



Tema 7. Modelado de arquitectura del sistema

Modelado del Software

Fernando Pereñíguez García

Escuela Politécnica



Contenidos

- Introducción
- Arquitectura lógica
 - Paquetes
 - Diagrama de paquetes
- Arquitectura física
 - Elementos:
 - Componentes
 - Diagrama de Componentes
 - Diagrama de Despliegue
 - Artefactos
 - Nodos



Tipos de modelos de arquitectura

- Un aspecto fundamental en el modelado de sistemas software reside en describir su composición desde un punto de **vista arquitectural**.
- Dos niveles:
 - **Lógico**: expresa cuáles son los **componentes lógicos** (subsistemas, funciones, etc.) que componen el sistema y la relación entre ellos.
 - **Físico**: indica los **componentes físicos** que conforman el sistema software.




Aspectos a representar en los modelos

- **Estructura:** son los “**sustantivos**” de UML, tales como clase, interfaz, atributo, componente, nodo...
- **Comportamiento:** son los “**verbos**” de UML, tales como acción, actividad, interacción, estado, mensaje...
- **Agrupamiento:** son los **paquetes**, que se usan para agrupar elementos relacionados semánticamente en unidades coherentes.
- **Anotación:** son las “**notas**”, que pueden añadirse en cualquier parte del modelo para capturar información no gráfica.

Modelado de requisitos	Diagrama de casos de uso	
Modelado estático	Diagrama de clases	
	Diagrama de objetos	
Modelado dinámico	Interacción	Colaboración
		Secuencia
	Diagrama de estados	
	Diagrama de actividad	
Modelado físico	Diagrama de componentes	
	Diagrama de despliegue	

Contenidos

- Introducción
- **Arquitectura lógica** 
- Paquetes
- Diagrama de paquetes
- Arquitectura física
- Elementos:
 - Componentes
- Diagrama de Componentes
- Diagrama de Despliegue
 - Nodo
 - Artefactos



Arquitectura lógica del sistema

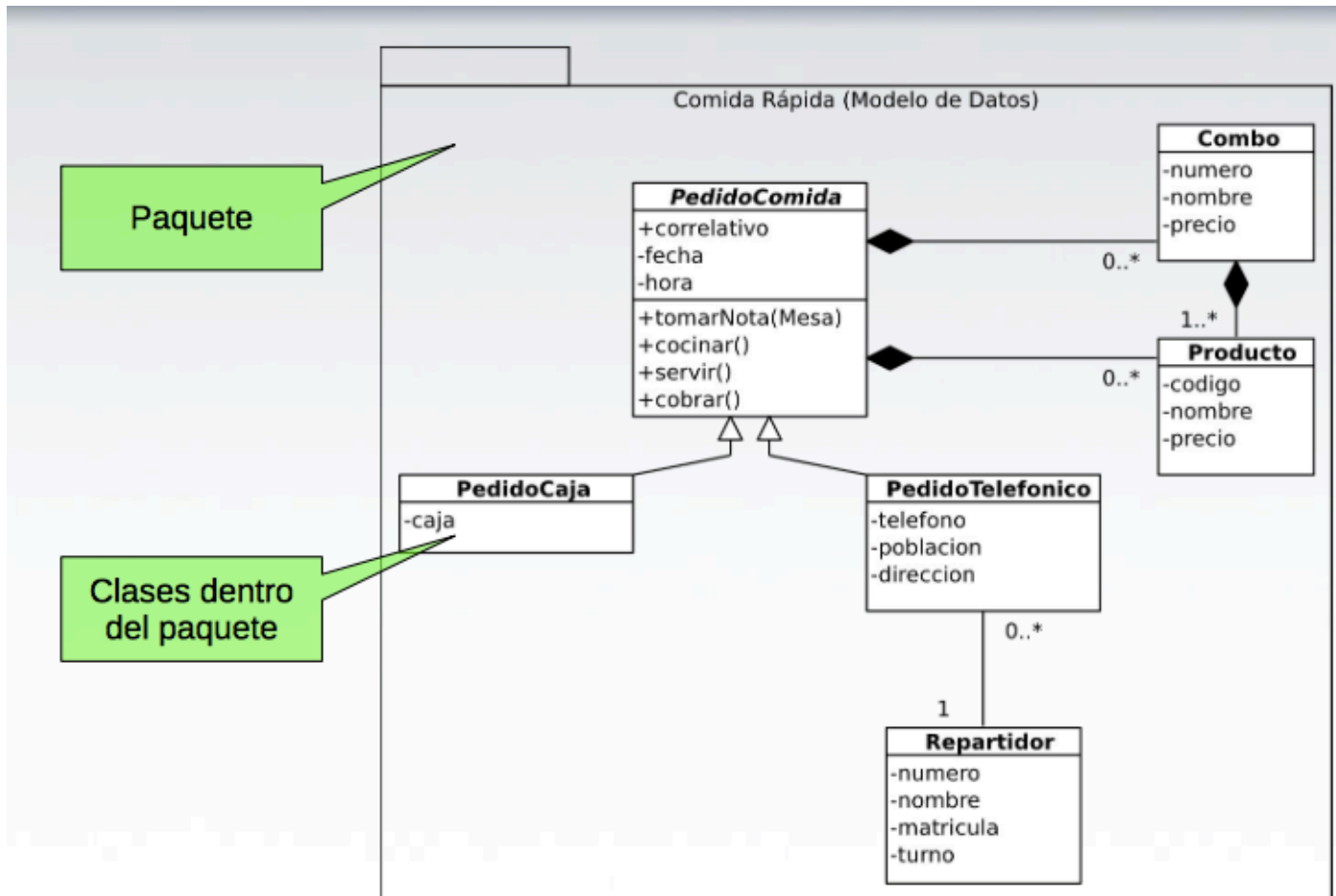
- Modelar sistemas medianos o grandes conlleva manejar una cantidad considerable de elementos
 - Clases, interfaces, nodos, relaciones, diagramas,...
- ¡¡ Necesario organizar !!
- En UML el elemento de organización es el **paquete**.
 - Elemento para organizar, pero no tiene identidad (no instancias de paquetes)
- Los **paquetes** permiten:
 - **Organizar elementos** de los modelos
 - **Controlar el acceso** a sus contenidos



Arquitectura lógica del sistema: paquetes

- Organizan elementos de un modelado en diferentes grupos (incluidos los diagramas).
- Pueden darse estructuras recursivas, donde un paquete incluye otro paquete.
- También organizan modelos grandes, agrupar elementos relacionados o crear namespaces.
- Puede ser incluido en cualquier tipo de diagrama y, a su vez, puede contener también otros diagramas.

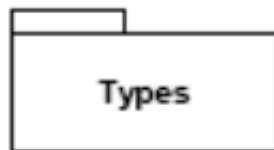
Notación



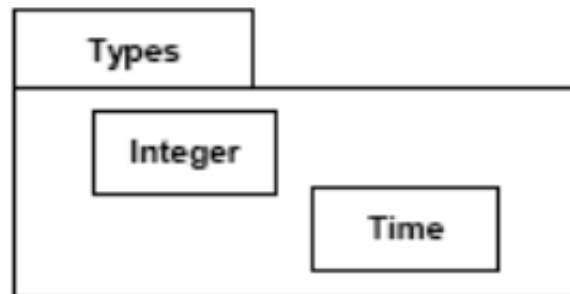
Notación

- Formas de dibujar un paquete:

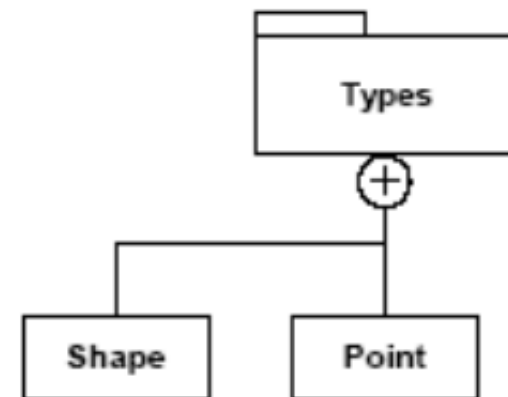
Notación interna



Sin especificar
su contenido



Notación externa



Uso de paquetes

- Un paquete debe agrupar elementos cercanos semánticamente.
- **Puede contener:** clases, interfaces, componentes, nodos, casos de uso, otros paquetes.
- Pueden formar relaciones jerárquicas.

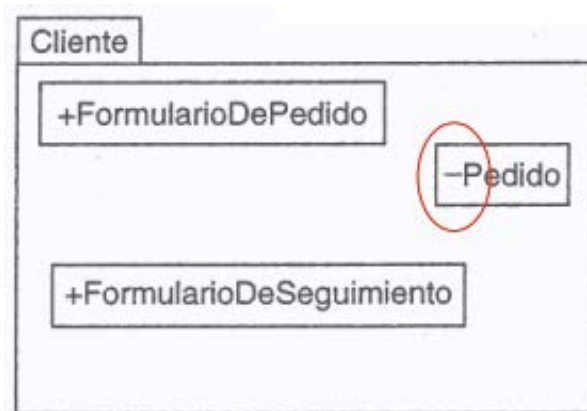


Uso de paquetes

- 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 “Persona” en el mismo paquete
 - P1::Cola y P2::Cola son elementos diferentes en paquetes diferentes (P1 y P2)
 - Sí puede haber una clase Tiempo y un componente Tiempo.
- Consejo: no repetir nombres, aunque sean de tipo diferente, incluso en paquetes diferentes.

Uso de paquetes

- Se puede controlar la **visibilidad** de los elementos que contiene un paquete
 - **Público (+)**: elemento disponible a otros elementos del paquete contenedor o uno de sus paquetes anidados, y a paquetes que importan el paquete contenedor.
 - **Privado (-)**: elemento no disponible fuera del paquete contenedor.
 - **Protegida (#)**: acceso permitido a elemento protegidos del padre



Uso de paquetes

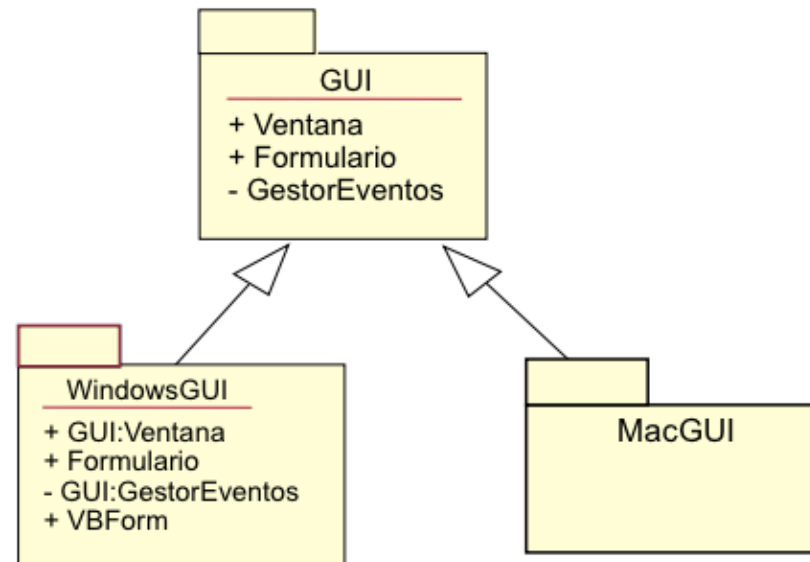
- Un elemento de un paquete puede referenciar:
 - Elementos **públicos** de cualquier paquete al que tenga permisos: **nombre cualificado** (P::A).
 - Elementos(incluso **privados**)de su paquete, o cualquier paquete **que incluya al suyo**.
 - Elementos **protegidos** de sus paquetes *padre*(relación de generalización).

Relaciones entre paquetes

- **Generalización:** el paquete especializado puede usarse en sustitución del general (similar entre clases).
- Un contenido puede ser **sobrescrito** en un paquete hijo.
- El paquete hijo hereda los contenidos públicos y protegidos del padre, incluyendo los contenidos importados.

Relaciones entre paquetes

- Generalización:



Relaciones entre paquetes

- Para que un elemento de un paquete tenga acceso (permisos) a otro elemento de otro paquete se debe crear una relación entre ambos paquetes.
- **Dependencia:** algún elemento de un paquete depende de los elementos de otro paquete
 - **Exportación:** partes públicas de un paquete
 - **Importación:** <<Import>>
 - **Acceso:** <<Access>>

¿Cuándo usar
uno u otro?

Diferencia entre <<Import>> y <<Access>>

- Los paquetes pueden importar elementos de otros paquetes, a través de una relación de dependencia entre paquetes con el estereotipo <<import>>.
 - Al menos un elemento del paquete cliente utiliza los servicios ofrecidos por al menos un elemento del paquete proveedor.
 - Cuando se importa un paquete, **sólo son accesibles sus elementos públicos.**

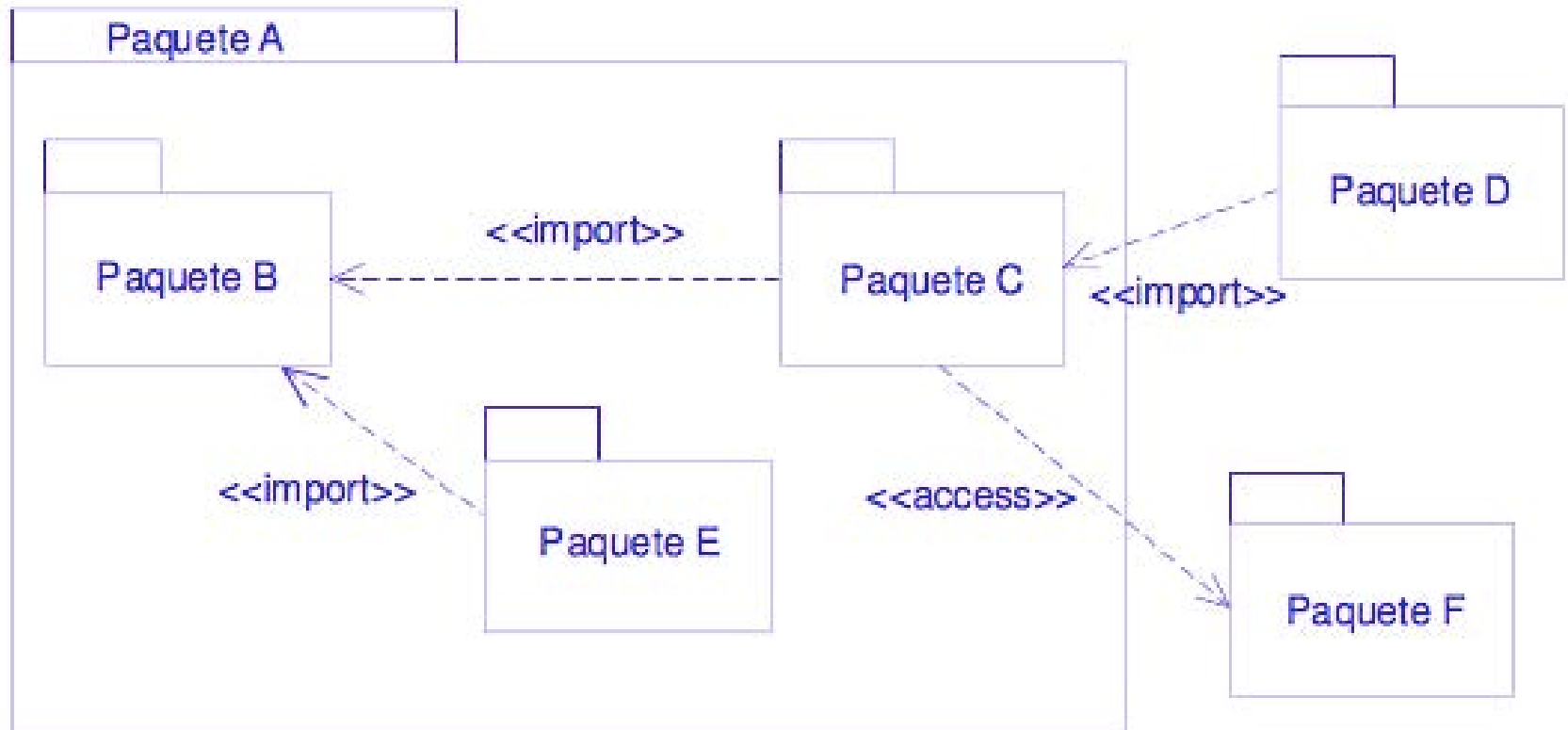
Diferencia entre <<Import>> y <<Access>>

- ¿Importación o acceso?
 - Ambas indican que el paquete origen tiene acceso la contenido del destino.
 - NO son **transitivas**
- <<import>> Incorpora el espacio de nombres de un paquete a otro (ojo a duplicados).
 - No hace falta cualificar nombres. Puede darse un alias a cada elemento importado y definir una nueva visibilidad.
- <<access>> Permite acceder al espacio de nombres de otro paquete sin llegar a incorporar sus elementos.
 - Los elementos accedidos se indican con su nombre cualificado.

Diferencia entre <<Import>> y <<Access>>

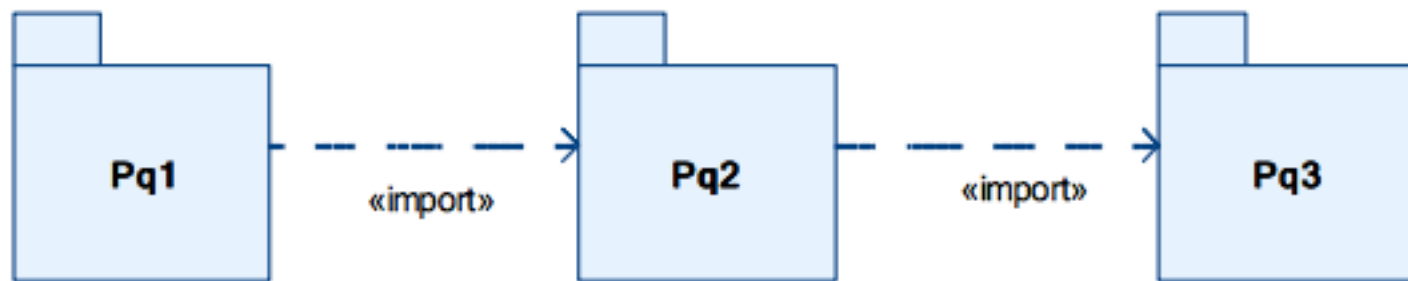
- La relación de importación puede ser **pública** (por defecto, <<import>>) o **privada** (<<access>>).
 - Una **importación pública** significa que los elementos importados tienen visibilidad pública en el paquete cliente.
 - Una **importación privada** significa que los elementos importados tienen visibilidad privada en el paquete cliente y no pueden usarse fuera de él, incluyendo cualquier otro paquete que lo importara.

Diferencia entre <<Import>> y <<Access>>



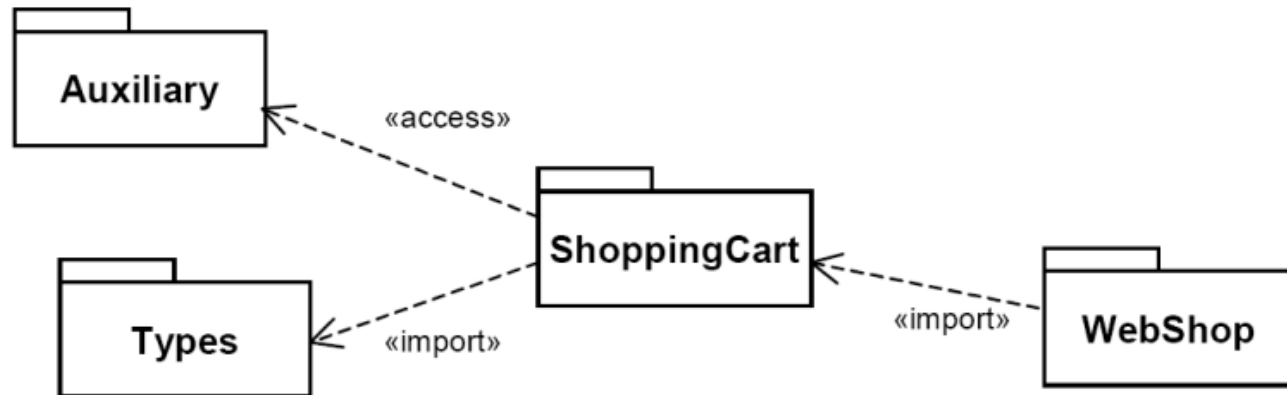
<<Import>> y <<Access>>

- Las relaciones de permiso no son simétricas ni transitivas.
- Pq1 puede acceder al contenido público de Pq2 y Pq2 al de Pq3 pero:
 - Pq2 no puede acceder a ningún contenido de Pq1.
 - Pq1 no puede acceder a ningún contenido de Pq3.



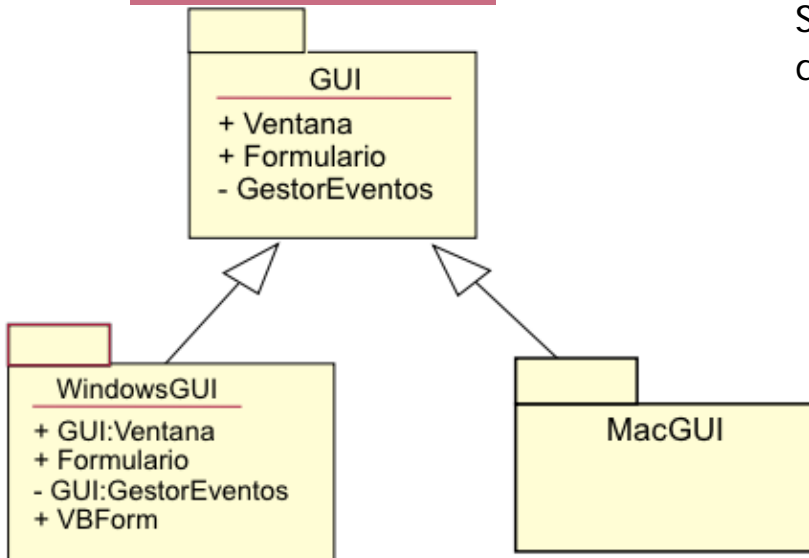
Ejemplos

Dependencia



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

Generalización



Recomendaciones

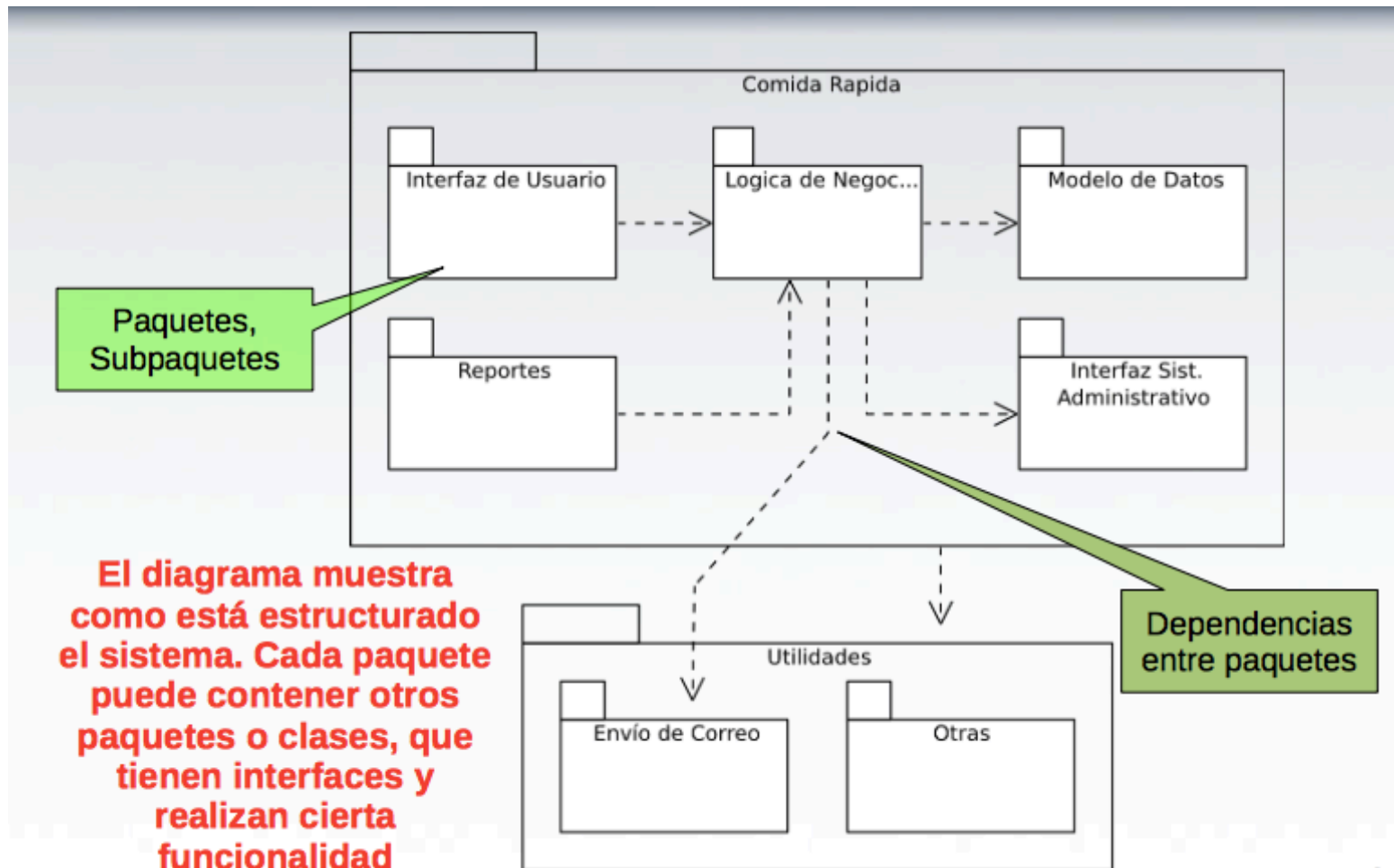
- Procurar que los paquetes estén bien **estructurados**:
 - Es **cohesivo**: proporciona un límite bien definido alrededor de un grupo de elementos relacionados.
 - Está **poco acoplado**: importa lo realmente necesario y exporta sólo los elementos que otros paquetes necesitan ver.
 - **No muy anidado** (max. 3 niveles)
 - Posee un conjunto **equilibrado de elementos** (5-9 nodos).

Aspectos importantes

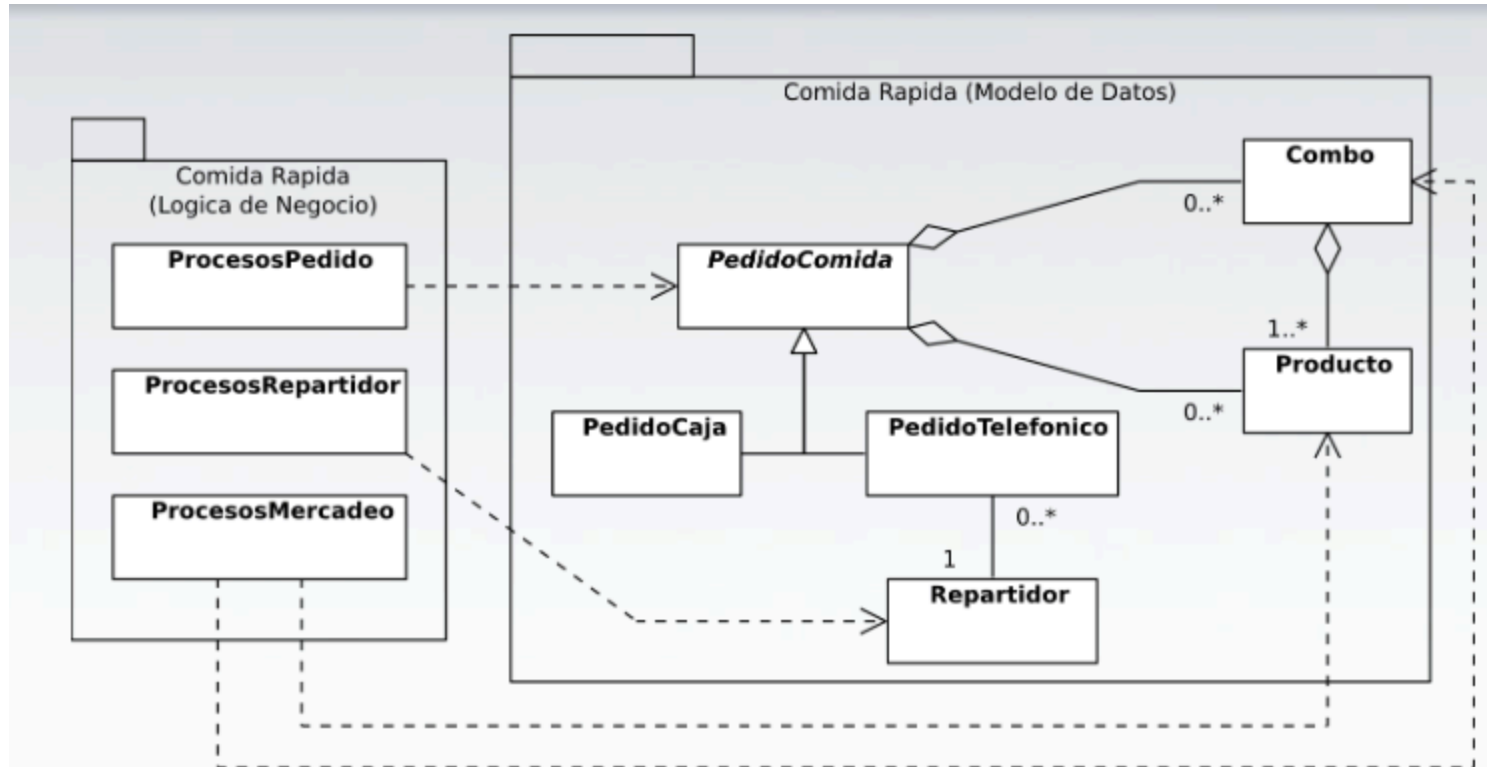
- Aspectos a tener en cuenta...
 - Agrupar en paquetes elementos que:
 - Tiene un objetivo común
 - Están relacionados a nivel conceptual
 - Pertenecen al mismo caso de uso
 - Buena idea: crear paquetes arquitecturales
 - “Clases e interfaces del modelo” , “Interfaces usuario” ,
“Servicios bases de datos” , “Modelo de análisis”



Ejemplos Diagramas de Paquetes

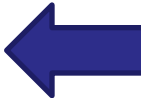


Ejemplos D. Paquetes



También se pueden mostrar algunas clases dentro de los paquetes, así como las relaciones de dependencia de estas clases con otras clases o paquetes

Contenidos

- Introducción
- Arquitectura lógica
 - Paquetes
 - Diagrama de paquetes
- **Arquitectura física** 
 - Elementos:
 - Componentes
 - Diagrama de Componentes
 - Diagrama de Despliegue
 - Nodo
 - Artefactos

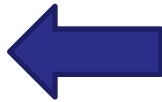


Arquitectura física del sistema

- En el desarrollo de cualquier producto (edificio, avión, etc.) los planos son muy importantes
 - Pero lo importante es dar lugar a una **construcción REAL**.
- UML nos ofrece dos clases de elementos para **modelar la arquitectura física de un sistema**:
 - **Componentes**
 - **Nodos**



Contenidos

- Introducción
- Arquitectura lógica
 - Elementos: Paquetes
 - Diagrama de paquetes
- Arquitectura física
 - **Elementos:**
 - **Componentes** 
 - Diagrama de Componentes
 - Diagrama de Despliegue
 - Nodo
 - Artefactos



Componente

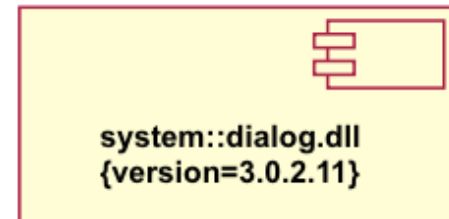
- **Parte física** de un sistema que conforma y proporciona la implementación de un conjunto de interfaces
 - Se usa para modelar elementos físicos que pueden encontrarse en un nodo:
 - Ejecutables, bibliotecas, tablas, archivos, documentos.
- **Encapsula el estado y comportamiento** de un conjunto de clasificadores (p.ej. clases)
- Especifica un **contrato** de los servicios que proporciona y de los que requiere.
- Unidad **reemplazable** por otro componente que ofrezca la misma funcionalidad.

A través de una interfaz un componente puede ofrecer servicios a otro componente.

Componente

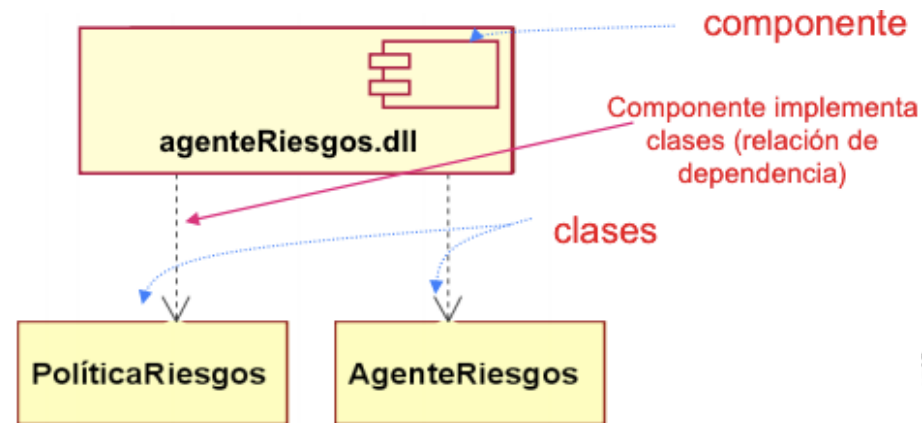
- **Notación:**

- Rectángulo con un icono especial en la esquina superior derecha.
- Normalmente sólo se muestra su nombre.
- Pueden estar estereotipados.



Diferencia entre clases y componentes

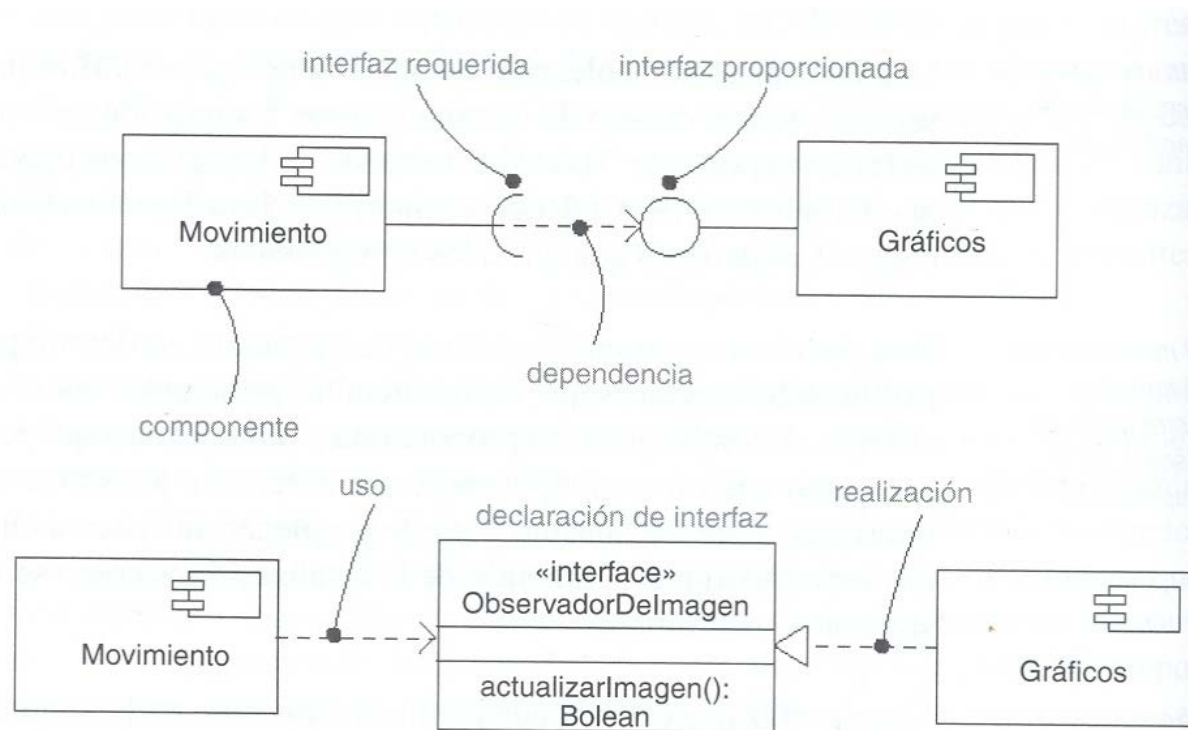
- Al igual que las clases tienen: nombres, interfaces, pueden participar en relaciones.
- Diferencias:
 - Clases
 - Son abstracciones lógicas
 - Tienen operaciones y atributos
 - Componentes
 - Son fragmentos físicos del sistema
 - Tienen interfaces



Un componente es la implementación de más de una clase. Los componentes no tienen atributos, sólo operaciones alcanzables a través de sus interfaces.

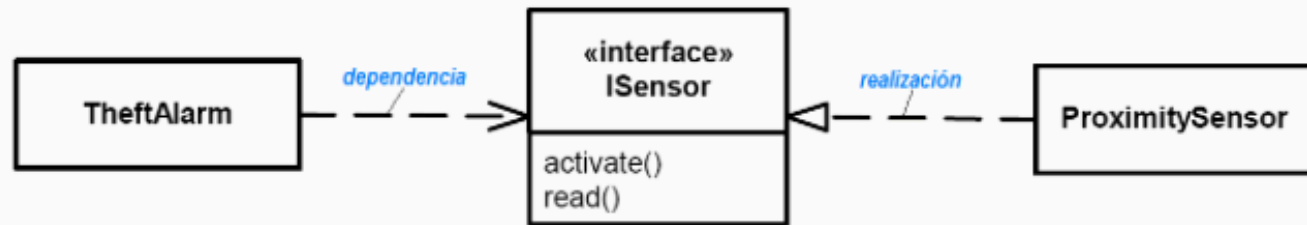
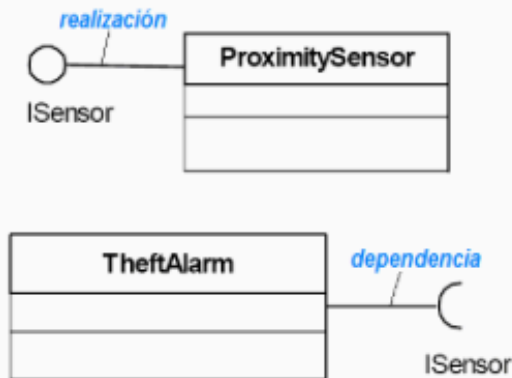
Conceptos clave de los componentes

- **Interfaz:** Colección de **operaciones** que especifican un servicio proporcionado o requerido por un componente.



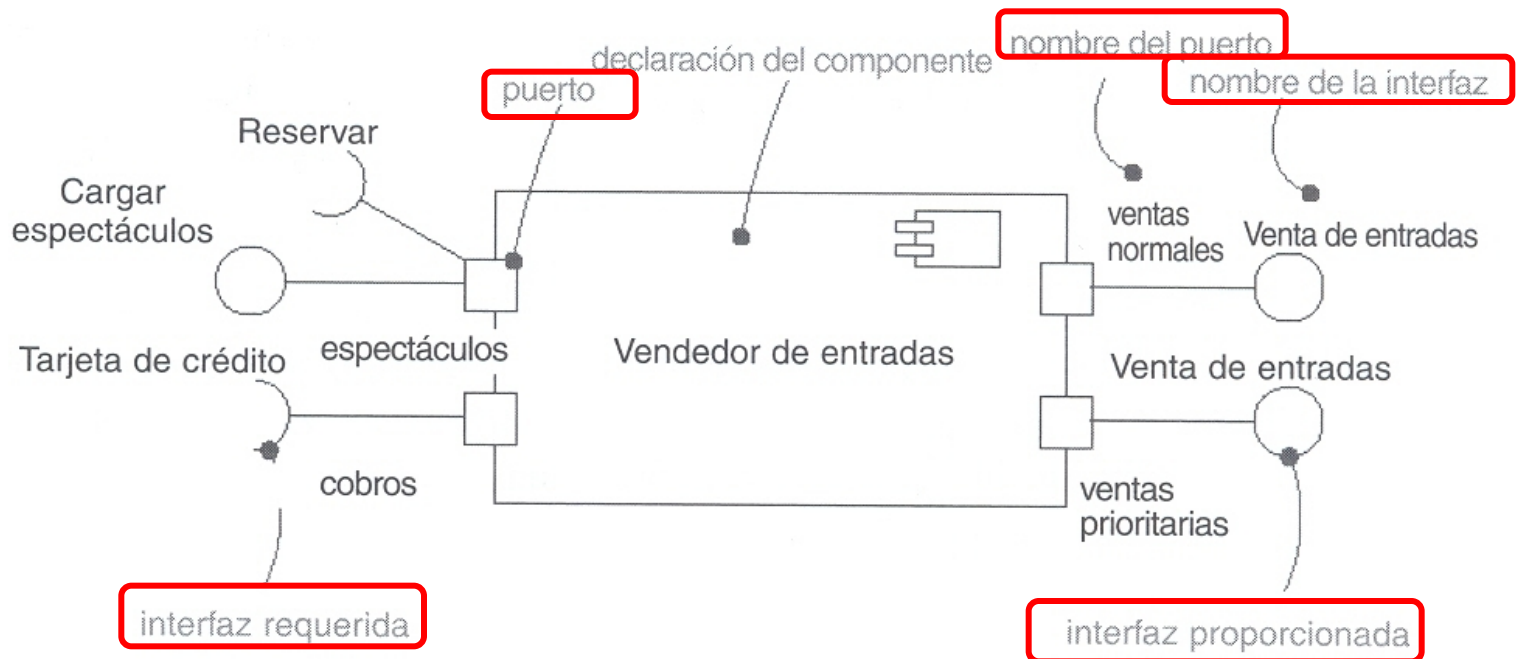
Conceptos clave de los componentes

- Entre los componentes debe existir un **acoplamiento débil**, para que los cambios en uno de ellos no afecten al resto del sistema.
- Acceso a los componentes por medio de interfaces.
- **Interfaz**: Colección de operaciones que especifican un servicio **proporcionado** o **requerido** por un componente.



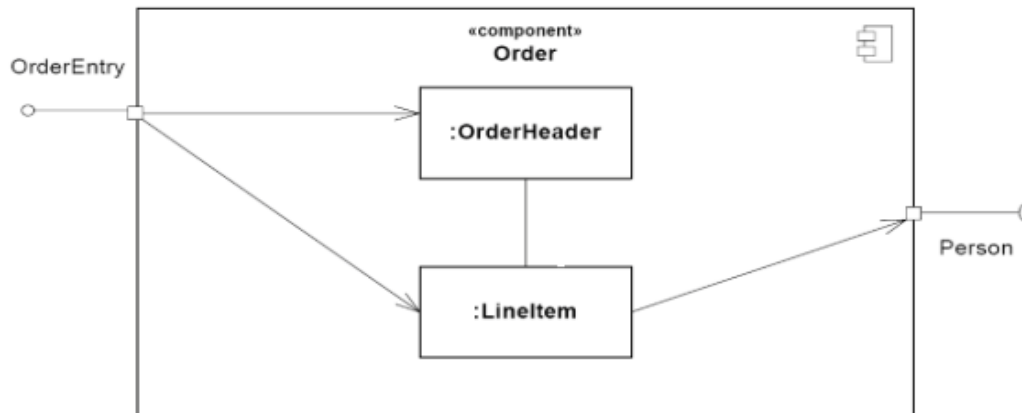
Conceptos clave de los componentes

- **Puerto:** Ventana específica de un componente encapsulado, que acepta mensajes hacia y desde el componente, que son conformes con las interfaces especificadas.

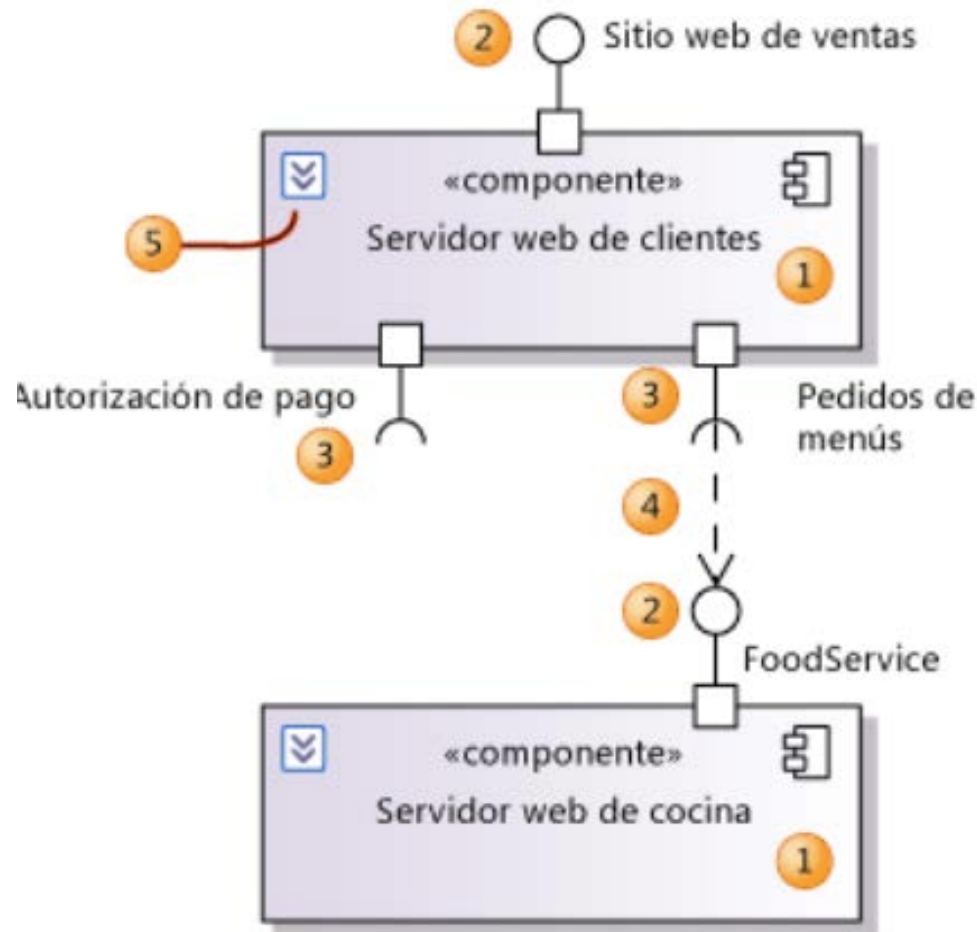


Conceptos clave de los componentes

- Un componente puede tener puertos para modelar las distintas formas de uso mediante las interfaces conectadas al puerto.
 - La interfaz **ofrecida** por un componente puede ser realizada por uno de sus elementos internos.
 - Una interfaz **requerida** por el componente puede ser usada por uno de sus elementos internos.

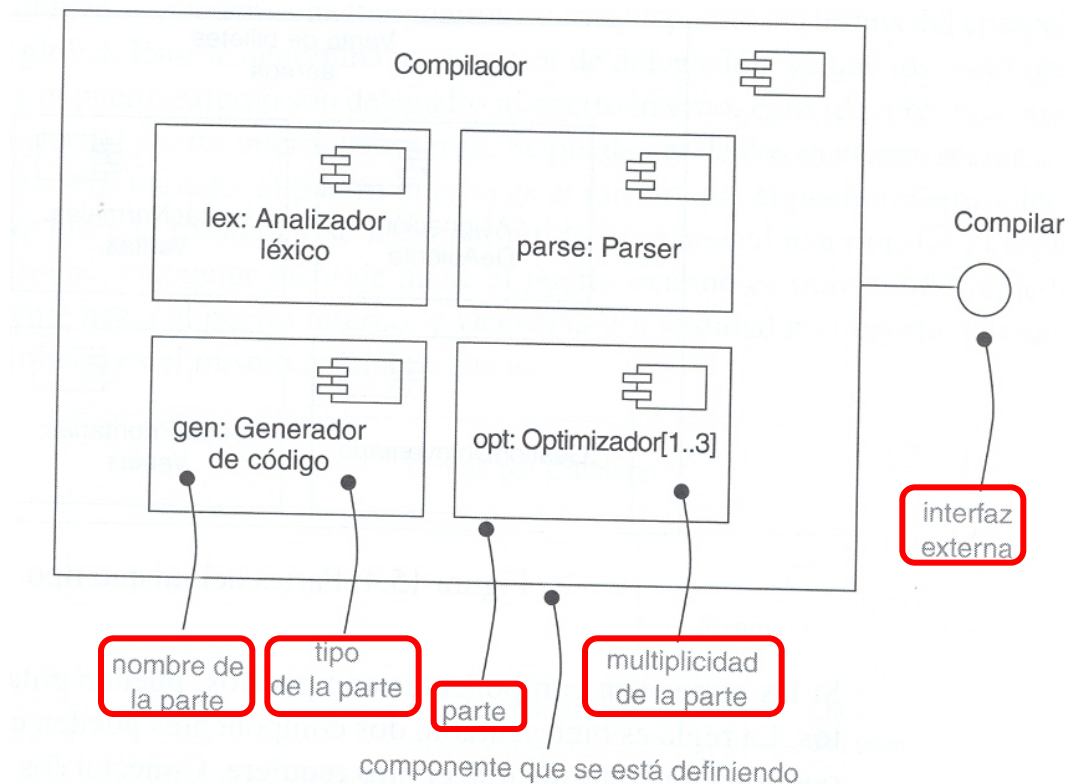


Conceptos clave de los componentes



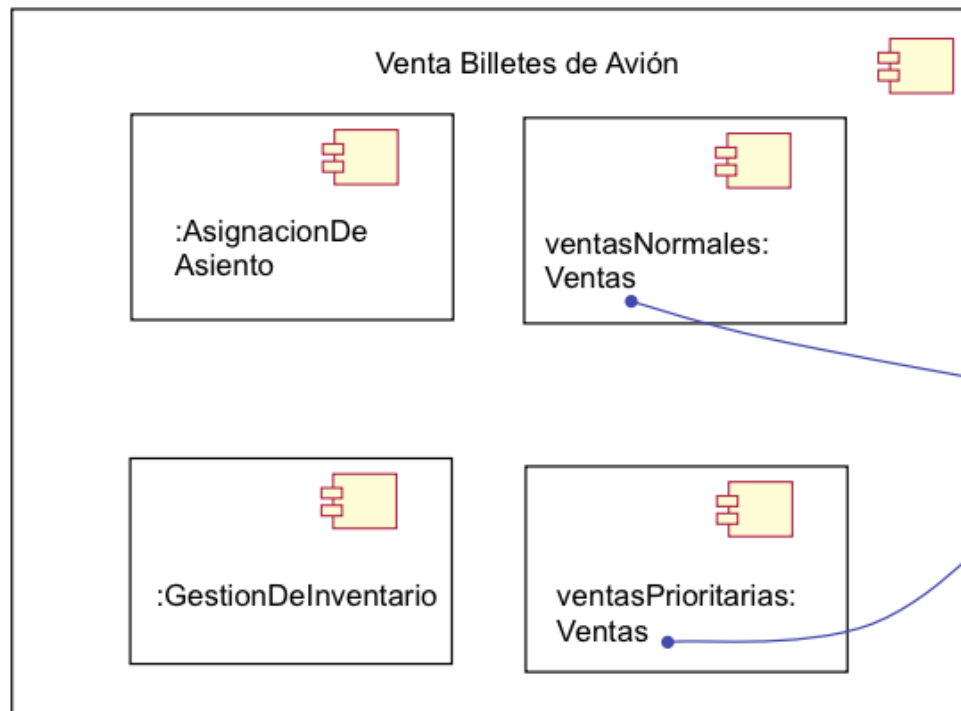
Conceptos clave de los componentes

- **Estructura interna:** Implementación de un componente a través de un conjunto de partes y sus conexiones.



Conceptos clave de los componentes

- **Parte:** Especificación de un rol que forma parte de un componente.
 - Juegan un papel similar a los atributos de una clase.



*dos partes con
igual tipo*



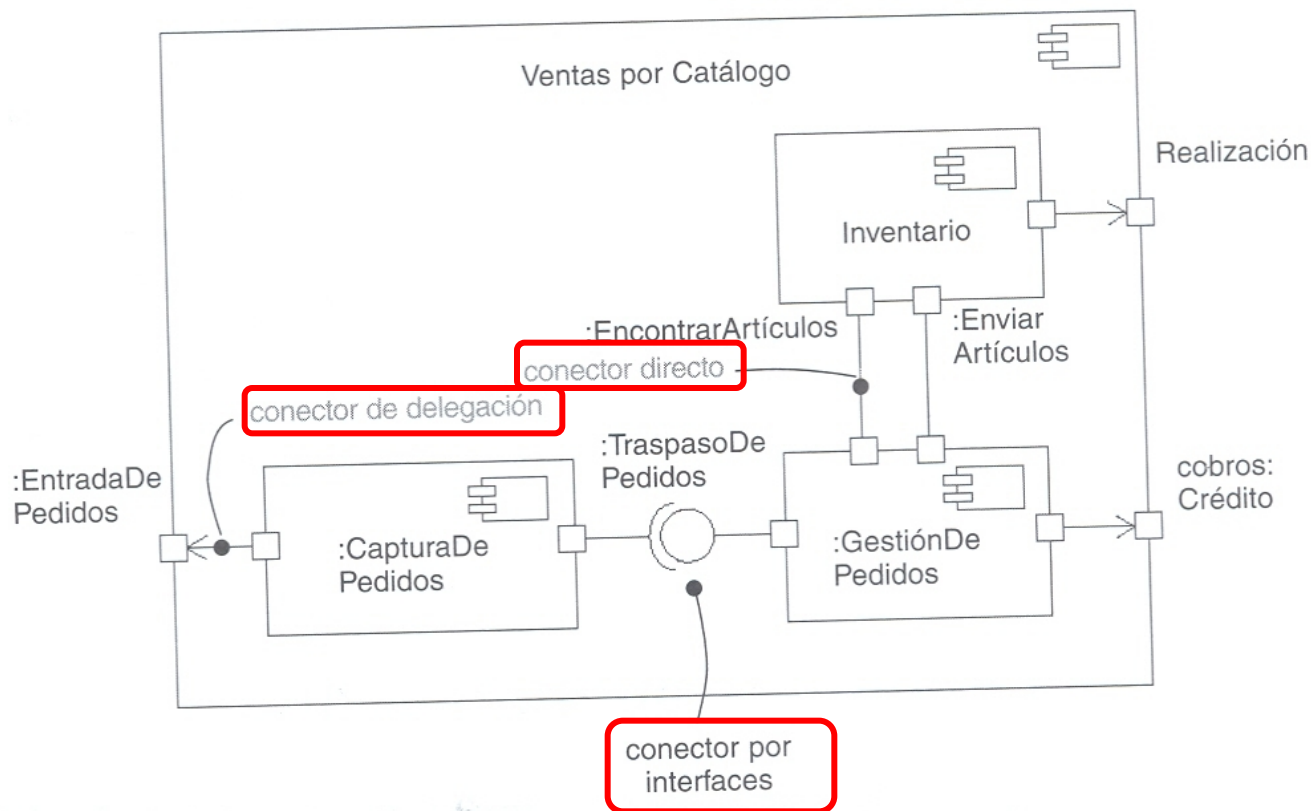
Conceptos clave de los componentes

- **Conector:** Relación de comunicación entre dos partes o puertos dentro del contexto de un componente.
 - Los componentes pueden ser conectados:
 - **Directamente** (mediante una línea entre ellos o sus puertos)
 - Porque tienen **interfaces compatibles** (mediante junta circular)
 - **Conector de delegación:**
 - Conecta un puerto interno a uno externo.
 - Se representa mediante una flecha desde el puerto interno al externo.
 - Actúa como si el puerto interno fuese el externo, es decir, **cualquier mensaje llegado al puerto externo es transmitido inmediatamente al puerto interno.**

Diagrama de Componentes

- Los **diagramas de componentes** muestran un conjunto de componentes y sus relaciones.
 - Describen los elementos físicos del sistema y sus relaciones.
- Un diagrama de componentes contiene:
 - **Componentes**
 - **Interfaces**
 - **Relaciones** de dependencia, generalización, asociación y realización.
- Cubren la **vista de implementación estática** de un sistema.

Diagrama de Componentes

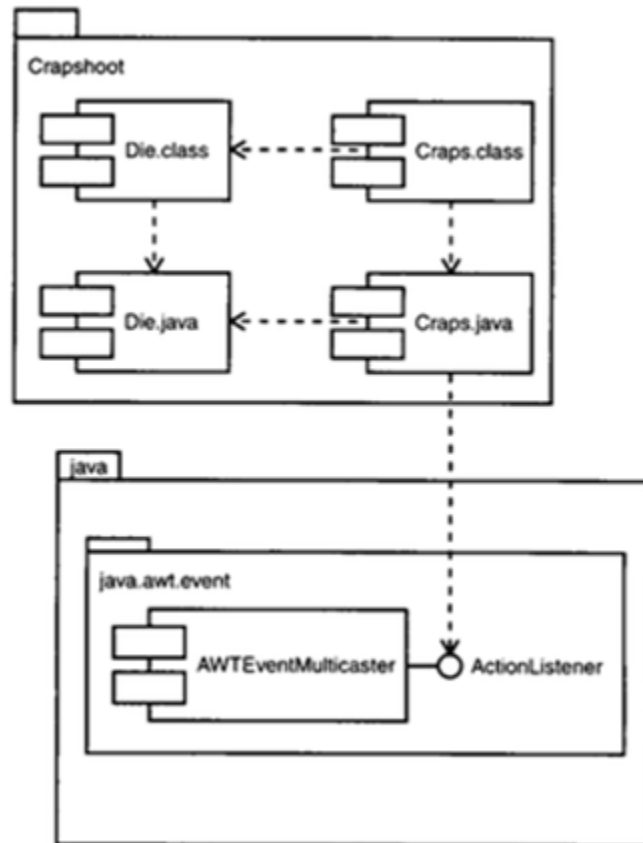


Ejercicios D.Componentes

Un juego de dados para web "Craps":

- El código fuente del applet se encuentra en el archivo Craps.java y código del objeto en Craps.class.
- El código fuente de la clase Die se encuentra en Die.java y el código del objeto en Die.class (Die se usa para crear los dados).
- Los cinco archivos se encuentran en el mismo directorio, que llamaremos tirodados. Cada archivo .class es un componente y cada uno es la implementación de una clase. Craps.java y Die.java utilizan la clase java.awt, conjunto de herramientas abstractas que muestran y controlan la GUI.

Ejercicios D.Componentes



Contenidos

- Introducción
- Arquitectura lógica
 - Paquetes
 - Diagrama de paquetes
- Arquitectura física
 - Componentes
 - Diagrama de Componentes
 - **Diagrama de Despliegue**
 - Nodo
 - Artefactos

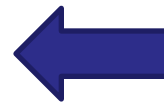


Diagrama de Despliegue

- Los **diagramas de despliegue** muestran un conjunto de **nodos, sus relaciones y los artefactos que residen en ellos**.
- Los diagramas de despliegue muestran:
 - El **hardware** sobre el que se ejecutará el sistema.
 - La **configuración** de los nodos que participan en la ejecución del software.
 - Los **artefactos** que residen en cada nodo.
- Describen la **vista de despliegue estática**.

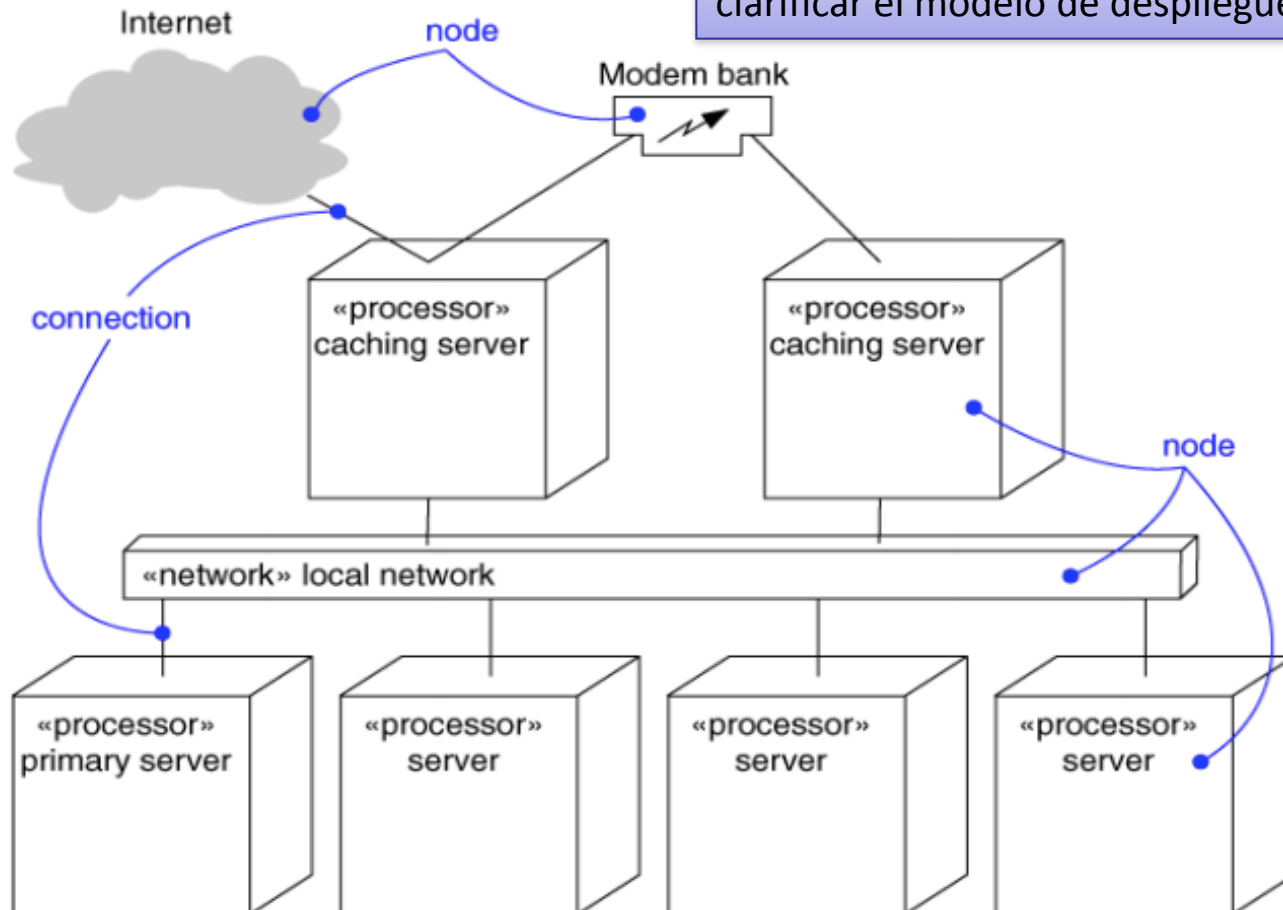
Diagrama de Despliegue

- Los diagramas de despliegue **no son necesarios** cuando:
 - El software reside en una única máquina
- Los diagramas de despliegue **son necesarios** cuando:
 - El sistema está distribuido físicamente sobre varios procesadores.
 - Es necesario razonar sobre la topología de procesadores y dispositivos sobre los que se ejecuta el software.

Diagrama de Despliegue

Se puede usar estereotipos para distinguir entre nodos y relaciones.

La nube que representa Internet no es parte de la simbología de UML pero es útil para clarificar el modelo de despliegue.

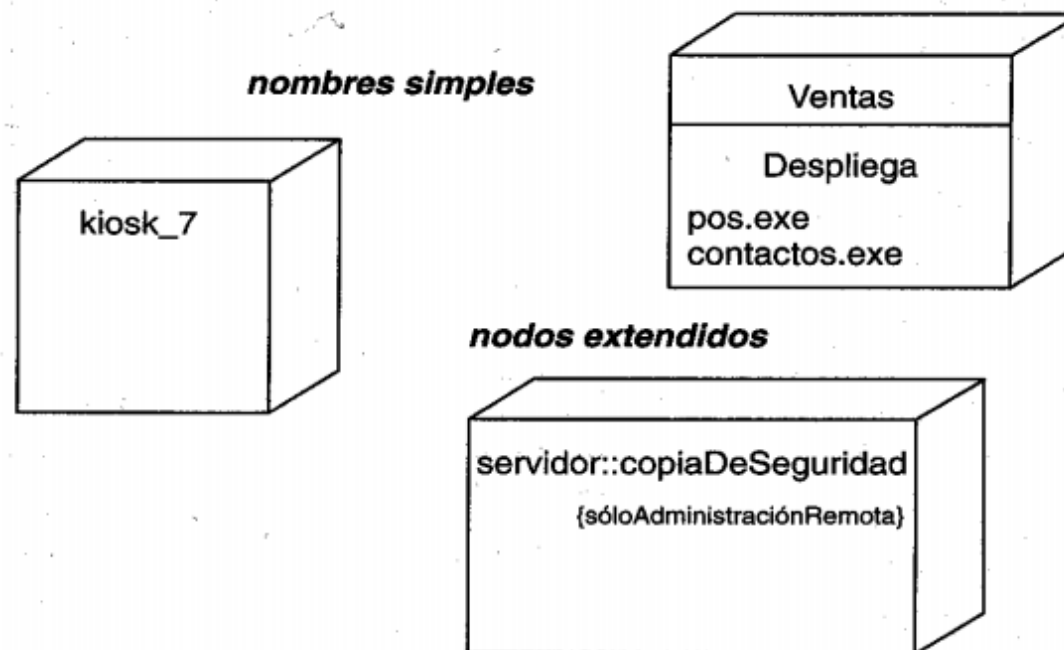


Nodo

- Un nodo se usa para representar **un recurso hardware o software**, en el que puede residir el software ejecutable o los ficheros relacionados.
- **Elemento físico** que existe en tiempo de ejecución
- Empleado para **modelar topología del hardware** donde se desplegará el sistema.
- **Los componentes representan el empaquetamiento físico lo nodos el despliegue físico de componentes.**

Nodo

- **Notación:**
 - Gráficamente se representa como un cubo.



Artefacto

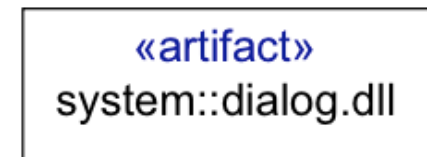
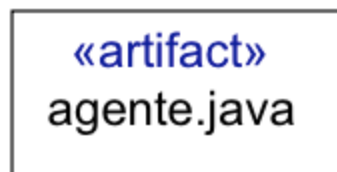
- El **software que se ejecuta en los nodos** se modela con artefactos, que son **ficheros físicos** que el software usa o ejecuta
- **Implementación física** de un conjunto de elementos lógicos tales como clases y componentes.
- Los artefactos residen en **nodos**.



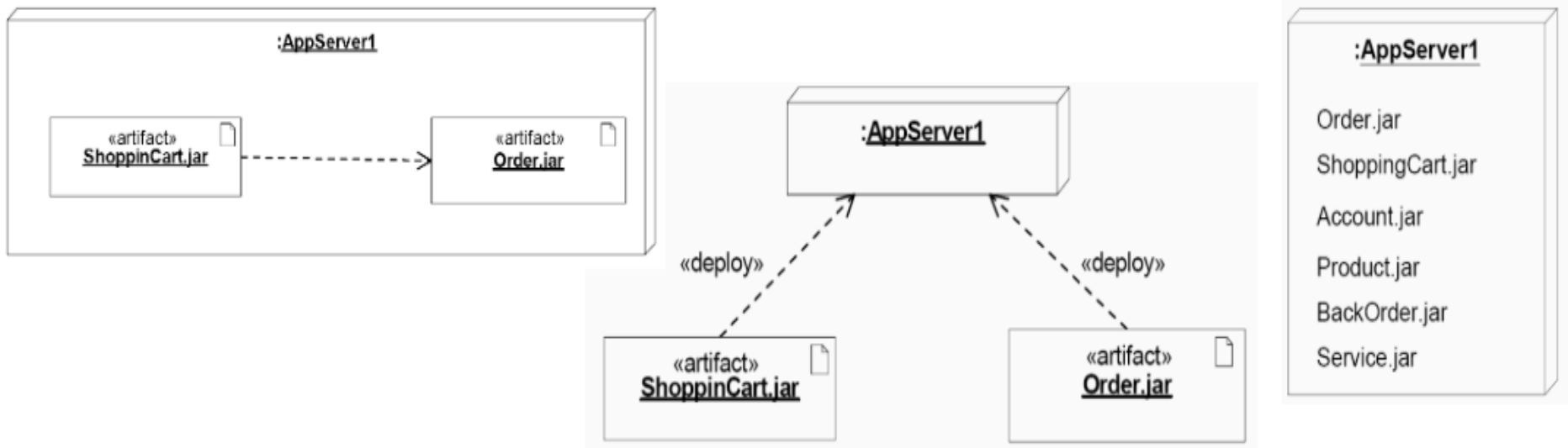
Artefacto

- **Notación**

- Se representa con el estereotipo **<<artifact>>**
- El nombre puede ser simple o cualificado.
- Se pueden adornar con valores etiquetados o compartimentos adicionales.



Artefacto



Diferencia entre clases y artefactos

- CLASES:
 - Las clases representan **abstracciones lógicas**.
 - Las clases pueden tener atributos y operaciones.
- ARTEFACTOS:
 - Los artefactos representan **elementos físicos** formados por bits.
 - No pueden tener atributos y operaciones.
 - Los artefactos implementan aspectos lógicos (p.ej. clases, componentes, etc.).
 - La relación entre artefacto y las clases que implementa se representa mediante una relación de manifestación (<<**manifest**>>).

Tipos de artefactos

- Producto de trabajo

- Permanecen al final del **proceso de desarrollo**
- No participan directamente en un sistema ejecutable.
- Ejemplos: archivos de código fuente, ficheros de datos, etc.

- Despliegue

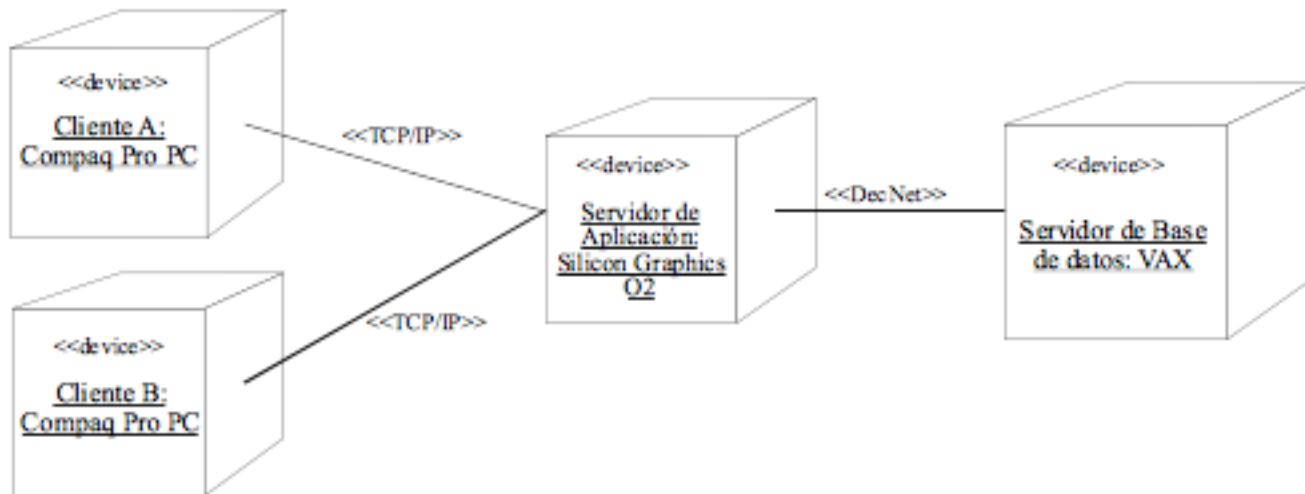
- Forman parte del **sistema ejecutable**
- Ejemplos: DLL, EXE, scripts, etc.

- Ejecución

- Se **crean durante la ejecución**
- Ejemplo: objetos de datos en tiempo de ejecución

Comunicación entre Nodos

- Los nodos se conectan entre sí mediante **caminos de comunicación**, que se dibujan con una línea que une los nodos.
- Las **conexiones** indican que hay alguna clase de comunicación entre los nodos y que los nodos intercambian mensajes a través de ese camino de comunicación.

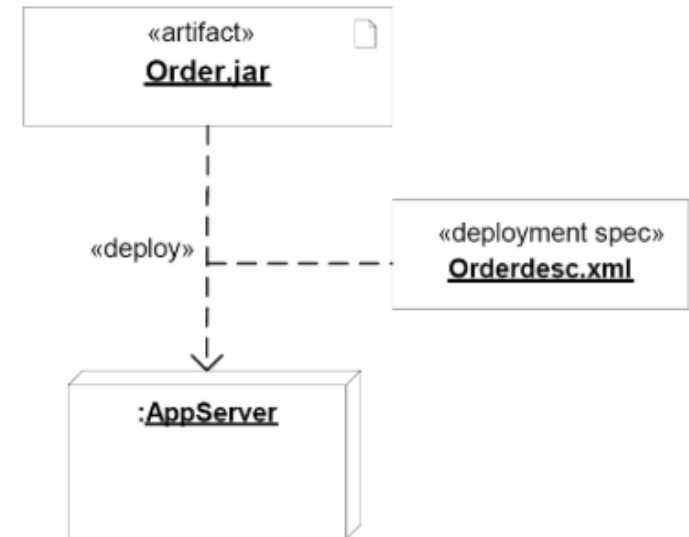
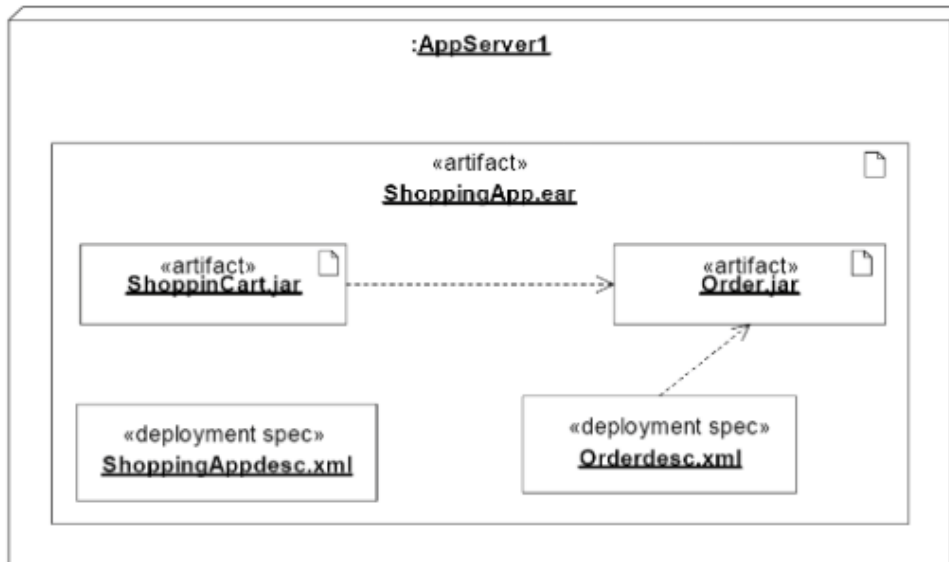


Especificación de Despliegue

- Propiedades que describen **cómo se despliega un artefacto en un nodo**, para que funcione adecuadamente en su entorno.
 - La relación entre un **artefacto** y su **especificación de despliegue** se muestra dibujando a ambos dentro del nodo y conectados con una flecha de dependencia desde la especificación hasta el artefacto.



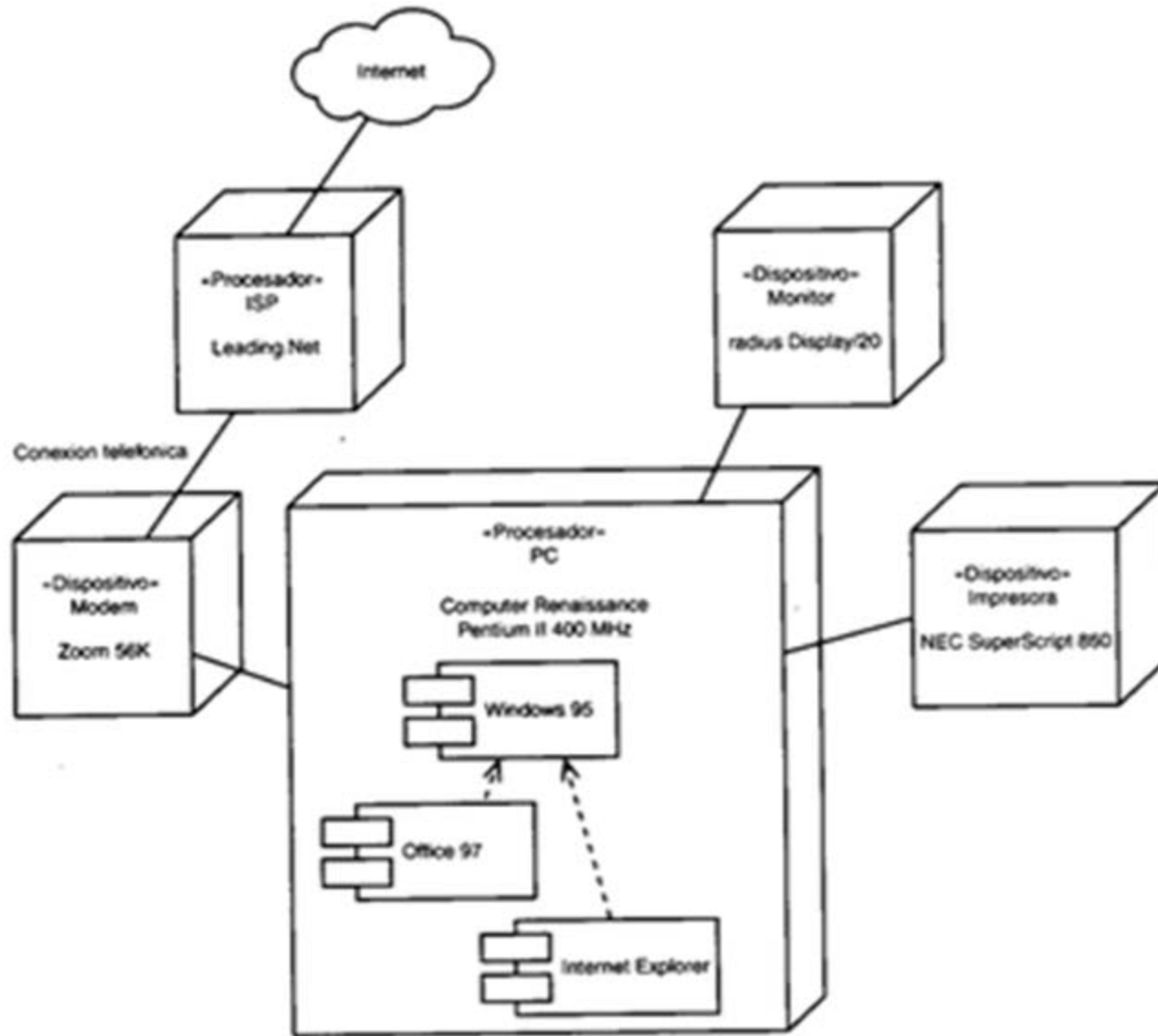
Especificación de Despliegue



Ejercicios Despliegue

Para modelar mi equipo he incluido al procesador y los dispositivos a la vez que he modelado mi conexión telefónica con mi proveedor de servicios de Internet y su conexión.





En una red token-ring los ordenadores equipados con una tarjeta de Interfaz (NIC) de red se conecta a una unidad central de acceso a multiestaciones (MSAU). Se conectan varias MSAU en una serie que puede parecer un anillo. El anillo se combina como un policia de transito mediante una señal conocida como token que permite a cada equipo de c'omputo saber cuando puede transmitir información.



