML 2** incluye **13 Tipos de Diagramas:**
- **Estructurales**
(ESTÁTICA)
 - Clases
 - Objetos
 - Componentes
 - Despliegue
 - Paquetes
 - Estructura Compuesta 
- **De Comportamiento**
(DINÁMICA)
 - Casos de uso
 - Estados
 - Actividades
 - Interacción
 - Secuencia
 - Comunicación
 - Tiempos 
 - Revisión de Interacciones 



nuevo en UML 2.0



Diagramas - Estructurales

- Los **diagramas estructurales** de UML 2 sirven para visualizar, especificar, construir y documentar los **aspectos estáticos** de un sistema.
- Se organizan en base a los principales grupos de elementos que aparecen al modelar (durante las diferentes fases del proceso de desarrollo).



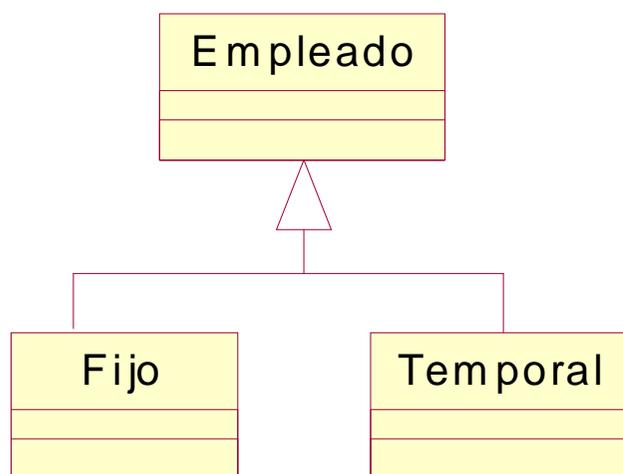
| Tipo de Diagrama | Elementos Centrales |
|----------------------|--|
| Clases | Clases, Interfaces, Colaboraciones |
| Componentes | Componentes |
| Estructura Compuesta | Estructura Interna de Clase o Componente |
| Objetos | Objetos |
| Paquetes | Paquetes |
| Despliegue | Nodos, Artefactos |



Diagramas - Estructurales

- **De Clases:**

- Muestran un conjunto de clases, interfaces y colaboraciones, así como las relaciones entre ellos.
- Son los diagramas más comunes en el análisis y diseño de un sistema:
 - Explorar conceptos del dominio (**Modelo de Dominio**).
 - Analizar requisitos (**Modelo de Análisis / Conceptual**).
 - Describir el diseño detallado de un software OO (**Modelo de Diseño**).
- Abarcan la **vista de diseño estática** de un sistema.
 - Con clases activas cubren la **vista de interacción estática**.

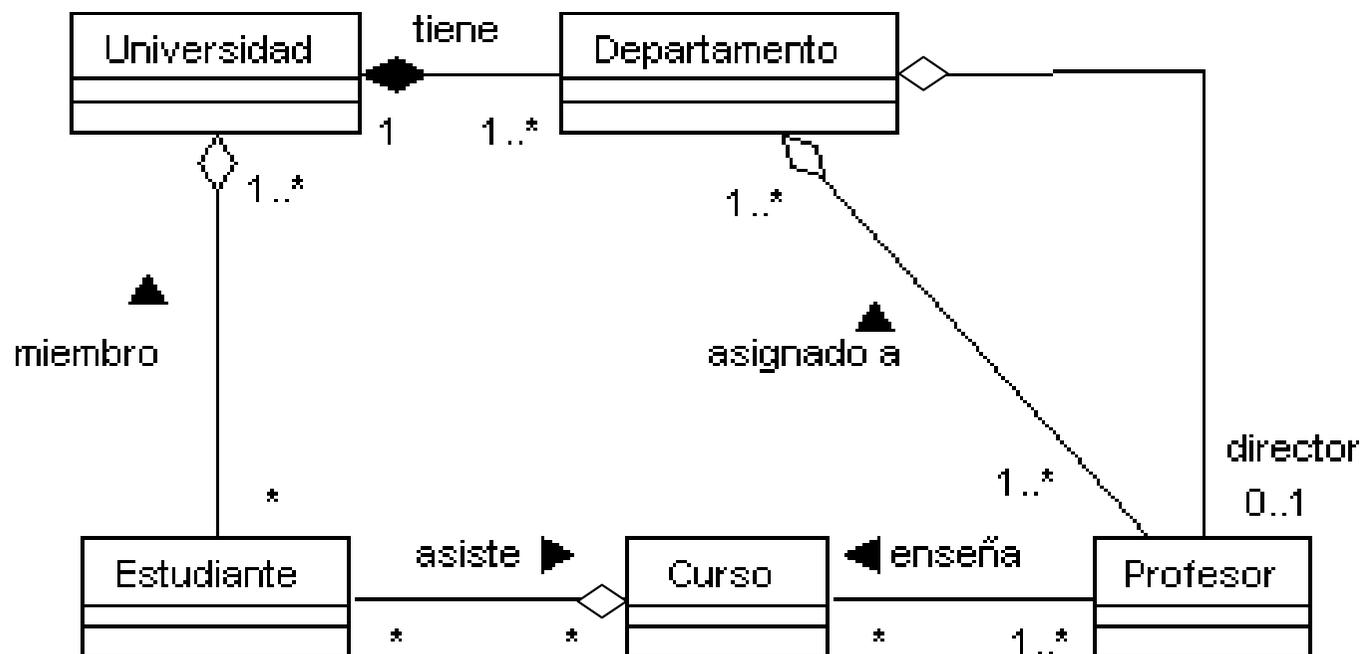




Diagramas - Estructurales

● De Clases:

- Principalmente, un diagrama de clases contiene:
 - **Clases** (con atributos, operaciones y visibilidad).
 - **Relaciones**: Dependencia, Generalización, Asociación, Agregación y Composición.

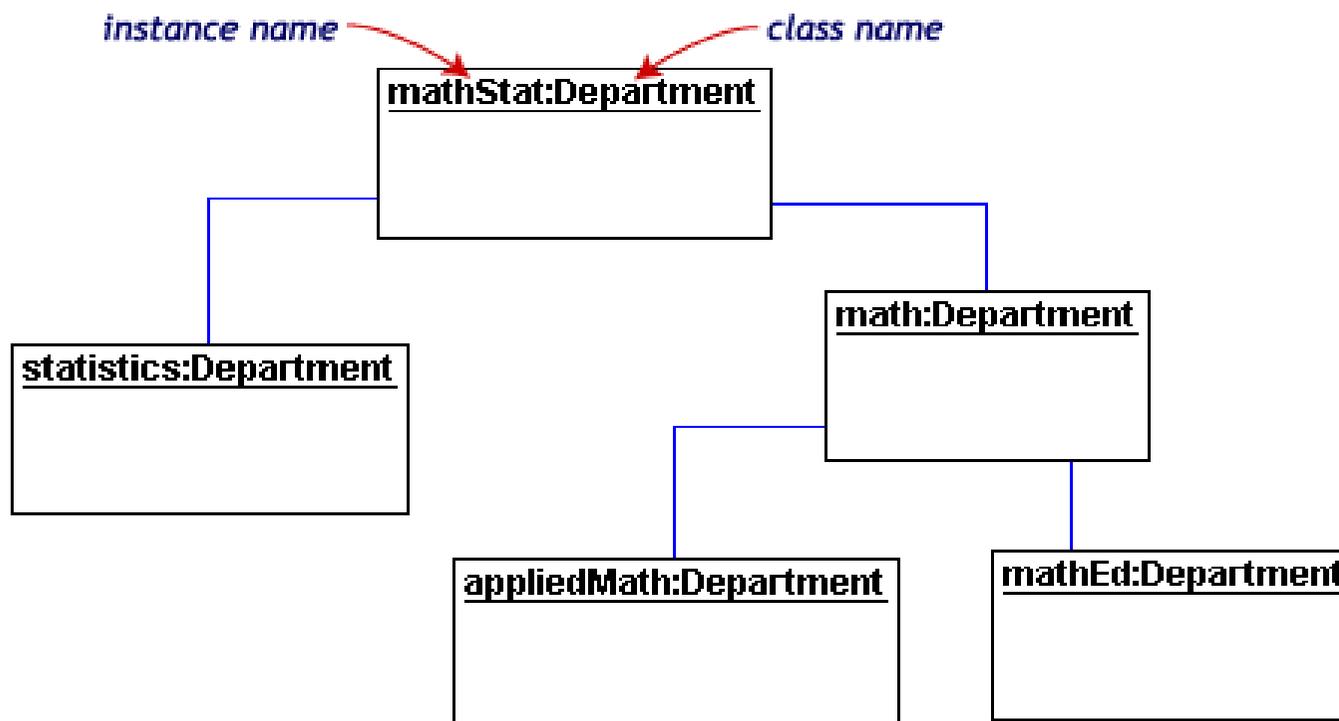




Diagramas - Estructurales

- **De Objetos:**

- Muestra un conjunto de objetos y sus relaciones.
- Representan instantáneas estáticas de instancias de los elementos existentes en los diagramas de clases.
- Describen la **vista de diseño estática** pero desde el punto de vista de **casos reales**

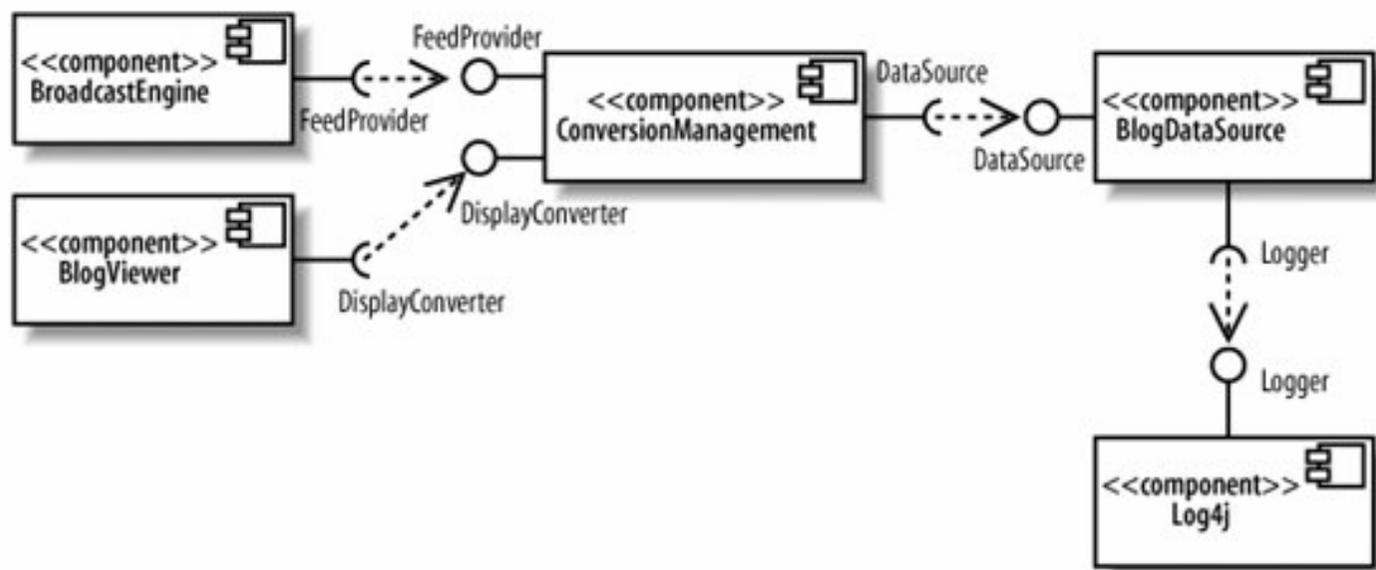




Diagramas - Estructurales

- **De Componentes:**

- Describen la estructura del software mostrando la **organización y las dependencias** entre un conjunto de **componentes**.
- Pueden representar la encapsulación de un componente con sus interfaces, puertos y estructura interna (posiblemente formada por otros componentes anidados y conectores).
- Cubren la **vista de implementación estática** del diseño de un sistema.

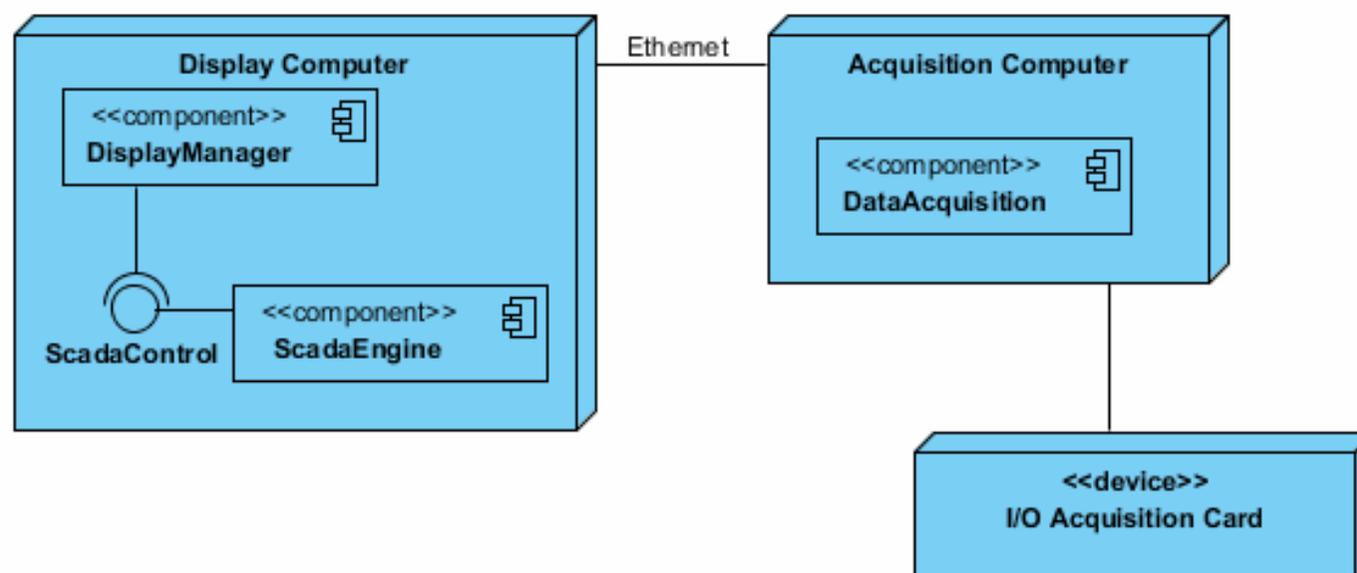




Diagramas - Estructurales

- **De Despliegue:**

- Muestran un conjunto de **nodos** y sus relaciones.
- Describen la **vista de despliegue estática** de una arquitectura.
- Cada nodo (hardware) suele albergar uno o más componentes.
- Muestran el hardware, el software y el middleware usado para conectar las máquinas.

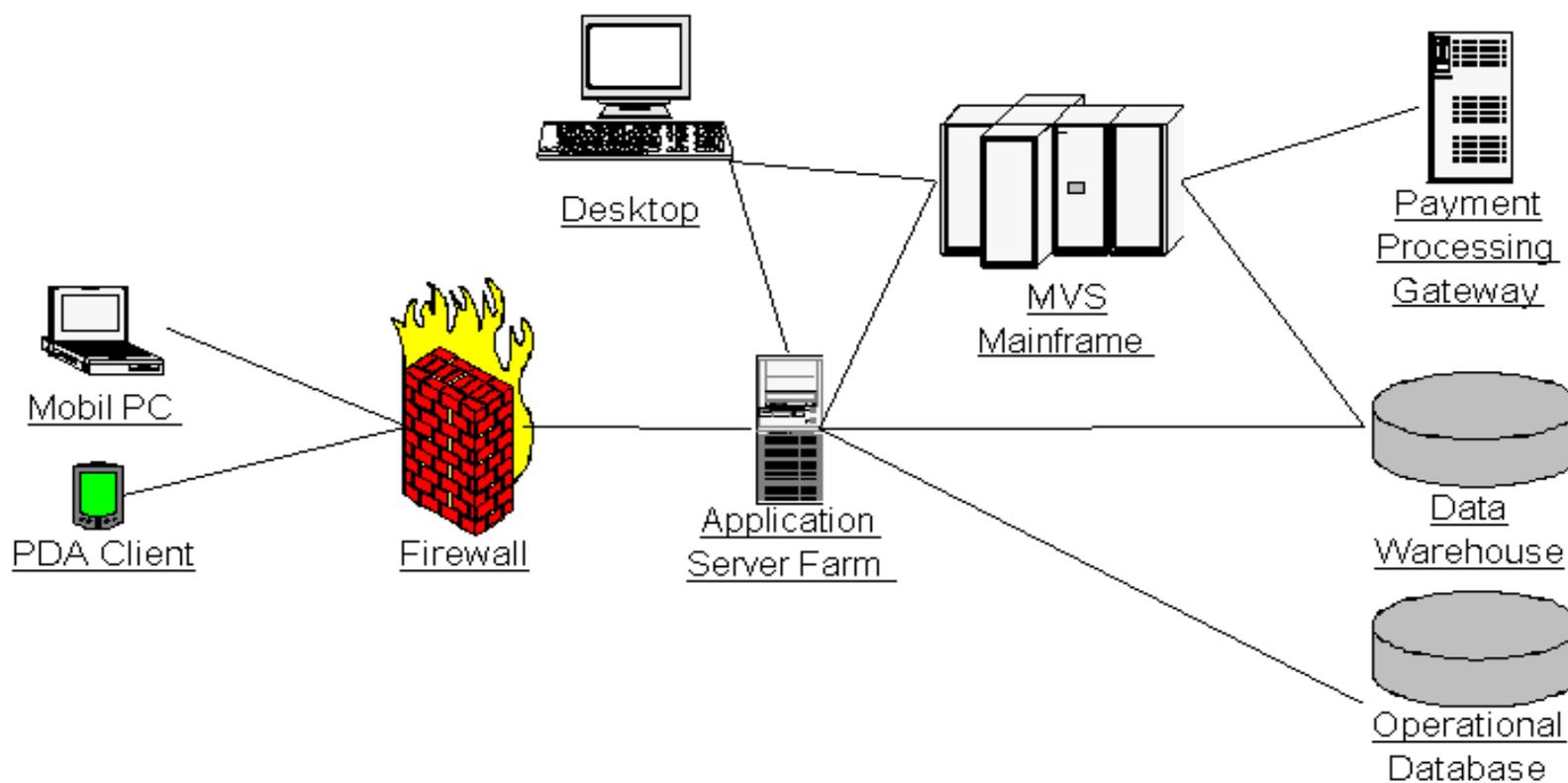




Diagramas - Estructurales

- **De Despliegue:**

- Mediante **iconos especializados** se puede precisar la naturaleza de los nodos como constituyentes físicos (dispositivos, archivos, bases de datos, etc.) de un sistema.

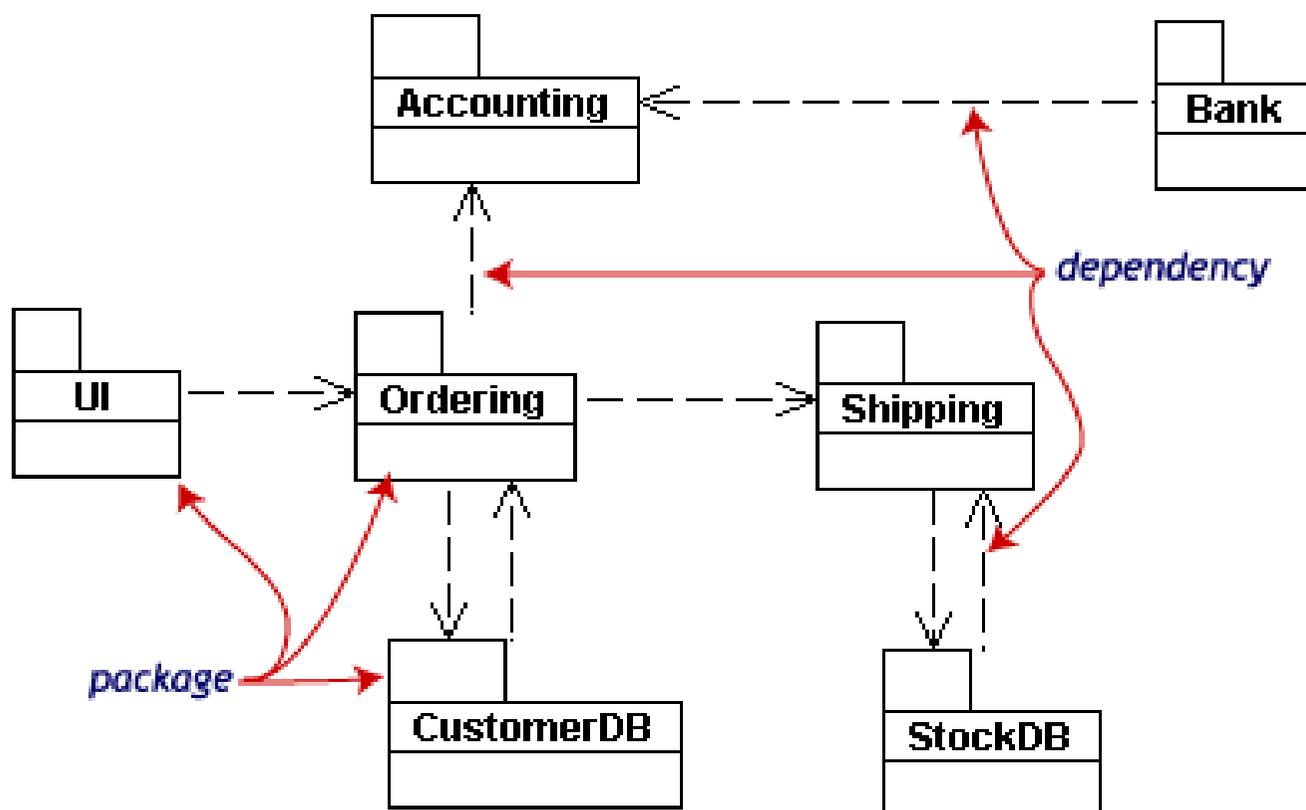




Diagramas - Estructurales

- **De Paquetes:**

- Muestran la descomposición del propio modelo en unidades organizativas (paquetes) y sus dependencias.
- Sirven para simplificar los diagramas de clases complejos, permitiendo el agrupamiento de los clasificadores en paquetes.

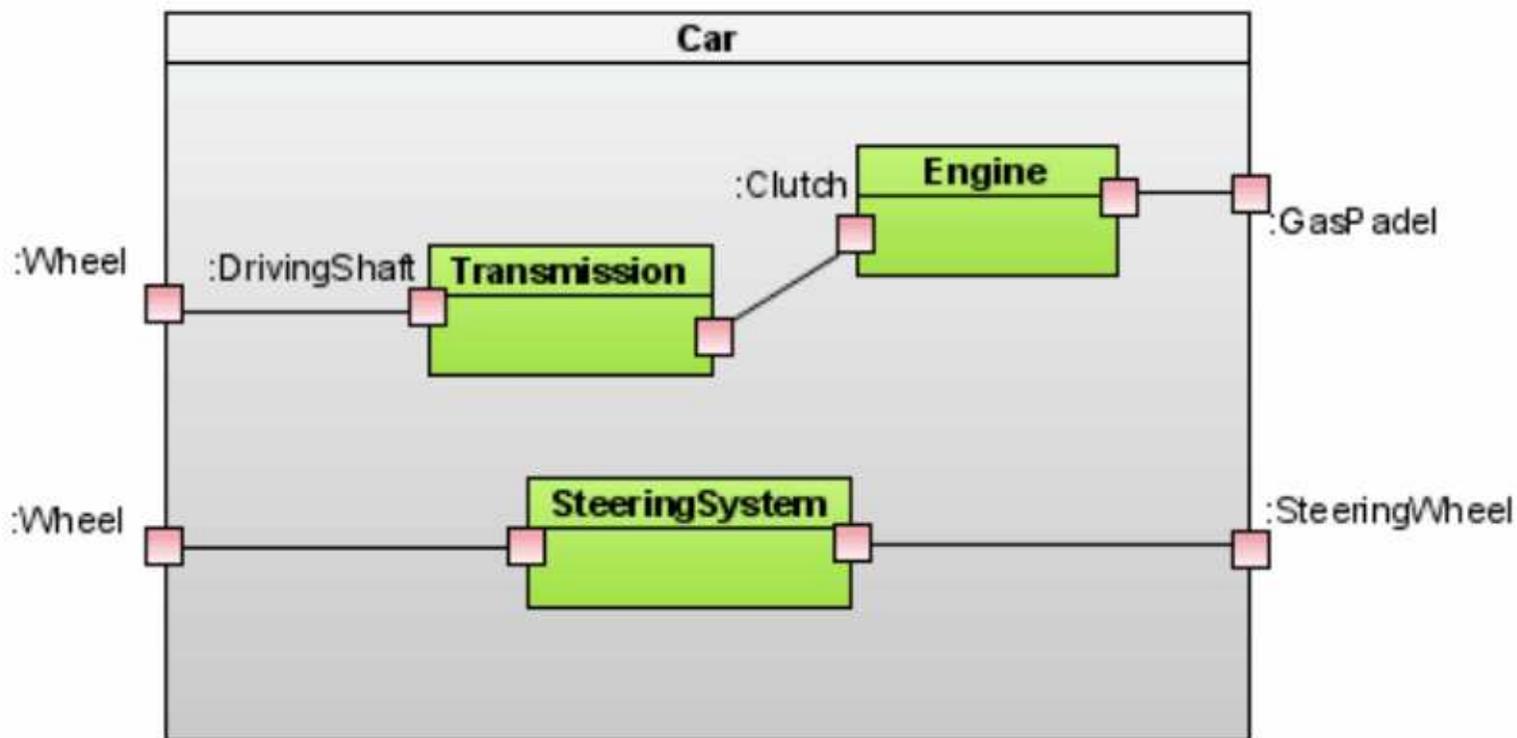




Diagramas - Estructurales

- **De Estructura Compuesta:**

- Muestran la **estructura interna** (incluyendo partes y conectores) de un clasificador estructurado o una colaboración.
- Muy parecidos a los diagramas de componentes.





Diagramas - De Comportamiento

- Los **diagramas de comportamiento** de UML 2 sirven para visualizar, especificar, construir y documentar los **aspectos dinámicos** de un sistema.
- Se organizan en base a las formas en que se puede modelar la dinámica de un sistema.



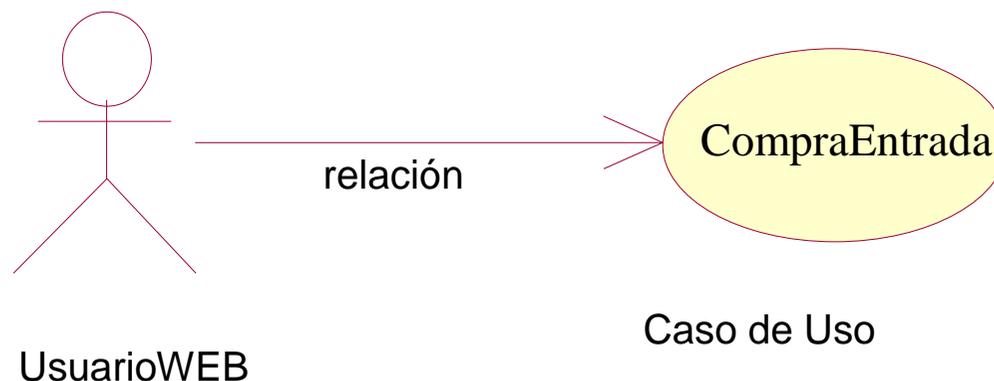
| Tipo de Diagrama | Forma de Modelar la Dinámica |
|---------------------------|---|
| Casos de Uso | Especificar el comportamiento externo |
| Secuencia | Ordenación temporal de los mensajes |
| Comunicación | Organización estructural de los objetos y envían y reciben mensajes |
| Estados | Estado cambiante de objetos dirigidos por eventos |
| Actividades | Flujo de control de actividades |
| Tiempos | Tiempo real de los mensajes y estados |
| Revisión de Interacciones | Vista general de las interacciones |



Diagramas – De Comportamiento

- **De Casos de Uso:**

- Muestran un conjunto de **casos de uso** y **actores** (tipo especial de clase) y sus relaciones.
- Cubren la **vista de casos de uso estática** de un sistema.
- Son importantes en el modelado y organización del comportamiento de un sistema. Modelan el comportamiento del sistema desde el punto de vista externo.
- Juegan un papel clave en metodologías muy usadas (desarrollo dirigido por casos de uso).

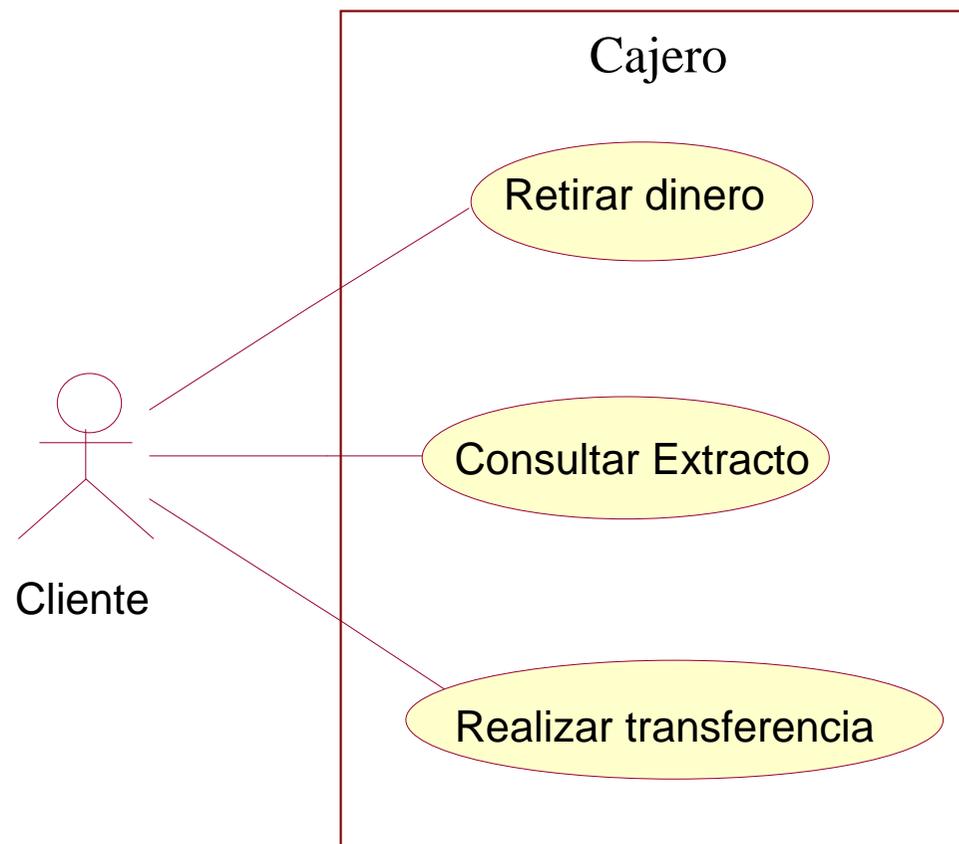




Diagramas – De Comportamiento

- **De Casos de Uso:**

- **Casos de Uso** es una técnica para capturar información respecto de los servicios que un sistema proporciona a su entorno (captura y especificación de requisitos).

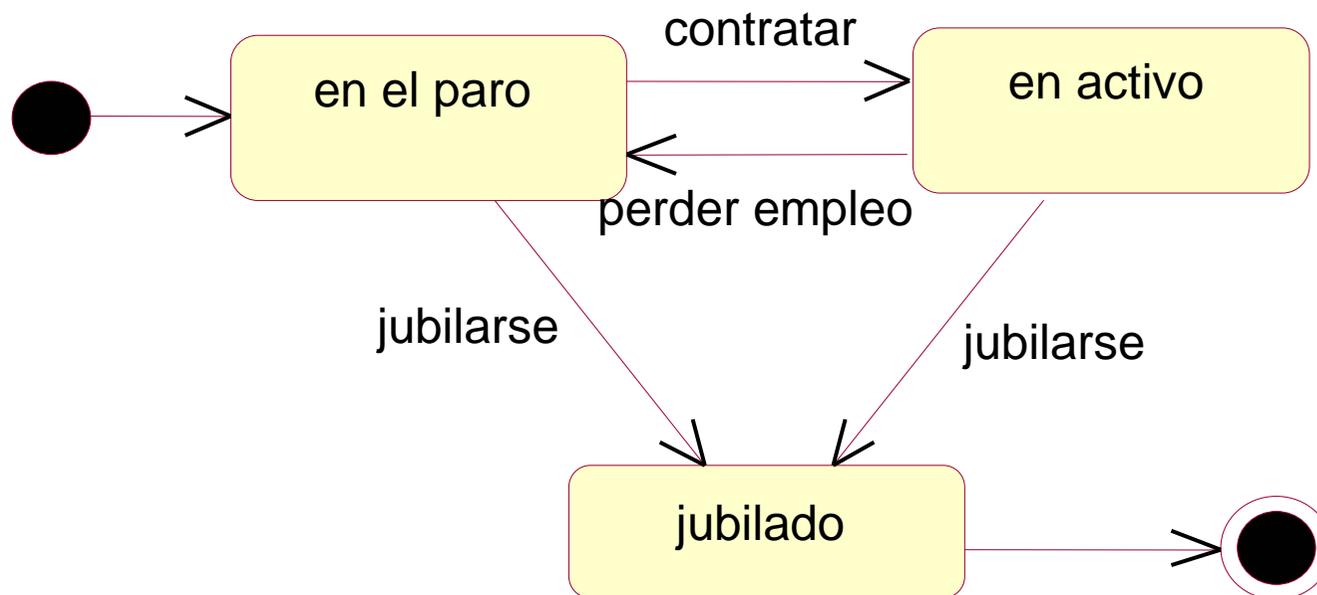




Diagramas – De Comportamiento

● De Estados:

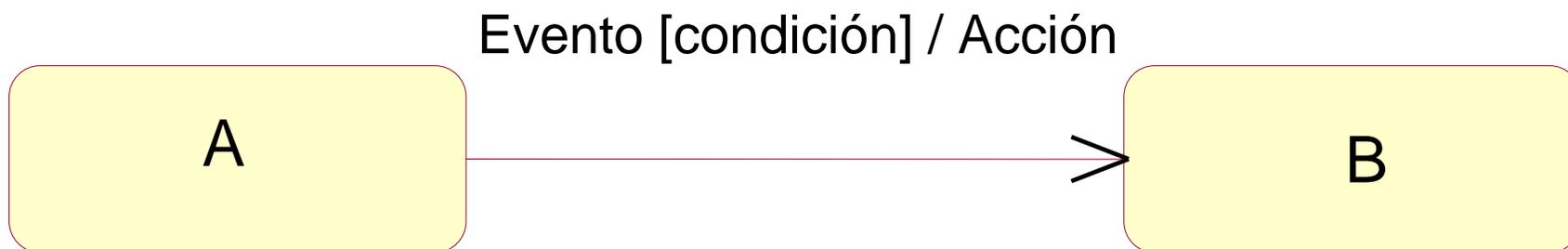
- Muestran **máquinas de estados**, que constan de:
 - estados, transiciones, eventos y acciones.
- Cubren la **vista dinámica de un objeto**.
- Son especialmente importantes en el modelado de una clase o colaboración con comportamiento significativo.
- Resaltan el comportamiento dirigido por eventos de un objeto.





Diagramas – De Comportamiento

- **De Estados:**
 - **Estados y Transiciones:**



Tanto el evento como la acción se consideran instantáneos

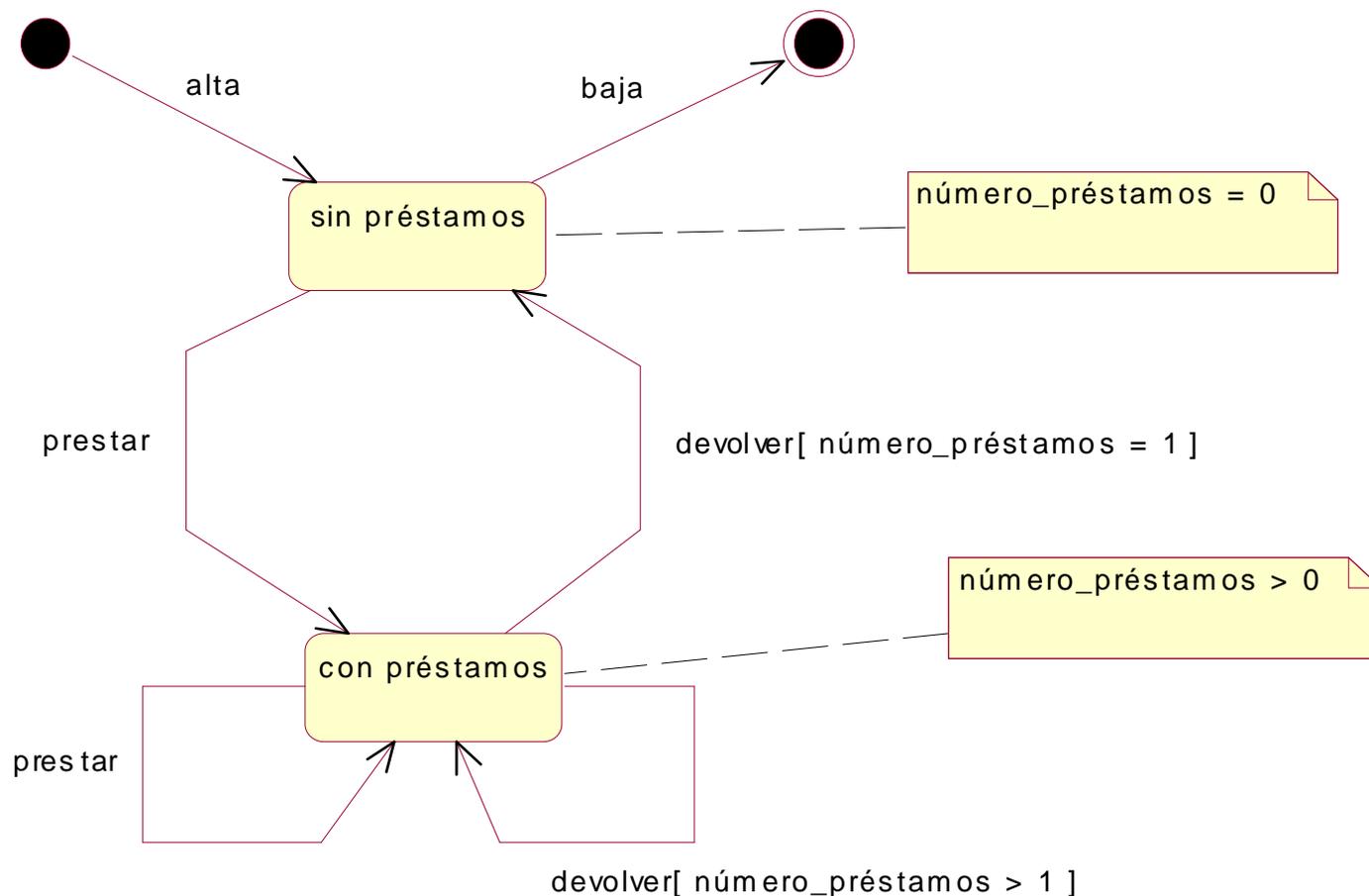


Diagramas – De Comportamiento

De Estados:

Ejemplo.

| Socio |
|---|
| número : int nombre : char[50] número_prestamos : int = 0 |
| alta() baja() prestar(código_libro : int, fecha : date) devolver(código_libro : int, fecha : date) |





Diagramas – De Comportamiento

- **De Actividad:**

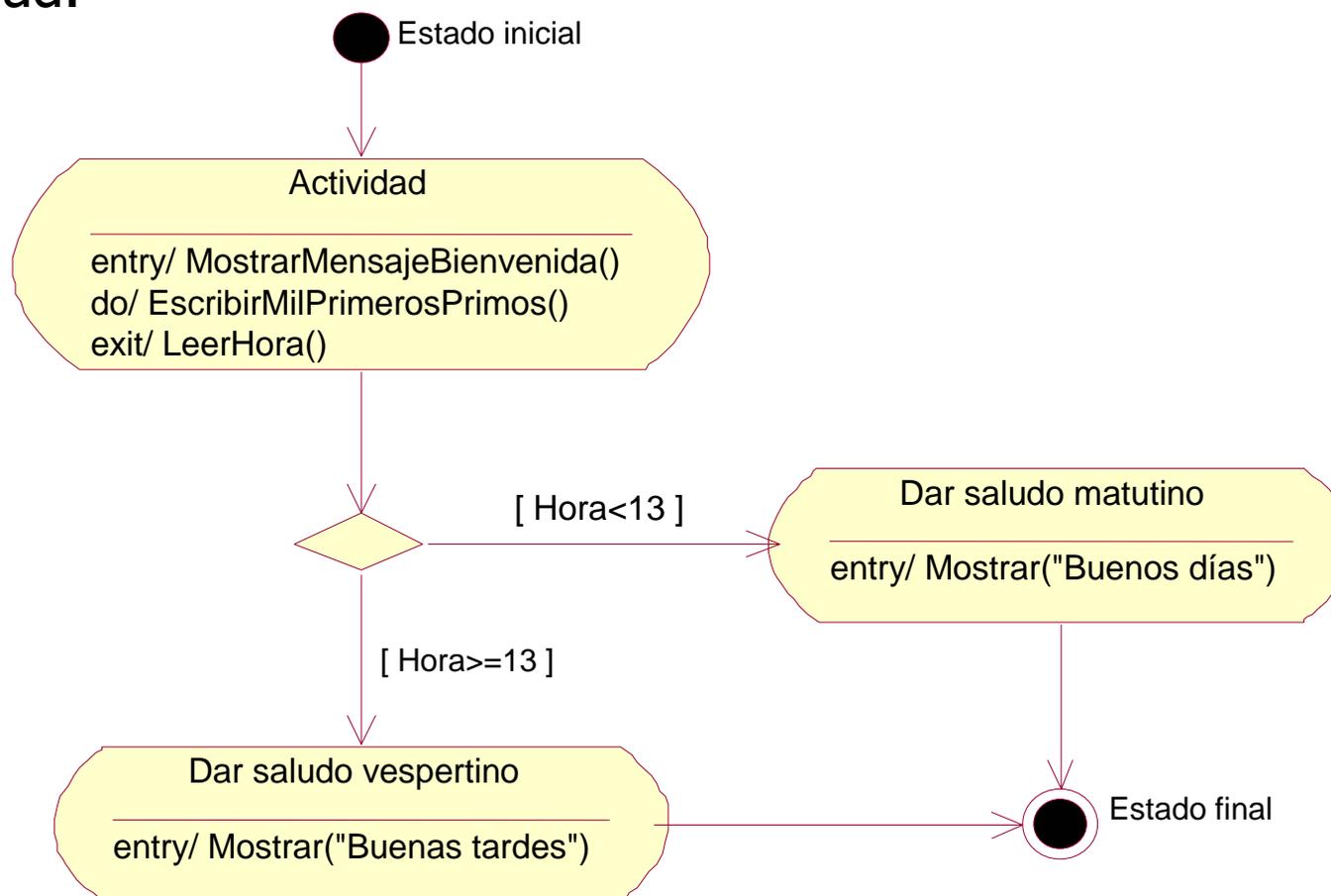
- Muestran el flujo paso a paso de una computación (proceso, flujo de control o flujo de datos).
- Una **actividad** muestra un conjunto de **acciones**, el **flujo** entre ellas y los **valores** producidos o consumidos.
- Cubren la **vista dinámica** de un sistema.
- Resaltan el flujo de control entre objetos.
- Son el equivalente en OO a los diagramas de flujo y DFDs.
- Se emplean para especificar:
 - Una operación compleja.
 - Un proceso de negocio (business process) o flujo de trabajo (workflow).
 - El proceso de negocio asociado a un caso de uso.



Diagramas – De Comportamiento

● De Actividad:

- Las actividades se enlazan por transiciones automáticas.
- Cuando una actividad termina se desencadena el paso a la siguiente actividad.

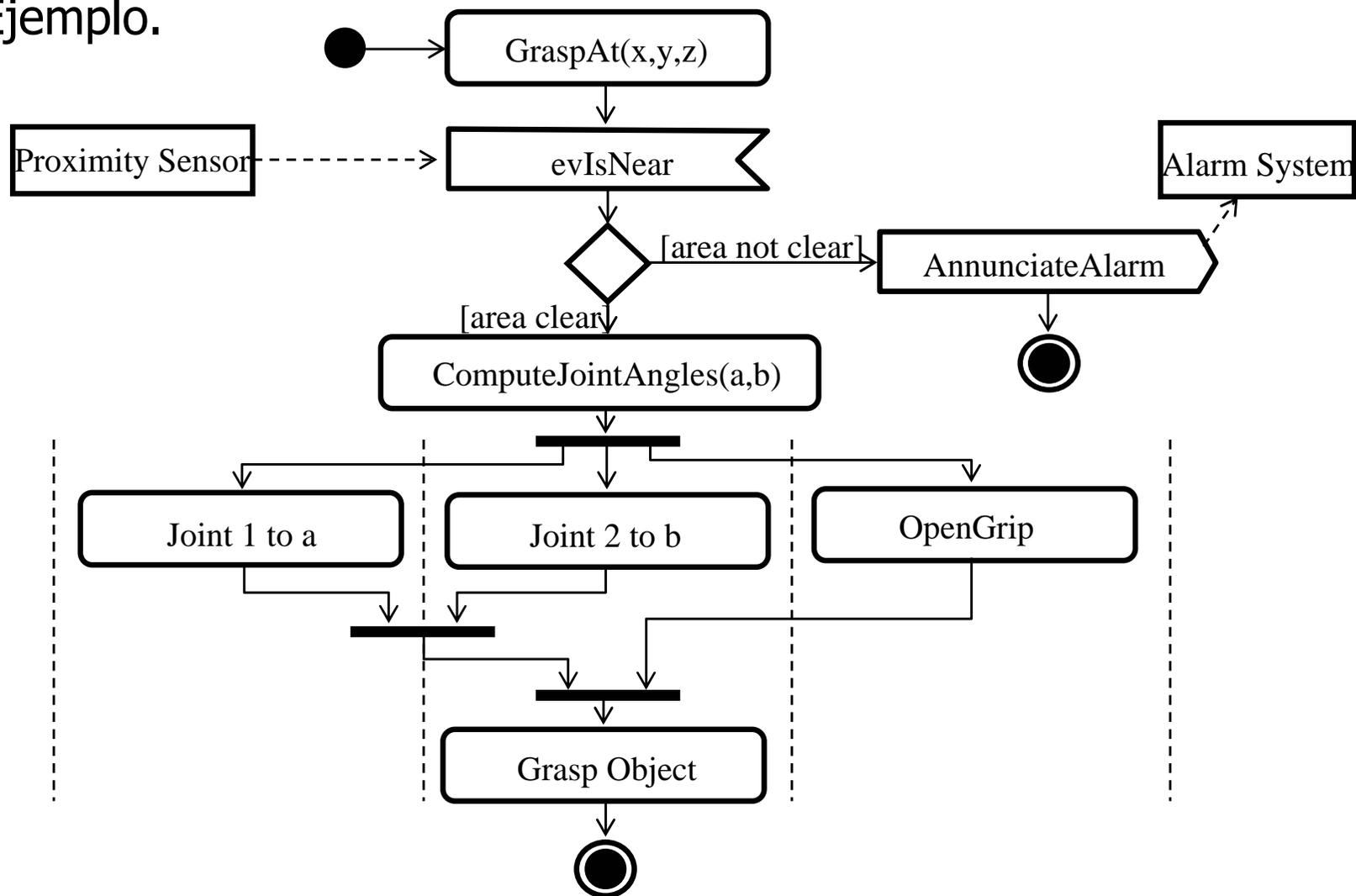




Diagramas – De Comportamiento

● De Actividad:

■ Ejemplo.





Diagramas – De Comportamiento

● Diagramas de Interacción

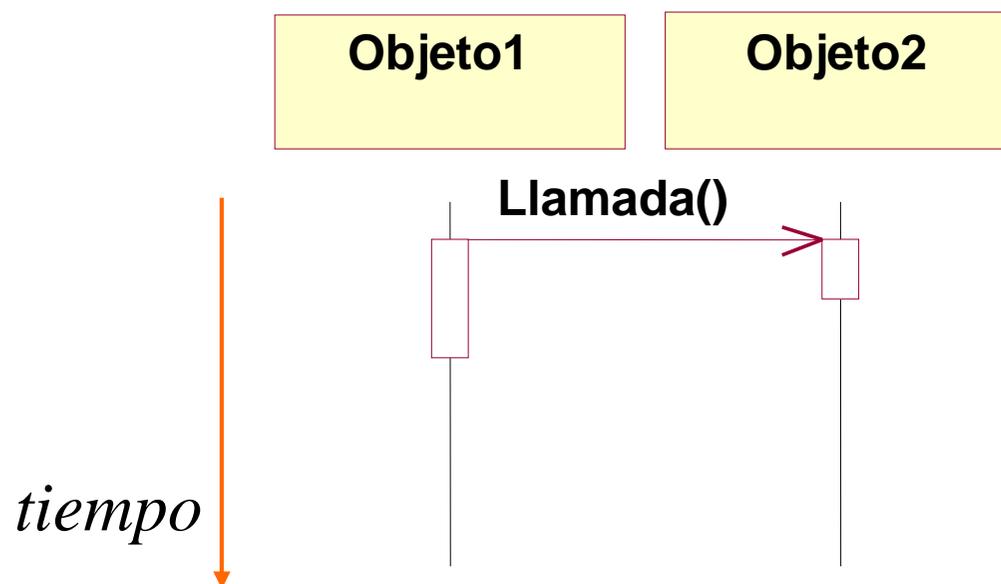
- Son un grupo especial de diagramas de comportamiento que muestran una **interacción**:
 - Conjunto de objetos o roles y mensajes que pueden ser enviados entre ellos.
- Cubren la **vista dinámica** de un sistema.
- Los objetos interactúan para realizar colectivamente los servicios ofrecidos por las aplicaciones.
- UML 2 incluye los siguientes
 - **Secuencia**
 - **Comunicación** (antiguo de Colaboración en UML 1.x)
 - **Tiempos**
 - **Revisión de las Interacciones**



Diagramas – De Comportamiento

- **De Secuencia:**

- Es un diagrama de interacción que resalta la **ordenación temporal** de los mensajes intercambiados durante la interacción.
- Presentan un conjunto de **roles** y los **mensajes** enviados y recibidos por las instancias que interpretan dichos roles.
- Habitualmente, sirven para mostrar como interaccionan unos objetos con otros en un caso de uso o un escenario de ejecución.

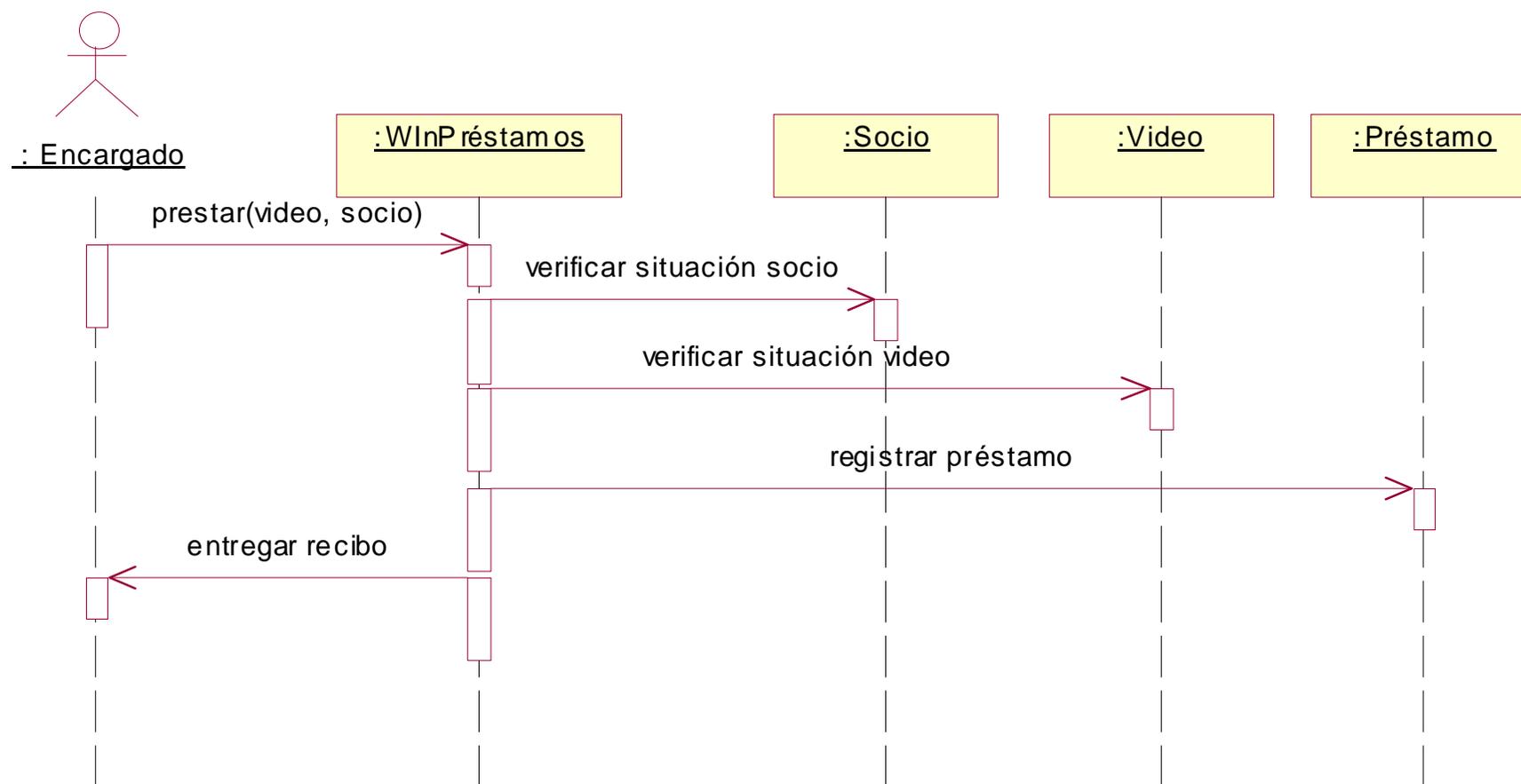




Diagramas – De Comportamiento

• De Secuencia:

■ Ejemplo.

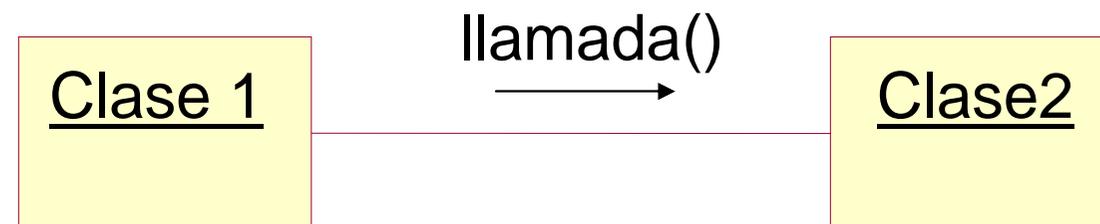




Diagramas – De Comportamiento

- **De Comunicación:**

- Es un diagrama de interacción que resalta la **organización estructural** de los objetos o roles que envían y reciben mensajes en la interacción.
- Muestran un conjunto de **objetos**, enlaces entre ellos y los **mensajes** enviados y recibidos.
- Se usan para modelar el comportamiento dinámico de un caso de uso.



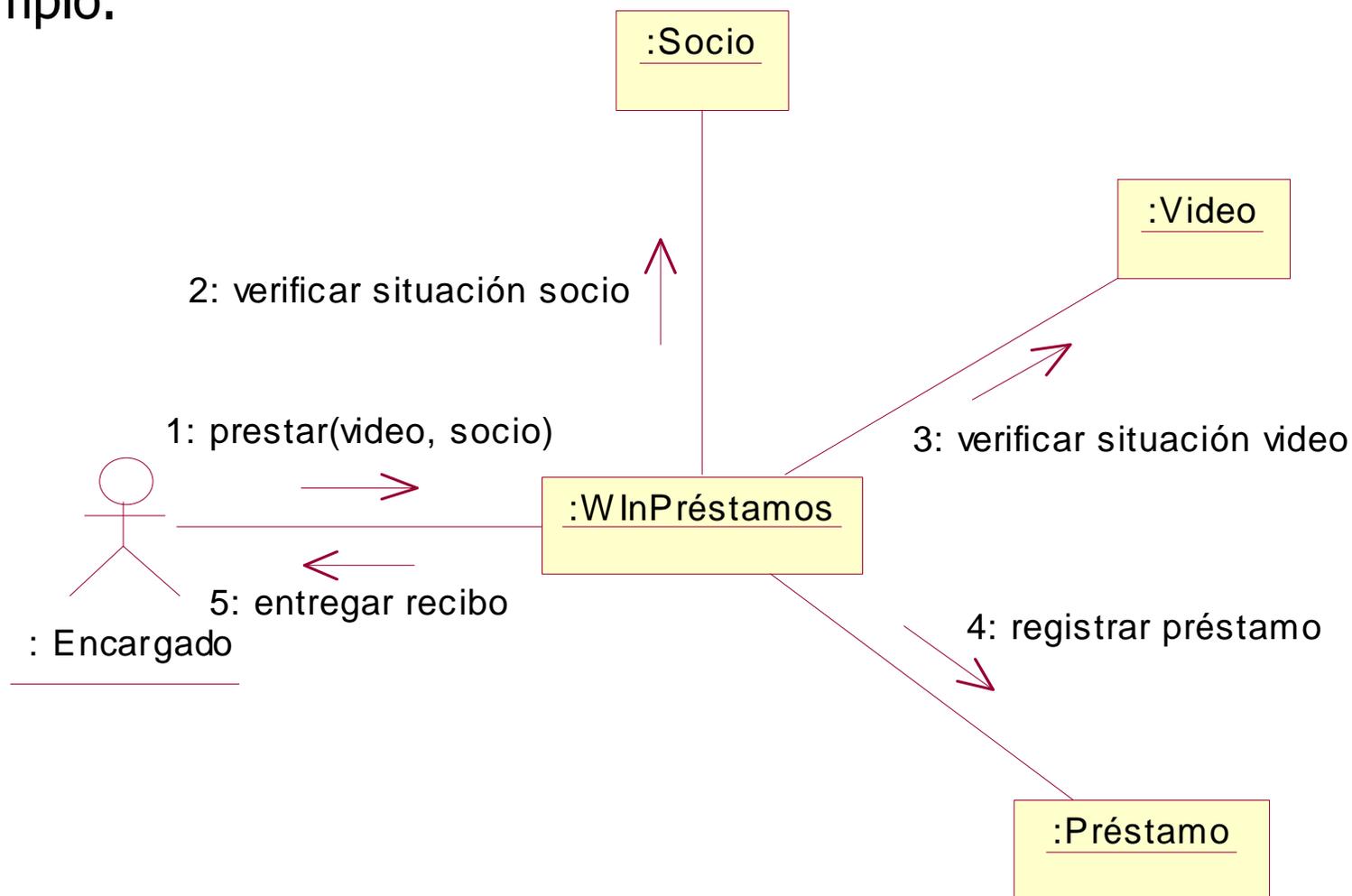
- En versiones anteriores a UML 2 se llamaban de colaboración.



Diagramas – De Comportamiento

• De Comunicación:

■ Ejemplo.



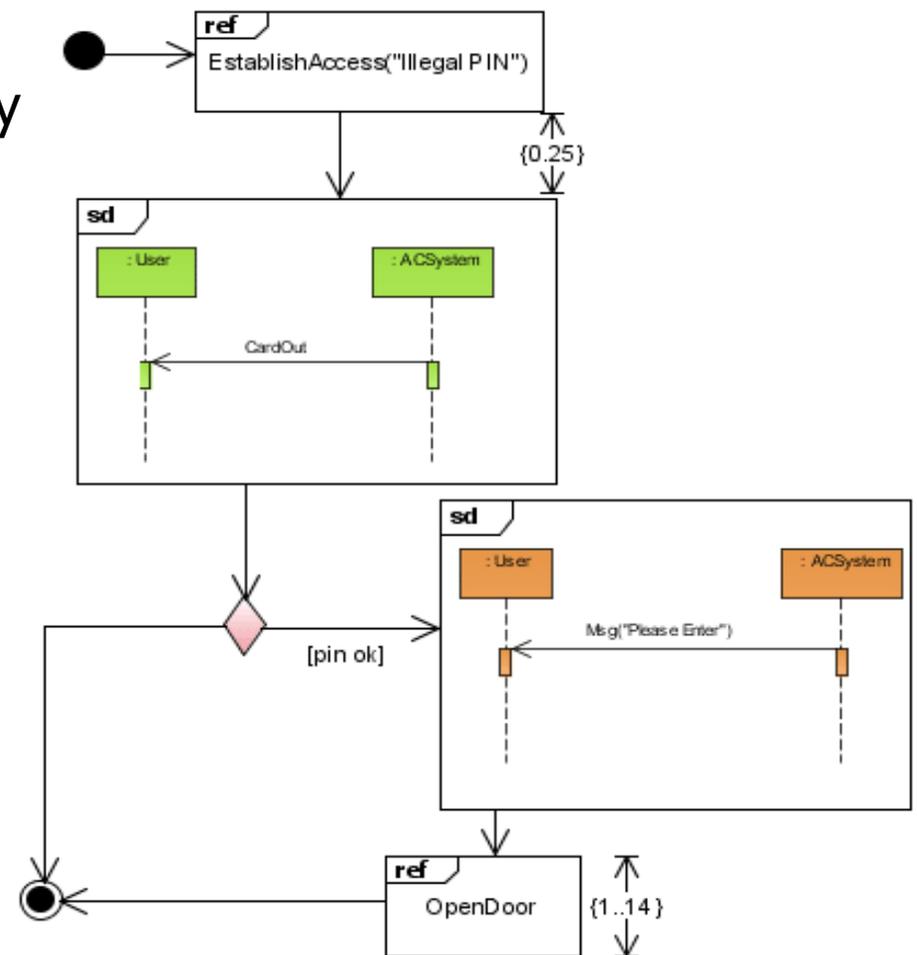


Diagramas – De Comportamiento

• De Revisión de Interacciones:

- Aportan una **visión general** del flujo de control de las interacciones.
- Híbrido entre diagrama de actividad y diagrama de secuencia.

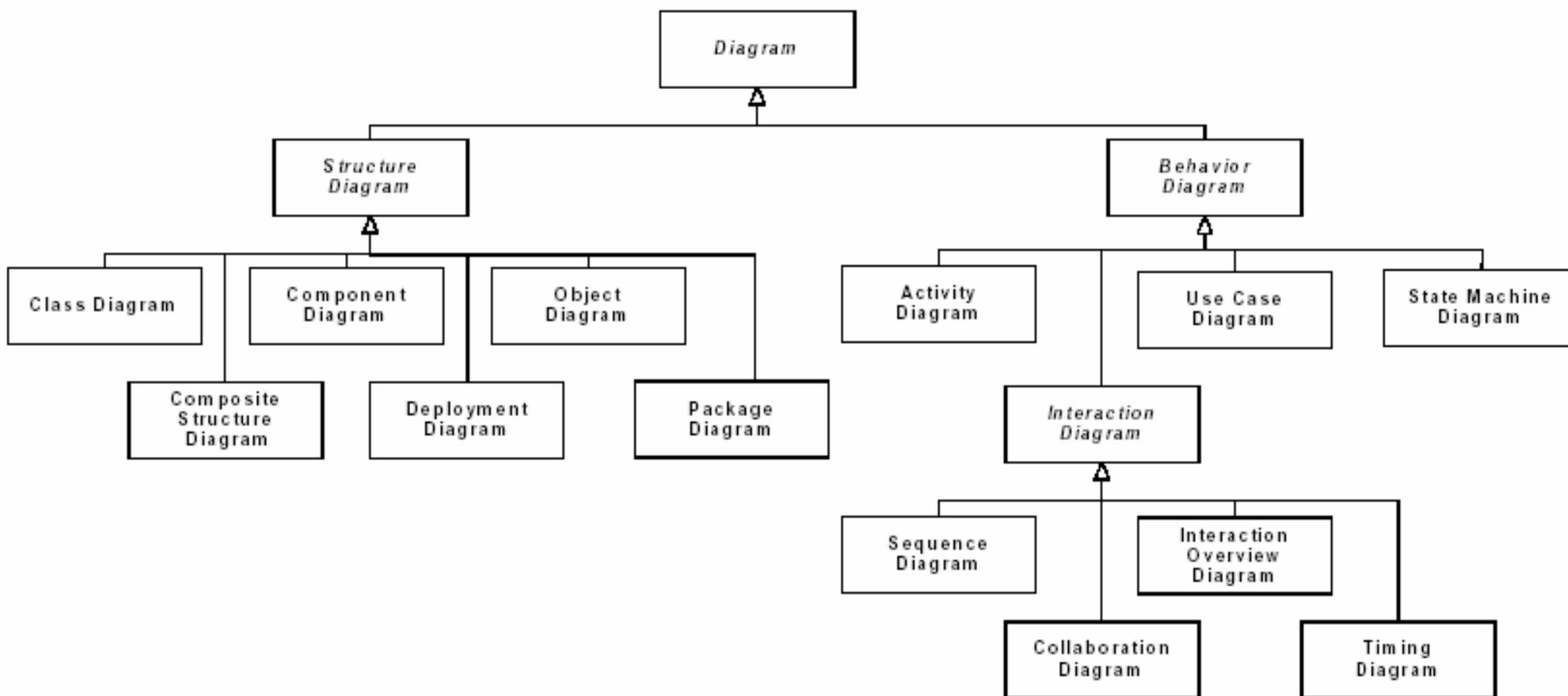
- También llamados
Visión Global de Interacciones





Diagramas - Jerarquía

- Jerarquía de diagramas en **UML 2**





Mecanismos Comunes

- UML tiene cuatro mecanismos comunes que se aplican de forma consistente a través de todo el lenguaje:
 - **Especificaciones**
 - **Adornos**
 - **Divisiones comunes**
 - **Extensibilidad**
- Dan cohesión y simplifican el modelo
- Se aplican a todos los bloques de construcción del modelo



Mecanismos Comunes - Especificaciones

- UML no es sólo un lenguaje gráfico: Detrás de la notación gráfica de cada elemento hay una **especificación** que explica la sintaxis y semántica de ese bloque de construcción.
- La **notación gráfica** de UML se utiliza para **visualizar** el sistema.
- La **especificación** se utiliza para **expresar los detalles** de dicho sistema.
- Los diagramas UML son proyecciones visuales de la base semántica provista por las especificaciones UML.



Mecanismos Comunes - Especificaciones

● Ejemplo de Especificación:

- Detrás del icono de una clase hay una especificación con información de los atributos, operaciones, firmas y comportamiento.
 - Visualmente el icono de la clase puede mostrar sólo parte de la especificación.

The image illustrates the relationship between a class icon and its detailed specification. On the left, a class icon for 'Vehiculo' displays attributes '-marca' and '-modelo' and an operation '+acelerar()'. On the right, another class icon for 'Vehiculo' displays attributes '-marca: String[1]' and '-modelo: String[1]' and an operation '+ acelerar(int kmh)'. In the center, the 'Class Specification' dialog box shows the 'General' tab with fields for Name (vehiculo), Parent (<None>), and Visibility (public). The dialog also has a Documentation field with a rich text editor. Arrows point from the icons to the corresponding fields in the dialog.



Mecanismos Comunes - Adornos

- Todos los elementos en la notación gráfica de UML parten de un símbolo básico, al cual pueden añadirse una variedad de **adornos** específicos de ese símbolo.
 - La notación básica proporciona una representación visual de los aspectos más importantes del elemento.
 - La especificación incluye otros detalles, muchos de los cuales se pueden incluir como adornos gráficos o textuales.
- **Ejemplo de Adornos** en una clase:
 - Es abstracta o no, visibilidad de los atributos y operaciones, tipos de los atributos, multiplicidad, etc.





Mecanismos Comunes - Divisiones

- En el modelado orientado a objetos, existen varias **divisiones comunes** del mundo:
- **Clase vs Objeto.**
 - Una clase es una **abstracción**.
 - Un objeto es una **manifestación concreta** de dicha abstracción.



- Se da en muchos otros bloques de construcción de UML: componente vs instancia de componente; nodo vs instancia de nodo, etc.



Mecanismos Comunes - Divisiones

- **Interfaz vs Implementación.**

- Una interfaz declara un **contrato**.
- Una implementación representa una **realización concreta** de ese contrato (hace efectiva la semántica completa de la interfaz).



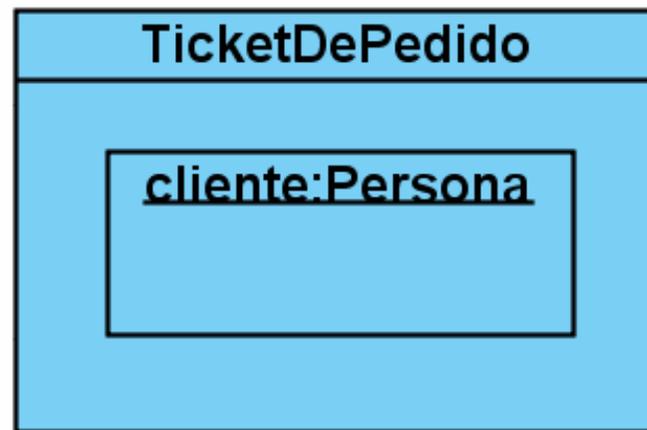
- Casos similares aparecen en casi todos los bloques de construcción de UML: caso de uso vs colaboraciones que los realizan; operaciones vs métodos que las implementan.



Mecanismos Comunes - Divisiones

- **Tipo vs Rol.**

- Un tipo declara la **clase de una entidad** (objeto, atributo, parámetro, ...).
- Un rol describe el **significado** de una entidad en un contexto (una clase, componente o colaboración).
- Cualquier entidad que forma parte de otra tiene ambas características y su significado depende de las dos.





Mecanismos Comunes - Extensibilidad

- **Mecanismos de Extensión** en UML
 - UML se puede extender en base a la definición de **perfiles**.
 - Un perfil extiende UML de manera controlada para poder **expresar** todos los **matices** posibles de un determinado dominio en cualquier momento.
 - Un perfil se define a través de:
 - **Estereotipos** (Stereotypes) => Para añadir nuevos bloques de construcción
 - **Valores Etiquetados** (Tagged Values) => Para modificar o caracterizar la especificación de los nuevos bloques de construcción
 - **Restricciones** (Constraints) => Cambiar o añadir una semántica particular a un elemento de modelado
 - OCL (*Object Constraint Language*)



Mecanismos Comunes - Extensibilidad

- **Estereotipo**

- **Extiende el vocabulario** de UML permitiendo crear **nuevos tipos de bloques de construcción** que derivan de los existentes pero que son específicos a un problema.
- **Ejemplo**
 - En Java las excepciones son clases, aunque se tratan de formas especiales.

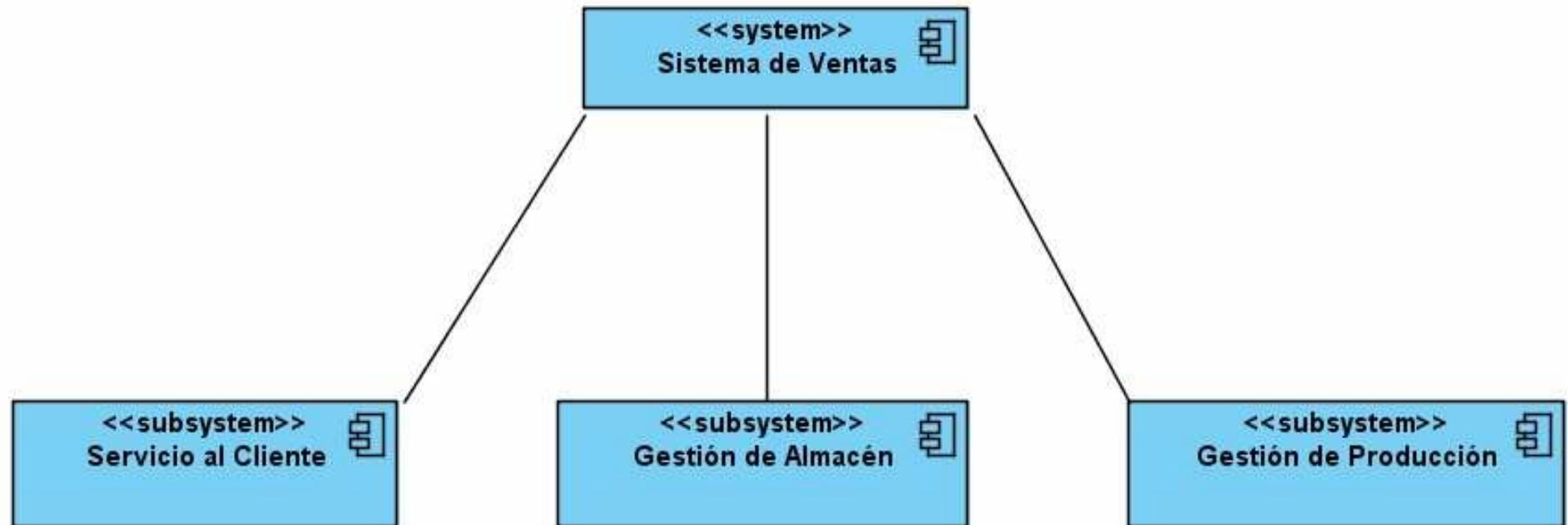
```
<<exception>>  
Overflow
```

- Se pueden asociar símbolos específicos a los bloques de construcción estereotipados.



Mecanismos Comunes - Extensibilidad

- UML 2 proporciona estereotipos de componente para modelar **sistemas** y **subsistemas**.

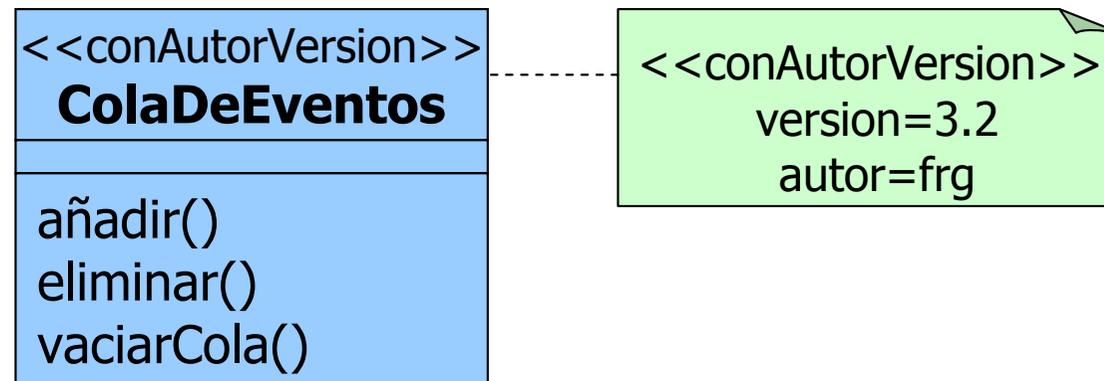




Mecanismos Comunes - Extensibilidad

• Valor Etiquetado

- **Extiende** las **propiedades** de un **estereotipo** de UML, permitiendo añadir nueva información en la especificación del estereotipo.
- Ejemplo
 - Añadir versión y autor mediante la creación del estereotipo `<<conAutorVersion>>` y asociándole una nota con los dos valores etiquetados.

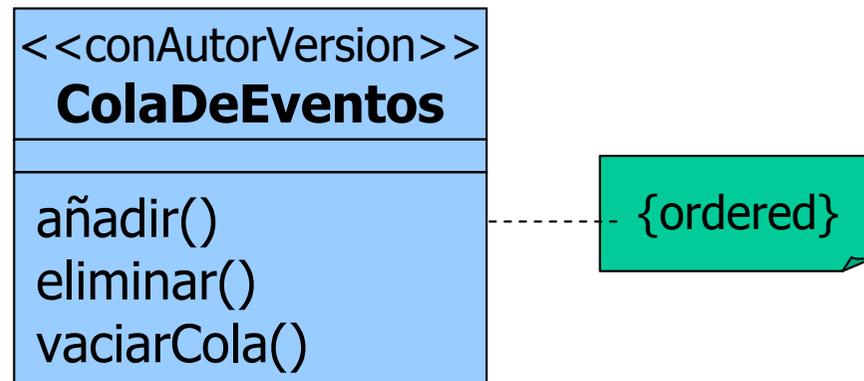




Mecanismos Comunes - Extensibilidad

● Restricción

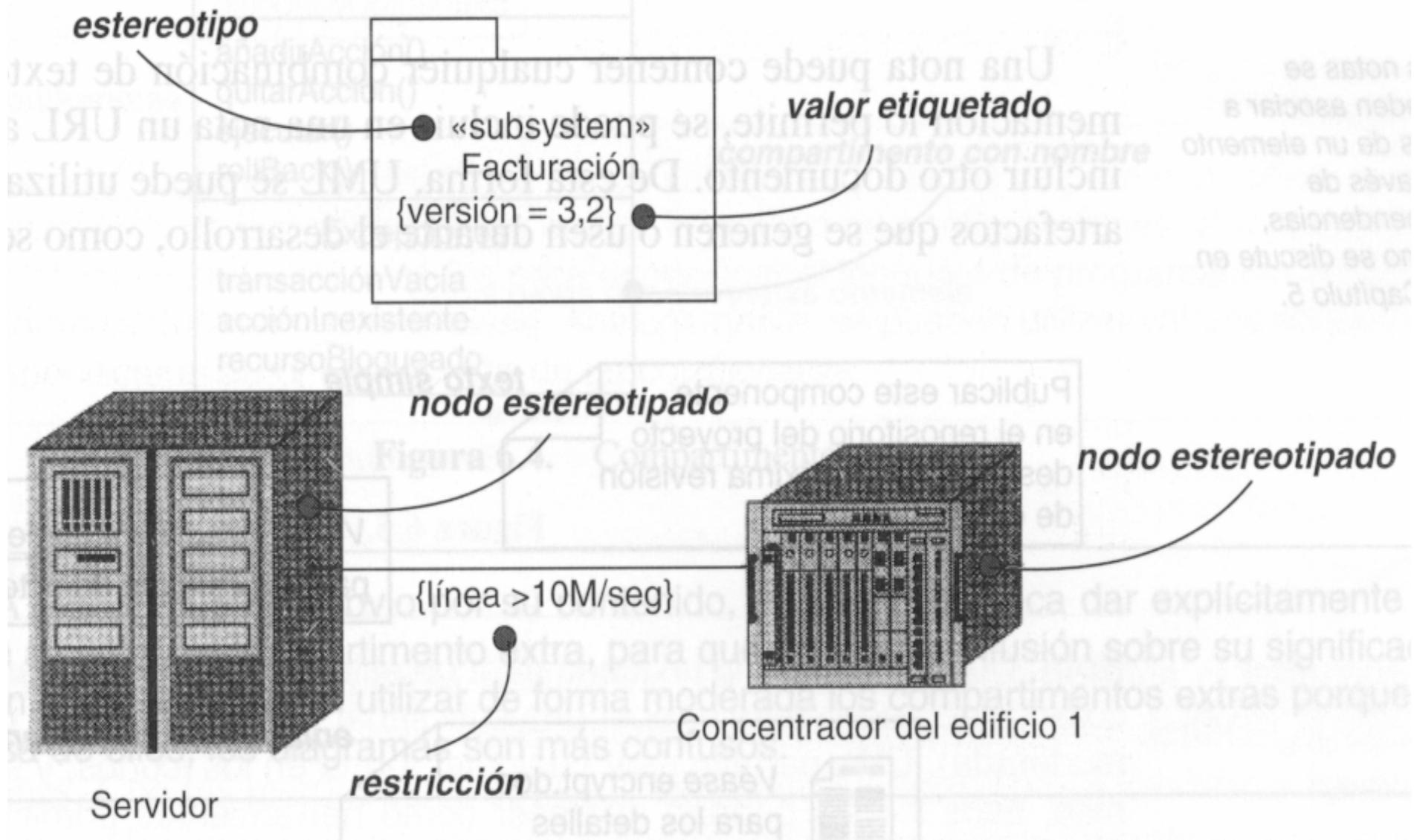
- **Extiende** la **semántica** de un bloque de construcción de UML, permitiendo añadir nuevas reglas o modificar las existentes.
- Ejemplo: Restringir la clase ColaDeEventos para que todas las adiciones se hagan en orden.



- Para especificar restricciones de forma precisa se usa OCL (Object Constraint Language).



Mecanismos Comunes - Extensibilidad



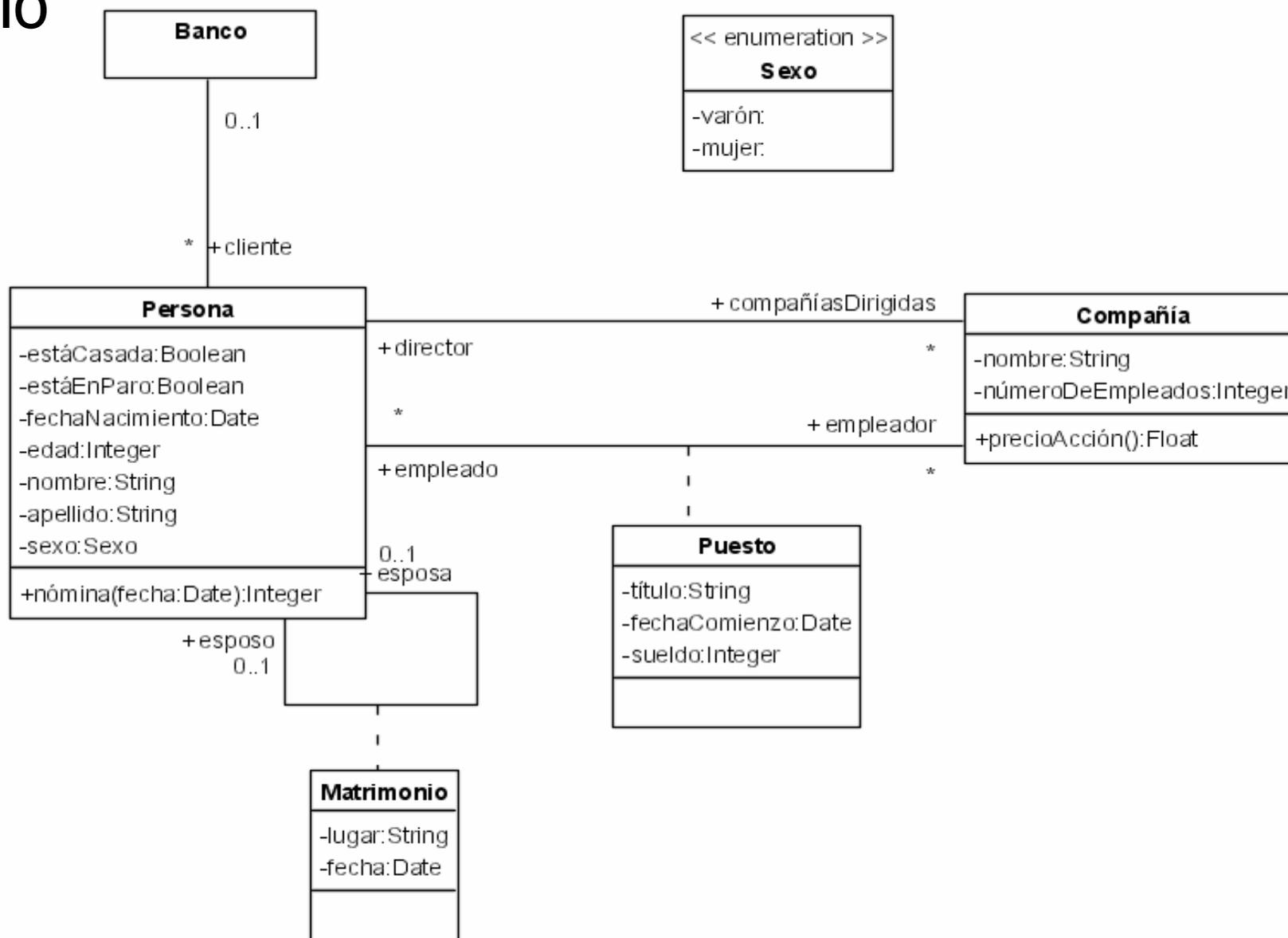


- **Object Constraint Language 2.0**
- Complemento a UML.
 - **Lenguaje** para escribir **expresiones formales** acerca de modelos UML.
 - Usos:
 - Especificación de **invariantes** en clases y tipos.
 - Especificación de invariantes de tipo para Estereotipos.
 - Describir pre- y post-**condiciones** en Operaciones.
 - Describir condiciones de **guarda**.
 - Especificar **destinatarios** para mensajes y acciones.
 - Especificar **restricciones** en Operaciones.
 - Especificar **reglas de derivación** para atributos.



OCL

● Ejemplo





- **Invariantes:**

- Condiciones o restricciones que deben cumplirse siempre.

- Ejemplo

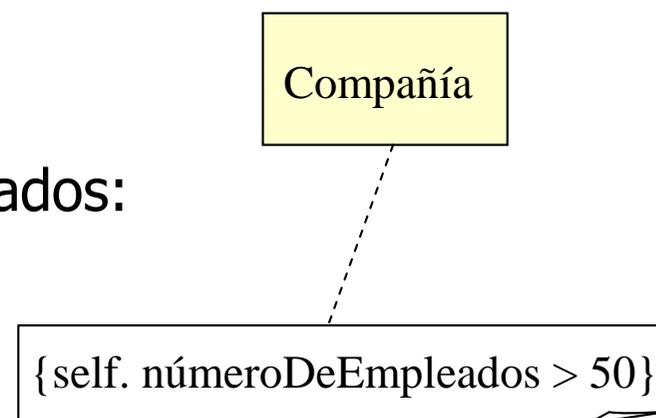
"El número de empleados debe ser mayor que 50"

context Compañía **inv**:

self. númeroDeEmpleados > 50

context c:Compañía **inv** suficientesEmpleados:

c.númeroDeEmpleados > 50





- **Condiciones**

- Pre-condiciones o post-condiciones que deben cumplirse en operaciones

- Sintaxis

context NombreTipo::NombreOperación(Param₁ : Tipo₁, ...):TipoRetorno

pre parametroOk: param₁ < ...

post resultadoOk : result > ...

- Ejemplo

context Persona::nómina(fecha : Date) : Integer

post: result > 650



- **Valores iniciales y derivados**

- Sintaxis

context NombreTipo::NombreAtributo: Tipo

init: -- alguna expresión representando el valor inicial

context NombreTipo::NombreRolAsociación: Tipo

derive: -- alguna expresión representando la regla de derivación

- Ejemplos

context Persona::estaEnParo: Boolean

init: true

derive: if self.empleador->notEmpty()

 false

 else

 true

 end if



- **Definiciones**

- Ejemplo

- context** Persona

- def:** ingresos : Integer = self.puesto.sueldo->sum()

- def:** apodo : String = 'Gallito rojo'

- **Navegación y combinación de expresiones**

- Ejemplos

- a) “Los casados tienen al menos 18 años de edad”

- context Persona inv:

- self.esposa >∅() implies self.esposa.edad >= 18 and

- self.esposø >∅() implies self.esposo.edad >= 18

- b) “Una compañía tiene como mucho 50 empleados”

- context Compañía inv:

- self.empleadø >size() <= 50