INGENIERÍA DEL SOFTWARE I

Tema 9

Arquitectura Lógica del Sistema

(en desarrollo OO)

Univ. Cantabria – Fac. de Ciencias

Carlos Blanco, Francisco Ruiz

Objetivos del Tema

- Conocer las características de los paquetes y los diagramas de paquetes.
- Aprender a agrupar elementos de modelado.
- Aprender a realizar vistas arquitecturales de un sistema empleando paquetes.
- Comprender las relaciones entre los distintos tipos de modelos y la arquitectura en las principales metodologías de desarrollo OO.

Contenido

1. Introducción
2. Paquetes
   - Contenido
   - Relaciones
   - Fusión
   - Tipos Especiales
3. Diagramas de Paquetes
   - Consejos
4. Modelado
   - Grupos de Elementos
   - Vistas Arquitecturales
5. Modelos, Arquitectura y Metodologías

Bibliografía

- Básica
  - Booch, Rumbaugh y Jacobson (2006): El Lenguaje Unificado de Modelado. 2ª edición.
    - Cap. 12.
  - Rumbaugh, Jacobson y Booch (2007): El Lenguaje Unificado de Modelado. Manual de Referencia. 2ª edición.
    - Cap. 11.
- Complementaria
  - Miles y Hamilton (2006): Learning UML 2.0.
    - Cap. 13.

1. Introducción

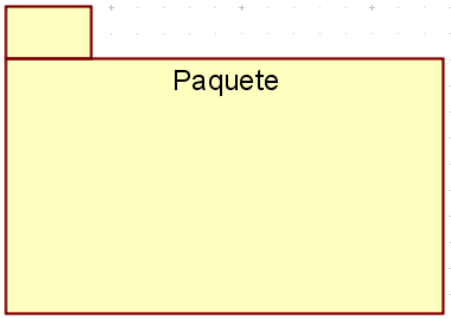
- Modelar sistemas medianos o grandes conlleva manejar una cantidad considerable de elementos de modelado (clases, interfaces, componentes, nodos, relaciones, diagramas).
  - A partir de un cierto tamaño, es necesario organizar estos elementos en bloques mayores.
  - La mejor forma de comprender un sistema complejo es agrupando las abstracciones en grupos cada vez mayores.
  - En UML las abstracciones que organizan un modelo se llaman **paquetes**.
  - Los paquetes son mecanismos de propósito general para organizar elementos en grupos.

2. Paquetes

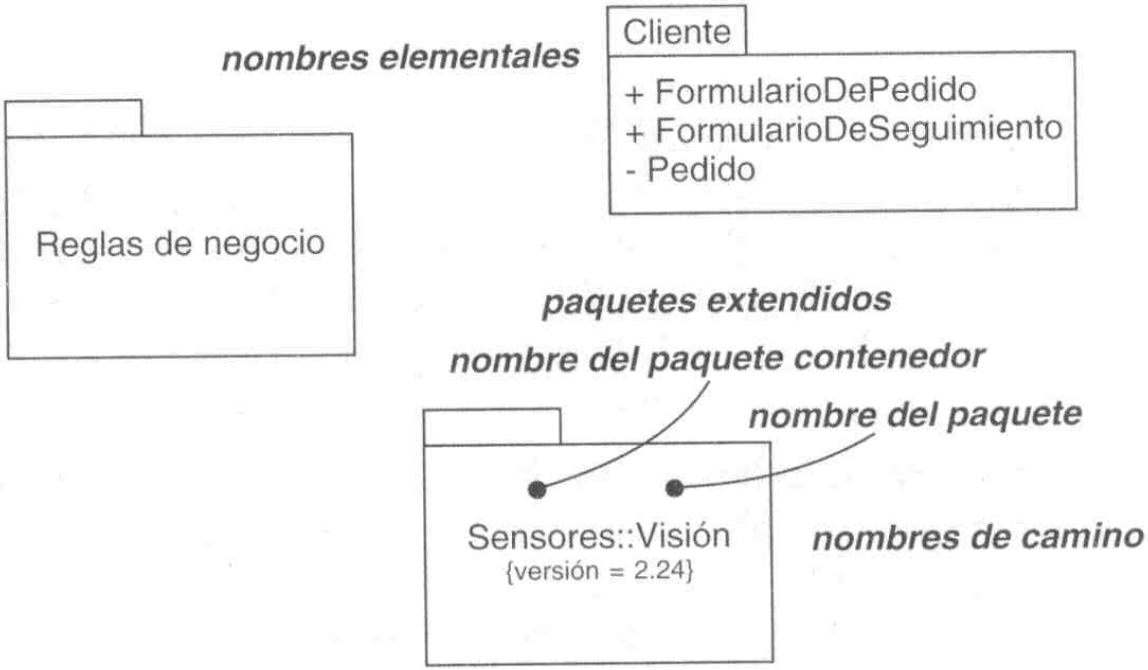
- Un **paquete** es un mecanismo de propósito general para organizar un modelo de manera jerárquica.
  - Estableciendo un espacio de nombres (*namespace*).
- Utilidades principales:
  - Organizar los elementos en los modelos para comprenderlos más fácilmente.
  - Controlar el acceso a sus contenidos para controlar las líneas de separación de la arquitectura del sistema.
- Son un mecanismo importante para gestionar la complejidad del modelado.

Paquetes

- Notación: carpetas.

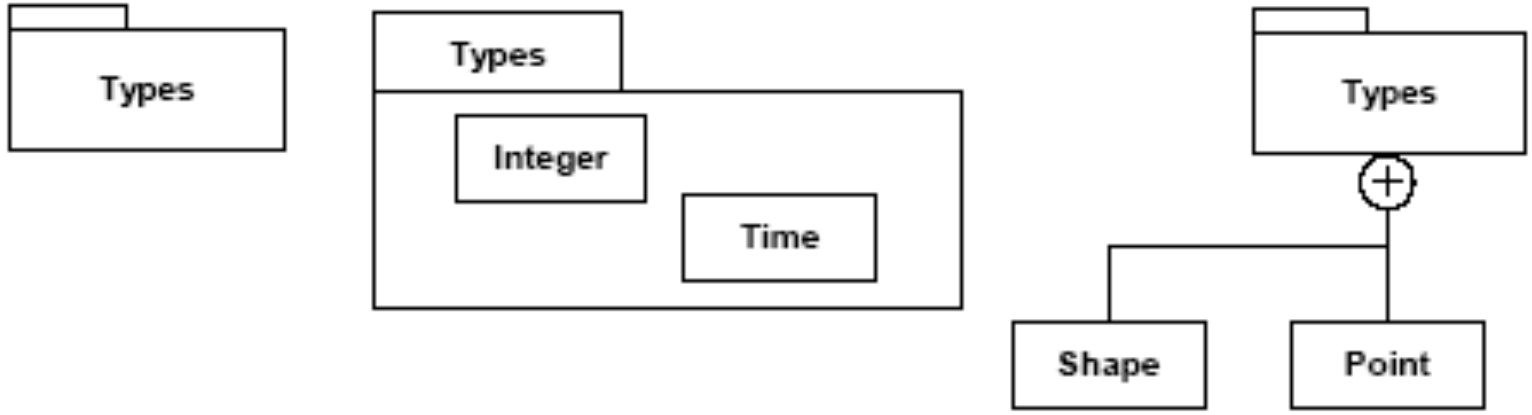


- Nombre
  - Unívoco
  - Simple o Calificado (precedido por el nombre del otro paquete en que se encuentra, separados por "::").
  - Al igual que las clases, se pueden dibujar adornados con valores etiquetados o con apartados adicionales para mostrar sus detalles.



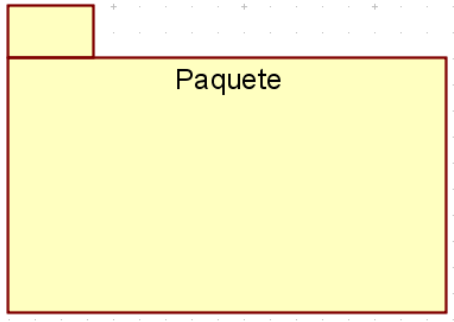
Paquetes

- Existen varias maneras de representar gráficamente el contenido de un paquete.
  - Sin especificar su contenido.
  - Notación interna: incluyéndolo dentro de la carpeta.
  - Notación externa: poniéndolo fuera y relacionado con el paquete mediante un símbolo "+" envuelto en un círculo.



Paquetes

- Los paquetes bien diseñados agrupan elementos cercanos semánticamente:
  - Fuertemente cohesionados
  - Débilmente acoplados



- Son completamente diferentes a las clases:
  - Las clases son abstracciones de aspectos del problema o la solución.
  - Los paquetes son mecanismos para organizar, pero no tienen identidad (no puede haber instancias de paquetes).

Paquetes - Contenido

- **Contenido**
- Un paquete puede contener **elementos** como clases, interfaces, componentes, nodos, colaboraciones, casos de uso e incluso otros paquetes.
  - Cuando se muestran los elementos del paquete, el nombre se coloca en la pestaña de la carpeta.
- Entre un paquete y sus elementos existe una relación de composición =>
  - Un elemento del modelo se declara en un sólo paquete, aunque puede ser referenciado desde otros.
  - Si el paquete se destruye, el elemento también es destruido.

Paquetes - Contenido

- Un paquete forma un **espacio de nombres** (*namespace*):
  - No puede haber dentro de un paquete dos elementos del mismo tipo – si de tipos diferentes – con el mismo nombre.
    - No puede haber dos clases Cola en el mismo paquete.
    - P1::Cola y P2::Cola son elementos diferentes en paquetes diferentes (P1 y P2).
    - Sí puede haber una clase Tempo y un componente Tempo.
  - Consejo: no repetir nombres aunque sean de tipo diferente, incluso en paquetes diferentes.

Paquetes - Contenido

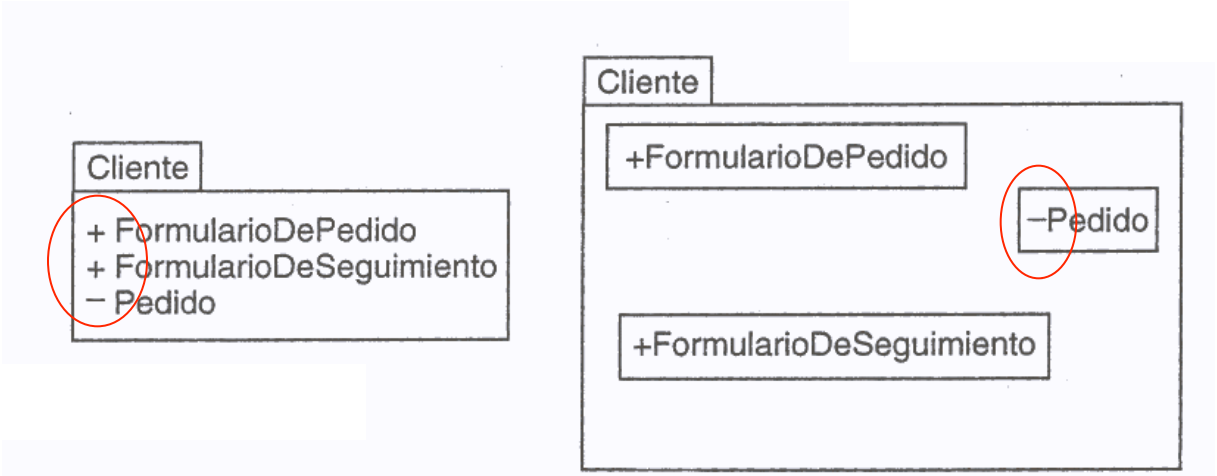
- También los paquetes pueden controlar la **visibilidad** de los elementos que contienen:

+ Público.

  - Elemento disponible a otros elementos del paquete contenedor o uno de sus paquetes anidados, y a paquetes que importan el paquete contenedor.

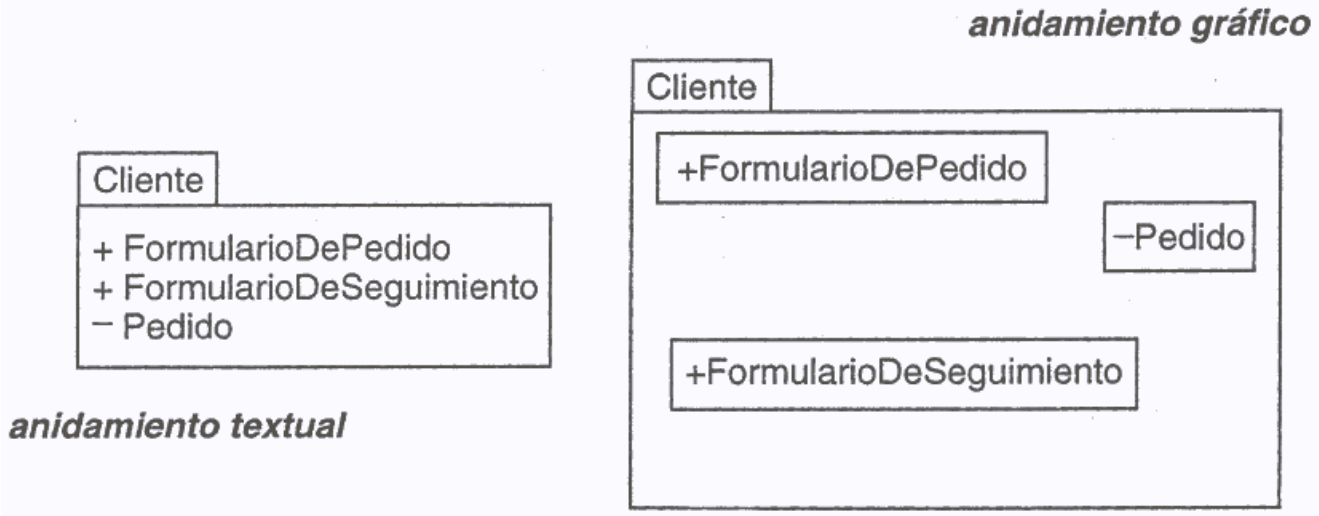
- Privado.

  - No disponibles fuera del paquete contenedor.



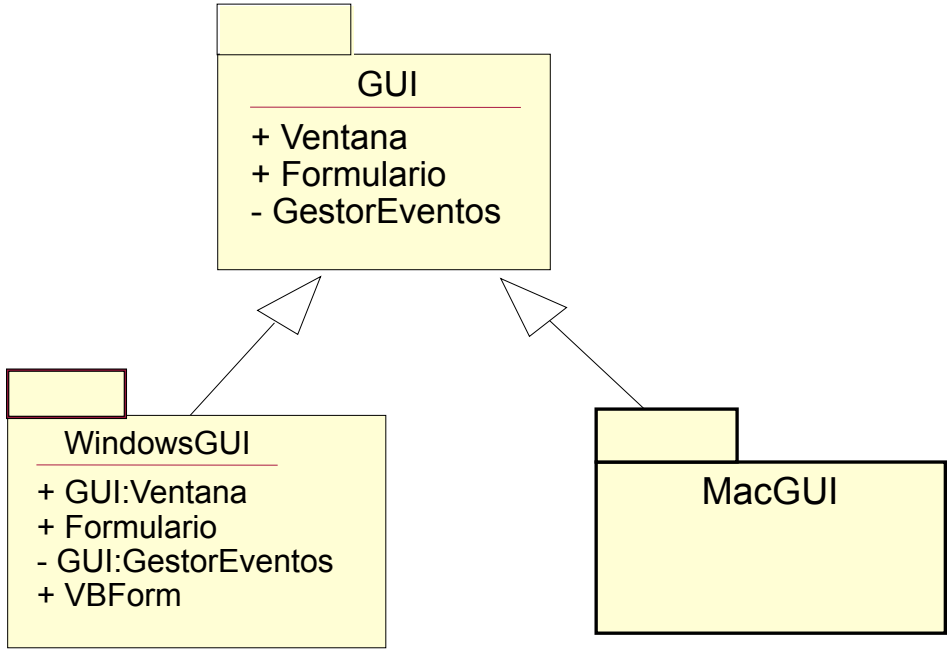
Paquetes - Contenido

- Los paquetes pueden contener a otros paquetes (se forman **jerarquías de paquetes**).
  - Los paquetes anidados tienen acceso al espacio de nombres del paquete que los contiene.
  - No recomendable más de 3 niveles.
  - UML asume que existe un paquete raíz anónimo (**root**).
- Se puede mostrar de forma explícita el contenido de un paquete, de forma textual o gráfica.



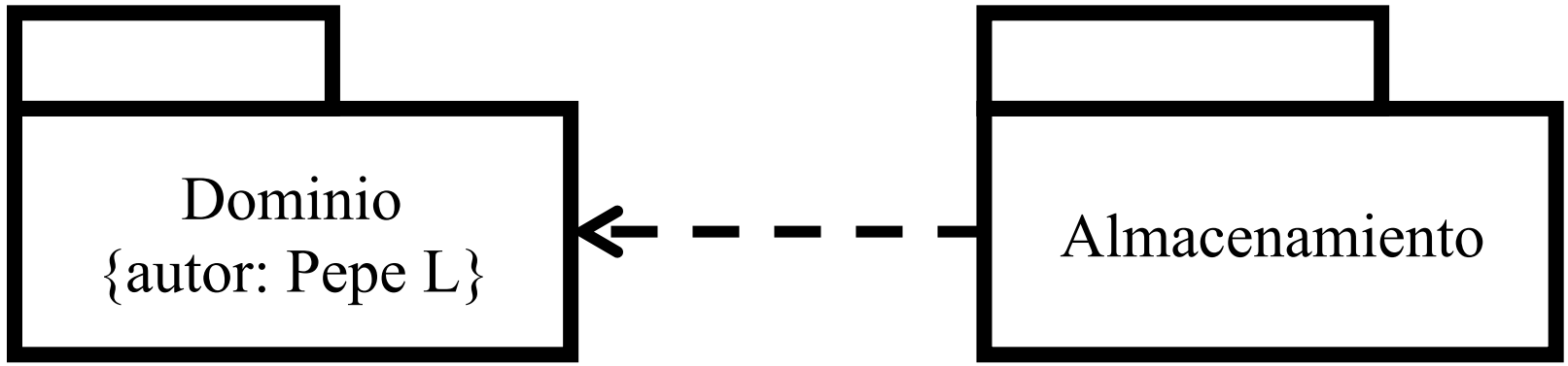
Paquetes - Relaciones

- **Relaciones**
- La **generalización** entre paquetes es muy parecida a la generalización entre clases.
  - Un paquete especializado puede usarse en sustitución de un paquete más general, del cual hereda.



Paquetes - Relaciones

- Las **dependencias entre paquetes** denotan que algún elemento de un paquete depende de los elementos en otro paquete.



Paquetes - Relaciones

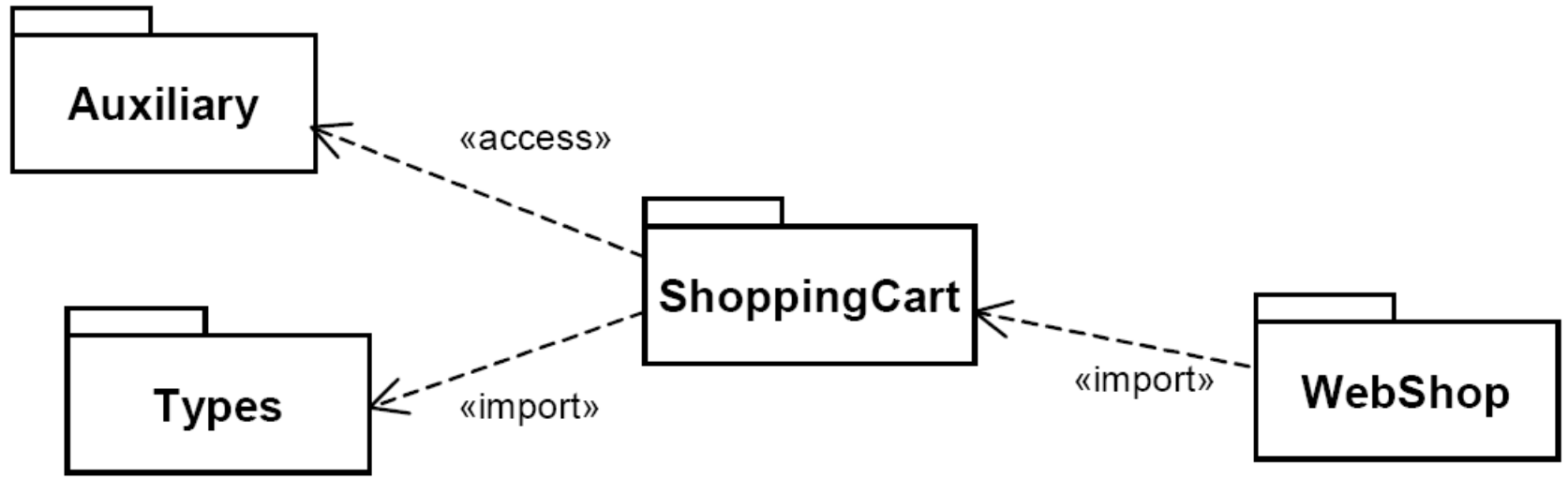
- Entre paquetes puede haber tres tipos de relaciones de dependencia:
  - **Importación**: modelado como una dependencia estereotipada con <<Import>>
  - **Exportación**: modeladas indicando la visibilidad pública en los elementos de un paquete.
    - No se exporta explícitamente a algún paquete.
    - Se pone público, para que cualquier otro paquete pueda importarlo.
  - **Acceso**: modelado como una dependencia estereotipada con <<Access>>.

Paquetes - Relaciones

- ¿Importación o Acceso?
  - Ambas indican que el paquete origen tiene acceso al contenido del destino.
  - Ambas son transitivas.
  - <<**import**>> añade el contenido del destino al espacio de nombres **público** del origen (paquete importador).
    - No hay que calificar los nombres => Posibilidad de colisión.
  - <<**access**>> añade el contenido del destino al espacio de nombres **privado** del origen.
    - No se pueden reexportar los elementos importados si un tercer paquete importa el origen.
  - La mayoría de las veces se usa <<import>>.

Paquetes - Relaciones

- **Ejemplo** de Importaciones y Accesos entre paquetes.



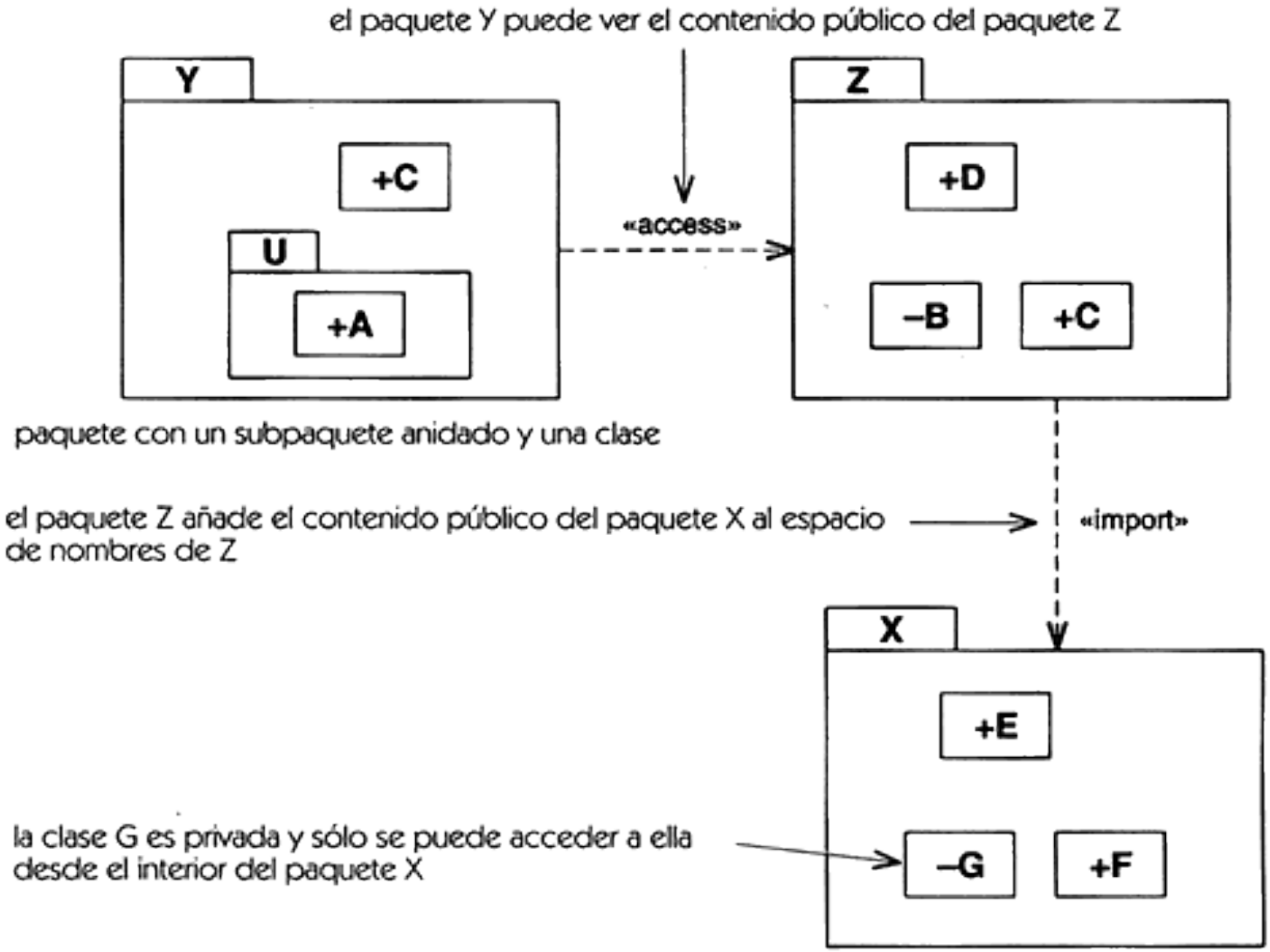
  - Los elementos en Types son importados a ShoppingCart.
  - Los elementos en Types son importados también a WebShop (por transitividad).
  - Los elementos de Auxiliary sólo son accedidos desde ShoppingCart y, por tanto, no pueden usarse sin calificar desde WebShop.

Paquetes - Relaciones

Import

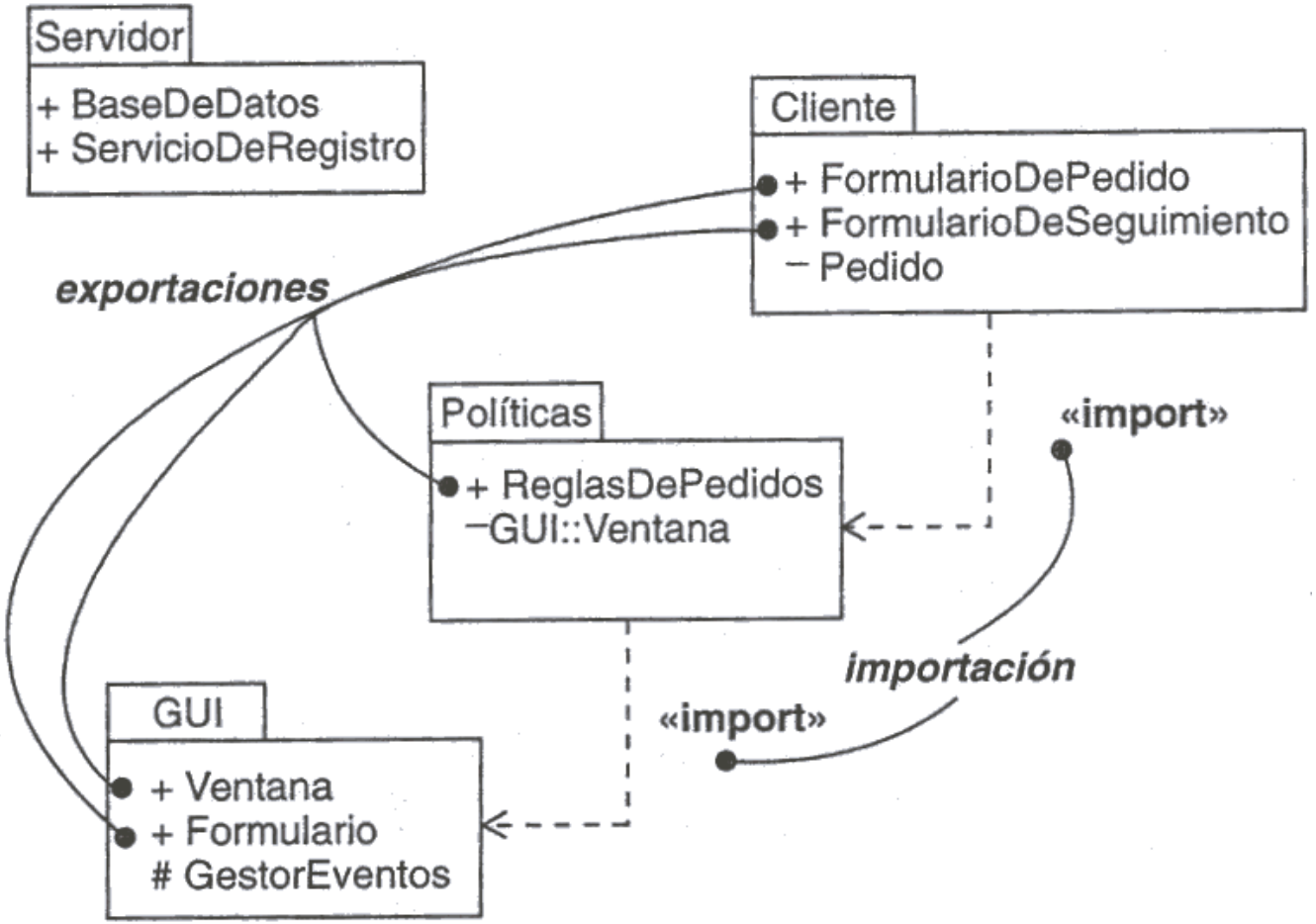
vs

Access



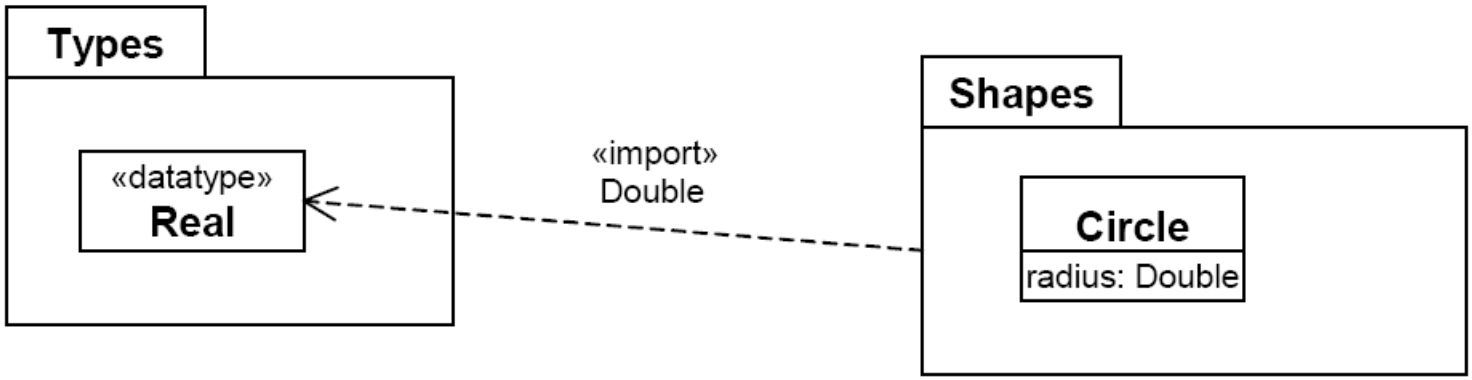
Paquetes - Relaciones

- **Exportación**
  - La parte pública de un paquete son sus exportaciones.



Paquetes - Relaciones

- También se pueden **importar o acceder elementos** de un paquete en vez de paquetes completos.
  - Se puede asignar un alias a un elemento importado/accedido.



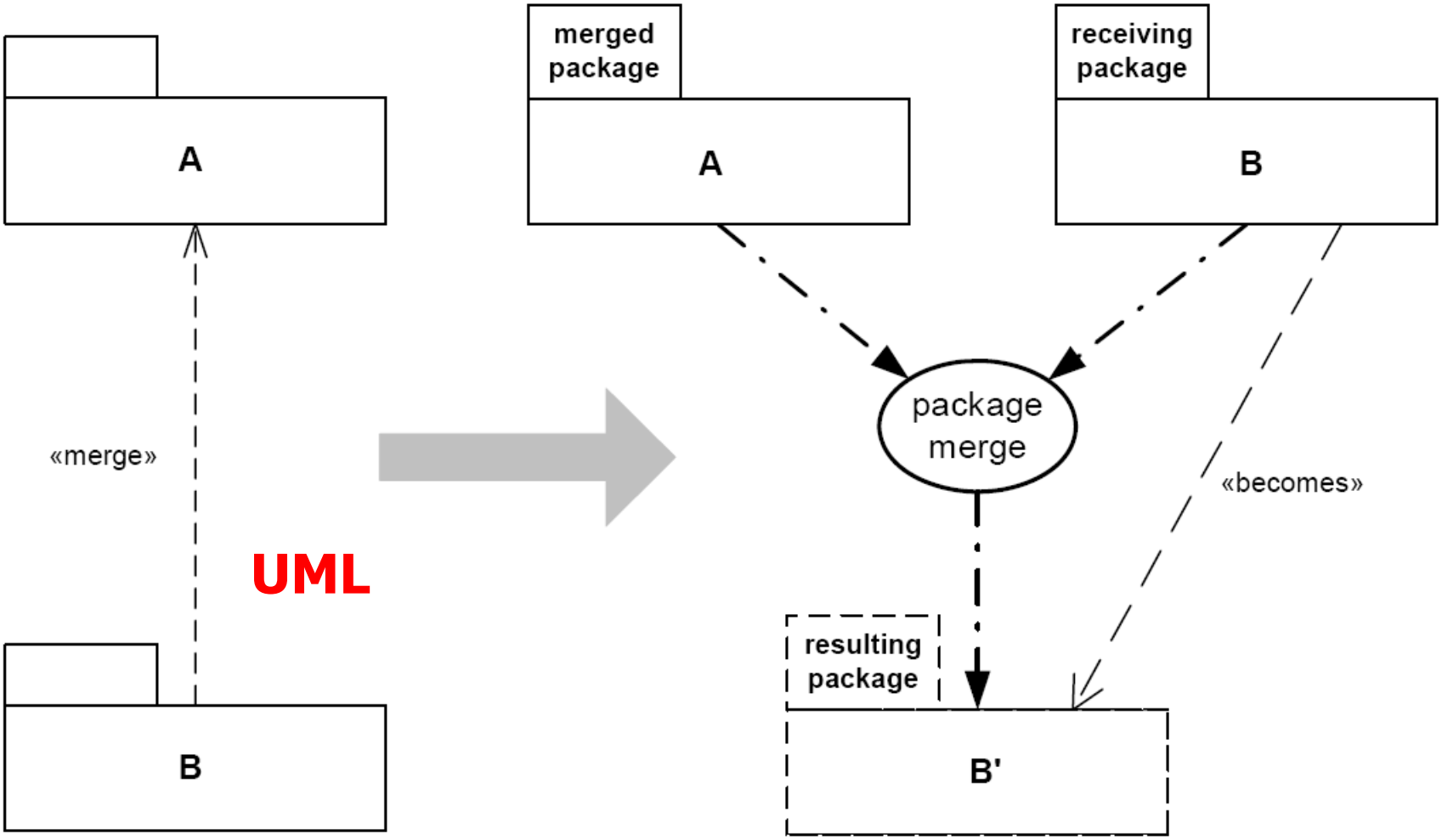
El tipo Types:Real está disponible en el paquete Shapes con el nombre Double.

Paquetes – Fusión

- **Fusión**
- Una relación de **fusión** (*merge*) entre dos paquetes especifica que el contenido del paquete origen (receptor) se extiende con el contenido del paquete destino.
  - Es necesario un mecanismo para fusionar los contenidos de ambos paquetes:
    - Resuelve los conflictos de nombres mediante especialización y redefinición.
    - Es bastante complicado.
    - Se define mediante restricciones (precondiciones) y transformaciones (post-condiciones).
  - Físicamente, en el repositorio de modelos no se produce ningún cambio en los paquetes.

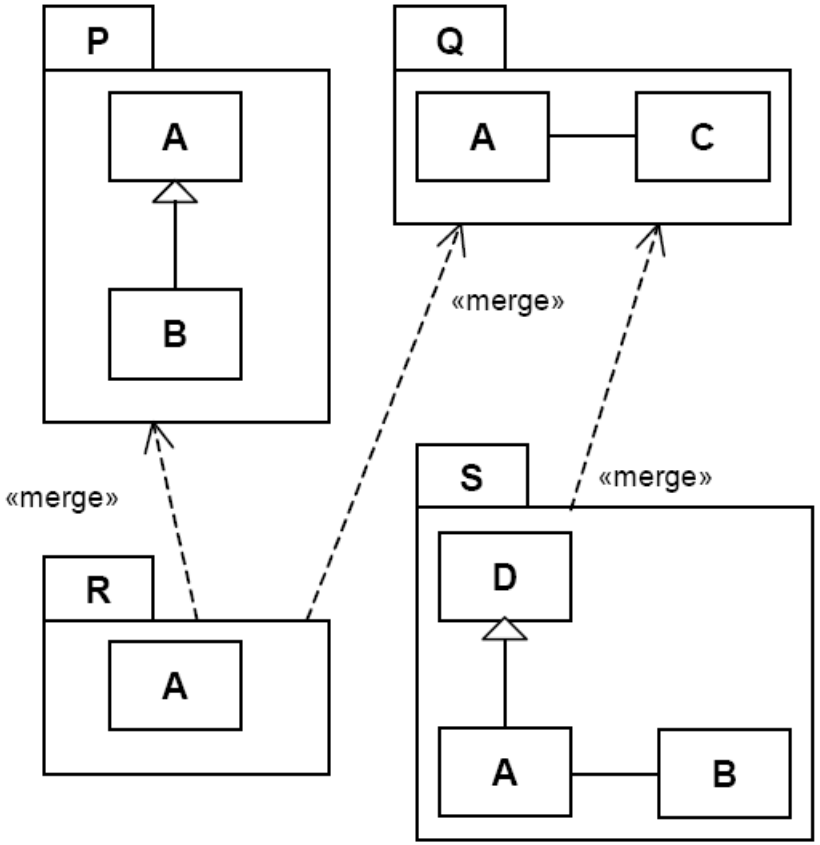
Paquetes – Fusión

- Significado conceptual de una relación <<merge>>.

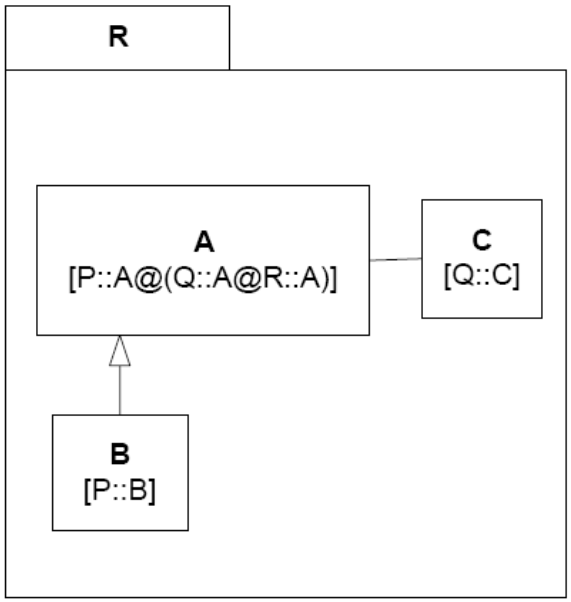


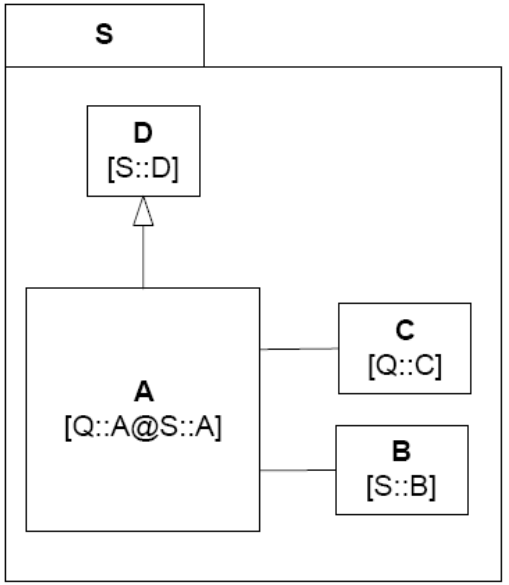
Paquetes – Fusión

- Ejemplo.



Resultado





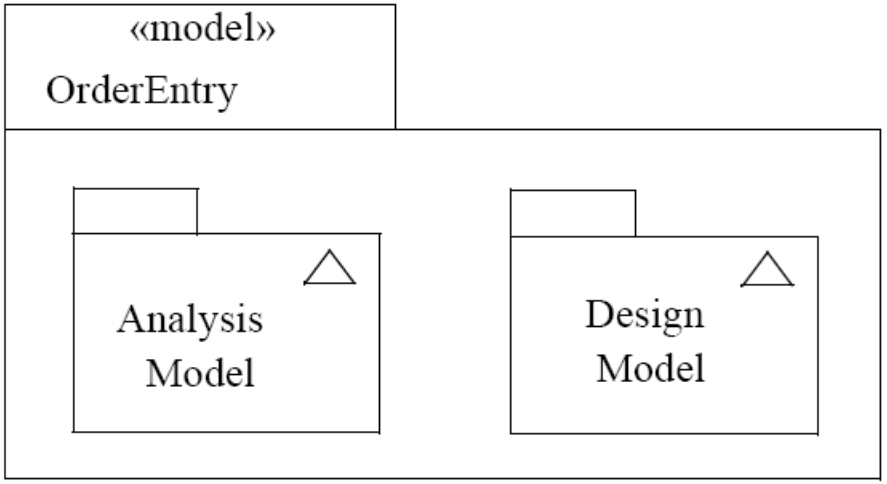
Paquetes – Tipos Especiales

- **Tipos Especiales**
- Todos los mecanismos de extensibilidad pueden ser aplicados a paquetes
  - Con frecuencia se añaden Valores etiquetados para incorporar nuevas propiedades a los paquetes.

- Existen varios **estereotipos** que definen nuevas categorías de paquetes (además del ya conocido "profile"):
  - **System** - Paquete que representa el sistema completo que se está modelando.
  - **Subsystem** - Expresa que el paquete es independiente del sistema completo que se modela.
  - **Framework** – Contiene elementos reusables como clases, patrones y plantillas.
  - **Facade** (fachada) - Proporciona una vista simplificada del paquete.
  - **Stub** - Un paquete que sirve de sustituto para el contenido público de otro paquete.
  - Otros: ModelLibrary, Perspective, ..

Paquetes – Tipos Especiales

- En UML 2, un **modelo** es un estereotipo de paquete que incluye un conjunto (jerárquico) de los elementos que forman una descripción completa de una vista particular del sistema.
  - Se suele estructurar en una jerarquía en forma de árbol.
  - El estereotipo utiliza un triángulo como identificador.
  - También puede contener elementos del entorno del sistema (actores y sus relaciones).



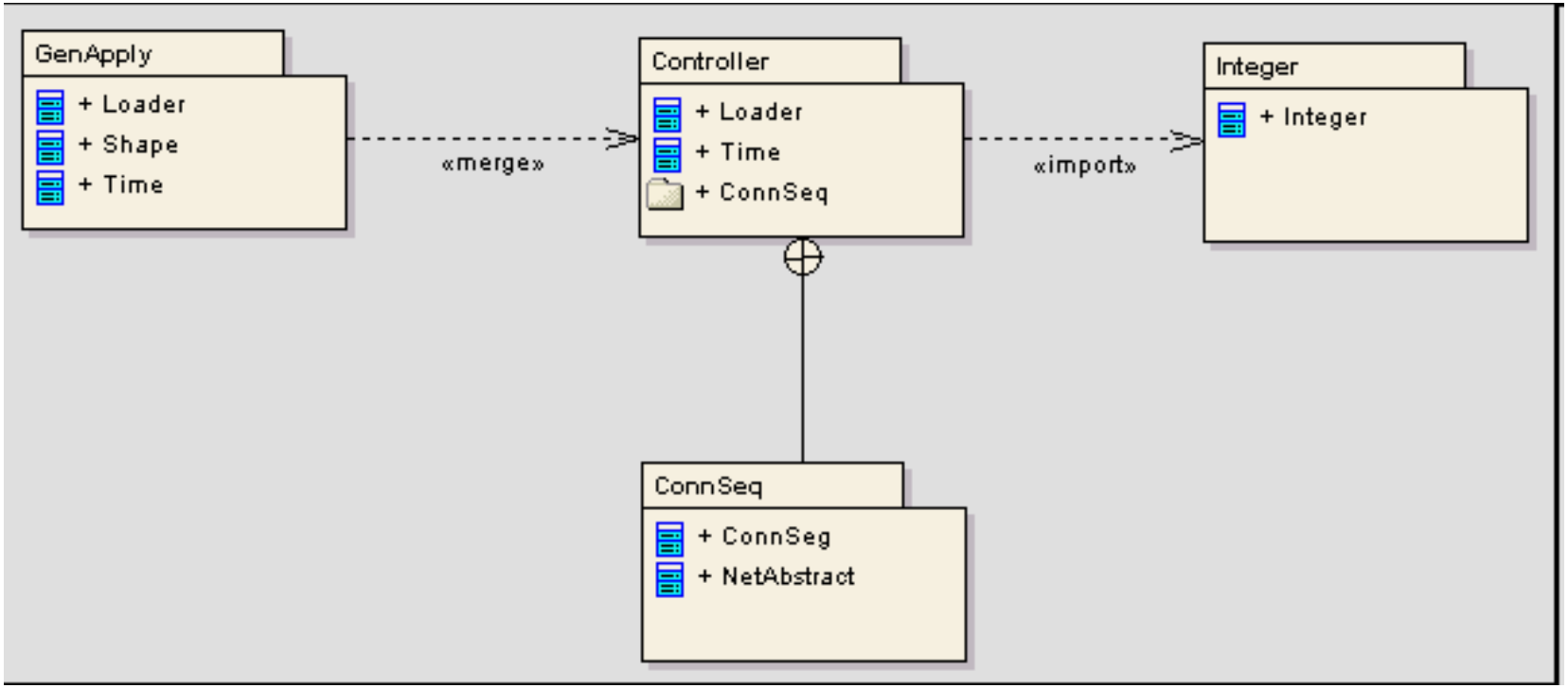
3. Diagramas de Paquetes

- Presentan cómo se organizan los elementos de modelado en paquetes y las dependencias entre ellos, incluyendo importaciones y extensiones de paquetes.

- Un **diagrama de paquetes** es un diagrama de estructura cuyo contenido es, principalmente, paquetes y sus relaciones.
  - La distinción entre los diversos tipos de diagramas de estructura (clases, objetos y paquetes) es relativa. Todos pueden incluir:
    - Como nodos del grafo:
      - Clases, Interfaces, Instancias o Paquetes.
    - Y como arcos (relaciones):
      - Agregaciones, asociaciones, composiciones, dependencias, generalizaciones, realizaciones, dependencias de uso, y fusiones, importaciones y accesos entre paquetes.

Diagramas de Paquetes

- Ejemplo.



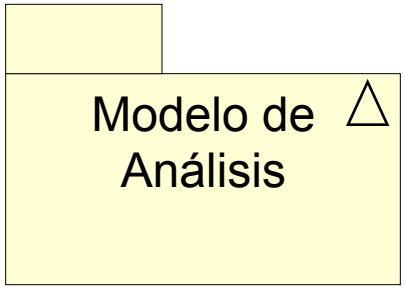
ConnSeq es un paquete anidado (incluido en) Controller

Diagramas de Paquetes - Consejos

- **Consejos**
- Un paquete está **bien estructurado** si:
  - Es cohesivo.
    - Proporciona un límite bien definido alrededor de un grupo de elementos relacionados.
  - Está poco acoplado.
    - Exportando sólo los elementos que otros paquetes necesitan ver realmente.
    - Importando sólo aquellos elementos necesarios y suficientes para que los elementos del paquete hagan su trabajo
  - No está profundamente anidado (max 3 niveles)
  - Posee un conjunto equilibrado de elementos.
    - Se recomiendan entre 5 y 9 nodos (casos de uso, clases, ..).

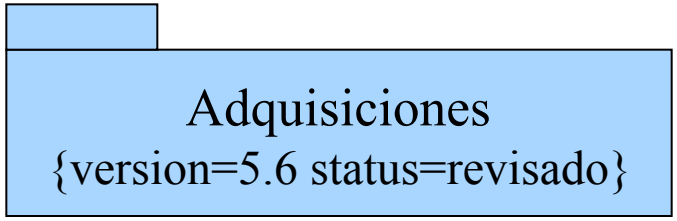
Diagramas de Paquetes - Consejos

- Se recomienda agrupar en paquetes los elementos que:
  - Tienen un objetivo común o relaciones conceptuales fuertes,
  - Pertenecen a un mismo árbol de herencia, o
  - Pertenecen al mismo caso de uso.
- O formando paquetes mas grandes de tipo arquitectural:
  - Paquete “Clases e interfaces del modelo”.
  - Paquete “Interfaces de usuario”.
  - Paquete “Servicios base de datos”.
  - Paquete “Modelo del análisis”.



Diagramas de Paquetes - Consejos

- Al dibujar un paquete en UML:
  - Emplear la forma simple del icono (sin contenido), a menos que sea necesario revelar dicho contenido explícitamente.
  - Al revelar el contenido, mostrar sólo los elementos necesarios para comprender el significado del paquete en el contexto.
  - Si los paquetes se emplean para modelar elementos sujetos a gestión de configuraciones, mostrara los valores etiquetados asociados a las versiones.



4. Modelado

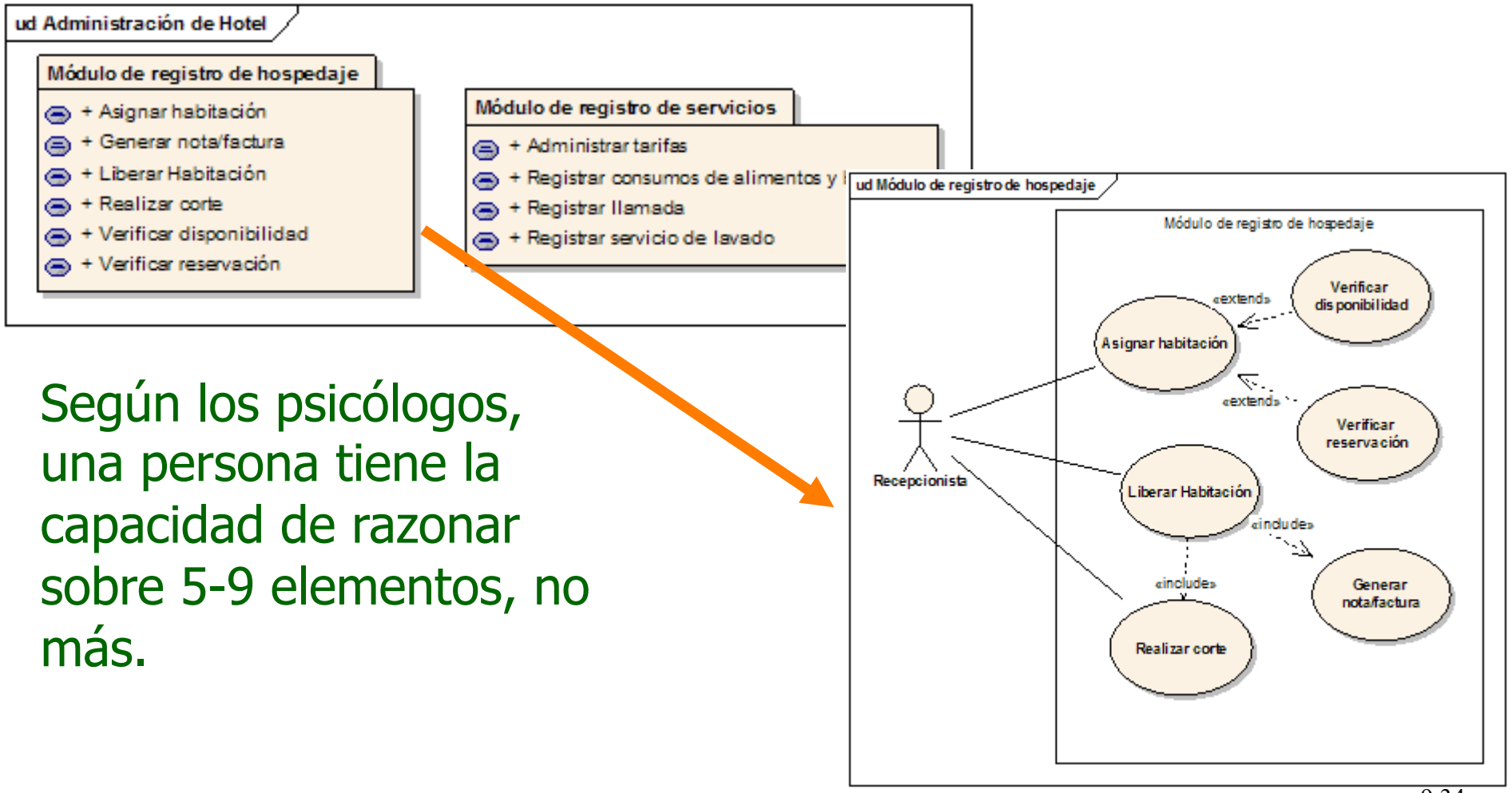
- Los paquetes son especialmente útiles en modelado de:
  - Grupos de Elementos Relacionados
  - Vistas Arquitecturales

Modelado – Grupos de Elementos

- **Grupos de Elementos**
- El uso más habitual es la **organización en grupos** de los elementos de modelado.
- En aplicaciones pequeñas no es necesario.
- Pasos a realizar:
  1. Examinar los modelos en busca de grupos de elementos cercanos semántica o conceptualmente.
  2. Englobar cada uno de dichos grupos en un paquete.
  3. Para cada paquete, diferenciar los elementos a los que se podrá acceder desde fuera (públicos) frente a los que no (privados).
     - En caso de duda, marcar como privado.
  4. Conectar los paquetes que dependen de otros por importaciones (import o access).
  5. Si hay familias de paquetes, es necesario utilizar la generalización.

Modelado – Grupos de Elementos

- Ejemplo. Diagrama de Casos de Uso simplificado con el uso de paquetes.



Según los psicólogos, una persona tiene la capacidad de razonar sobre 5-9 elementos, no más.

Modelado – Vistas Arquitecturales

- **Vistas Arquitecturales**
- Los paquetes se pueden emplear también para modelar las **vistas de una arquitectura**.
- Esto tiene dos implicaciones:

a) Se puede descomponer un sistema en paquetes, casi ortogonales, cada uno de los cuales cubre un conjunto de decisiones significativas a nivel arquitectónico.
   - Ejemplo: Un paquete para cada una de las vistas de diseño, interacción, implementación, despliegue y casos de uso.

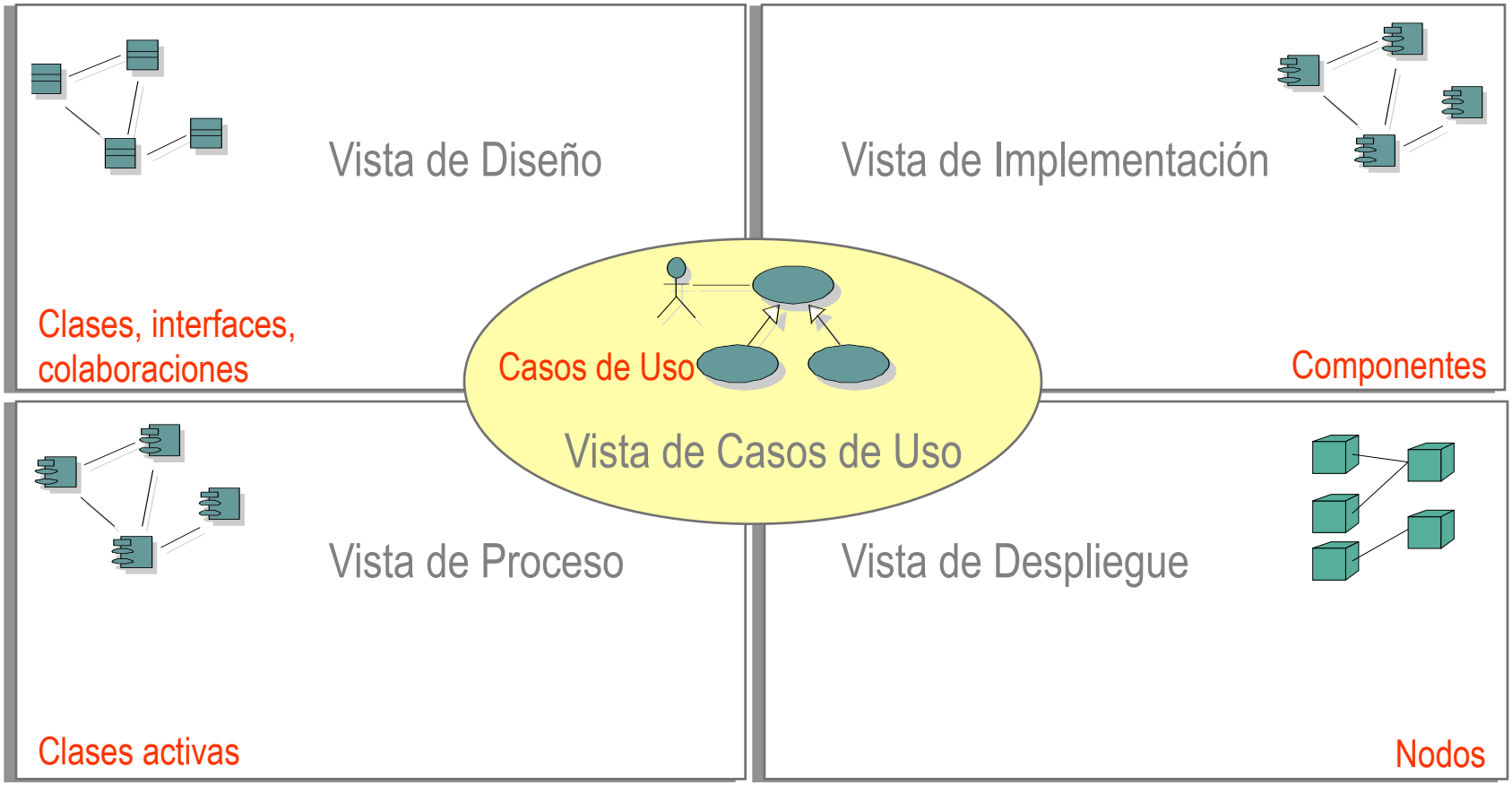
b) Cada paquete contiene todas las abstracciones pertinentes para una vista.
   - Ejemplo: Todos los componentes pertenecen al paquete de la vista de implementación.

Modelado – Vistas Arquitecturales

- Para **modelar vistas arquitecturales**:
  1. Identificar el conjunto de vistas arquitectónicas significativas del problema.
     - Suelen ser una vista de diseño, una vista de interacción, una de implementación, una de despliegue y una de casos de uso.
  2. Colocar en el paquete adecuado los elementos (y diagramas) necesarios y suficientes para visualizar, especificar, construir y documentar la semántica de cada vista.
  3. Si es necesario, agrupar más estos elementos en sus propios paquetes.
  4. Permitir a cada vista en la cima del sistema estar abierta al resto de las vistas en el mismo nivel.
     - Esto es necesario porque normalmente existen dependencias entre elementos de vistas diferentes.

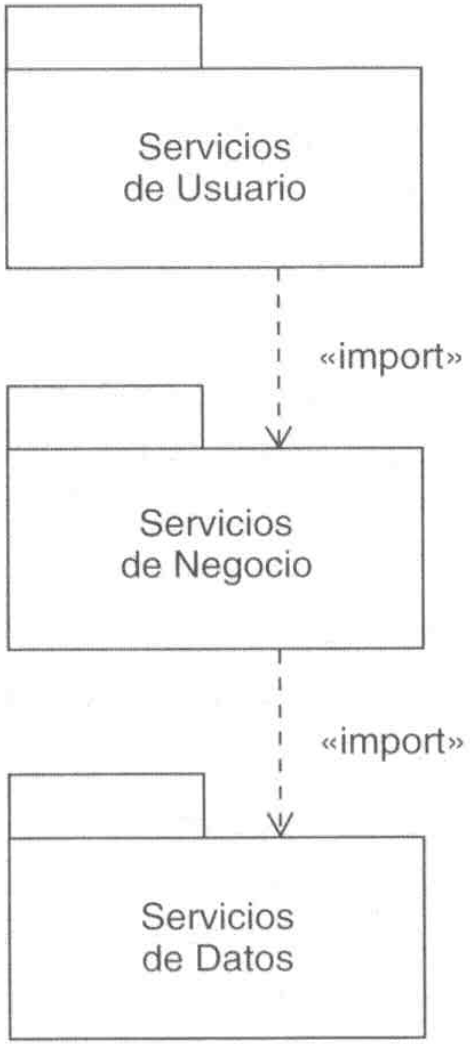
Modelado – Vistas Arquitecturales

- **Ejemplo.**
  - Descomposición canónica de nivel superior válida para sistemas de muy diversos tamaños, incluidos muy complejos.



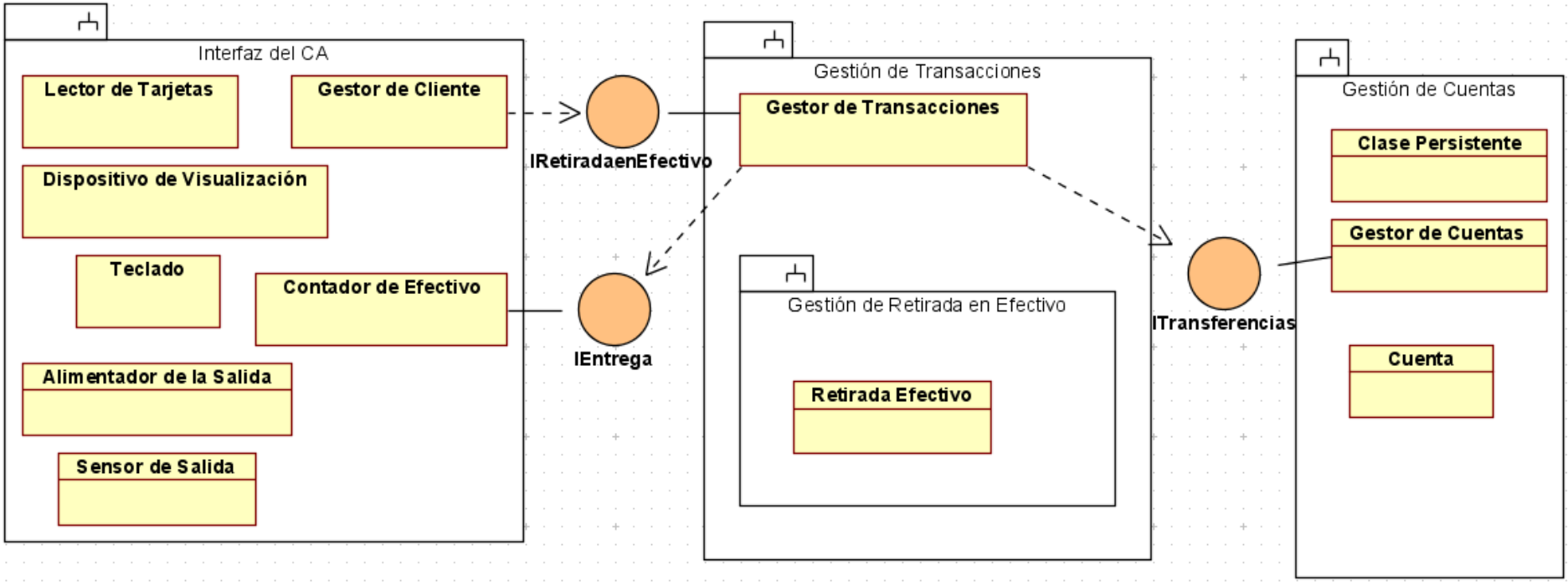
Modelado – Vistas Arquitecturales

- Ejemplo.
  - Arquitectura clásica en tres capas.
  - La capa de interfaz (superior) importa los elementos de la capa de dominio (medio) que, a su vez, importa los elementos de la capa de almacenamiento (inferior).



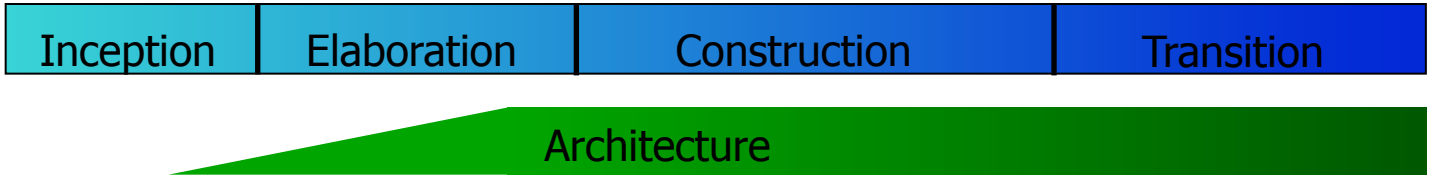
Modelado – Vistas Arquitecturales

- Ejemplo.
  - Arquitectura detallada.



5. Modelos, Arquitectura y Metodologías

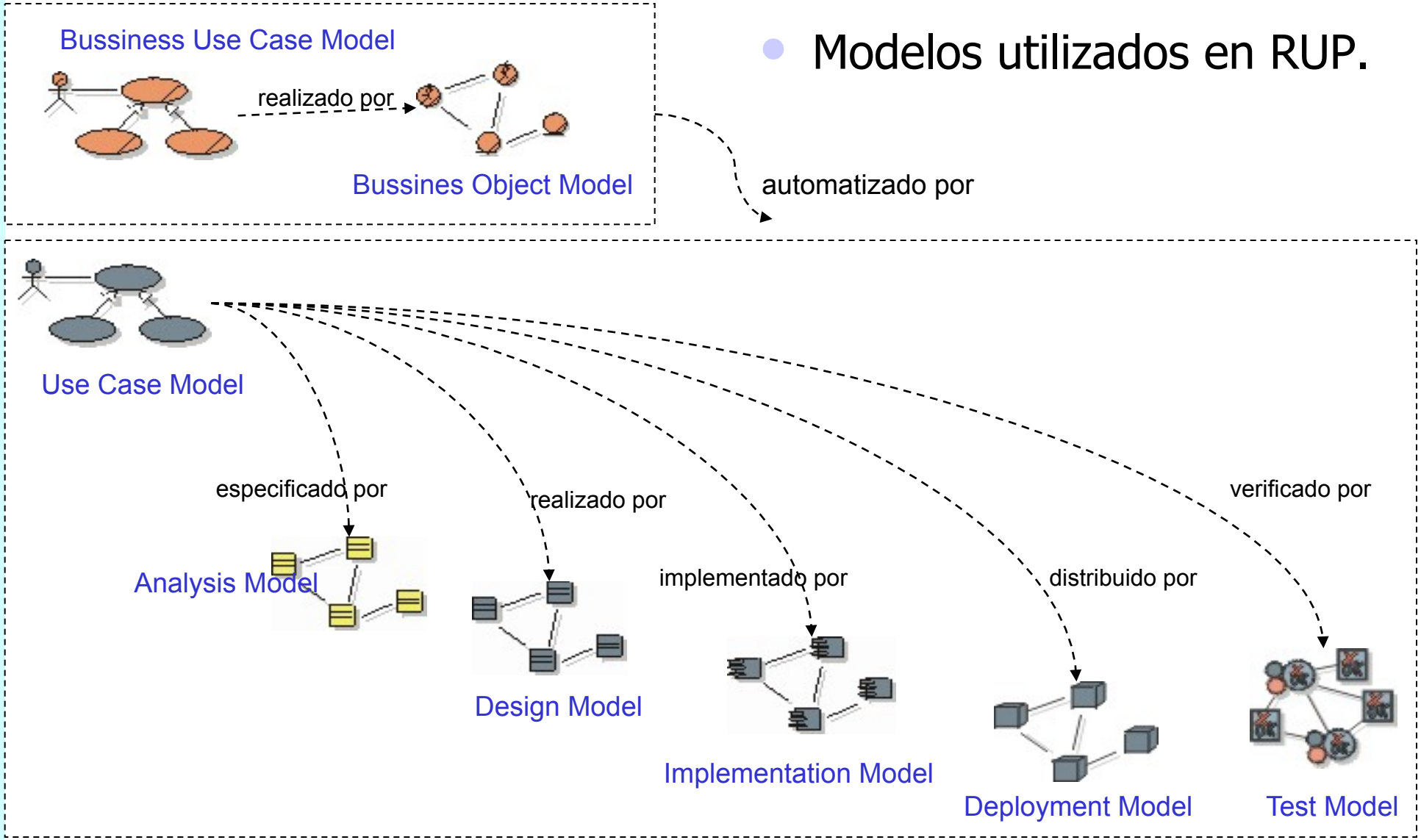
- Casi todas las metodologías de desarrollo de software asignan un papel central a la **arquitectura**.
  - RUP establece refinamientos sucesivos de una arquitectura, construida como un prototipo evolutivo.



- Cada metodología también propone una forma específica de organizar y usar los diagramas UML en Vistas Arquitectónicas y Modelos.

Modelos, Arquitectura y Metodologías

- Modelos utilizados en RUP.

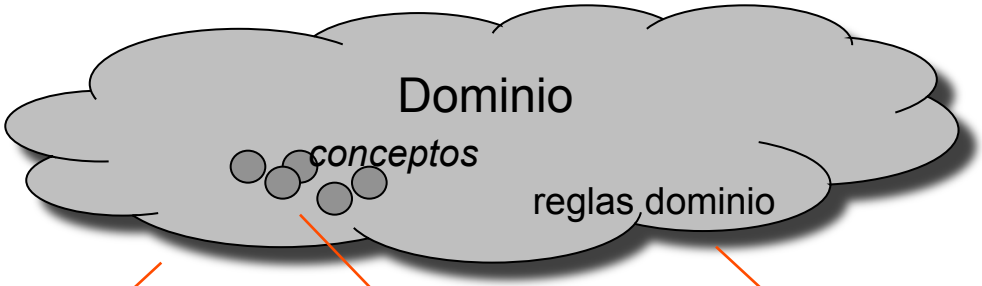


Modelos, Arquitectura y Metodologías

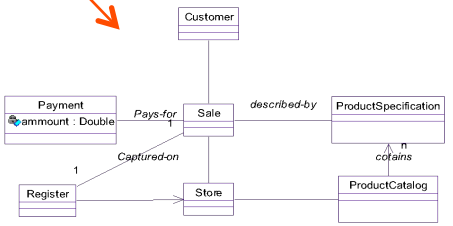
- En general, se puede considerar que las metodologías de desarrollo OO basadas en UML suponen ir refinando un sistema en distintos niveles de abstracción, cada vez más elaborados, a partir del conocimiento del dominio del problema:
  - Modelos Conceptuales del Problema
  - Requisitos del Sistema
  - Diseño del Sistema

Modelos, Arquitectura y Metodologías

Niveles de Abstracción y Modelos



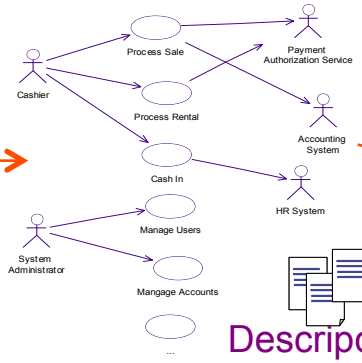
Glosario



Modelo Conceptual

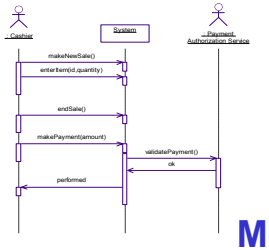
Restricciones OCL

Requisitos Automatización



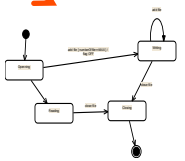
Modelo Casos de Uso

Descripciones c.u.



Secuencia Eventos
(actores-sistema)

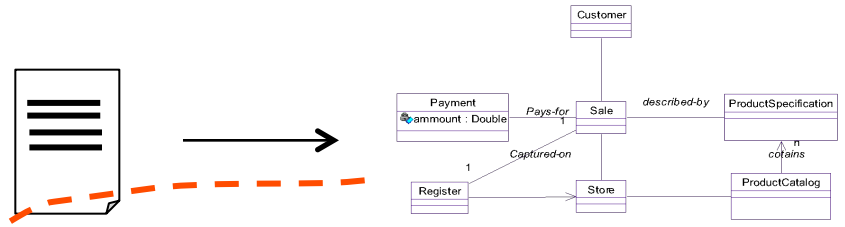
Modelo Comportamiento



Diagramas Estados
(objetos dominio / Sistema)

Modelos, Arquitectura y Metodologías

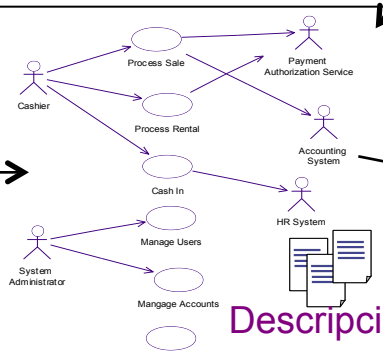
Niveles de Abstracción y Modelos



Glosario

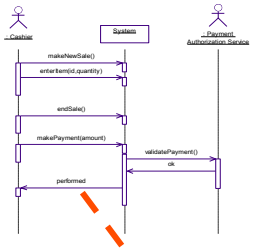
Modelo Conceptual
(atributos-relaciones)

Requisitos Automatización

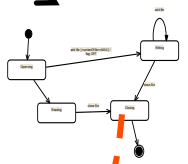


Descripciones c.u.

Modelo Casos de Uso

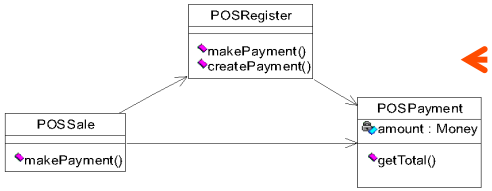


Secuencia Eventos

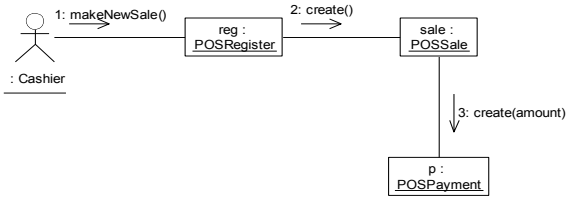


Diagramas Estados
(objetos dominio Sistema)

Arquitectura



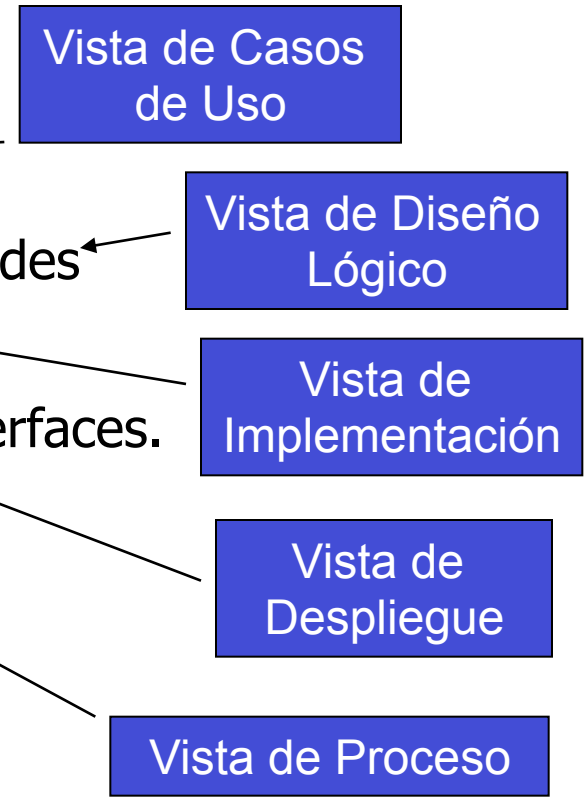
Diseño Clases
(atributos/operaciones-relaciones)



Diseño Colaboraciones Clases para evento
(secuencia de llamadas a métodos)

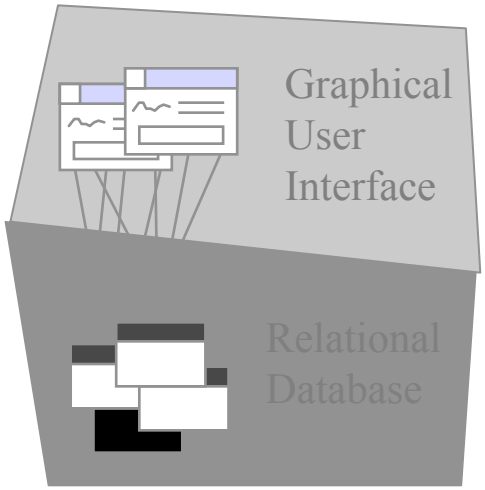
Modelos, Arquitectura y Metodologías

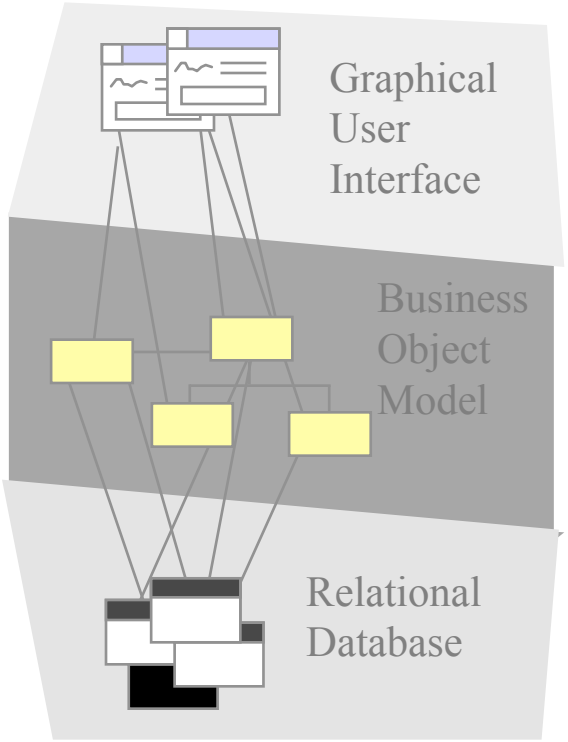
- El diseño arquitectural suele estar formado por los siguientes pasos, que forman un ciclo de refinamiento:
  - Seleccionar Escenarios
  - Identificar clases principales y sus responsabilidades
  - Distribuir el comportamiento entre las clases
  - Estructurar en subsistemas y capas, y definir interfaces.
  - Definir la distribución y concurrencia.
  - Implementar un prototipo arquitectural.
  - Derivar pruebas desde los casos de uso.
  - Evaluar la arquitectura.

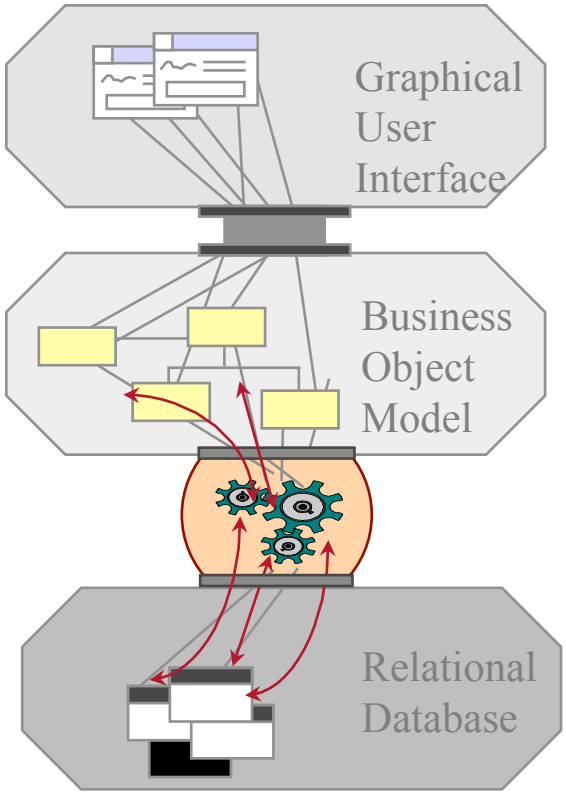


Modelos, Arquitectura y Metodologías

- Se emplean unos modelos, vistas y diagramas para la Arquitectura Lógica







Modelos, Arquitectura y Metodologías

- Y otros diferentes para la Arquitectura Física