INGENIERÍA DEL SOFTWARE I
Tema 11

Arquitectura Lógica del Sistema
(en desarrollo OO)

Univ. Cantabria – Fac. de Ciencias

Francisco Ruiz y Patricia López

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

- Introducción
- Paquetes
  - Contenido
  - Relaciones
  - Fusión
  - Tipos Especiales
- Diagramas de Paquetes
  - Uso de paquetes en diagramas de clase
  - Consejos
- Modelado
  - Grupos de Elementos
  - Vistas Arquitecturales
- 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.

Introducción

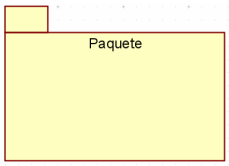
- Modelar sistemas medianos o grandes conlleva manejar una cantidad considerable de elementos de modelado (clases, interfaces, componentes, nodos, relaciones, diagramas).
  - A partir de un cierto tamaño, es necesario organizar estos elementos en bloques mayores.
  - La mejor forma de comprender un sistema complejo es agrupando las abstracciones en grupos
    - Se agrupan aquellos elementos relacionados entre sí de acuerdo a algún criterio.
- En UML las abstracciones que permiten organizar un modelo se llaman **paquetes**.

Paquetes

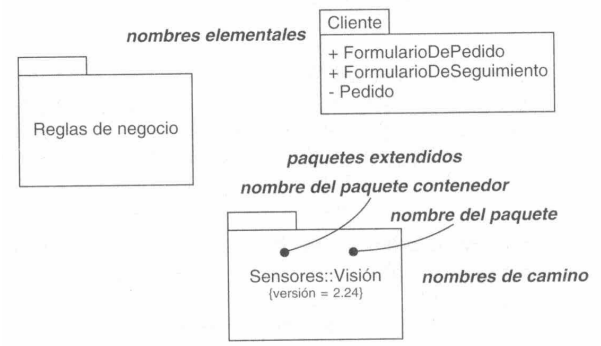
- Un **paquete** es un mecanismo de propósito general para organizar un modelo de manera jerárquica.
  - Cada paquete establece un espacio de nombres (*namespace)*.
- Utilidades principales:
  - Organizar los elementos en los modelos para comprenderlos más fácilmente.
  - Controlar el acceso a sus contenidos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

- Notación: Carpeta

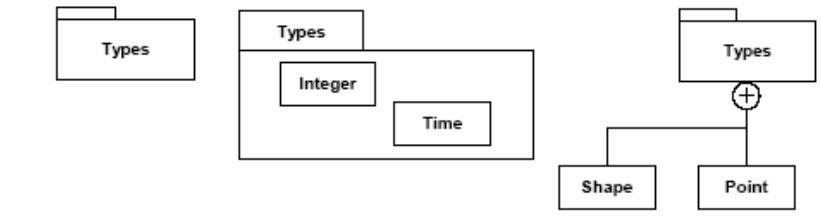


- Nombre
  - Unívoco
  - Simple o Calificado (precedido por el nombre del otro paquete en que se encuentra, separados por "::").



Paquetes

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



Paquetes - Contenido

- Un paquete puede contener diferentes tipos de **elementos** como clases, interfaces, componentes, nodos, colaboraciones, casos de uso e incluso otros paquetes.
- Los paquetes bien diseñados agrupan elementos cercanos semánticamente:
  - Fuertemente cohesionados
  - Débilmente acoplados
- Entre un paquete y sus elementos existe una relación de composición =>
  - Cada elemento del modelo pertenece a un único paquete (aquel en el que es declarado)
    - aunque puede ser referenciado desde otros.
  - Si el paquete se destruye, todos los elementos que contiene son también destruidos.

Paquetes - Contenido

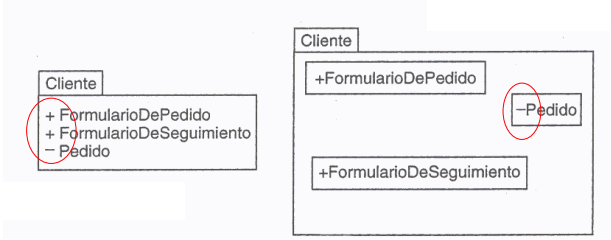
- Un paquete forma un **espacio de nombres** (*namespace*):
  - No puede haber dentro de un paquete dos elementos del mismo tipo – si de tipos diferentes – con el mismo nombre.
    - No puede haber dos clases Lista en el mismo paquete.
    - P1::Lista y P2::Lista son elementos diferentes en paquetes diferentes (P1 y P2).
    - Sí puede haber una clase Logger y un componente Logger.
- En UML, cuando se referencia una clase desde un paquete externo hay que utilizar el nombre completo cualificado => Indicando el namespace (paquete) al que pertenecen
  - Entre clases que pertenecen al mismo paquete no es necesario

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 hace falta cualificar los nombres
  - No recomendable más de 3 niveles.
  - UML asume que existe un paquete raíz anónimo (**root**).
  - Cuidado: En Java los elementos de un paquete anidado no tienen visibilidad directa sobre los elementos del paquete contenedor

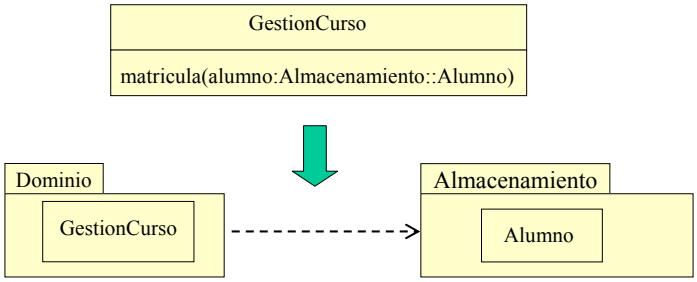
Paquetes - Contenido

- Los paquetes controlan la **visibilidad** de los elementos que contienen:
  - **+ Público**
    - Disponibles fuera del paquete contenedor.
      - El acceso desde un paquete externo es siempre con nombre cualificado.
  - **- Privado**
    - No disponibles fuera del paquete contenedor
- No son posibles el resto de visibilidades (package o protected)



Paquetes - Relaciones

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

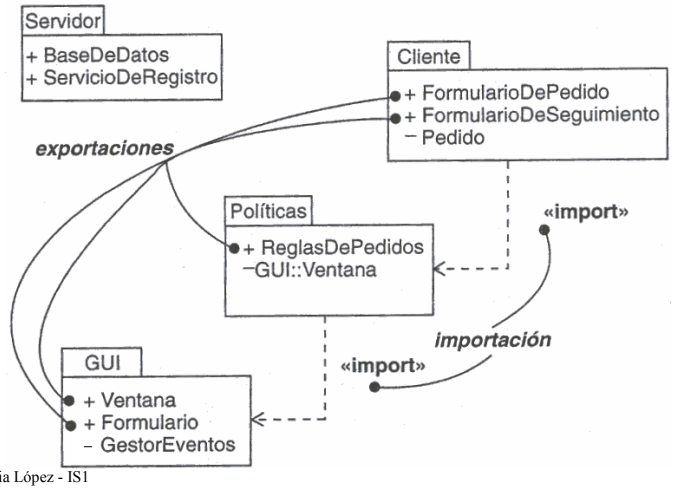


Paquetes - Relaciones

- Entre paquetes puede haber tres tipos de relaciones de dependencia:
  - **Importación**: modelado como una dependencia estereotipada con <<Import>>
  - **Acceso**: modelado como una dependencia estereotipada con <<Access>>
  - **Exportación**: modelado implícitamente a través de 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.

Paquetes - Relaciones

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

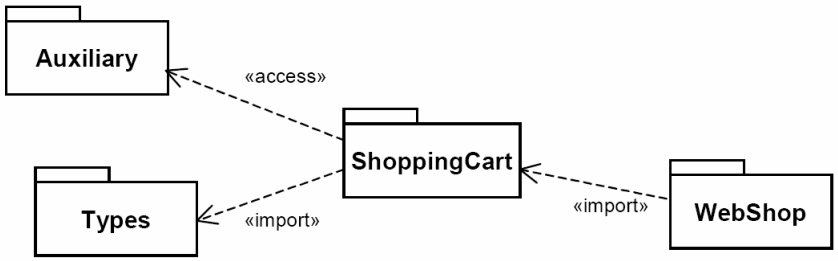


Paquetes - Relaciones

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

Paquetes - Relaciones

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



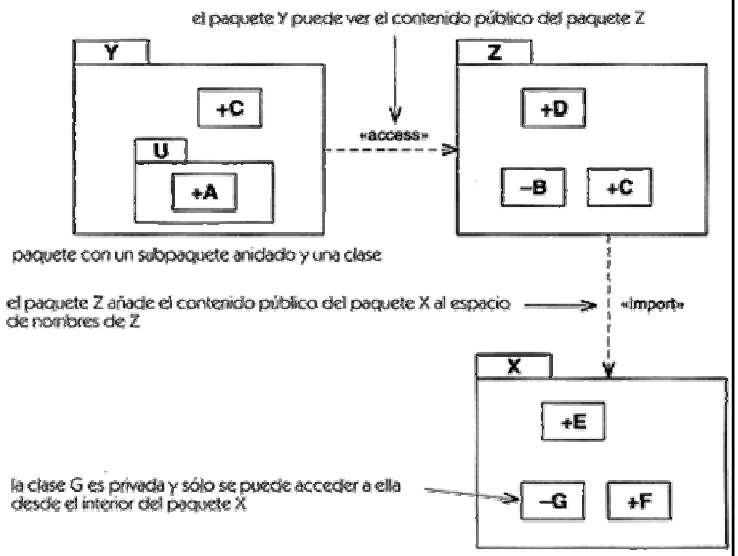
- Los elementos en Types son importados a ShoppingCart.
- Los elementos en Types son importados también a WebShop (por transitividad).
- Los elementos de Auxiliary sólo son accedidos desde ShoppingCart y, por tanto, no pueden usarse sin calificar desde WebShop.

Paquetes - Relaciones

Import

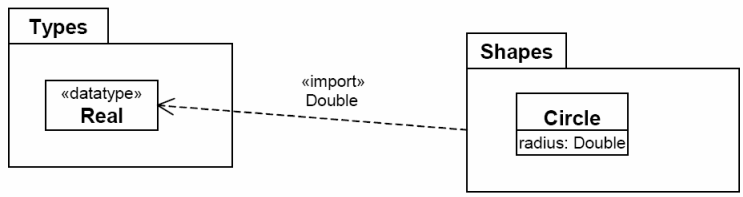
vs

Access



Paquetes - Relaciones

- 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.



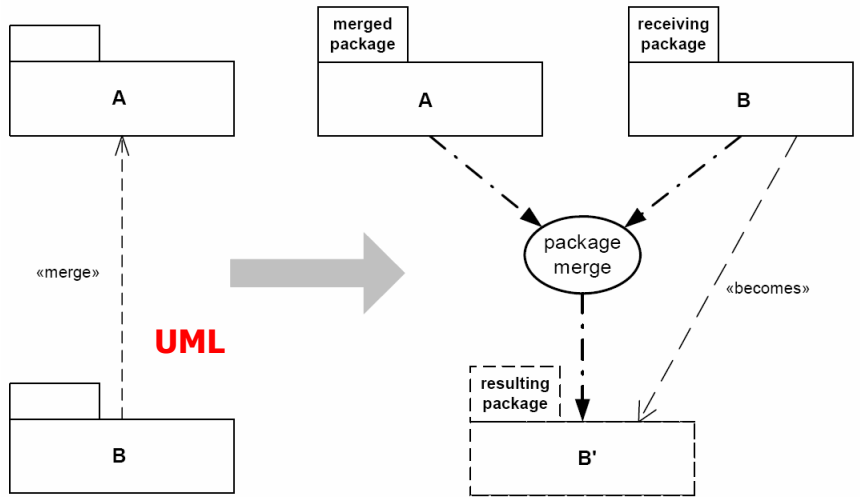
El tipo Types:Real está disponible en el paquete Shapes con el nombre Double.

Paquetes – Fusión

- 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 para realizar la fusión) y transformaciones (postcondiciones después de la fusión).
  - Físicamente, en el repositorio de modelos no se produce ningún cambio en los paquetes.

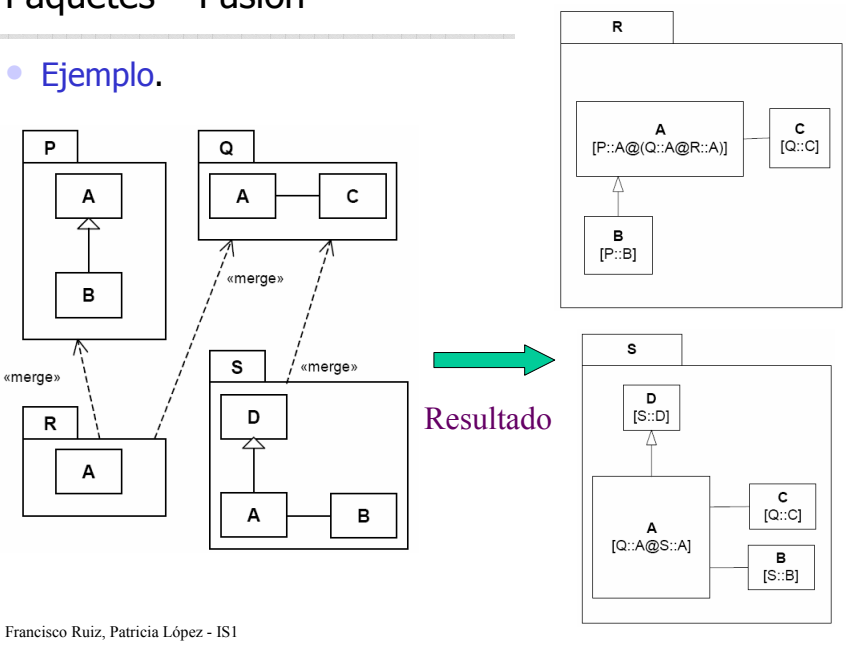
Paquetes – Fusión

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



Paquetes – Fusión

- Ejemplo.

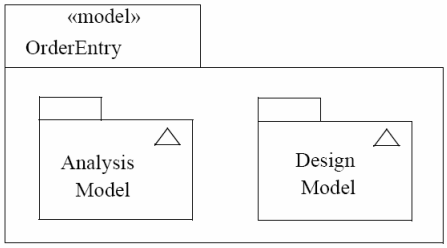


Paquetes – Tipos Especiales

- Todos los mecanismos de extensibilidad pueden ser aplicados a paquetes
  - Con frecuencia se añaden estereotipos y valores etiquetados para incorporar nuevas propiedades a los paquetes.
- Existen varios **estereotipos** que definen nuevas categorías de paquetes (además del ya conocido "profile"):
  - **<<framework>>:** Contiene elementos reusables como clases, patrones y plantillas que definen una arquitectura aplicable a un sistema.
  - **<<modelLibrary>>:** Contiene elementos de modelado que pueden ser reutilizados por diferentes modelos.

Paquetes – Tipos Especiales

- En UML 2, un **modelo** se representa como un paquete estereotipado (<<model>>) que:
  - incluye un conjunto de 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).



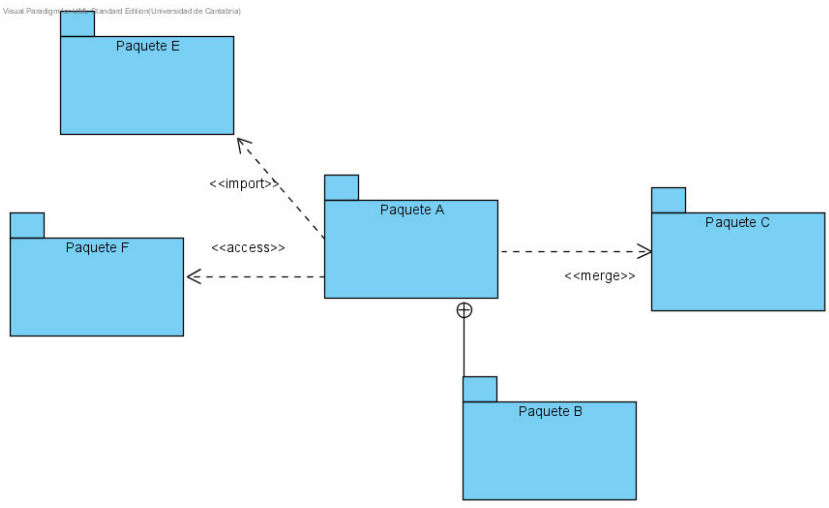
Diagramas de Paquetes

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

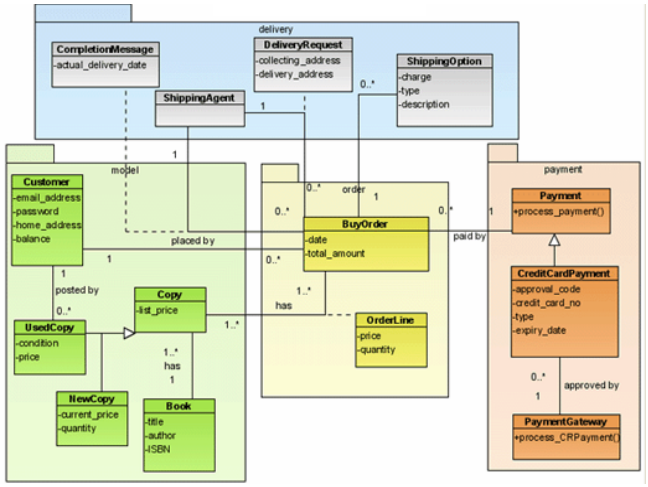
- **Contenido** de un diagrama de paquetes:
  - Paquetes
  - Dependencias entre paquetes
    - Import
    - Access
    - Merge

Diagramas de Paquetes

- Ejemplo.



Paquetes en diagramas de clases



Diagramas de Paquetes - 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

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 más grandes de tipo arquitectural:
  - Paquete "Modelo del diseño":
    - Paquete "Clases e interfaces del modelo".
    - Paquete "Interfaces de usuario".
    - Paquete "Servicios base de datos".

Modelado

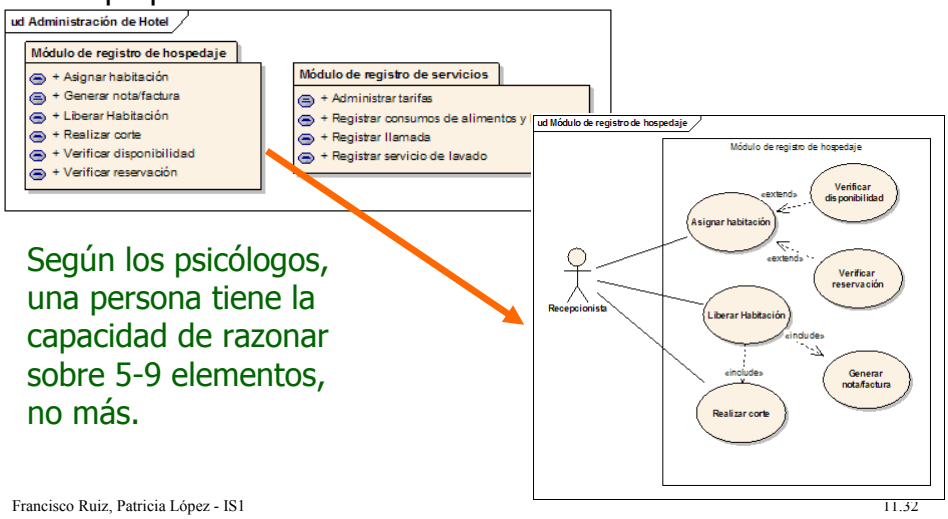
- Los paquetes son especialmente útiles en modelado de:
  - Grupos de Elementos Relacionados
  - Vistas Arquitecturales

Modelado – 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 dependencias (import o access).

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

- 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.

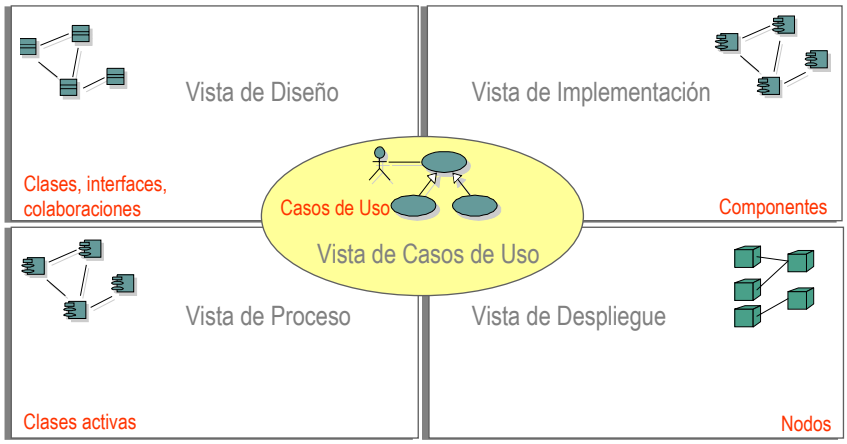
Modelado – Vistas Arquitecturales

- 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.

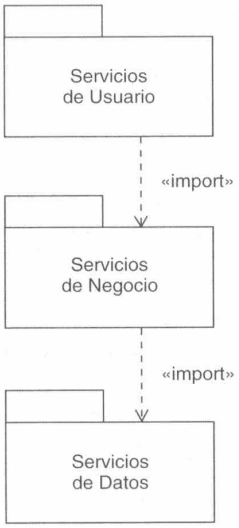
Modelado – Vistas Arquitecturales

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



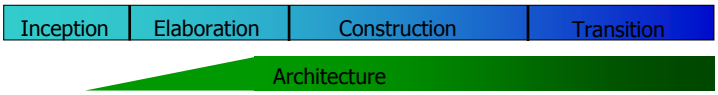
Modelado – Vistas Arquitecturales

- Ejemplo de organización de la vista de diseño
  - 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).



Modelos, Arquitectura y Metodologías

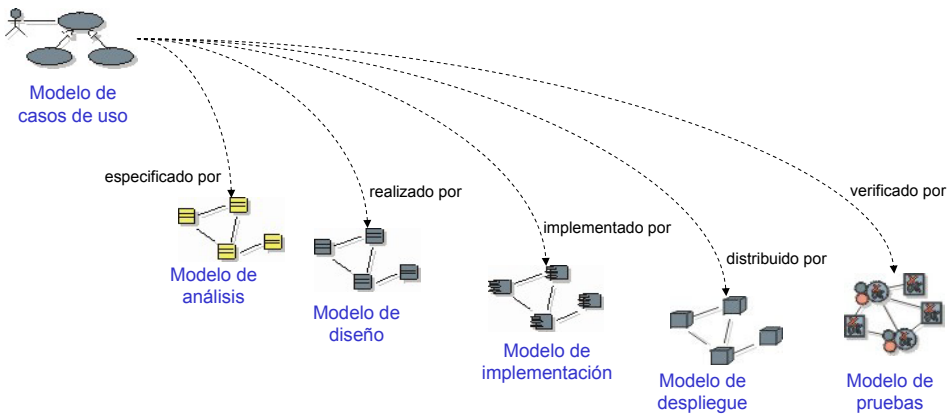
- Casi todas las metodologías de desarrollo de software asignan un papel central a la **arquitectura**.
  - RUP establece refinamientos sucesivos de una arquitectura, construida como un prototipo evolutivo.



- 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.

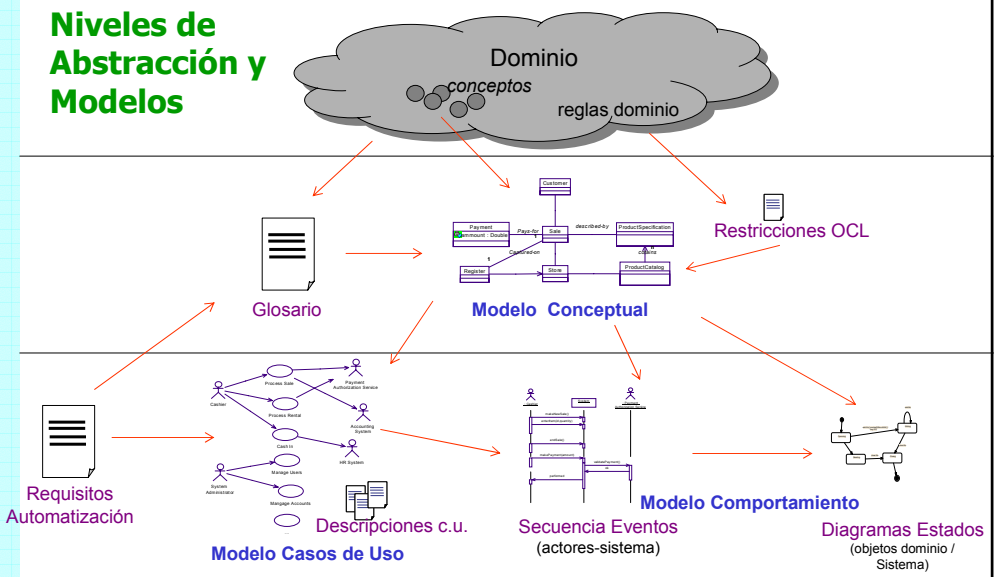


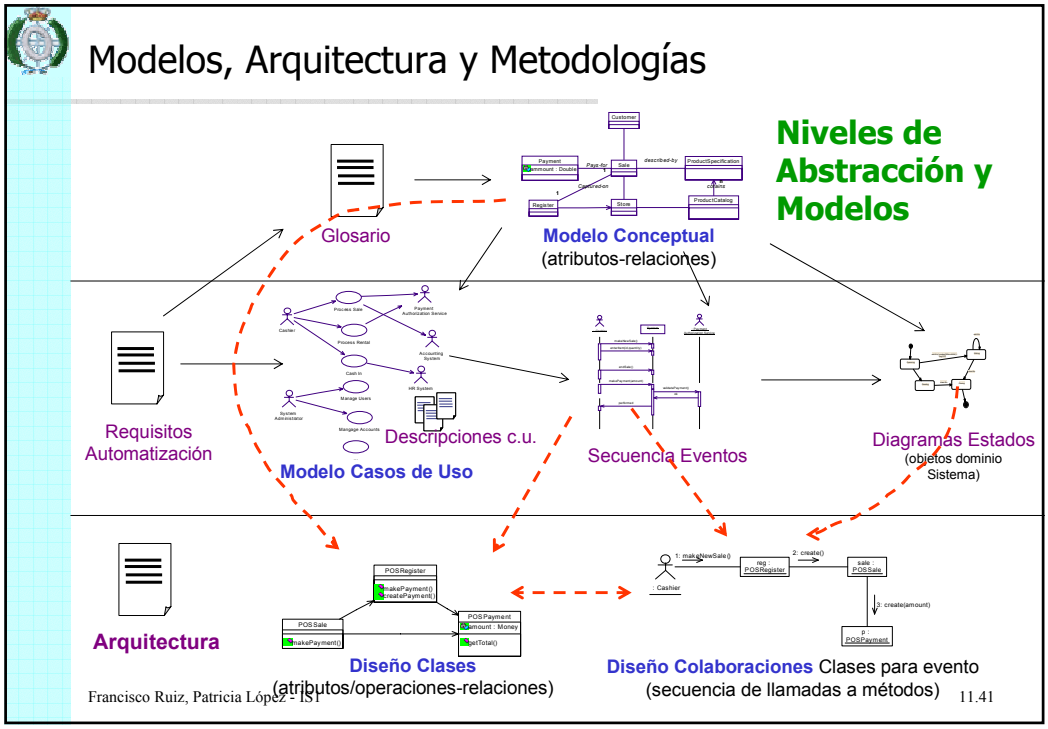
Modelos, Arquitectura y Metodologías

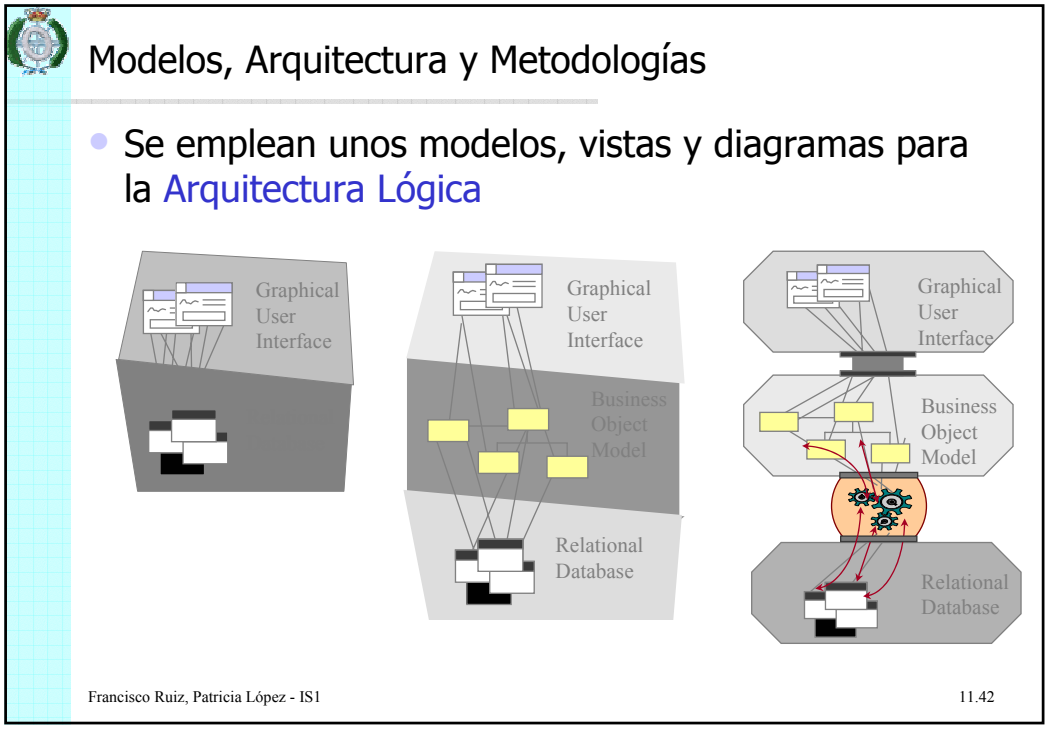
- En general, las metodologías de desarrollo OO basadas en UML suponen ir refinando un sistema en distintos niveles de abstracción, cada vez más elaborados, a partir del conocimiento del dominio del problema:
  - Modelos Conceptuales del Problema
  - Requisitos del Sistema
  - Diseño del Sistema
  -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

Modelos, Arquitectura y Metodologías

Niveles de Abstracción y Modelos




Modelos, Arquitectura y Metodologías
Niveles de Abstracción y Modelos
Glosario
Modelo Conceptual
(atributos-relaciones)
Requisitos Automatización
Modelo Casos de Uso
Descripciones c.u.
Secuencia Eventos
Diagramas Estados
(objetos dominio Sistema)
Arquitectura
Diseño Clases
(atributos/operaciones-relaciones)
Diseño Colaboraciones Clases para evento
(secuencia de llamadas a métodos)
Francisco Ruiz, Patricia López - IS1
11.41


Modelos, Arquitectura y Metodologías
Se emplean unos modelos, vistas y diagramas para la Arquitectura Lógica
Graphical User Interface
Business Object Model
Relational Database
Graphical User Interface
Business Object Model
Relational Database
Graphical User Interface
Business Object Model
Relational Database
Francisco Ruiz, Patricia López - IS1
11.42

Modelos, Arquitectura y Metodologías

- Y otros diferentes para la Arquitectura Física