IN77J – Orientación al Objeto para el e-business

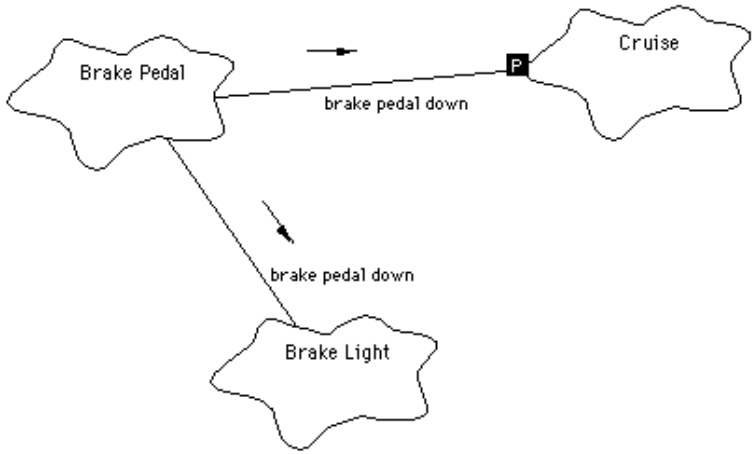
3. UML

Temario

- 3. UML
  - Breve Historia
  - ¿Por qué Modelar con UML?
  - Diagramas de Comportamiento
  - Diagramas de Interacción
  - Diagramas de Estructura

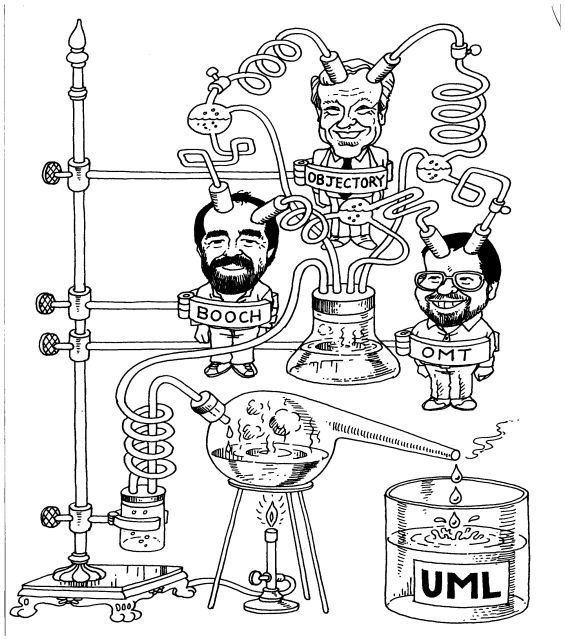
Breve Historia

- A comienzos de los años '90 aparecieron múltiples metodologías y notaciones gráficas para análisis y diseño orientados a objetos:
  - Grady Booch
  - James Rumbaugh: OMT (Object Modeling Technique)
  - Ivar Jacobson: OOSE (Object-Oriented Software Engineering)
  - Peter Coad y Edward Yourdon
  - Sally Shlaer y Stephen Mellor
  - ...



Breve Historia

- En 1994 Rumbaugh se unió a Rational (donde se desempeñaba Booch), y comenzaron a trabajar en la unificación de sus métodos
- En 1995 Booch y Rumbaugh presentaron el "método unificado", basado en la unión de sus métodos
- Posteriormente Jacobson se unió a Rational, y en 1997 Rational entregó una definición de Unified Modeling Language a OMG (Object Management Group, http://www.omg.org)



UML

- UML (Unified Modeling Language) es hoy la notación estándar para el modelamiento de aplicaciones con tecnologías orientadas a objetos
- Se encuentra en su versión 2.0 (aparecida en octubre del año 2004), y es mantenido por OMG
- Las especificaciones pueden obtenerse en http://www.uml.org
- Actualmente existen diversas herramientas que soportan UML, entre las que se cuentan:
  - Rational Rose, WebSphere Application Developer Studio, Rational Software Architect, etc.
  - Borland Together
  - Enterprise Architect
  - Sybase PowerDesigner
  - Poseidon
  - ArgoUML
  - ...

¿Por Qué Modelar con UML?

- Modelar es diseñar las aplicaciones de software antes de construirlas, actividad esencial en el desarrollo de proyectos grandes
- Modelar ayuda al desarrollo, permitiendo mantener un nivel alto de abstracción
- UML ayuda a especificar, visualizar y documentar aplicaciones de software

Unified Modeling Language

- *"The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.*
- *The UML represents the culmination of best practices in practical object-oriented modeling. The UML is the product of several years of hard work, in which we focused on bringing about a unification of the methods most used around the world, the adoption of good ideas from many quarters of the industry, and, above all, a concentrated effort to make things simple."*

Especificación de UML 1.5, Foreword

Booch, Jacobson, Rumbaugh

Diagramas de UML 2.0

- UML define un conjunto de diagramas que describen diferentes vistas de un sistema de software:

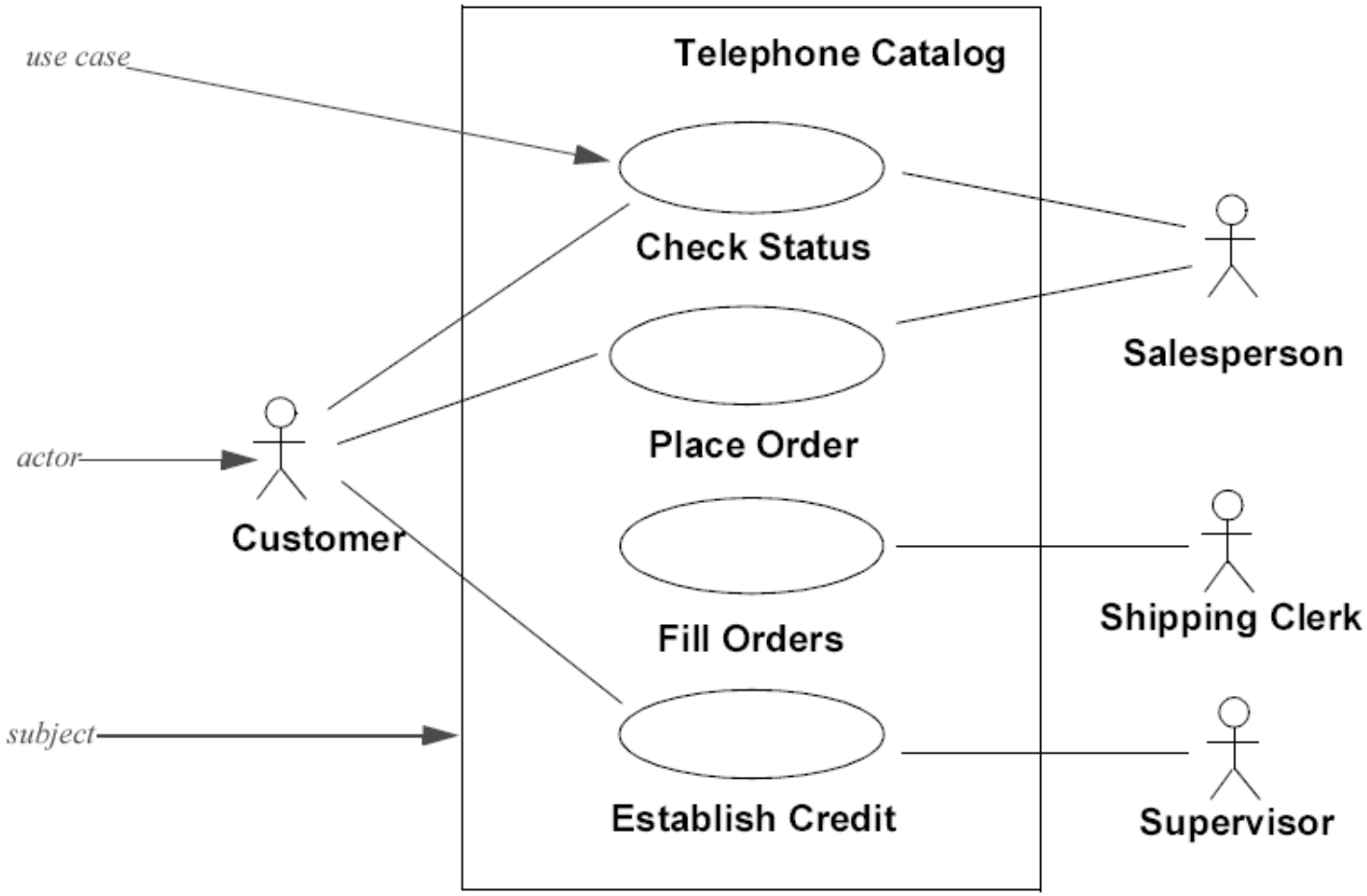
Tipo de Diagrama	Diagrama
Comportamiento	Casos de Uso (Use Case Diagram)
	Actividades (Activity Diagram)
	Máquinas de Estados (State Machine Diagram)
Interacción	Secuencia (Sequence Diagram)
	Comunicación (Communication Diagram)
	Tiempo (Timing Diagram)
	Interacción Global (Interaction Overview Diagram)
Estructura	Paquetes (Package Diagram)
	Clases (Class Diagram)
	Objetos (Object Diagram)
	Estructura Compuesta (Composite Structure Diagram)
	Componentes (Component Diagram)
	Despliegue (Deployment Diagram)

Diagramas de Comportamiento

- Los diagramas de comportamiento dan una vista de alto nivel del comportamiento de un sistema
  - Diagramas de Casos de Uso (Use Case Diagram)
  - Diagramas de Actividades (Activity Diagram)
  - Diagramas de Máquinas de Estado (State Machine Diagram)

Diagrama de Casos de Uso

- Modela requerimientos funcionales, expresados en la interacción entre actores y módulos del sistema



Casos de Uso

- Los casos de uso son una técnica para definir requerimientos creada por Ivar Jacobson en 1986, e incorporada en UML (pese a no ser una técnica orientada a objetos)
- *"Use cases are a means for specifying required usages of a system. Typically, they are used to capture the requirements of a system, that is, what a system is supposed to do. The key concepts associated with use cases are actors, use cases, and the subject. The subject is the system under consideration to which the use cases apply. The users and any other systems that may interact with the subject are represented as actors. Actors always model entities that are outside the system. The required behavior of the subject is specified by one or more use cases, which are defined according to the needs of actors."*

[especificación UML 2.0]

Definición de Caso de Uso

- *"A UseCase is a kind of behaviored classifier that represents a declaration of an offered behavior. Each use case specifies some behavior, possibly including variants, that the subject can perform in collaboration with one or more actors. Use cases define the offered behavior of the subject without reference to its internal structure. These behaviors, involving interactions between the actor and the subject, may result in changes to the state of the subject and communications with its environment. A use case can include possible variations of its basic behavior, including exceptional behavior and error handling."*

[especificación UML 2.0]

Definición de Caso de Uso

- *"Use cases can be used both for specification of the (external) requirements on a subject and for the specification of the functionality offered by a subject. Moreover, the use cases also state the requirements the specified subject poses on its environment by defining how they should interact with the subject so that it will be able to perform its services."*

- *"The behavior of a use case can be described by a specification that is some kind of behavior, such as interactions, activities, and state machines, or by pre-conditions and post-conditions as well as by natural language text where appropriate."*

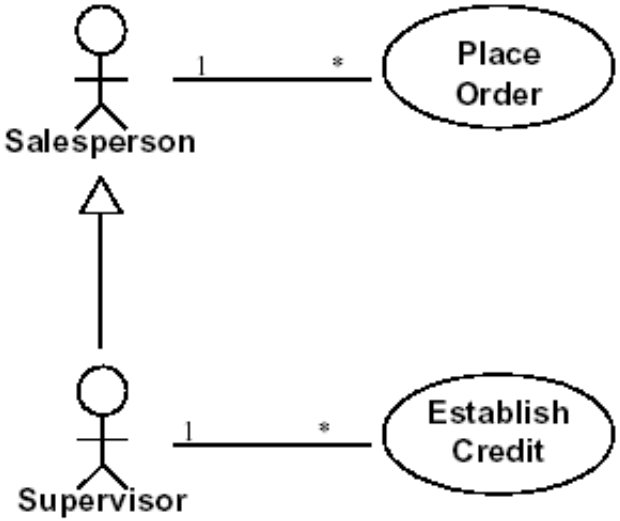
[especificación UML 2.0]

Diagrama de Casos de Uso

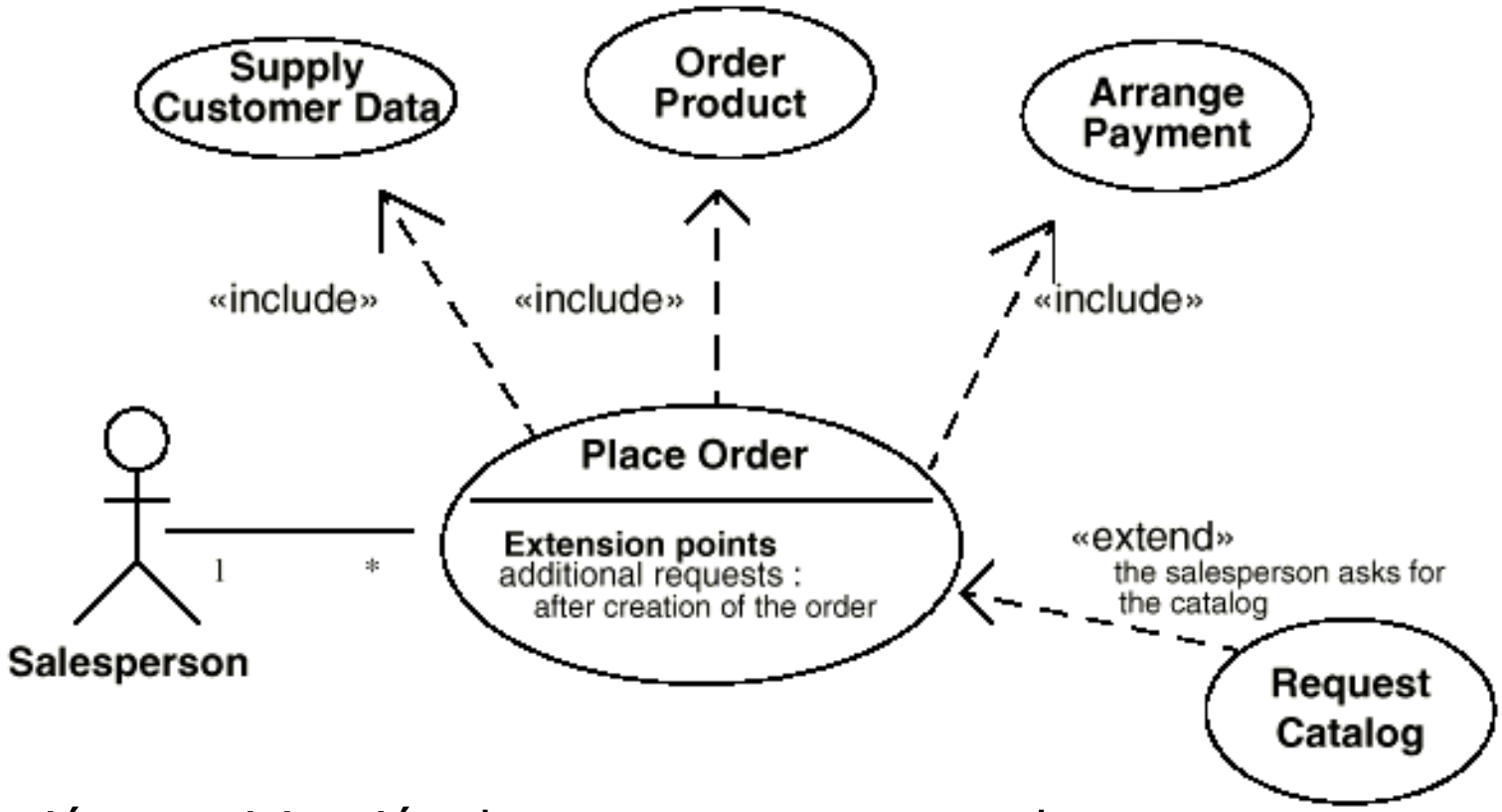
- Un diagrama de casos de uso representa una parte de la funcionalidad de un sistema, en la interacción con objetos externos a él
- Los diagramas de casos de uso muestran actores y casos de uso en conjunto con sus relaciones
- Un caso de uso muestra qué hace el sistema, no cómo lo hace

Actores

- Actor: *"An Actor models a type of role played by an entity that interacts with the subject (e.g., by exchanging signals and data), but which is external to the subject (i.e., in the sense that an instance of an actor is not a part of the instance of its corresponding subject). Actors may represent roles played by human users, external hardware, or other subjects.*
- Generalización: *"A generalization from an actor A to an actor B indicates that an instance of A can communicate with the same kinds of use-case instances as an instance of B"*

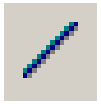


Relaciones en Casos de Uso

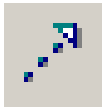


- Asociación: participación de un actor en un caso de uso
- Extensión: un caso de uso extiende el comportamiento de otro
- Generalización: especialización entre casos de uso o actores
- Inclusión: un caso de uso incluye el comportamiento de otro

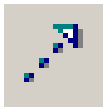
Relaciones en Casos de Uso



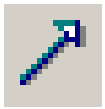
- Asociación: *"The participation of an actor in a use case; that is, instances of the actor and instances of the use case communicate with each other"*



- Extensión: *"An extend relationship from use case A to use case B indicates that an instance of use case B may be augmented (subject to specific conditions specified in the extension) by the behavior specified by A"*



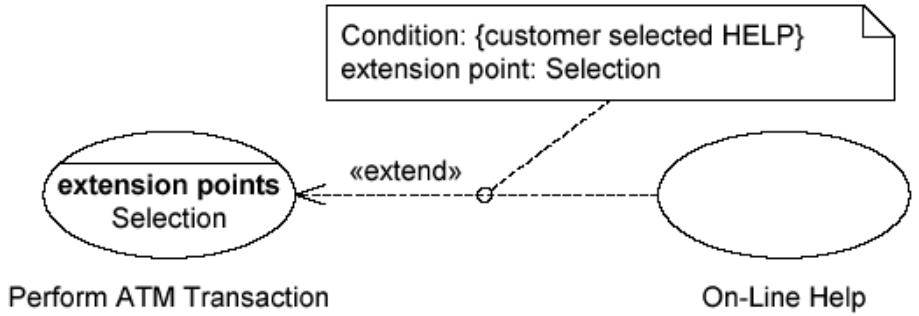
- Inclusión: *"An include relationship from use case E to use case F indicates that an instance of the use case E will also contain the behavior as specified by F"*



- Generalización: *"A generalization from use case C to use case D indicates that C is a specialization of D"*

Extensión de Casos de Uso

- Punto de extensión: *"An extension point identifies a point in the behavior of a use case where that behavior can be extended by the behavior of some other (extending) use case, as specified by an extend relationship"*
- Extensión: *"An extend relationship specifies that the behavior of a use case may be extended by the behavior of another (usually supplementary) use case. The extension takes place at one or more specific extension points defined in the extended use case. Note, however, that the extended use case is defined independently of the extending use case and is meaningful independently of the extending use case. On the other hand, the extending use case typically defines behavior that may not necessarily be meaningful by itself. Instead, the extending use case defines a set of modular behavior increments that augment an execution of the extended use case under specific conditions."*



Inclusión de Casos de Uso

- *"Include is a directed relationship between two use cases, implying that the behavior of the included use case is inserted into the behavior of the including use case."*

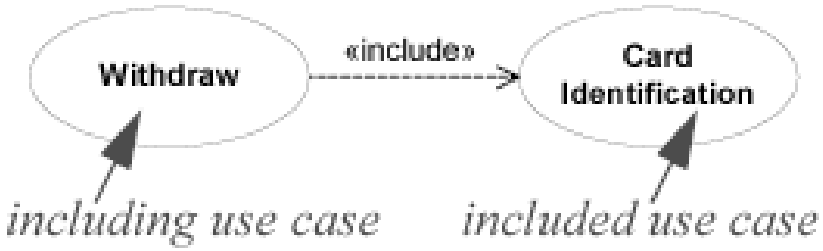
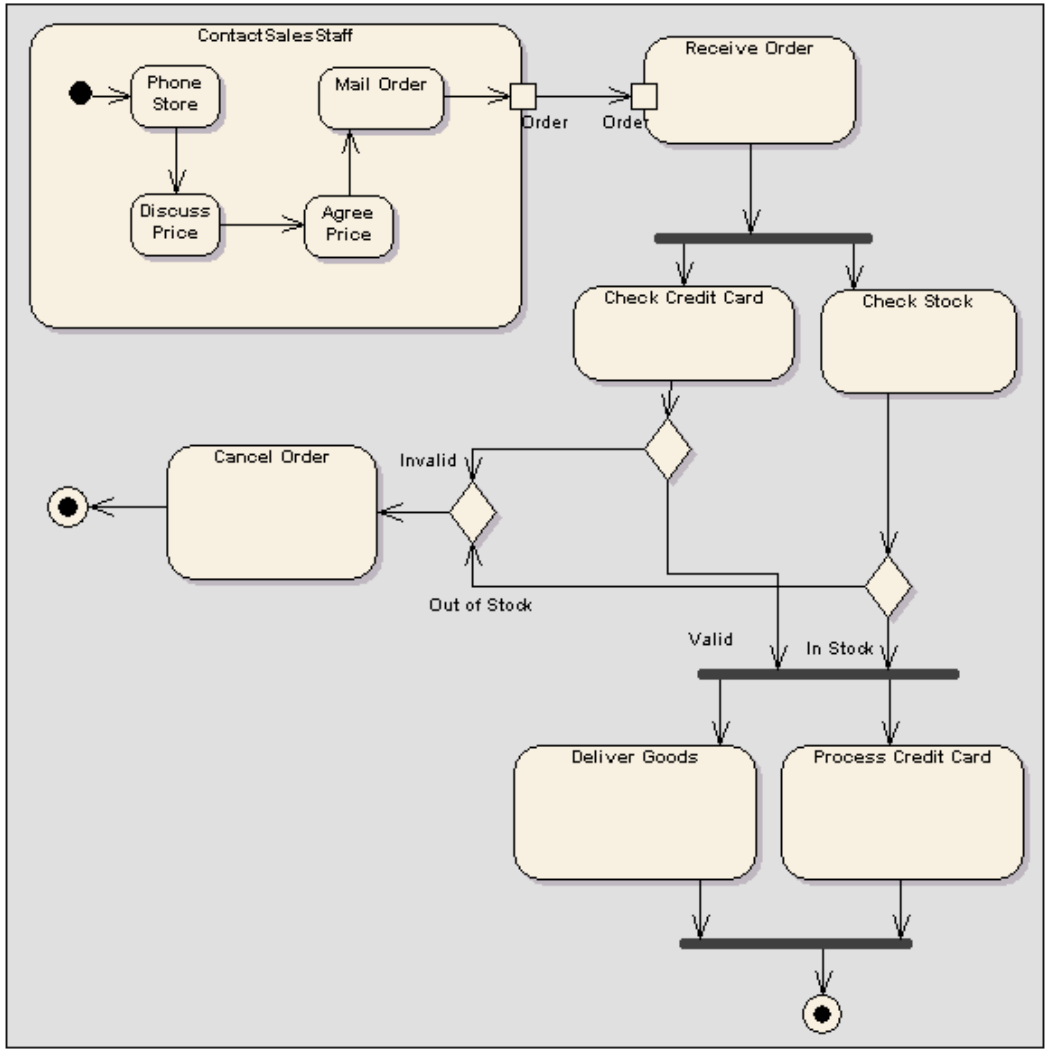


Diagrama de Actividades

- Los diagramas de actividades modelan el comportamiento de un sistema, módulo, o proceso de negocio



Actividades

- *"An activity specifies the coordination of executions of subordinate behaviors, using a control and data flow model. The subordinate behaviors coordinated by these models may be initiated because other behaviors in the model finish executing, because objects and data become available, or because events occur external to the flow. The flow of execution is modeled as activity nodes connected by activity edges. A node can be the execution of a subordinate behavior, such as an arithmetic computation, a call to an operation, or manipulation of object contents. Activity nodes also include flow-of-control constructs, such as synchronization, decision, and concurrency control. Activities may form invocation hierarchies invoking other activities, ultimately resolving to individual actions. In an object-oriented model, activities are usually invoked indirectly as methods bound to operations that are directly invoked."*

[especificación UML 2.0]

Actividades

- *"Activities may describe procedural computation. In this context, they are the methods corresponding to operations on classes. Activities may be applied to organizational modeling for business process engineering and workflow. In this context, events often originate from inside the system, such as the finishing of a task, but also from outside the system, such as a customer call. Activities can also be used for information system modeling to specify system level processes.*
- *Activities may contain actions of various kinds:*
  - *occurrences of primitive functions, such as arithmetic functions.*
  - *invocations of behavior, such as activities.*
  - *communication actions, such as sending of signals.*
  - *manipulations of objects, such as reading or writing attributes or associations."*

[especificación UML 2.0]

Acciones

- *"Actions have no further decomposition in the activity containing them. However, the execution of a single action may induce the execution of many other actions. For example, a call action invokes an operation which is implemented by an activity containing actions that execute before the call action completes."*

[especificación UML 2.0]

Principales Elementos

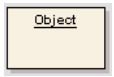
- Actividad: tarea que puede incluir subactividades y acciones, puede modelarse en un diagrama

- Acción: tarea elemental, no puede subdividirse

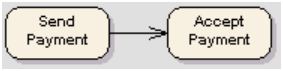
- Inicio: define el comienzo del flujo de la actividad

- Fin: indica que la actividad se completó

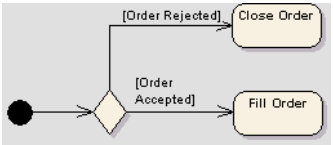
- Fin de flujo: fin de actividad, sin que se complete



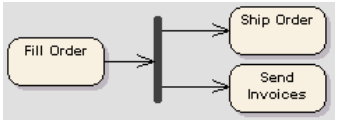
- Objeto: instancia de una clase, puede utilizarse en la transferencia de información entre actividades



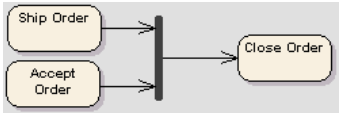
- Flujo: establece que una actividad se inicia al terminar otra



- Decisión: dependiendo de una condición, el procesamiento continúa de una manera u otra



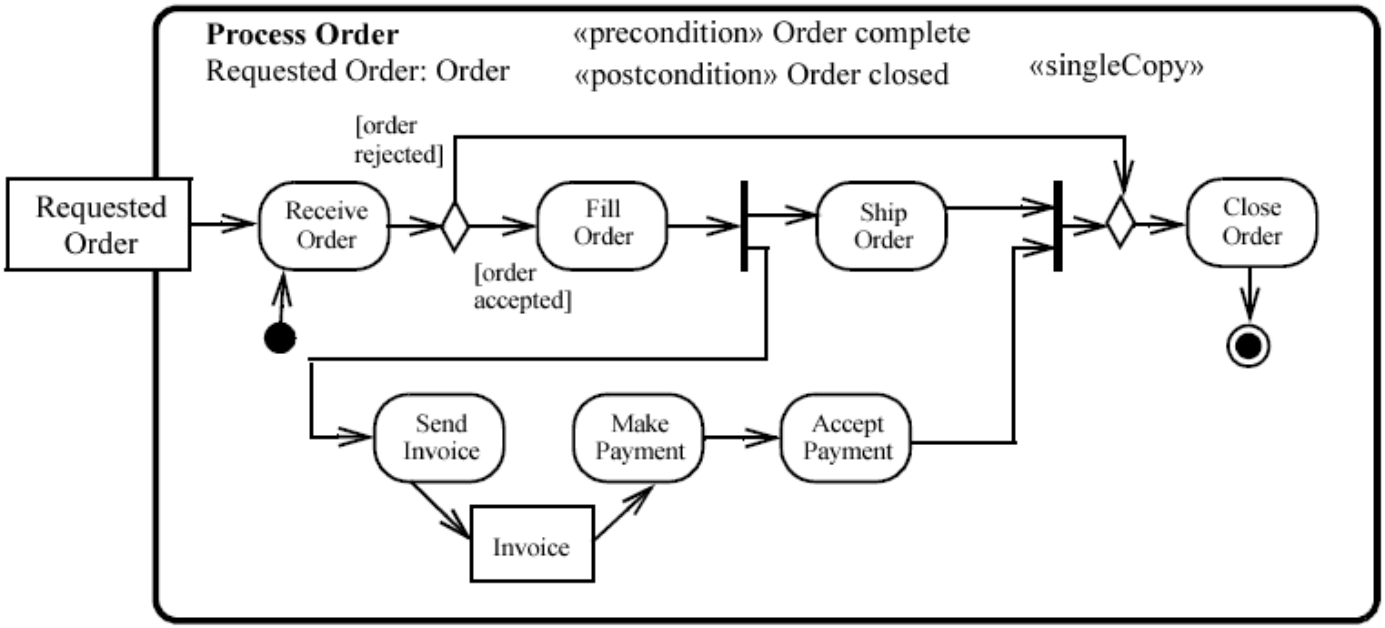
- Fork: un flujo se divide en varios flujos concurrentes



- Join: varios flujos concurrentes se unen

Ejemplo

- El siguiente diagrama (tomado de la especificación) hace uso de los siguientes elementos:
  - Inicio y fin
  - Actividades (receive order)
  - Decisiones, flujos y condiciones (order rejected)
  - Fork, join
  - Objetos (invoice) y parámetros de entrada (requested order)



Particiones

- Es posible definir particiones para organizar lógicamente un diagrama de actividades

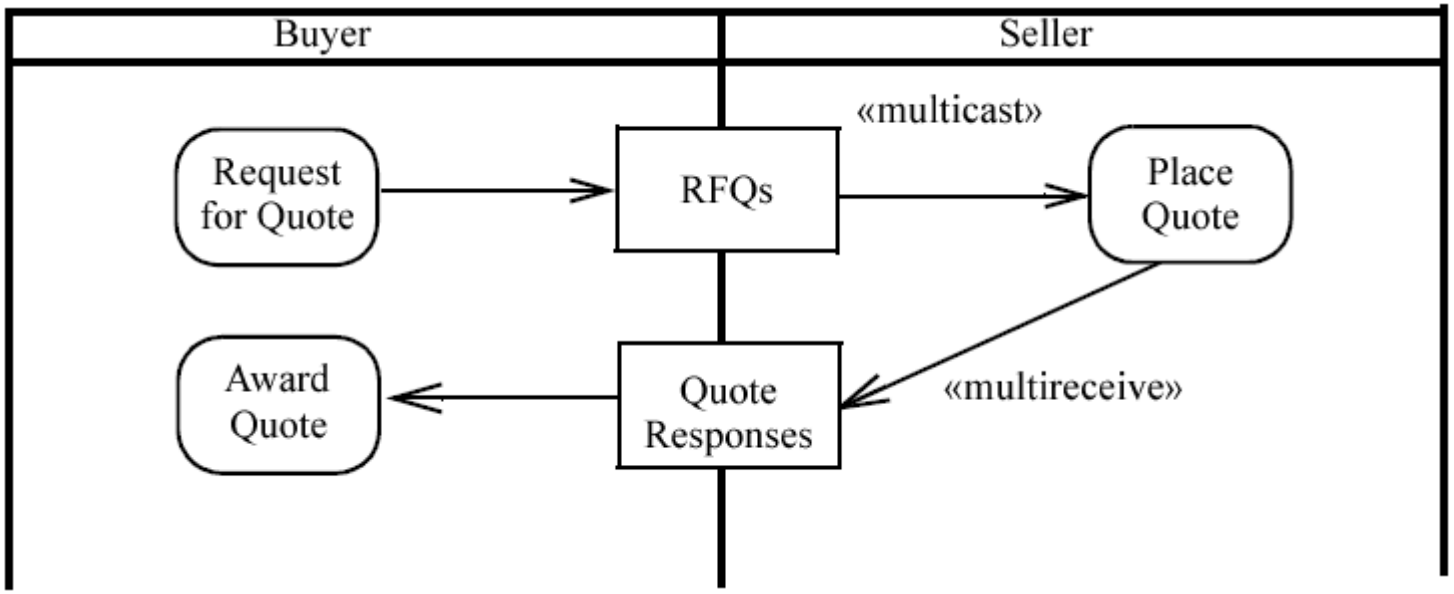


Diagrama de Máquina de Estados

- Describe la manera en que una clase u otro elemento transita por diferentes estados durante su ciclo de vida

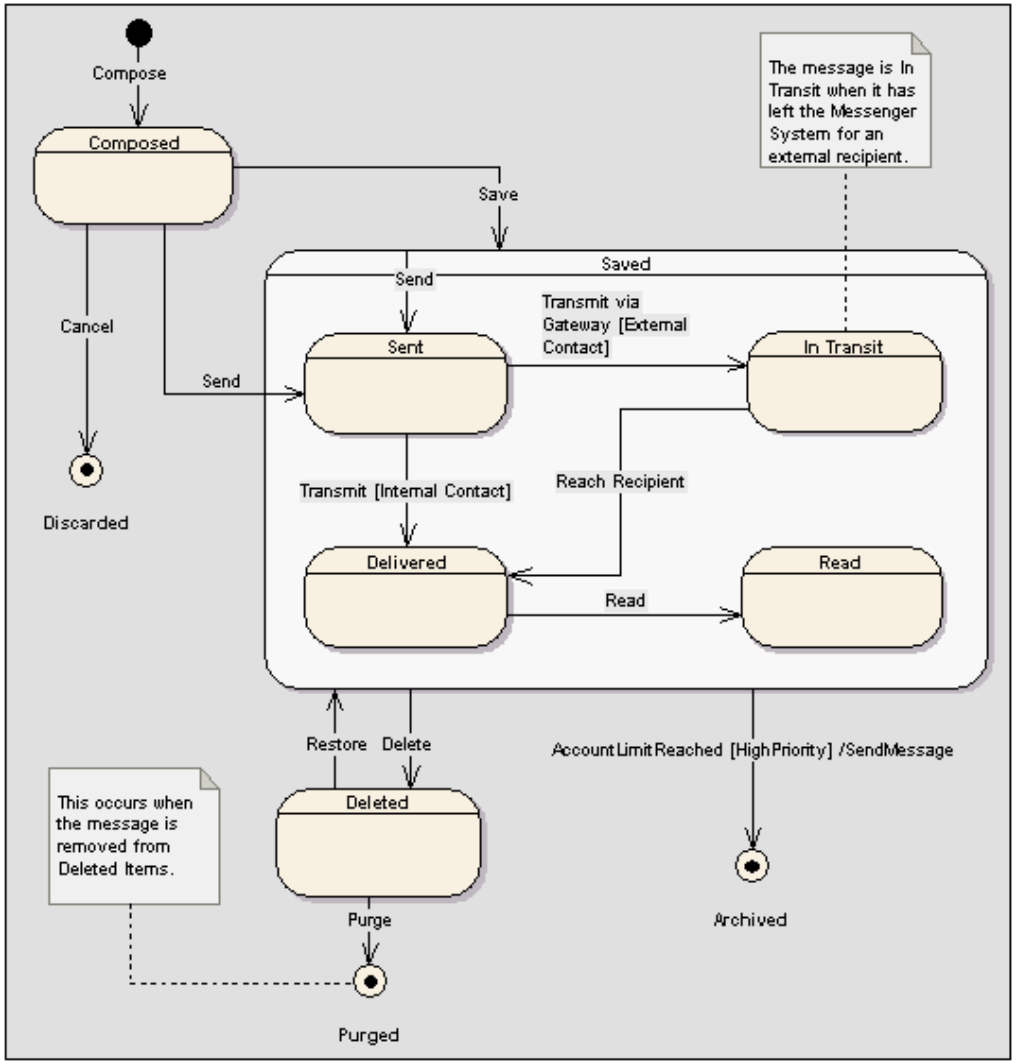
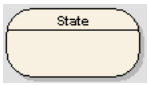


Diagrama de Máquina de Estados

- *"The behaviored classifier context owning a state machine defines which signal and call triggers are defined for the state machine, and which attributes and operations are available in activities of the state machine. Signal triggers and call triggers for the state machine are defined according to the receptions and operations of this classifier. As a kind of behavior, a state machine may have an associated behavioral feature (specification) and be the method of this behavioral feature. In this case the state machine specifies the behavior of this behavioral feature. The parameters of the state machine in this case match the parameters of the behavioral feature and provide the means for accessing (within the state machine) the behavioral feature parameters. A state machine without a context classifier may use triggers that are independent of receptions or operations of a classifier, i.e. either just signal triggers or call triggers based upon operation template parameters of the (parameterized) statemachine."*

[especificación UML 2.0]

Principales Elementos



- Estado: representa una situación en la cual ocurre una condición (esperando un evento, realizando una tarea, etc.)

- Transición: define el movimiento de un estado a otro, controlado por las siguientes propiedades:
  - Trigger: evento que gatilla la transición
  - Guard: condición para habilitar la transición
  - Effect: actividad realizada durante la transición

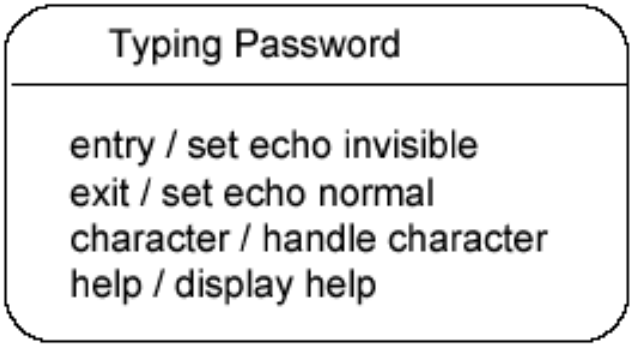
  La transición es rotulada de la siguiente forma:
  - *trigger [guard] / effect*

- Entry: entrada a una máquina de estados, debe definirse un entry para cada región

- Exit: punto de salida de una máquina de estados
- Otros elementos (ya definidos): inicio, fin, decisión, objeto, fork, join

Estados

- Cuando un objeto se encuentra en un **estado**:
  - satisface alguna **condición**,
  - realiza alguna **acción**,
  - o espera algún **evento**



- Al interior del estado, es posible indicar eventos y acciones asociadas (por ejemplo, evento "help" y acción "display help")
- Los eventos **entry** (al entrar al estado), **exit** (al salir del estado) y **do** (estando en el estado) se encuentran predefinidos

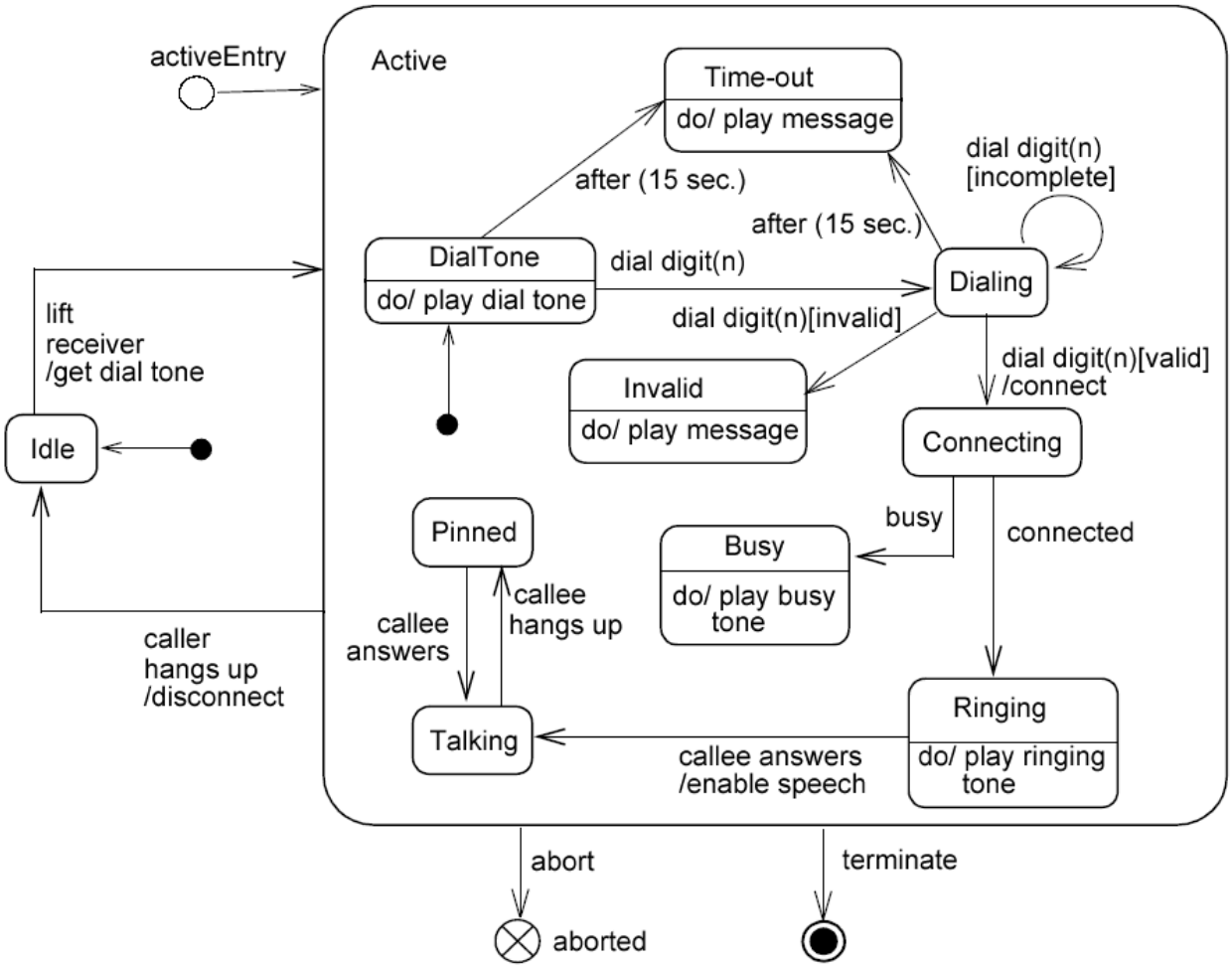
Transiciones

- Una transición es una relación entre dos estados, que indica que un objeto que se encuentre en el primer estado pasará al segundo, si se produce el evento (trigger) asociado, y se cumple la condición (guard) asociada

- Rótulo (label) de la transición (todos los elementos son opcionales):
  - *evento(parámetros) [condición] / acción*

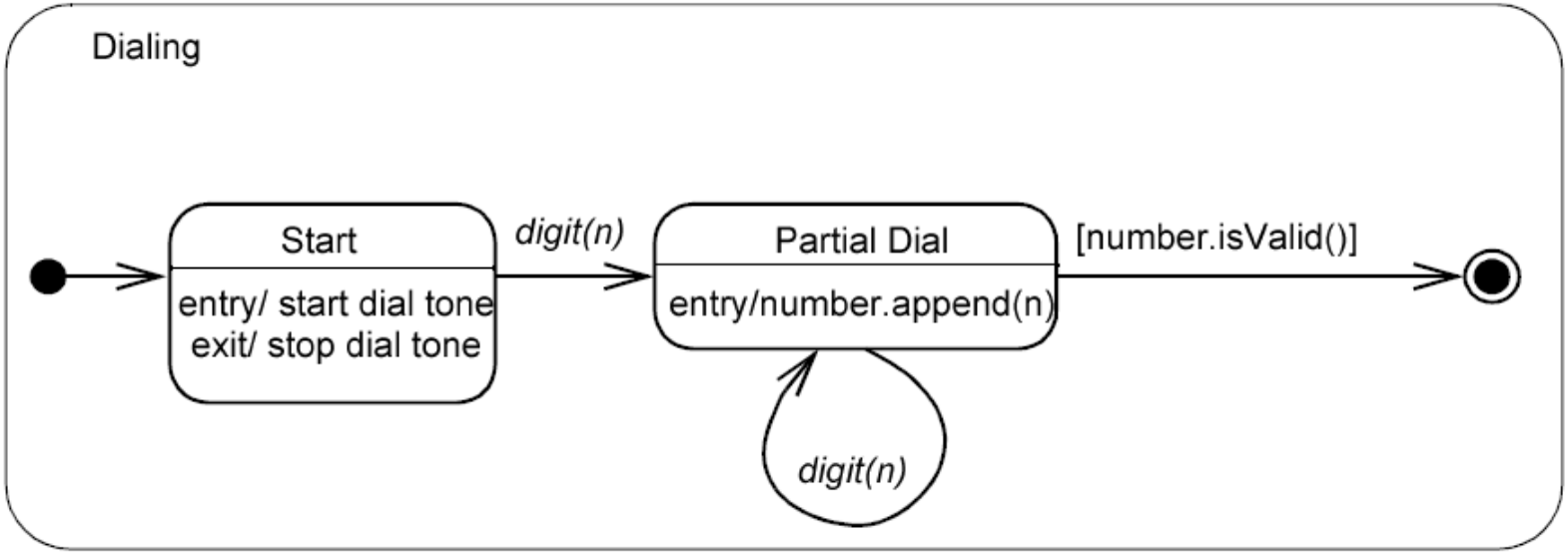
Ejemplo

- El siguiente ejemplo muestra una máquina de estados con diversos estados y transiciones



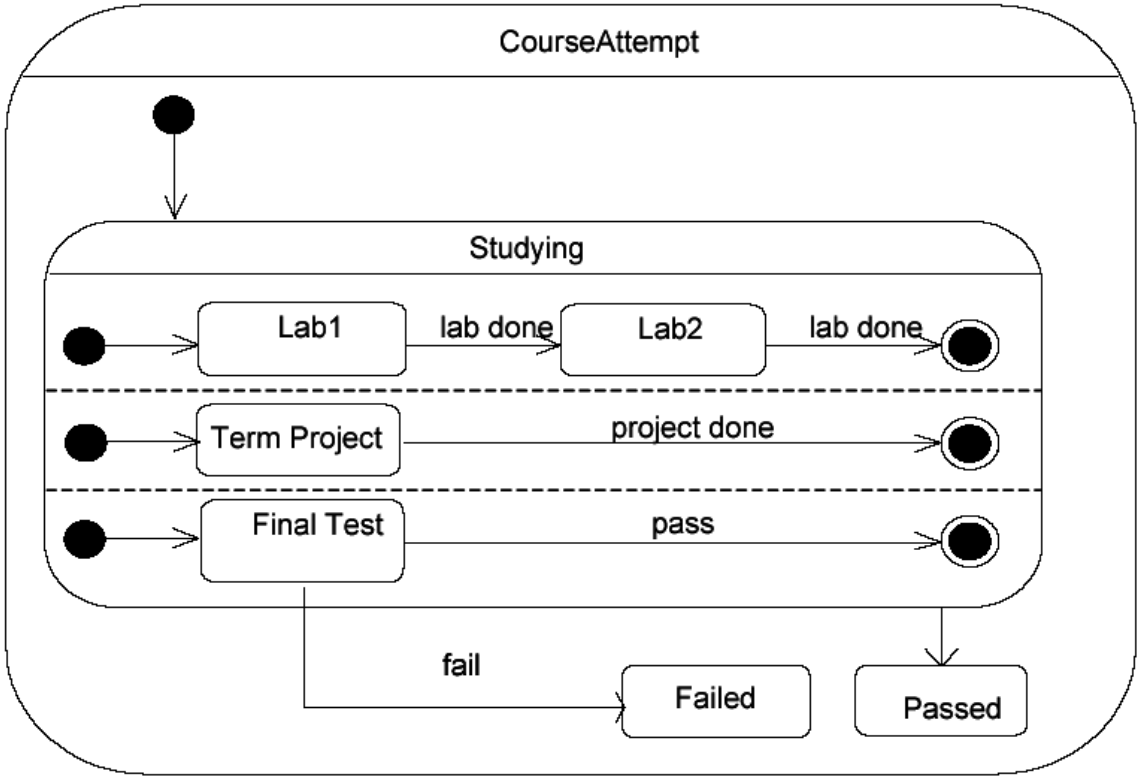
Estado Compuesto

- El siguiente ejemplo muestra un estado compuesto (“Dialing”) con dos estados (“Start” y “Partial Dial”)



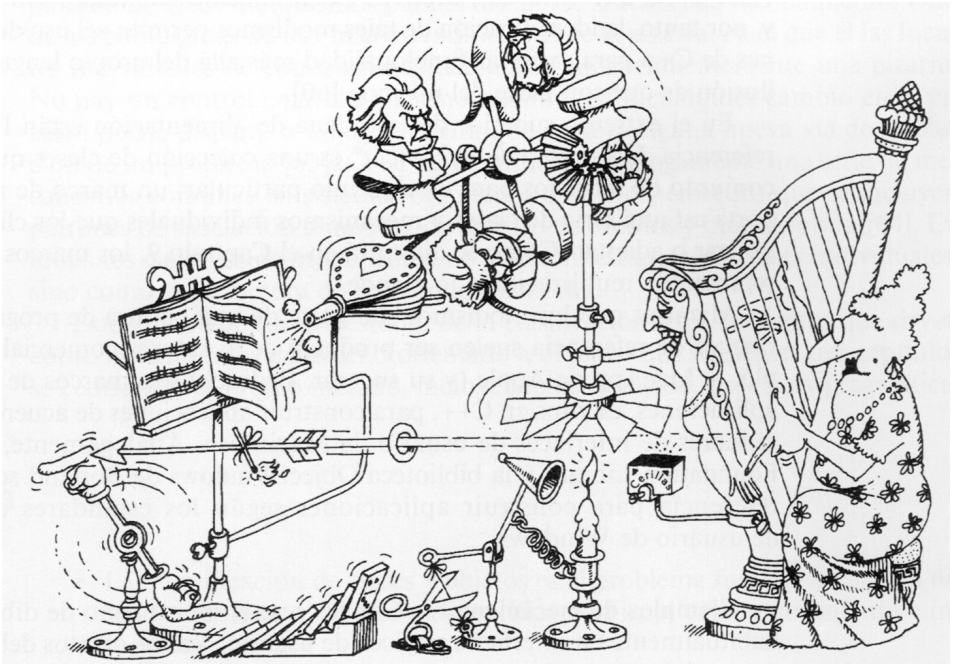
Regiones

- Una región es una parte “ortogonal” de un estado compuesto
- El ejemplo muestra un estado compuesto (“Studying”) con 3 regiones (o subestados concurrentes)



Diagramas de Interacción

- Los diagramas de interacción describen la interacción entre diferentes objetos y componentes del sistema
  - Diagramas de Secuencia (Sequence Diagram)
  - Diagramas de Comunicación (Communication Diagram)
  - Diagramas de Tiempo (Timing Diagram)
  - Diagramas de Interacción Global (Interaction Overview Diagram)



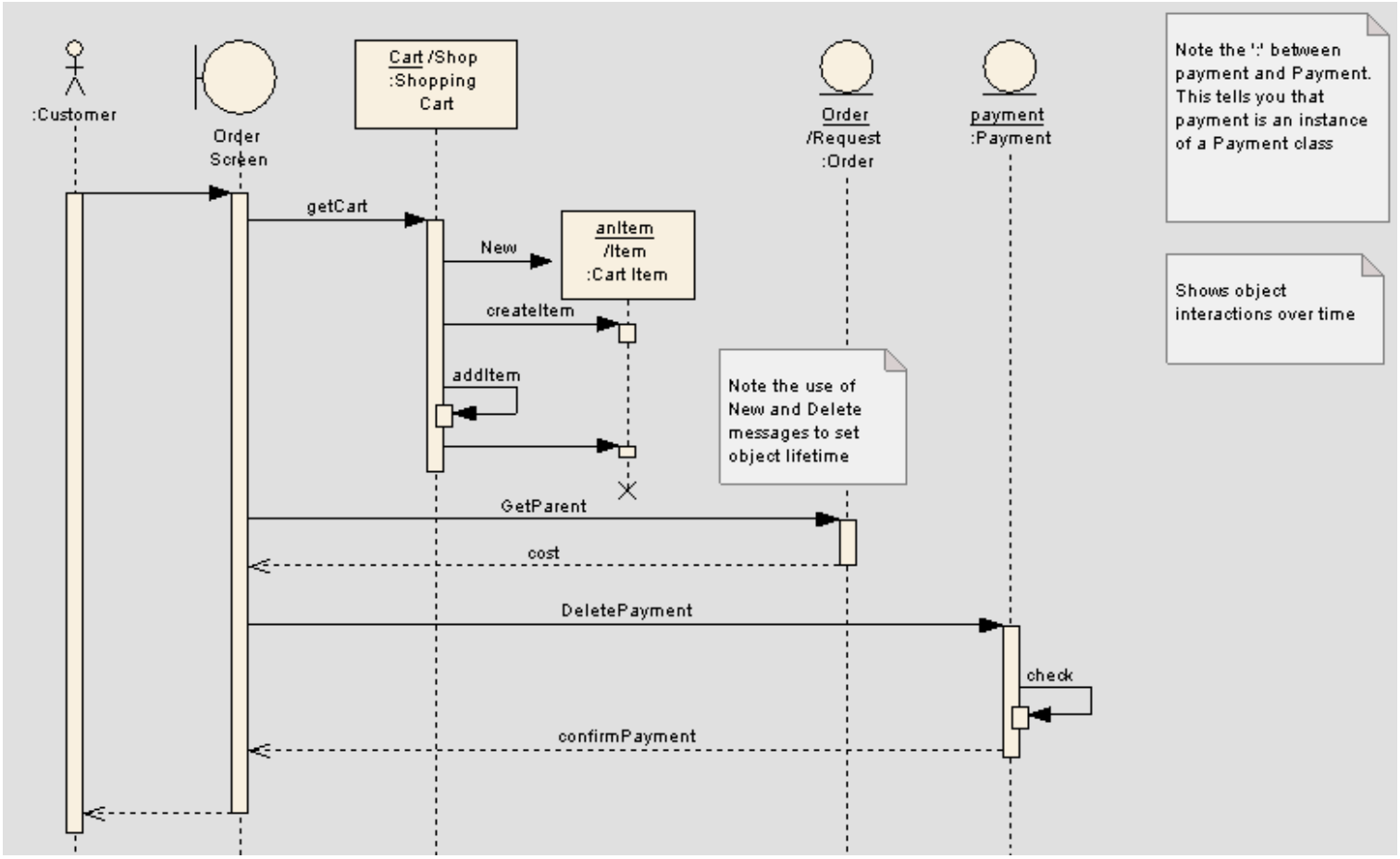
Interacciones

- *"Interactions are used in a number of different situations. They are used to get a better grip of an interaction situation for an individual designer or for a group that need to achieve a common understanding of the situation. Interactions are also used during the more detailed design phase where the precise inter-process communication must be set up according to formal protocols.*
- *Depending on their purpose, an interaction can be displayed in several different types of diagrams: Sequence Diagrams, Interaction Overview Diagrams and Communication Diagrams. Optional diagram types such as Timing Diagrams and Interaction Tables come in addition.*
- *The most visible aspects of an Interaction are the messages between the lifelines. The sequence of the messages is considered important for the understanding of the situation. The data that the messages convey and the lifelines store may also be very important, but the Interactions do not focus on the manipulation of data even though data can be used to decorate the diagrams."*

[especificación UML 2.0]

Diagrama de Secuencia

- Describe la interacción entre objetos en el tiempo
- El tiempo transcurre hacia abajo en el diagrama



Principales Elementos

- Actor: puede iniciar una interacción

- Lifeline: línea de vida de un objeto

- Boundary: objeto de interfaz

- Control: objeto con lógica de control

- Entity: objeto de datos

- Message: a nivel de objetos, invocación de un método

- Self-Message: mensaje de un objeto a sí mismo

Diagrama de Comunicación

- Describe la interacción entre objetos en el tiempo
- La secuencia está dada por la numeración de los mensajes
- Similar al diagrama de secuencia, pero con un mayor énfasis en la estructura

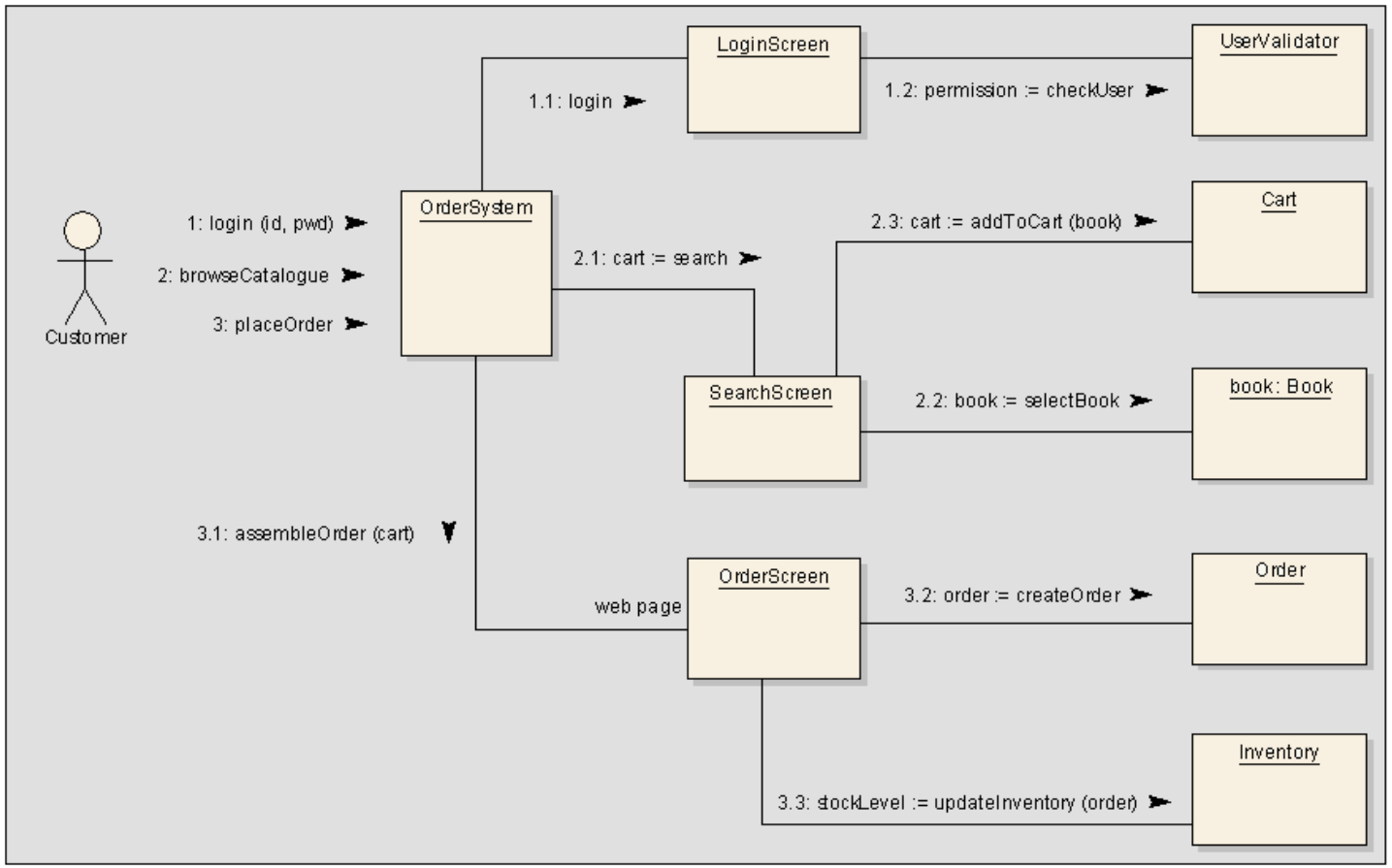


Diagrama de Tiempo

- Describe el comportamiento de diversos objetos dentro de una escala de tiempo
- Util para sistemas de tiempo real, de control automático, etc.

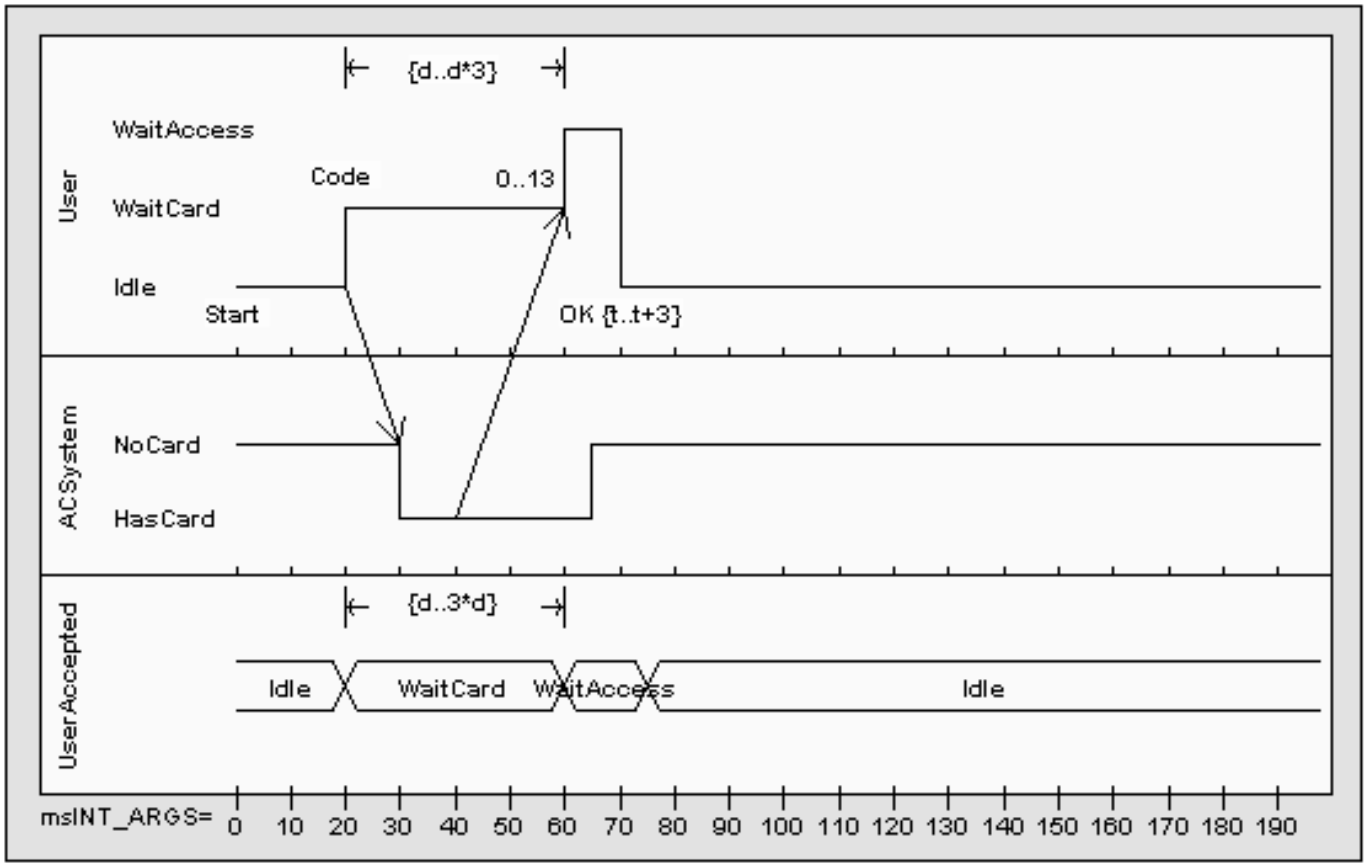
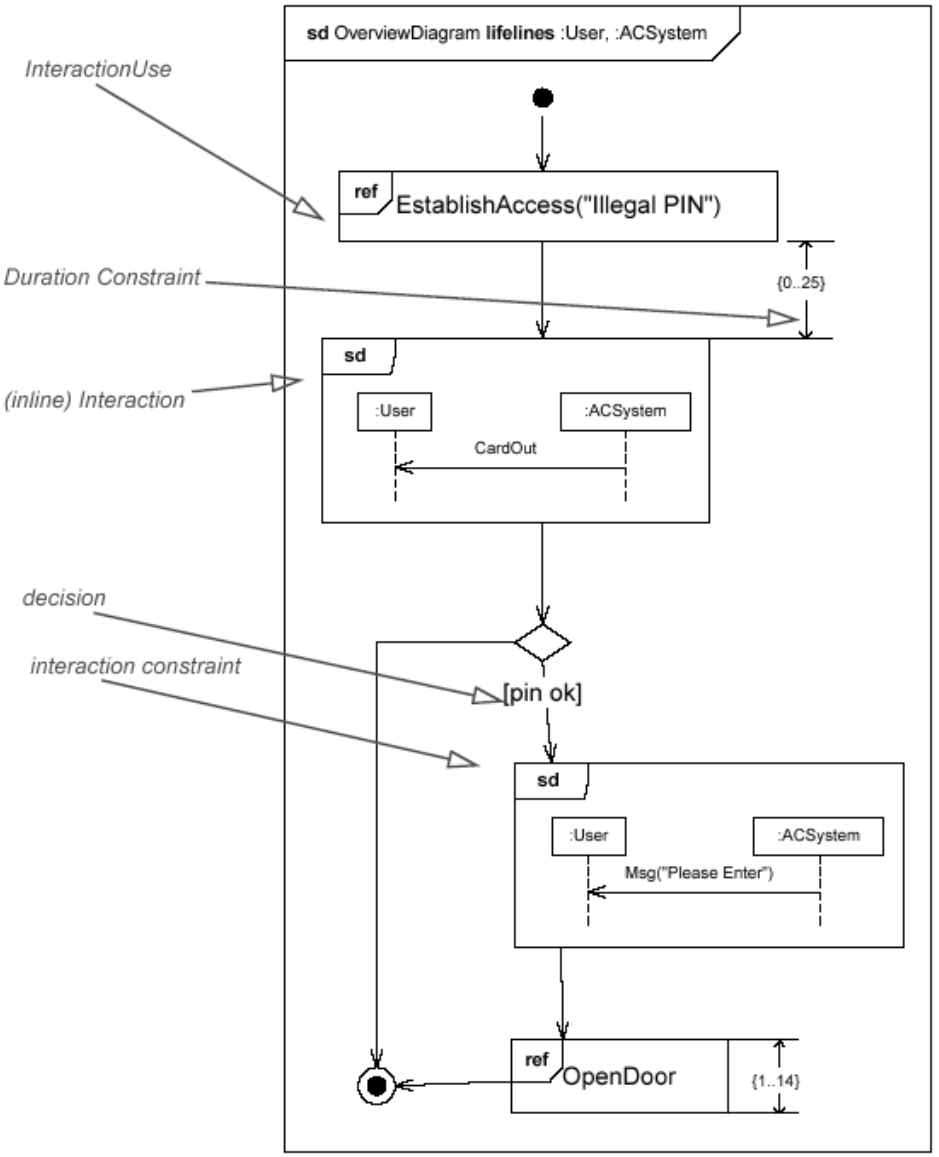


Diagrama de Interacción Global

- Un Diagrama de Interacción Global (Interaction Overview Diagram) describe la cooperación entre diagramas de interacción
- Es una variante de un diagrama de actividades en que los nodos corresponden a otros diagramas de interacción ("inline" o referencias a diagramas existentes)



Diagramas de Estructura

- Los diagramas de estructura describen los elementos estructurales que componen un sistema
  - Diagrama de Paquetes (Package Diagram)
  - Diagrama de Clases (Class Diagram)
  - Diagrama de Objetos (Object Diagram)
  - Diagrama de Componentes (Component Diagram)
  - Diagrama de Estructura Compuesta (Composite Structure Diagram)
  - Diagrama de Despliegue (Deployment Diagram)
- Estos diagramas pueden reflejar la estructura estática del sistema (por ejemplo diagramas de paquetes y de clases) o la estructura del sistema en ejecución (por ejemplo diagramas de objetos)

Diagrama de Paquetes

- Describe la organización de un sistema en sus paquetes y componentes

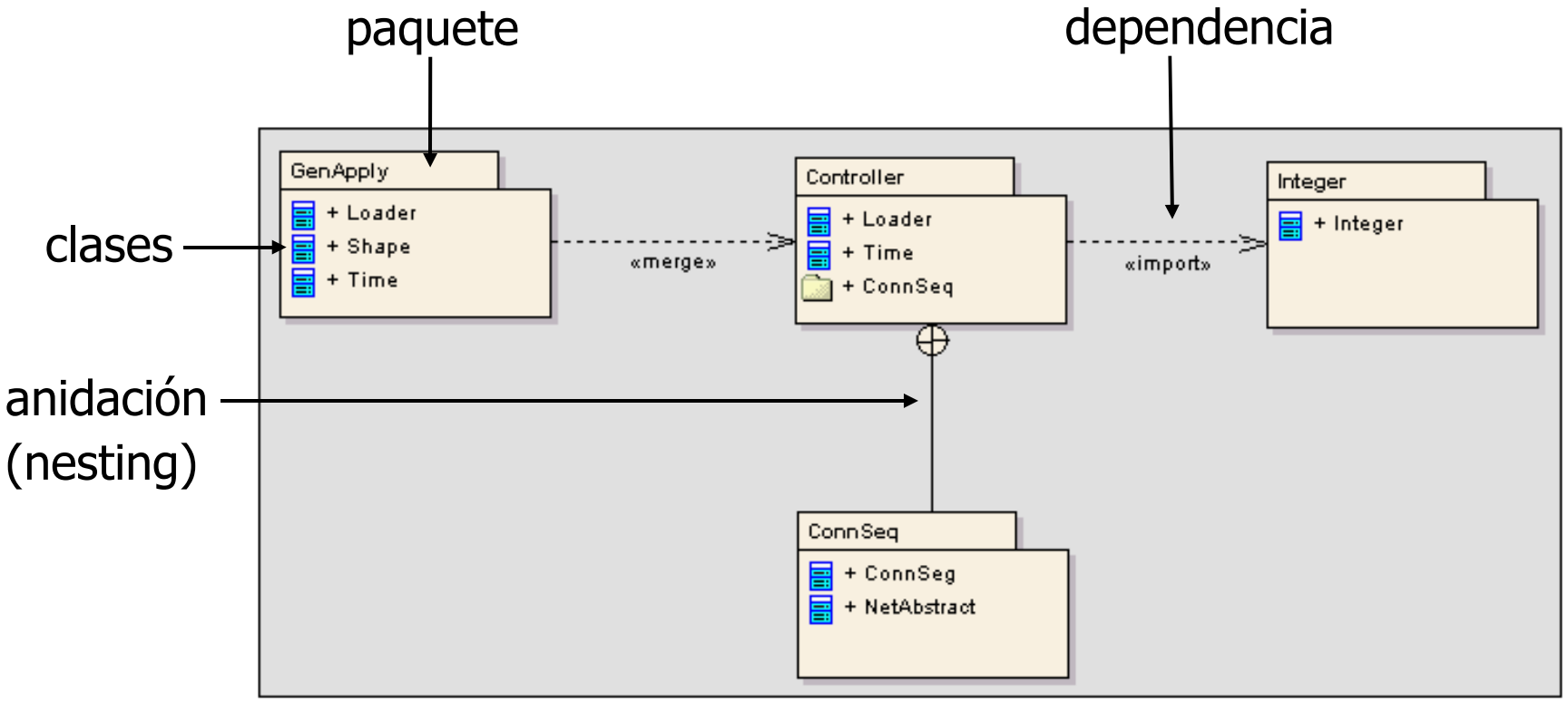
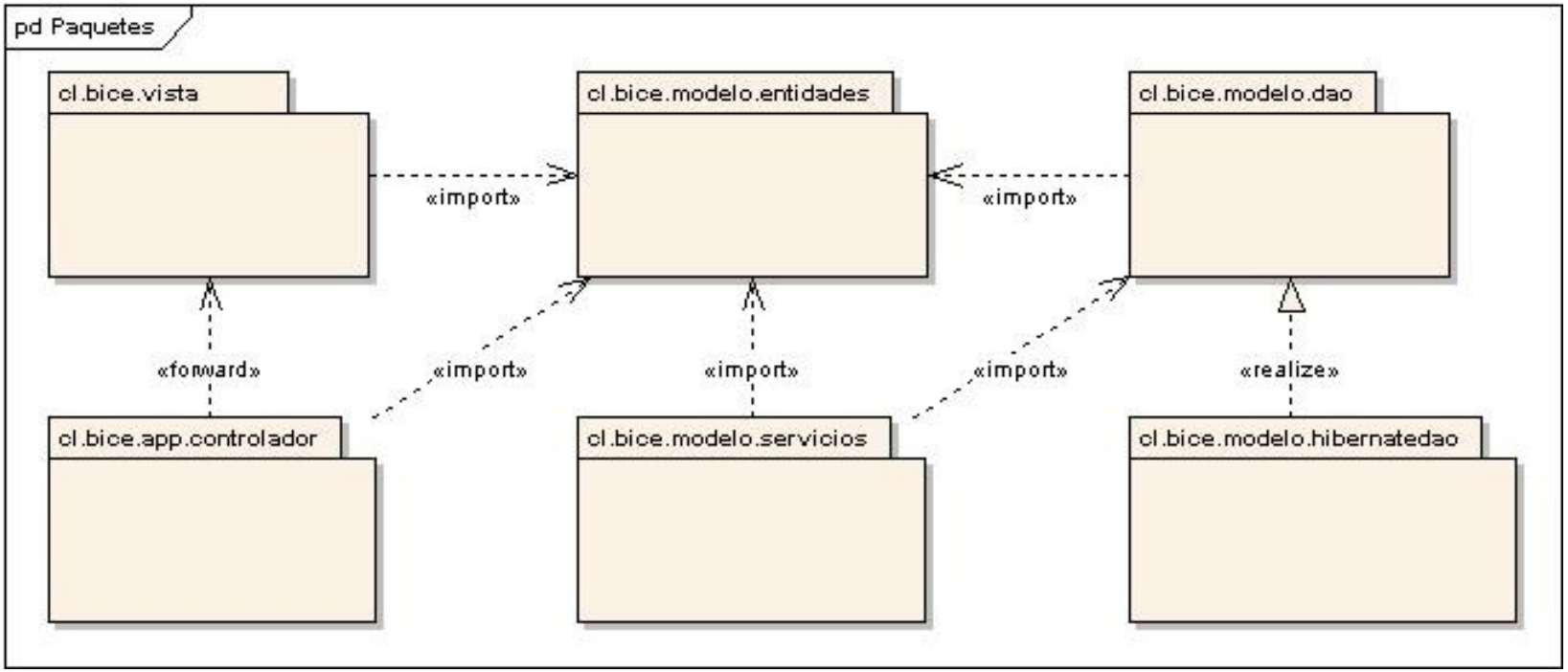


Diagrama de Paquetes

- El diagrama de paquetes entrega una visión global de los paquetes y sus dependencias
- Es importante que la dependencia entre los paquetes se mantenga controlada, y en lo posible que sea pequeña



Herencia entre Paquetes

- La figura muestra un uso conceptual de la relación de herencia entre paquetes

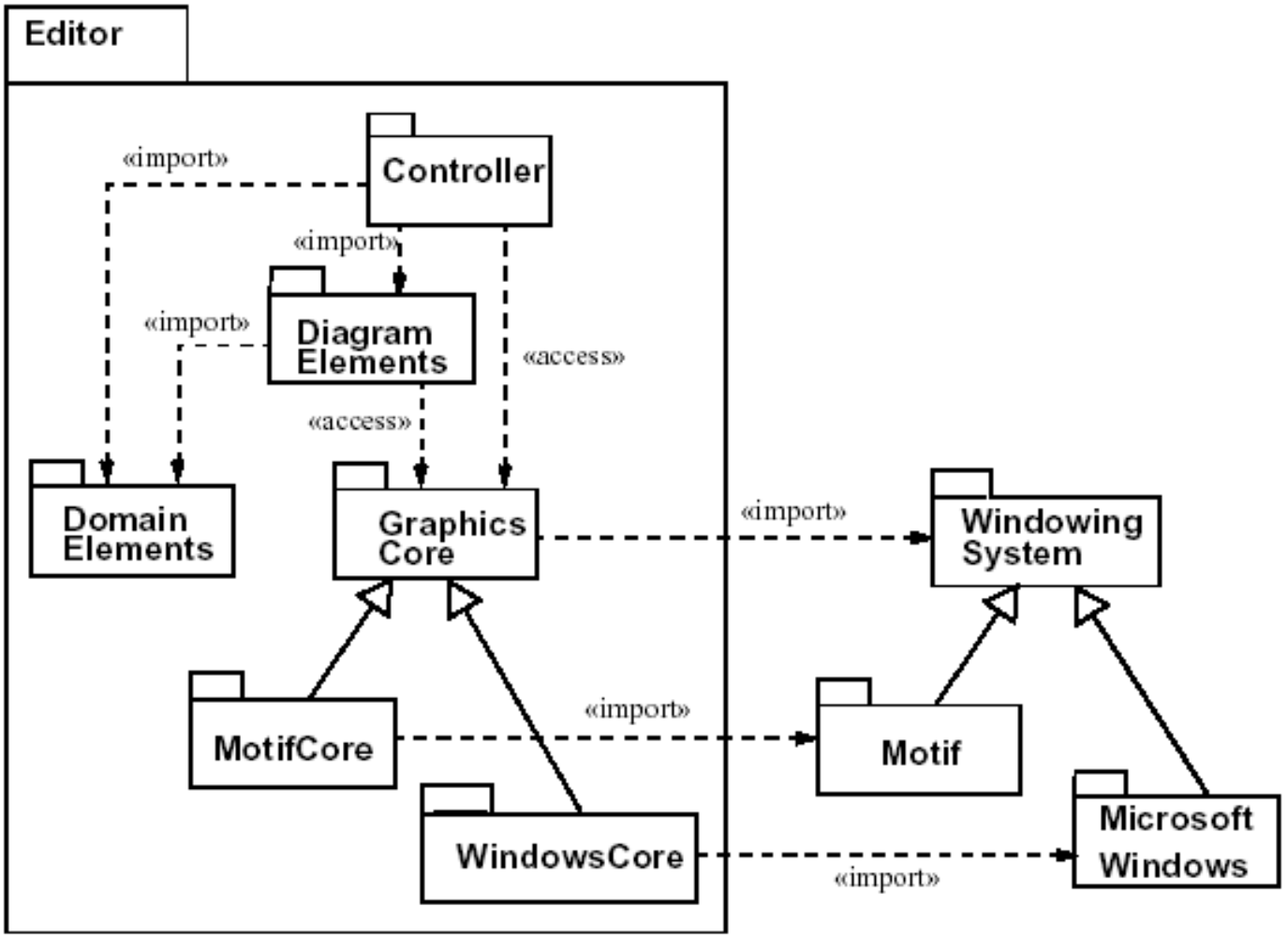
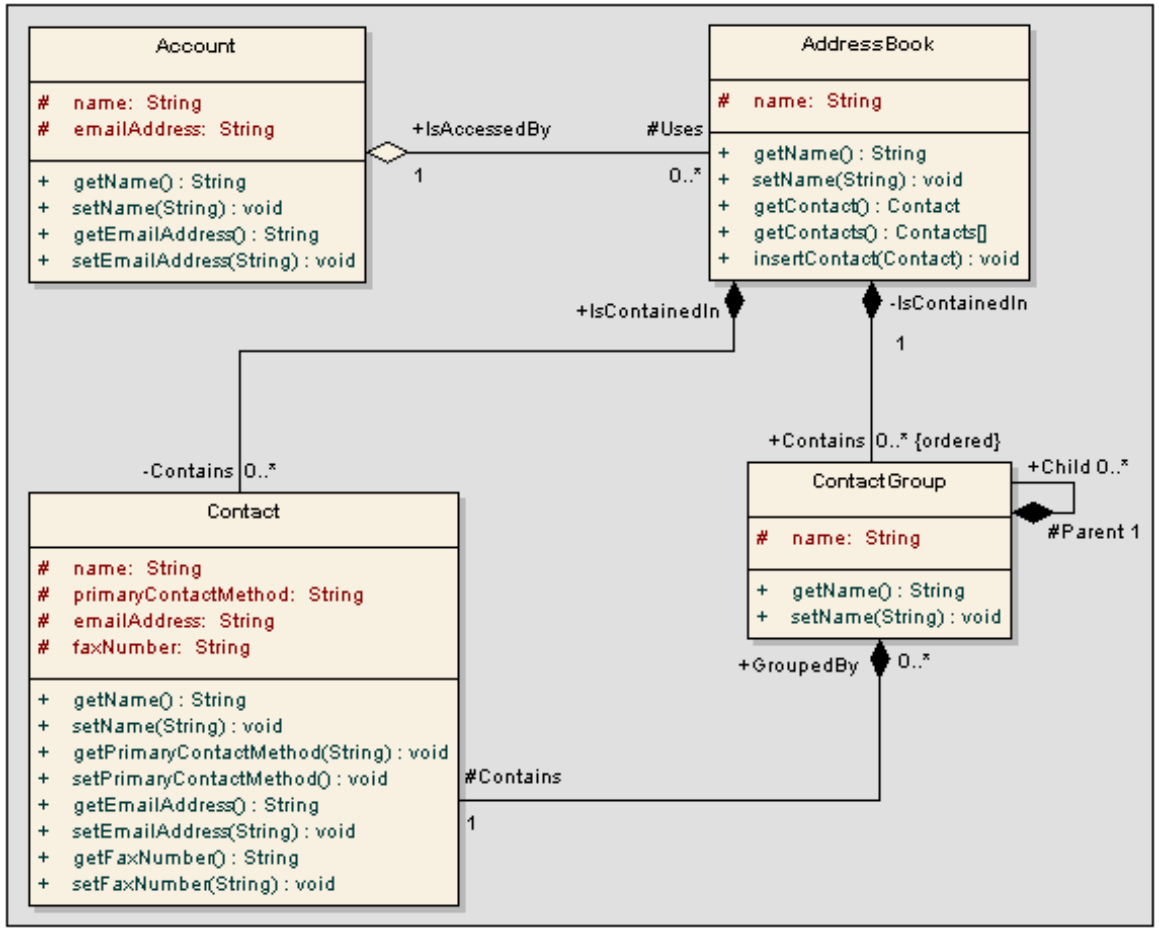


Diagrama de Clases

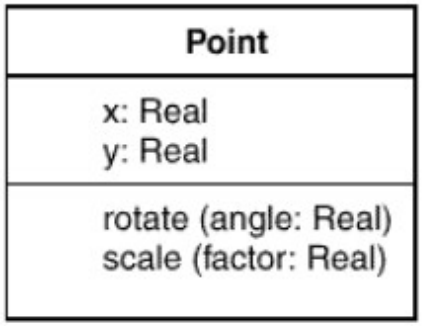
- Usado principalmente para describir las clases de un módulo o sistema y sus relaciones
- Es usado también para describir modelos de dominio y de datos

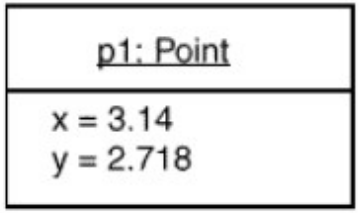


Clases y Objetos

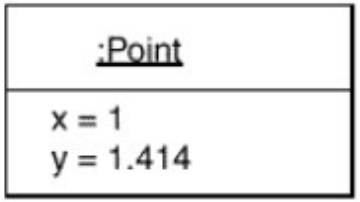
- Las clases se representan en rectángulos con 3 compartimentos:
  - El nombre de la clase
  - Los atributos
  - Las operaciones
- Los objetos se representan en rectángulos con 2 compartimentos
  - Nombre y clase del objeto
  - Valores de los atributos

clase Point



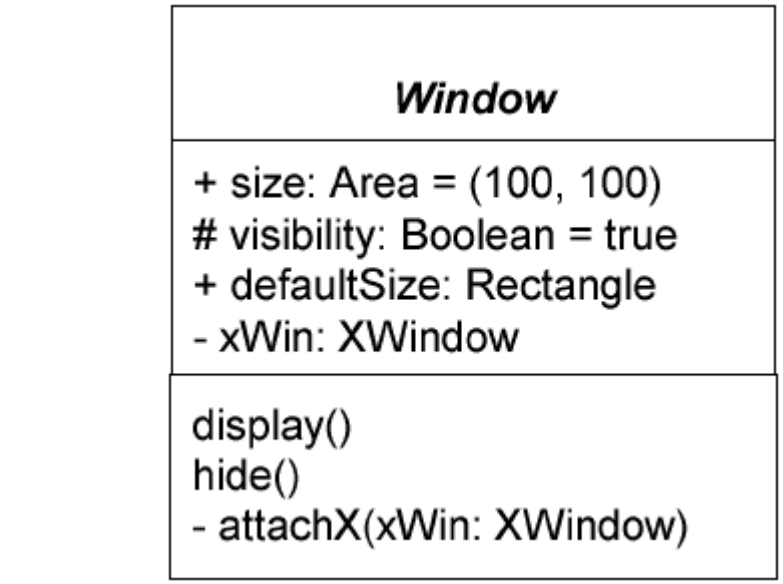


2 objetos Point



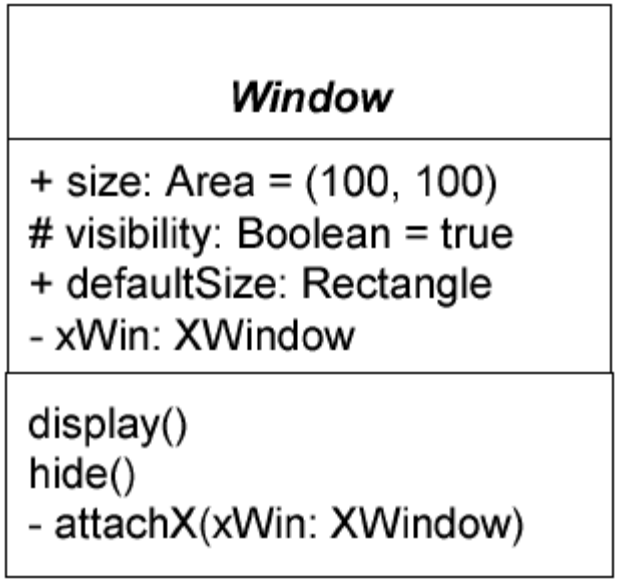
Atributos

- Notación:
  - <visibilidad> <nombre> : <tipo> = <valor inicial> <restricción>
- Visibilidad:
  - + public
  - # protected
  - ~ package
  - - private
- Multiplicidad: [ ]
  - colors[3]: Color
  - points[2..*]: Point
- Atributo derivado: /
  - / area
- Ejemplos:
  - + size: Integer[0..1] = (100,100)
  - + size: Area = (100,100)
  - + color: String {readOnly}



Operaciones

- Notación:
  - <visibilidad> <nombre> (<parámetros>) : <retorno>
- Parámetros (separados por ','):
  - <{in,out,inout}> <nombre> : <tipo> = <valor default>
- Ejemplos:
  - + sum(m1: Matrice, m2: Matrice): Matrice
  - + deposito(in monto: double, in moneda: Moneda = PESO)
  - display()
  - hide()
  - - attachX(xWin: XWindow)

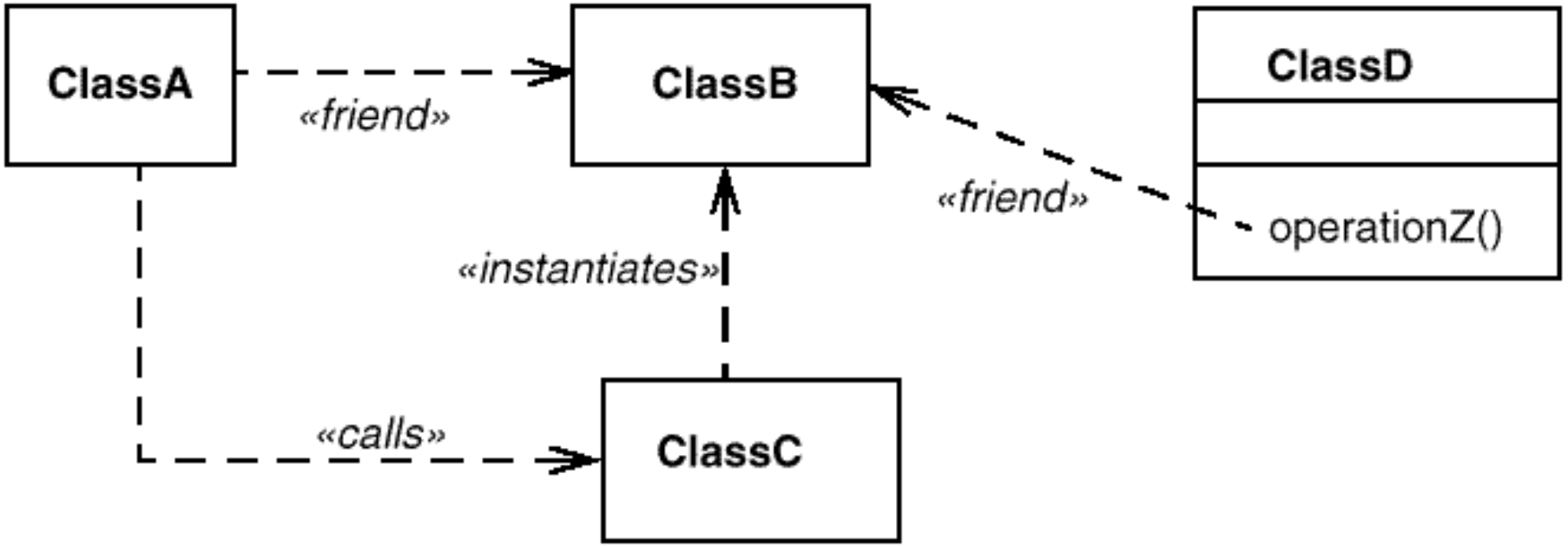


Presentación

- La especificación establece algunas definiciones de presentación adicionales:
  - Atributos y operaciones static: subrayado
  - Clases y operaciones abstractas: *letra itálica*

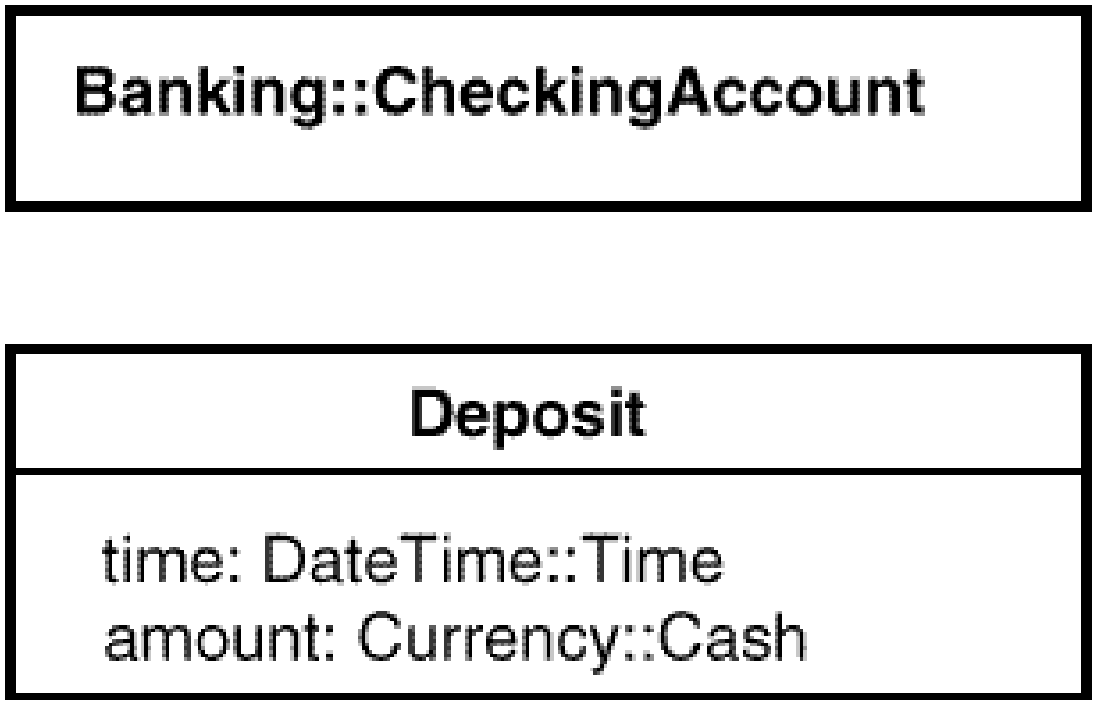
Dependencia

- La dependencia entre clases se representa mediante una flecha punteada, que va desde la clase dependiente hacia la clase de la cual depende



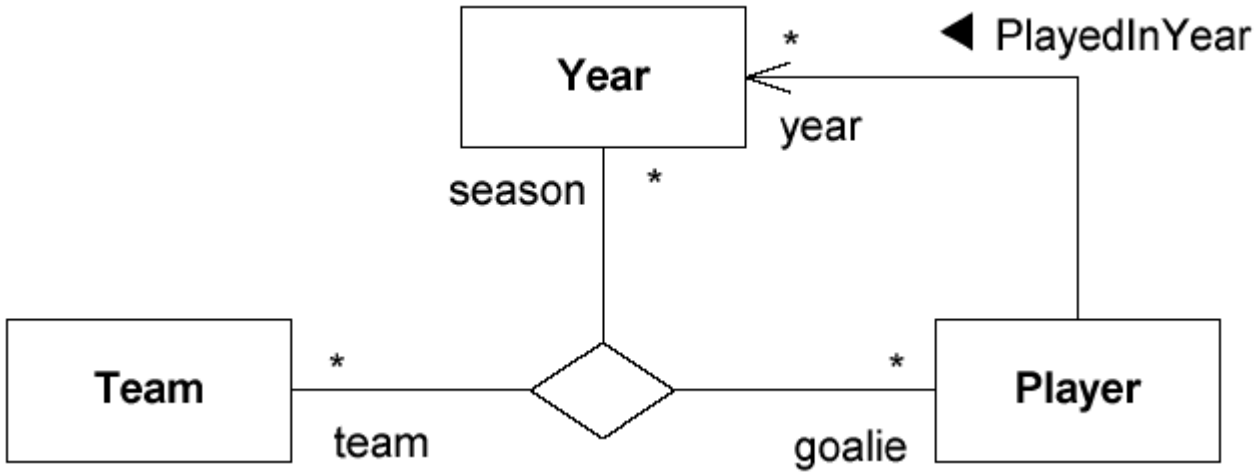
Clases en Otros Paquetes

- Al describir las clases de un paquete, en ocasiones es necesario hacer uso de clases de otros paquetes



Asociaciones

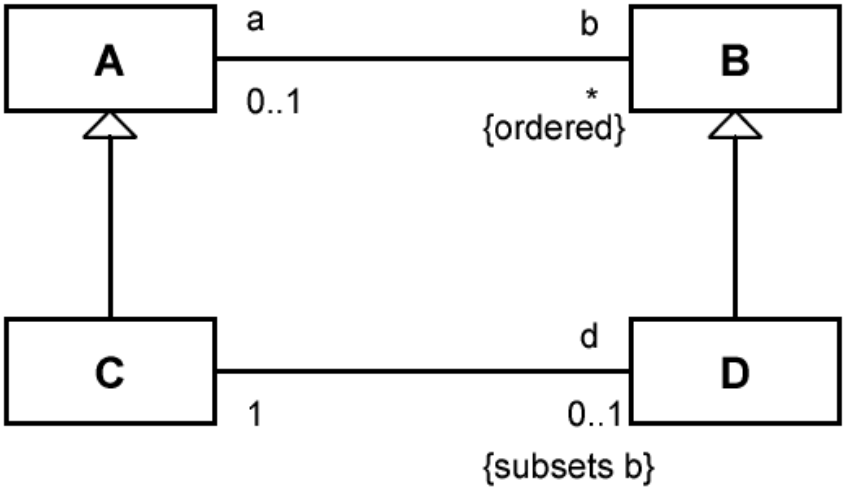
- Una asociación indica que puede haber "links" entre instancias de los tipos indicados
- Un "link" es una tupla con un valor para cada parte de la asociación



- La figura muestra una asociación binaria "PlayedInYear" entre las clases Player y Year (se lee desde Player hacia Year) y una asociación ternaria entre las clases Year, Team y Player

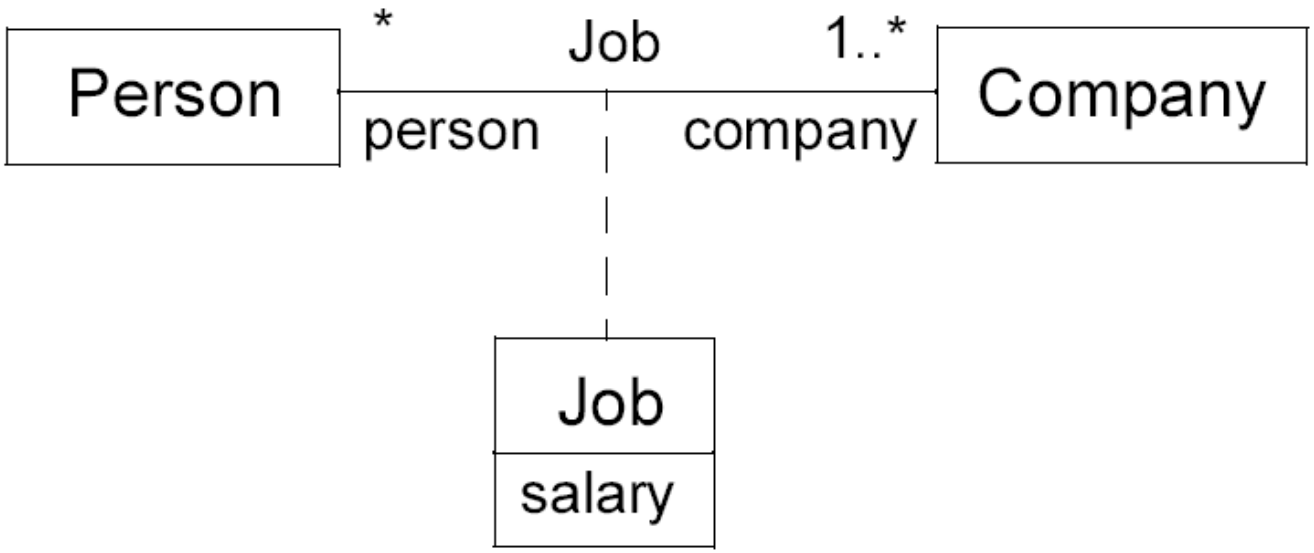
Adornos en Asociaciones

- Las asociaciones pueden tener adornos (adornments) en los extremos
- El ejemplo de la figura muestra:
  - Nombres “a”, “b” y “d”
  - Multiplicidad 0..1 en “a”, * en “b”, 1 en el extremo sin nombre, y 0..1 en “d”
  - Especificación de ordenamiento en “b”
  - Subconjunto en “d”: para una instancia de C, la colección “d” es un subconjunto de la colección “b”



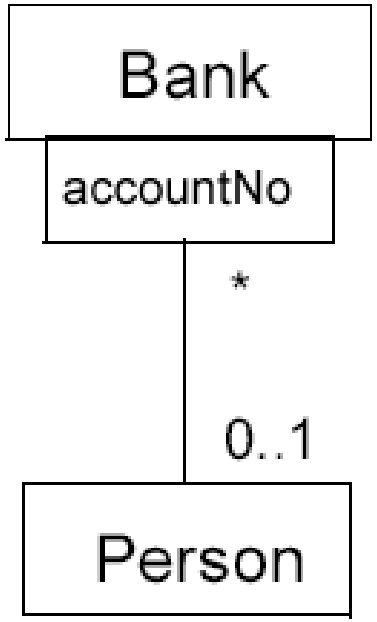
Clase de Asociación

- Una clase de asociación (association class) representa una clase cuyas instancias corresponden a las tuplas de la asociación
- En la figura, las instancias de la clase Job corresponden a tuplas (links) de la asociación entre las clases Person y Company



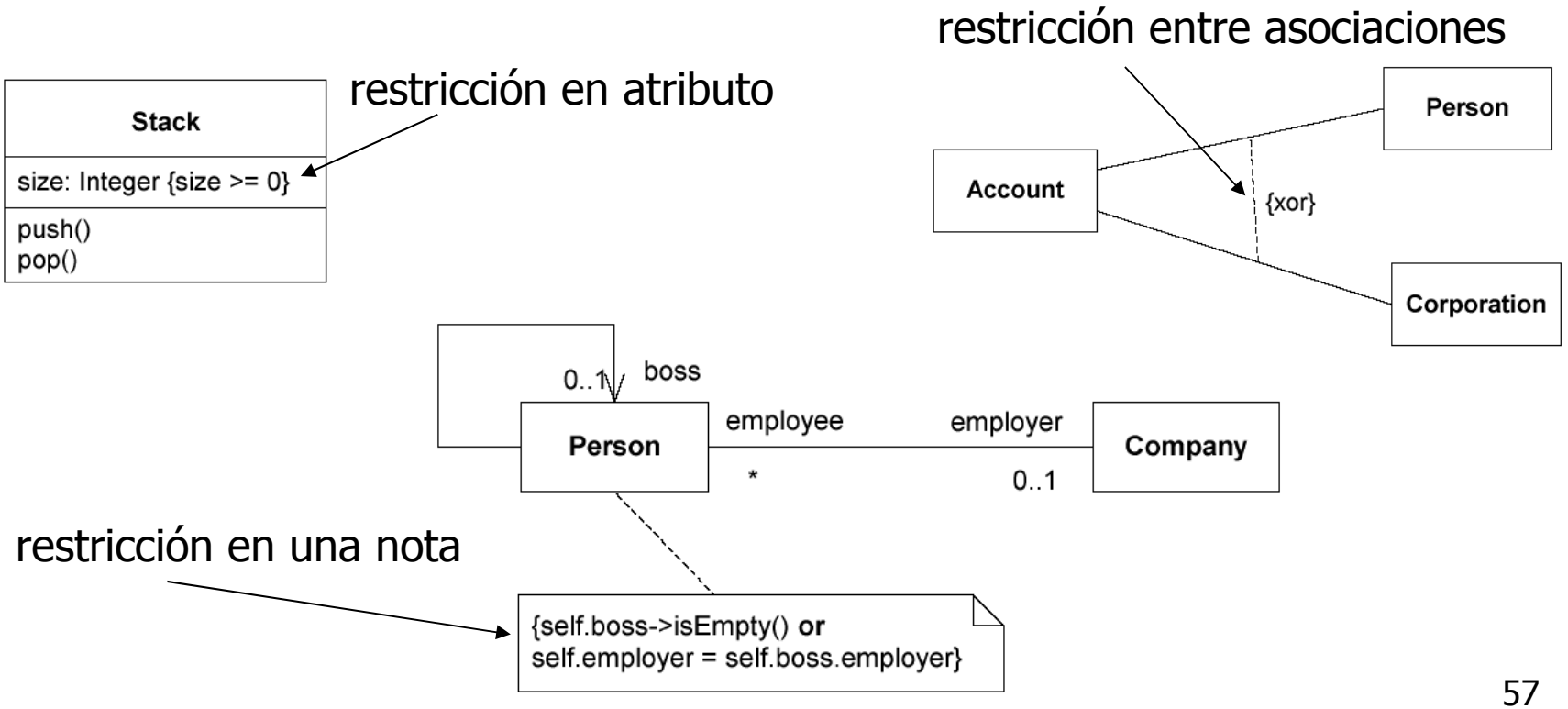
Asociaciones Calificadas

- Un calificador define una partición del conjunto de instancias asociadas con la instancia del extremo del calificador
- En la figura, el calificador "accountNo" particiona el conjunto de instancias "Person" asociadas con una instancia "Bank"



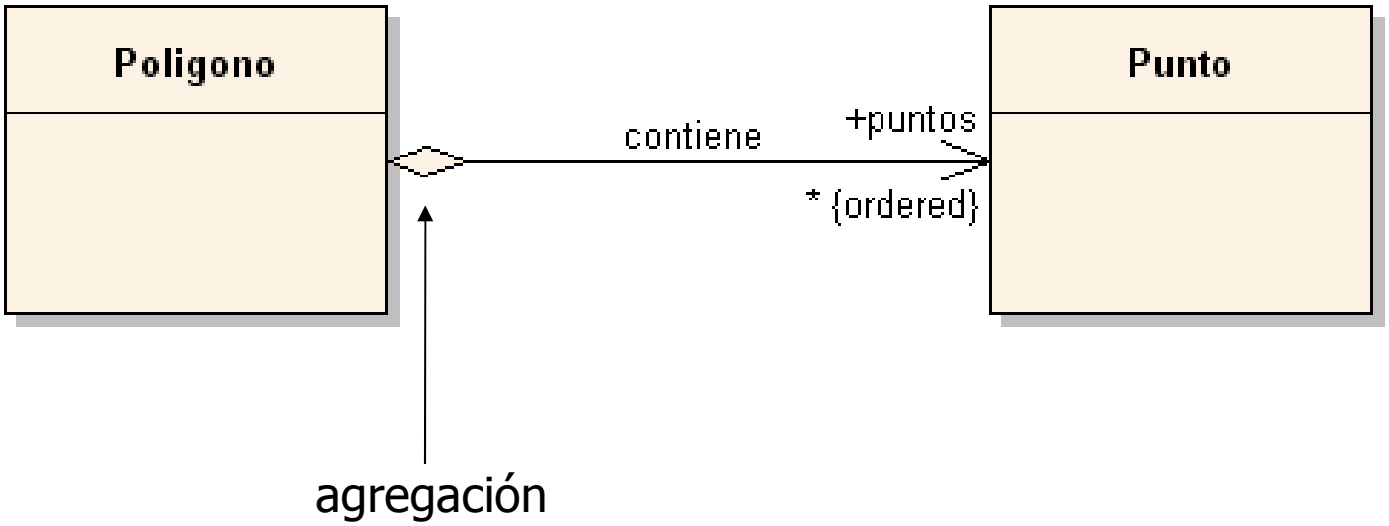
Restricciones (Constraints)

- Una restricción es una condición entre elementos del modelo que debe cumplirse
- OMG está trabajando en la especificación de OCL (Object Constraint Language), el lenguaje para expresar restricciones



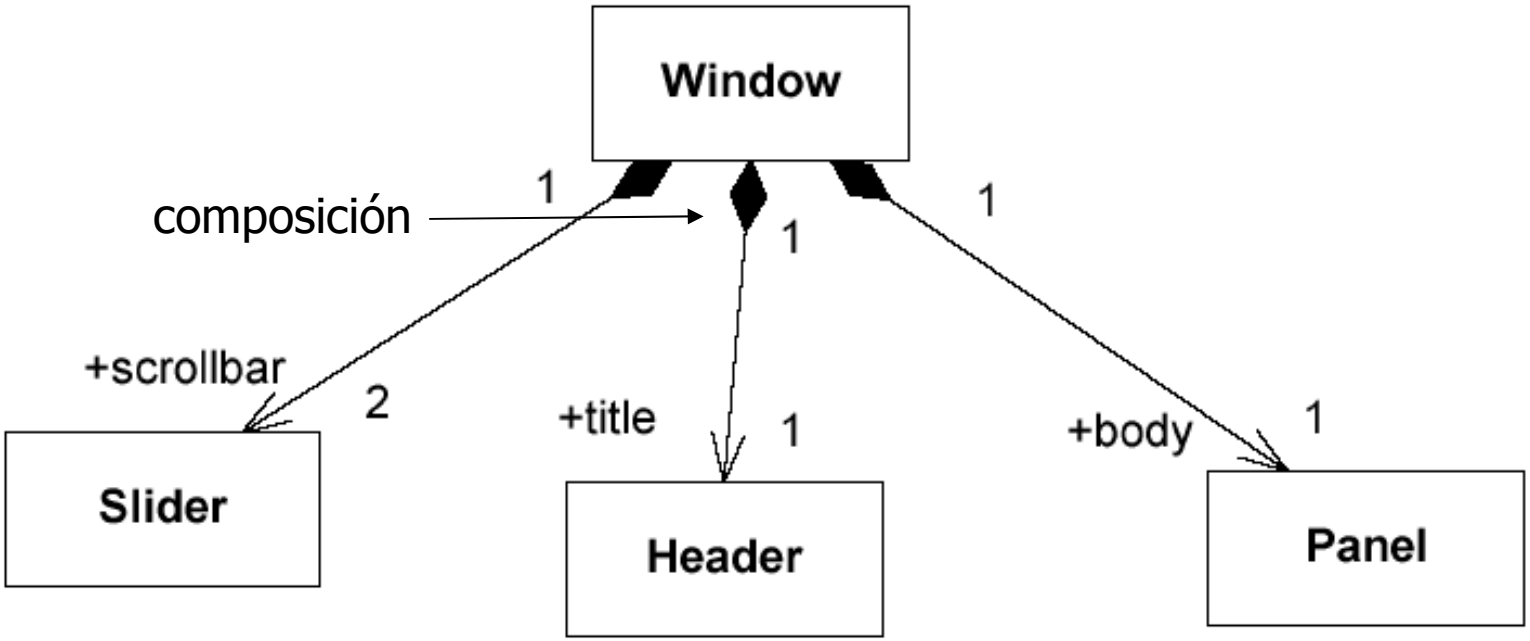
Agregación

- Una agregación (aggregation) establece que un elemento se compone de componentes más pequeños



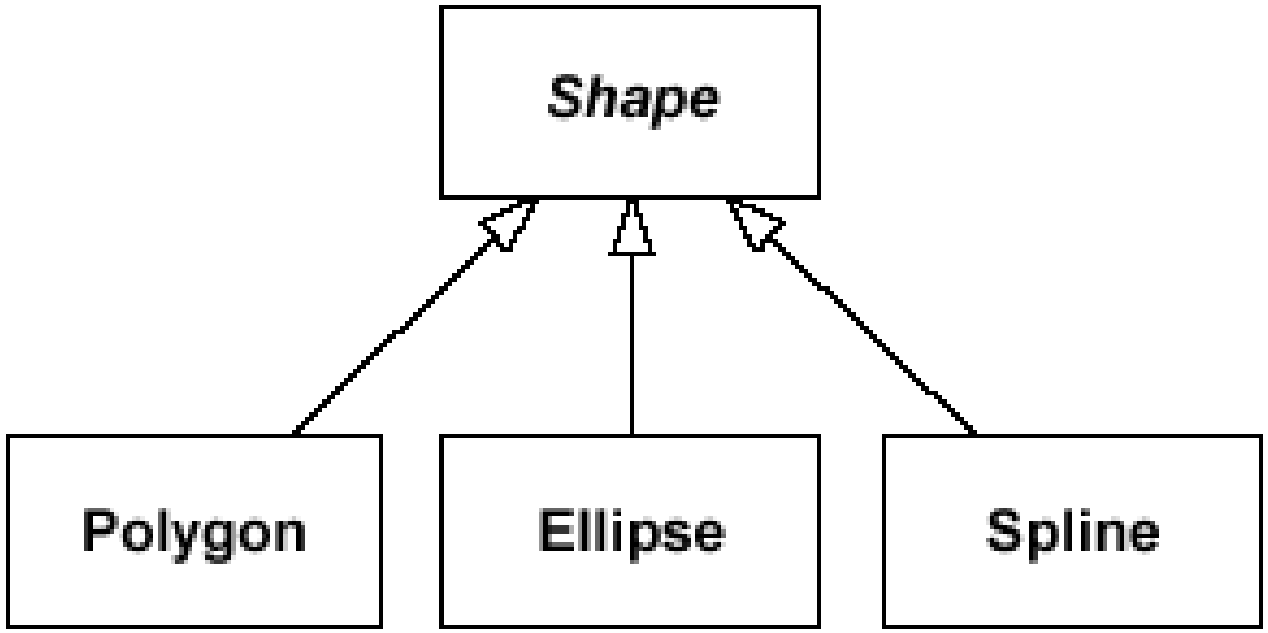
Composición

- Una agregación de composición (composite aggregation) representa una agregación más fuerte, en que los componentes pequeños le pertenecen al componente compuesto: si éste es eliminado, usualmente sus partes son eliminadas con él



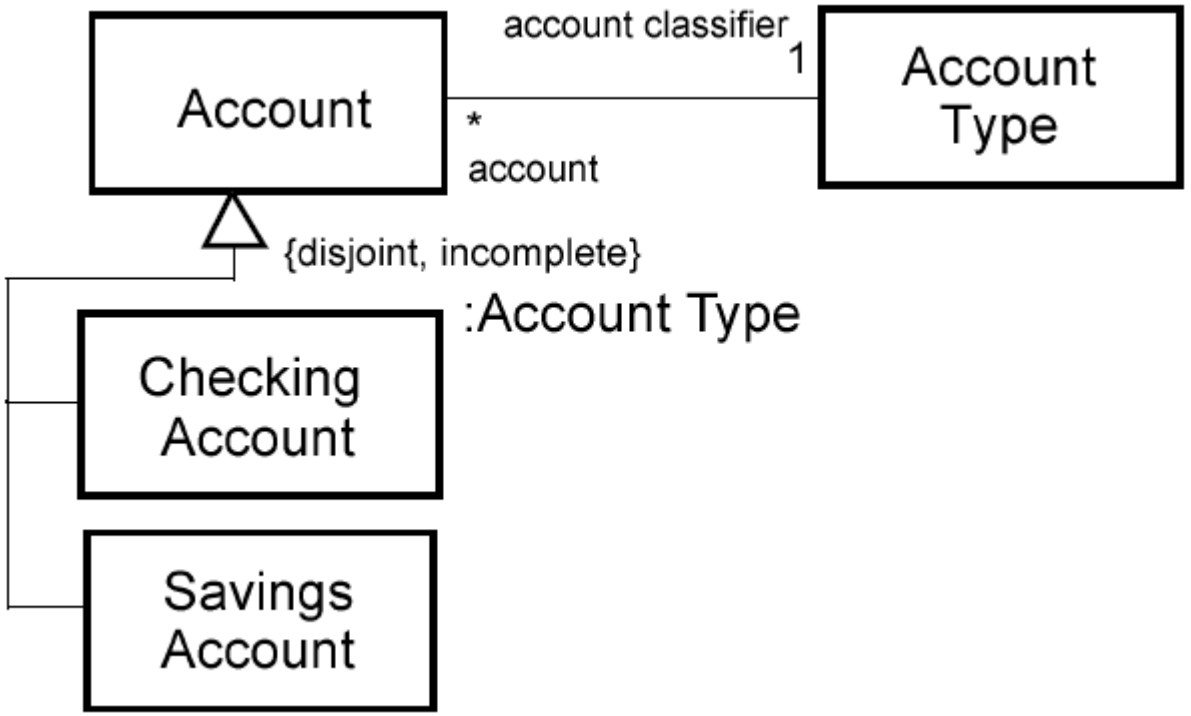
Herencia (Generalización)

- La herencia se representa mediante una flecha que va desde la clase derivada hacia la clase base



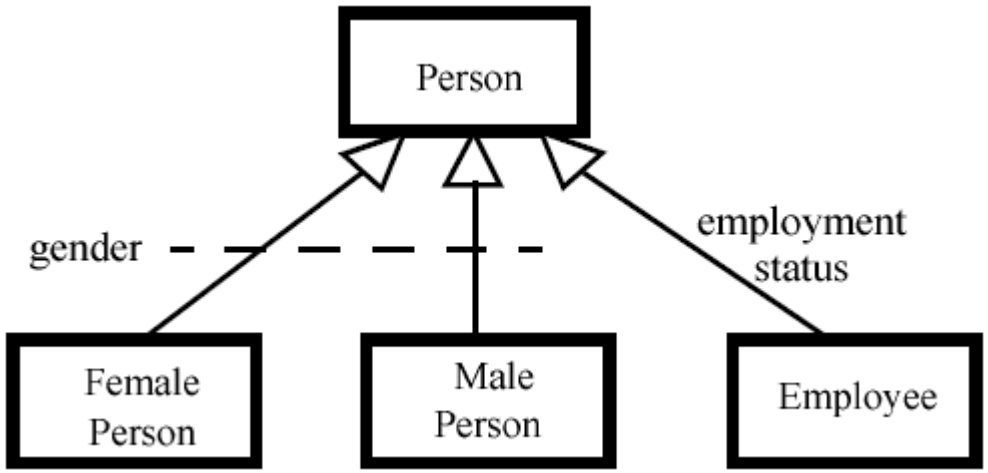
Restricciones en la Clasificación

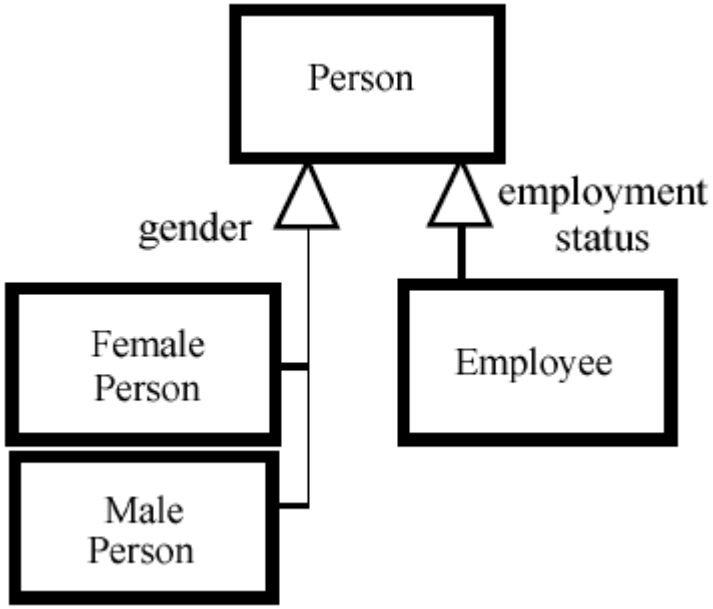
- Es posible establecer restricciones en la clasificación de herencia utilizada



Particiones

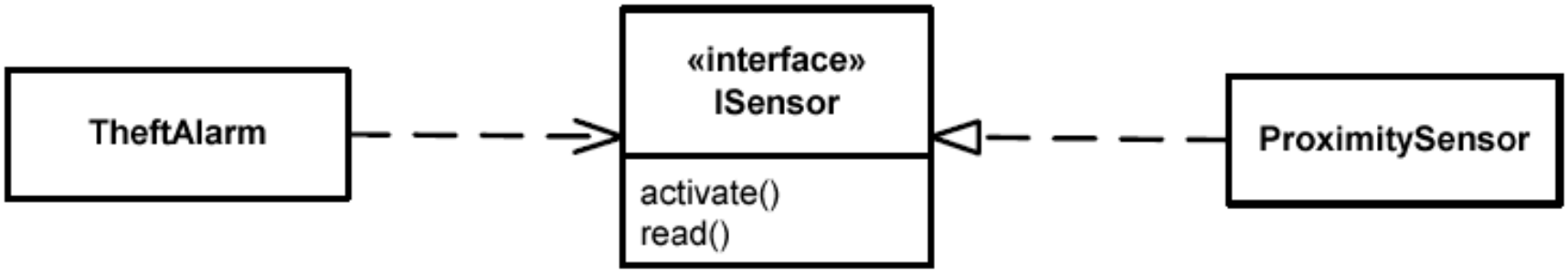
- Es posible definir particiones de subtipos con diferentes criterios





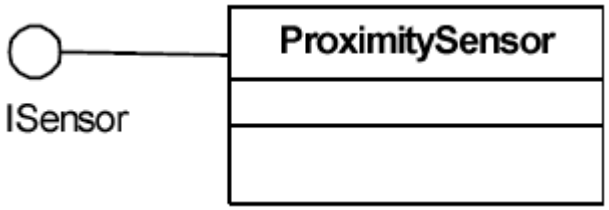
Interfaces

- Una interfaz se representa como una clase con el estereotipo “interface”
- La implementación de una interfaz se representa mediante una flecha similar a la de herencia, pero punteada

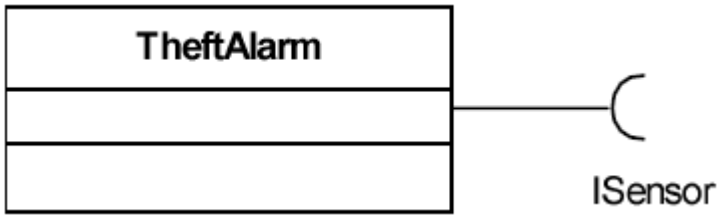


Realización y Uso de Interfaces

- La siguiente figura muestra que la clase ProximitySensor implementa la interfaz Isensor



- La siguiente figura muestra que la clase TheftAlarm hace uso de la interfaz ISensor



Conector de Ensamble

- Un conector de ensamble (assembly) entre dos componentes establece que uno de ellos provee los servicios que el otro requiere

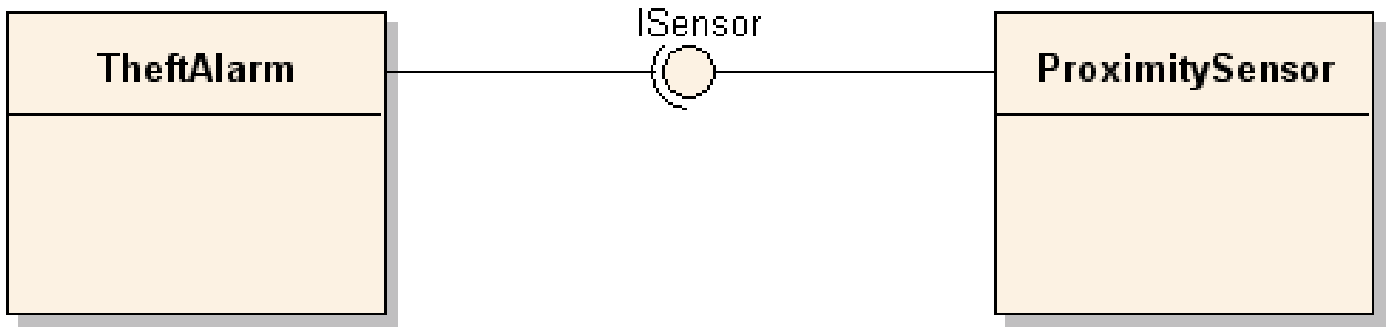


Diagrama de Objetos

- Describe una vista de objetos y sus relaciones en un momento de la ejecución de un sistema

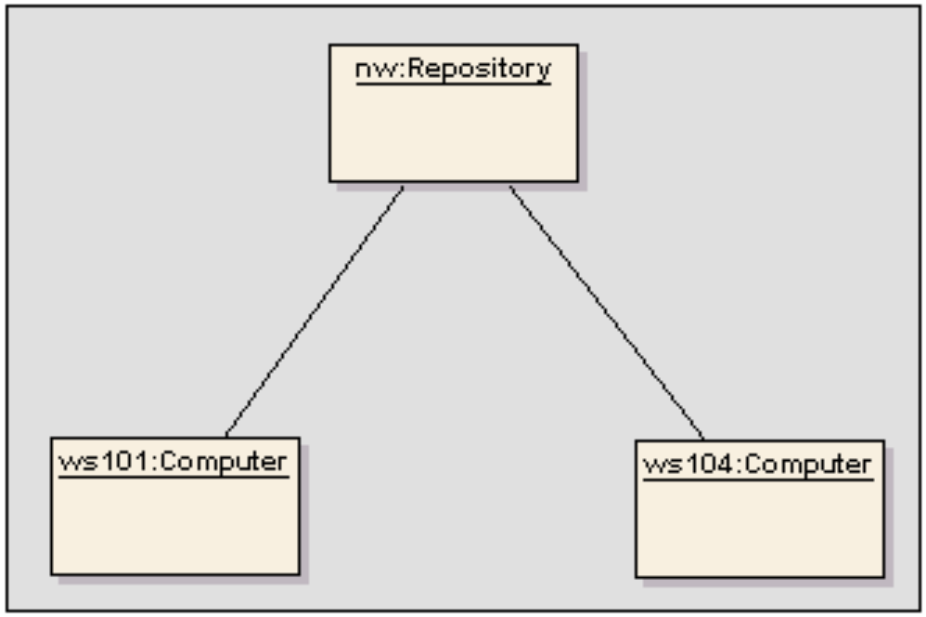


Diagrama de objetos

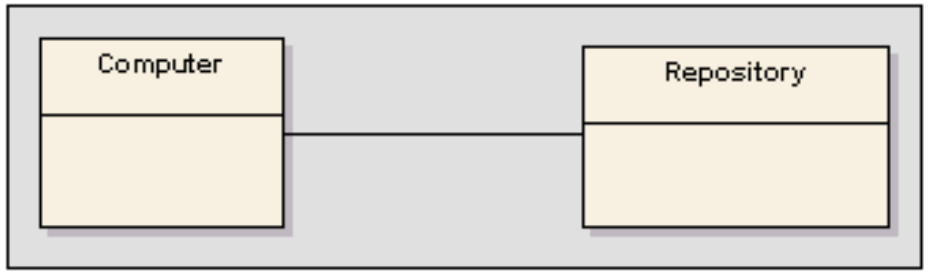
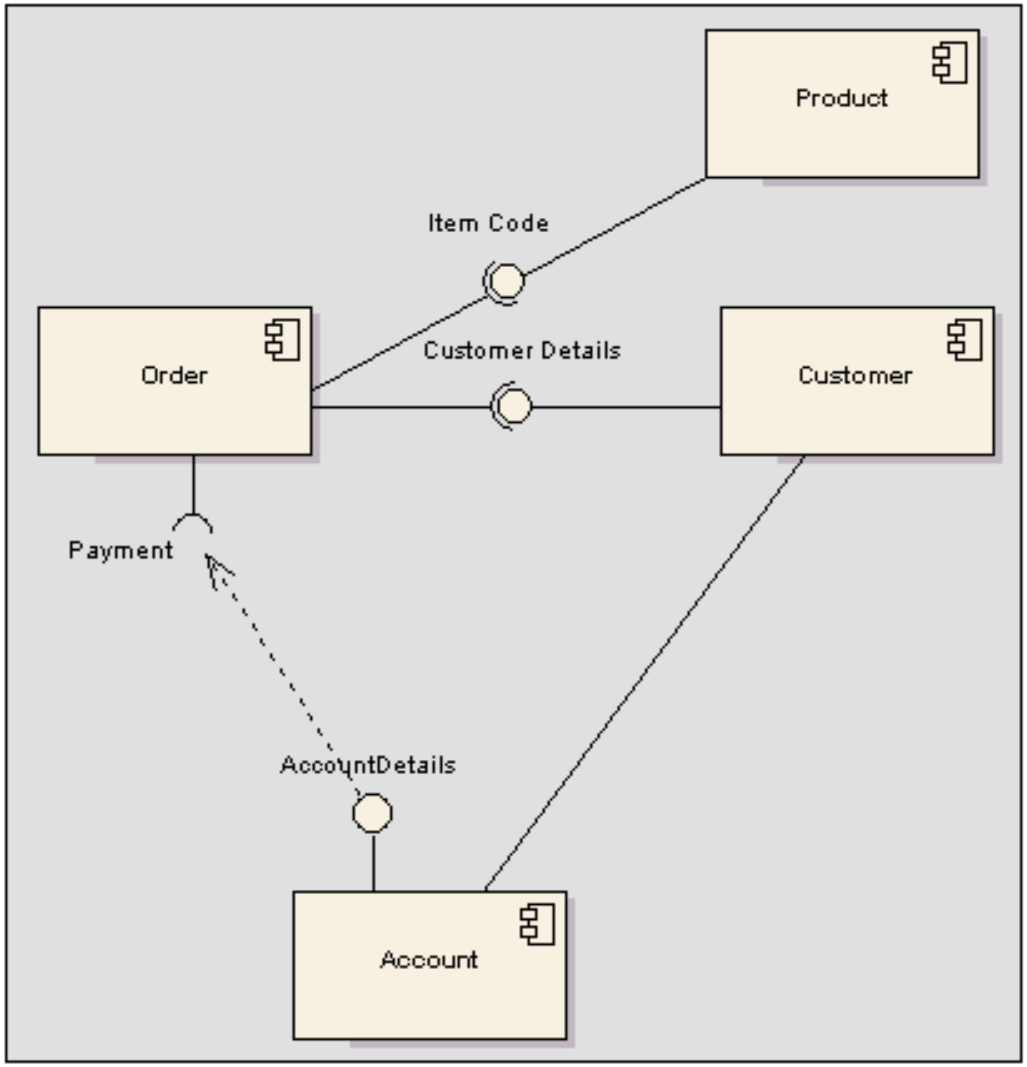


Diagrama de clases

Diagrama de Componentes

- Describe las piezas de software que forman un módulo o sistema



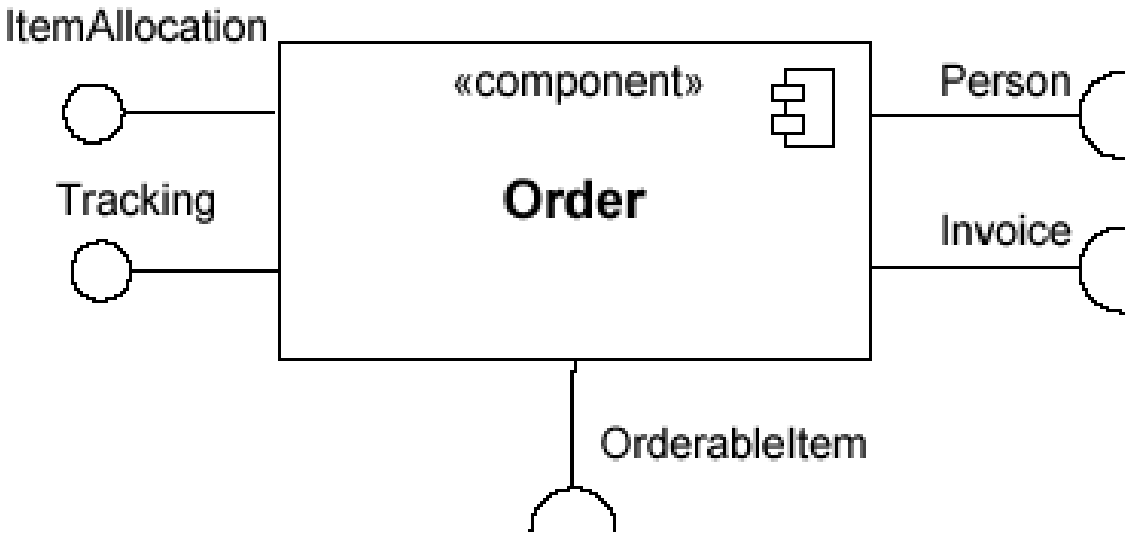
Componentes

- *"A component is a self contained unit that encapsulates the state and behavior of a number of classifiers. A component specifies a formal contract of the services that it provides to its clients and those that it requires from other components or services in the system in terms of its provided and required interfaces."*
- *"The component concept addresses the area of component-based development and component-based system structuring, where a component is modeled throughout the development life cycle and successively refined into deployment and run-time."*
- *"An important aspect of component-based development is the reuse of previously constructed components. A component can always be considered an autonomous unit within a system or subsystem. It has one or more provided and/or required interfaces (potentially exposed via ports), and its internals are hidden and inaccessible other than as provided by its interfaces."*

[especificación UML 2.0]

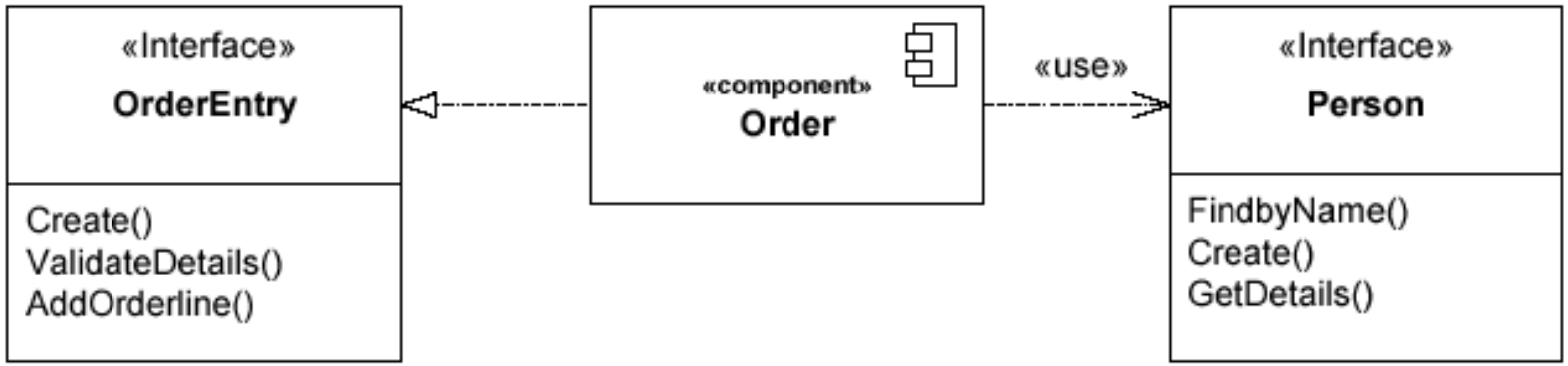
Componentes e Interfaces

- Componente Order
  - Interfaces provistas: ItemAllocation, Tracking
  - Interfaces requeridas: Person, Invoice, OrderableItem



Componentes e Interfaces

- Las interfaces provistas y requeridas pueden representarse explícitamente, lo que permite desplegar más información (por ejemplo, las operaciones de las interfaces)



Vista Interna de un Componente

- La figura muestra la estructura interna de un componente que contiene otros componentes
- Los conectores de delegación asocian el contrato externo de un componente (definido por sus puertos) con la implementación interna del comportamiento requerido

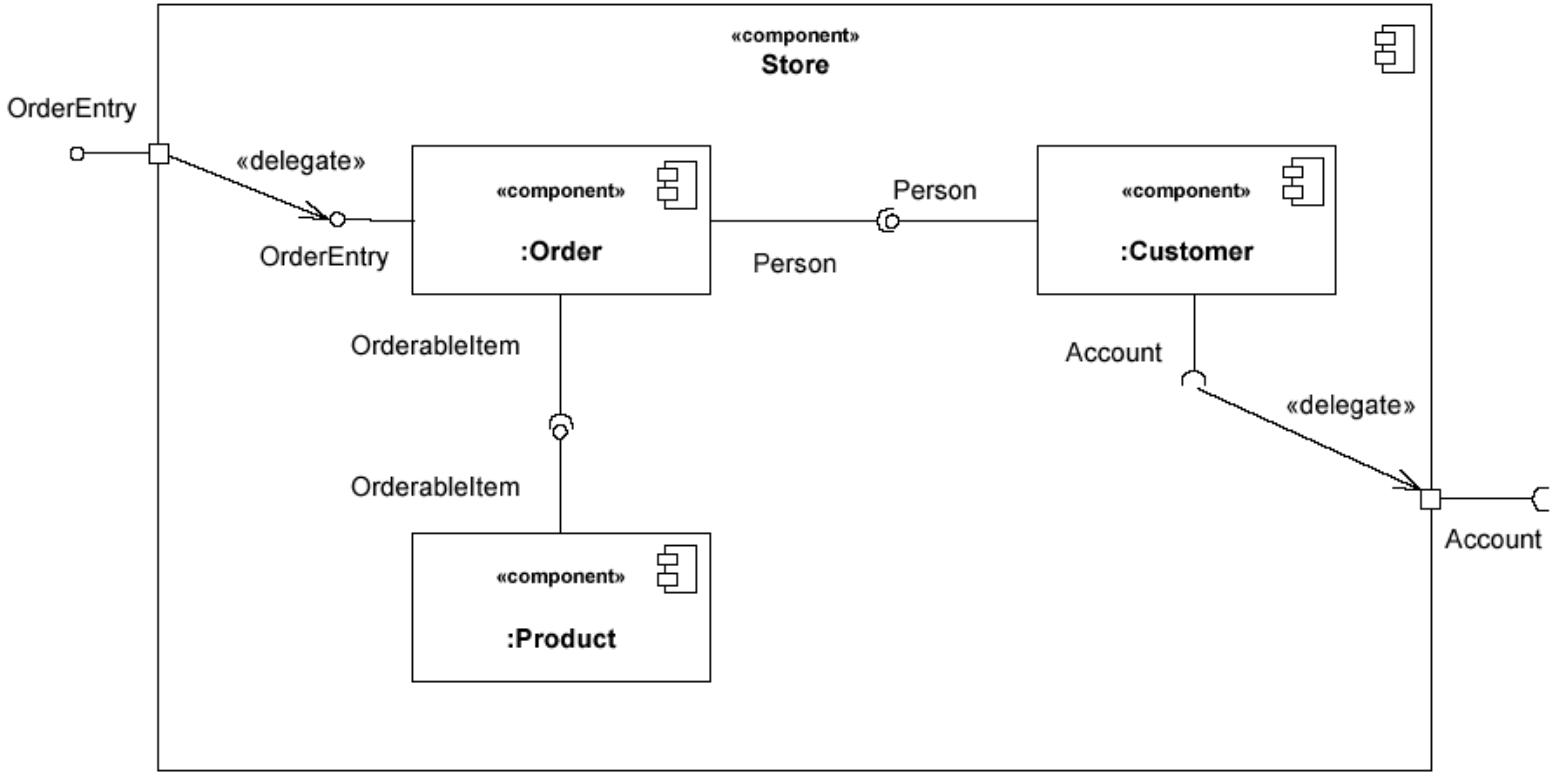
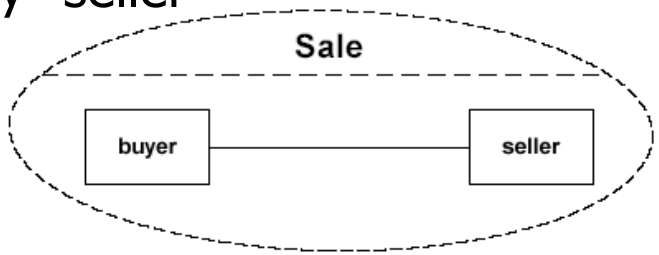


Diagrama de Estructura Compuesta

- Refleja la colaboración interna de clases, interfaces y componentes para describir una funcionalidad (útil para describir patrones de diseño)
- La figura muestra una colaboración "Sale", en la cual intervienen dos roles: "buyer" y "seller"



- La figura muestra la colaboración "BrokeredSale", que incluye 3 roles ("producer", "broker", y "consumer") y 2 ocurrencias de la colaboración "Sale"

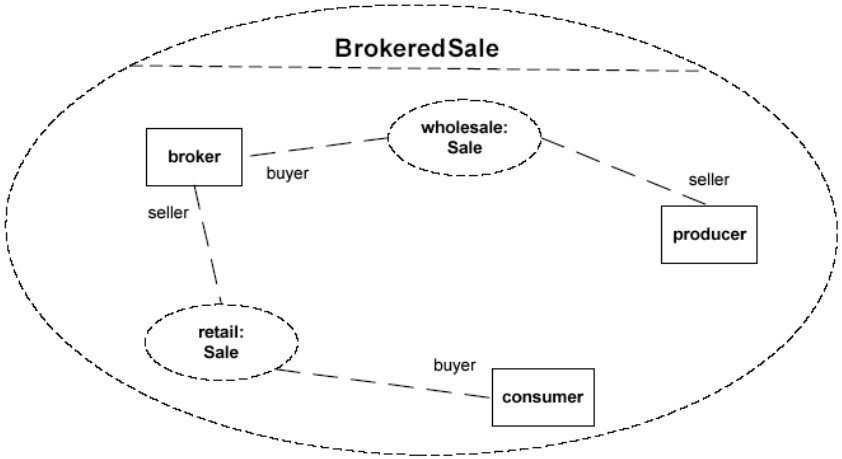


Diagrama de Despliegue

- *"The Deployments package specifies a set of constructs that can be used to define the execution architecture of systems that represent the assignment of software artifacts to nodes. Nodes are connected through communication paths to create network systems of arbitrary complexity. Nodes are typically defined in a nested manner, and represent either hardware devices or software execution environments. Artifacts represent concrete elements in the physical world that are the result of a development process."*

[especificación UML 2.0]