INGENIERÍA DEL SOFTWARE I
Tema 13

Arquitectura Física del Sistema
(en desarrollo OO)

Univ. Cantabria – Fac. de Ciencias
Francisco Ruiz

Objetivos del Tema

- Presentar los conceptos básicos de UML 2 para el modelado de las vistas de implementación y despliegue: **componentes, nodos y artefactos**.
- Aprender a usar los diagramas de componentes y de despliegue para tal fin, conociendo los principales usos de tales diagramas y conceptos.

Contenido

- Introducción
- Componentes
  - Organización
  - Puertos
  - Estructura Interna
  - Subsistemas
- Diagramas de Componentes
  - Consejos
- Artefactos
  - Tipos
  - Estereotipos
- Nodos
  - Organización
- Diagramas de Despliegue
  - Consejos
- Diagramas de Artefactos
  - Consejos
- Modelado
  - Clase Estructurada
  - API
  - Dispositivos Físicos
  - Distribución de Artefactos
  - Ejecutables y Bibliotecas
  - Versiones Ejecutables
  - Tablas, Archivos y Documentos
  - Base de Datos Física
  - Código Fuente
  - Sistema Embebido
  - Sistema Cliente/Servidor
  - Sistema Distribuido

Bibliografía

- Básica
  - Booch, Rumbaugh y Jacobson (2006): El Lenguaje Unificado de Modelado. 2ª edición.
    - Caps. 15, 26, 27, 30 y 31.
- Complementaria
  - Rumbaugh, Jacobson y Booch (2007): El Lenguaje Unificado de Modelado. Manual de Referencia. 2ª edición.
    - Caps. 5 y 10.
  - Miles y Hamilton (2006): Learning UML 2.0.
    - Caps. 12 y 15.

Introducción

- Modelado Arquitectónico …
  - Nos referimos a la arquitectura física del sistema
  - No a la **OTRA** arquitectura.
- Mundo Real →
  - En la construcción de un edificio, los planos son muy importantes … pero finalmente, lo más 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

Componentes

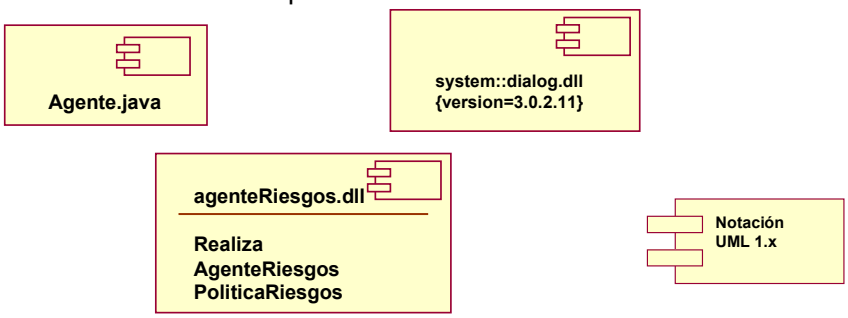
- Un **Componente** es una parte física reemplazable de un sistema que conforma y proporciona la implementación de un conjunto de **interfaces**.
  - Se utiliza para modelar elementos físicos que pueden hallarse en un **nodo**
    - ejecutables, bibliotecas (DLLs), tablas, archivos, documentos, ...
  - Es una parte modular de un sistema que encapsula el estado y comportamiento de un conjunto de clasificadores (p.e. clases).
  - Especifica un contrato de los servicios que proporciona y de los que requiere en términos de interfaces requeridas y proporcionadas.
  - Es una unidad reemplazable que se puede sustituir en tiempo de diseño o ejecución por otro componente que ofrezca la misma funcionalidad en base a la compatibilidad de sus interfaces.

Componentes

- **Propiedades** de un **Componente**.
- Tres Principales:
  - Es una parte de un sistema.
  - Es reemplazable.
  - Conforma y proporciona la realización de un conjunto de interfaces.
- Adicionales:
  - Es una unidad de despliegue independiente.
  - Puede ser conectado con otros componentes.
  - En una aplicación dada existirá una única copia.
  - Realiza una función bien definida (cohesión física y lógica).
  - Abarca más de una colaboración de clases.
  - Existe en el contexto de una arquitectura bien definida.
  - Presupone una infraestructura tecnológica que se piensa utilizar.

Componentes

- Componentes. **Notación**.
  - Gráficamente se representan como un rectángulo con un icono especial en la esquina superior derecha.
  - Normalmente se dibujan mostrando sólo su nombre.
  - Se pueden adornar con valores etiquetados o con compartimentos adicionales.
  - Pueden ser estereotipados.

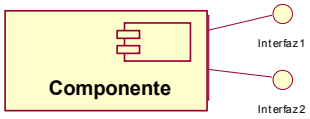


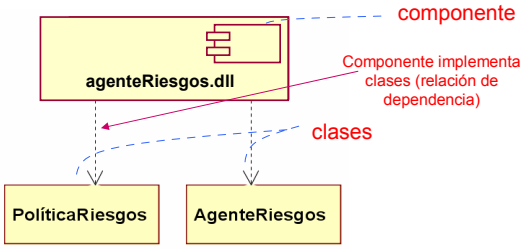
Componentes - Organización

- Los componentes se pueden agrupar en paquetes.
- También se pueden **organizar** mediante relaciones entre ellos de:
  - Dependencia, generalización, asociación (incluida agregación) y realización.
  - Un componente se puede construir a partir de otros componentes (agregación).

Componentes vs Clases

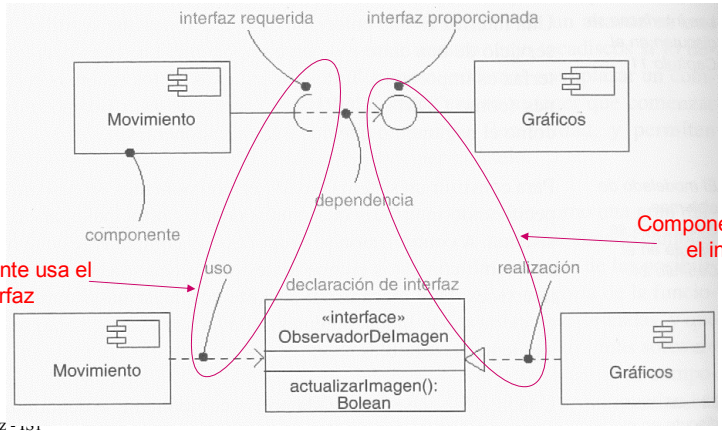
- Se parecen a las **clases** en que:
  - tienen nombres, realizan interfaces, pueden participar en relaciones,
- Pero se diferencian en que:
  - Las Clases
    - Son abstracciones lógicas
    - Tienen operaciones y atributos
  - Los Componentes
    - Son fragmentos físicos del sistema
    - Tienen interfaces





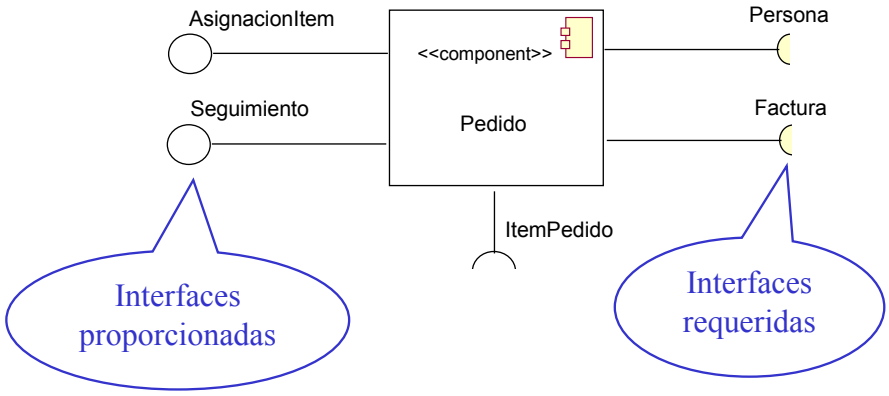
Componentes - Interfaz

- La relación entre componente e **interfaz** es importante.
  - Unos componentes implementan las interfaces y otros acceden a los servicios proporcionados por esas interfaces.
  - Estas relaciones se pueden mostrar en forma icónica o expandida:



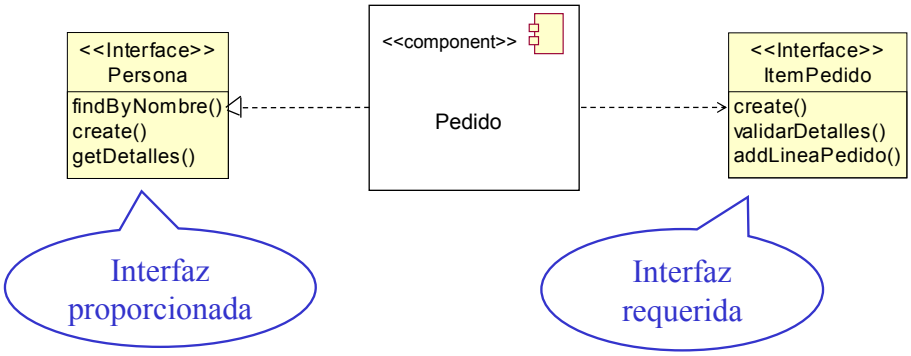
Componentes - Interfaz

- Ejemplo de interfaces requeridas y proporcionadas.
  - Notación icónica (piruleta).



Componentes - Interfaz

- Ejemplo de interfaces requeridas y proporcionadas.
  - Notación extendida.



Componentes

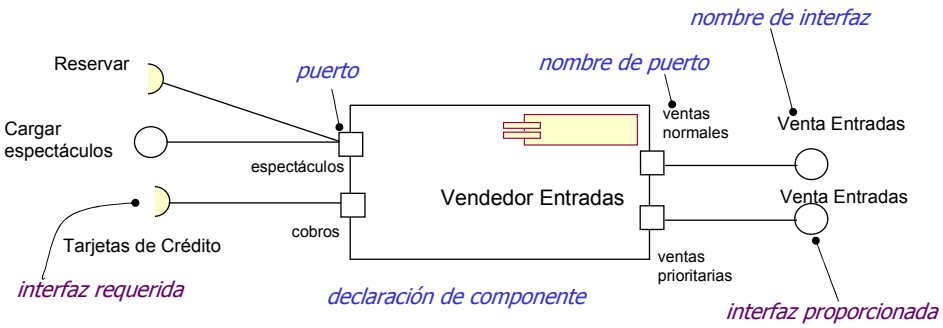
- **Conceptos** de Componentes en **UML 2**:
  - **Interfaz**. Colección de operaciones que especifican un servicio proporcionado o solicitado por una clase o componente.
  - **Puerto**. Una ventana específica de un componente encapsulado, que acepta mensajes hacia y desde el componente, que son conformes con las interfaces especificadas.
  - **Estructura Interna**. Implementación de un componente a través de un conjunto de partes conectadas de una manera específica.
  - **Parte**. Especificación de un rol que forma parte de la implementación de un componente.
  - **Conector**. Relación de comunicación entre dos partes o puertos dentro del contexto de un componente.

Componentes - Puertos

- Un **Puerto** es una ventana explícita dentro de un componente encapsulado.
  - En un componente encapsulado, todas las interacciones dentro y fuera pasan a través de sus puertos.
  - Representa un punto de interacción entre una instancia de un clasificador (clase, componente) con su entorno o con las instancias que contiene (estructura interna).
  - Cuando se crea una instancia de un componente, se crean instancias de sus puertos.
    - La instancia de un puerto es un objeto de una clase que implementa las interfaces proporcionadas.
  - Un puerto tiene
    - Identidad (nombre).
    - Multiplicidad (número posible de instancias de un puerto dentro de una instancia de componente). => Vector de Instancias del Puerto.

Componentes - Puertos

- Ejemplo. Puertos de un componente.

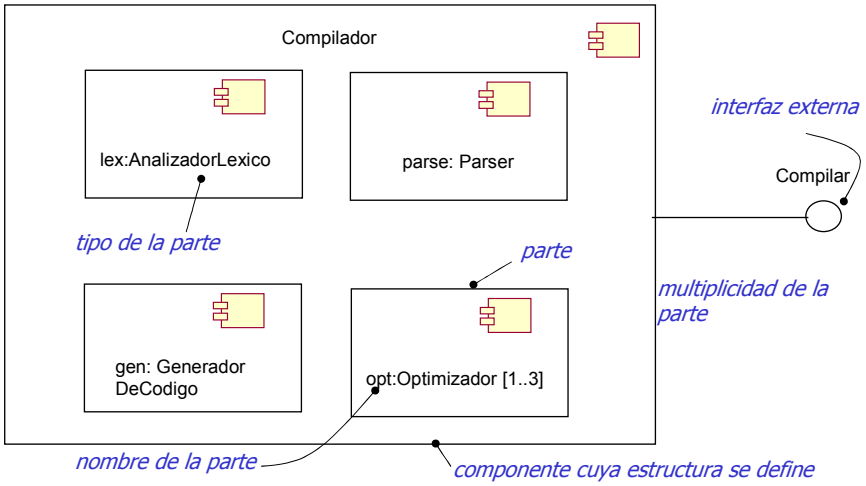


Componentes – Estructura Interna

- La **estructura interna** de un componente está formada por las **partes** que componen su implementación junto con las **conexiones** entre ellas.
  - Las partes pueden ser componentes conectados a través de sus puertos.
  - Una parte es una unidad de implementación de un componente, que tiene un nombre y un tipo.
  - Una instancia de un componente tiene una o más instancias de cada una de sus partes.
    - Las partes tienen multiplicidad.

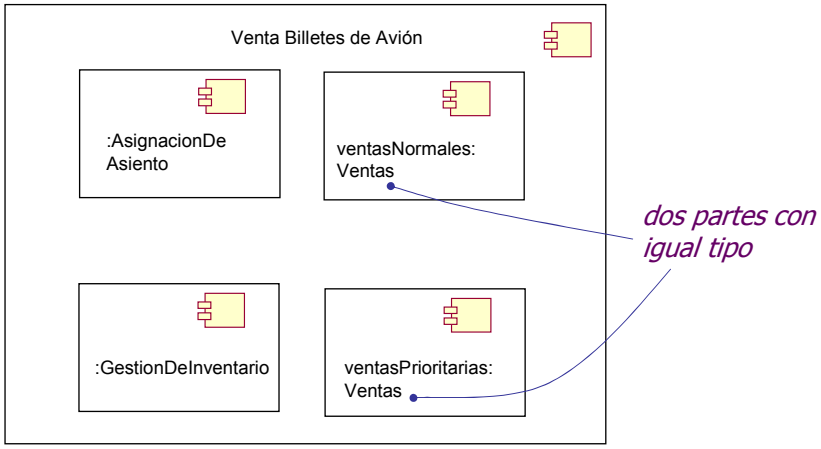
Componentes – Estructura Interna

- Ejemplo de Estructura Interna de un Compilador.



Componentes – Estructura Interna

- Las **partes** en un componente juegan un papel similar a los atributos de una clase.

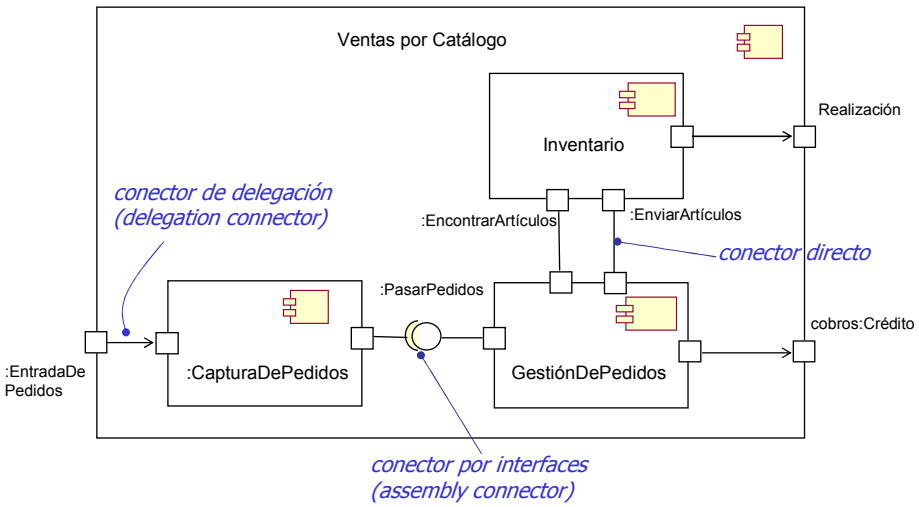


Componentes – Estructura Interna

- Una conexión entre dos puertos se denomina **conector** y denota un enlace en una instancia del componente.
  - Los componentes pueden ser conectados:
    - Directamente (mediante una línea entre ellos o sus puertos), o
    - Porque tienen interfaces compatibles (mediante junta circular).
  - Un **conector de delegación** (*delegate*) conecta un puerto interno a uno externo.
    - Se representa mediante una flecha desde el puerto interno al externo.
    - Actua como si el puerto interno fuese el externo, es decir, cualquier mensaje llegado al puerto externo es transmitido inmediatamente al puerto interno.

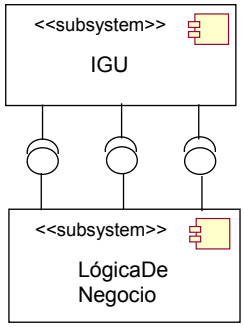
Componentes – Estructura Interna

- Ejemplo con **conectores**.



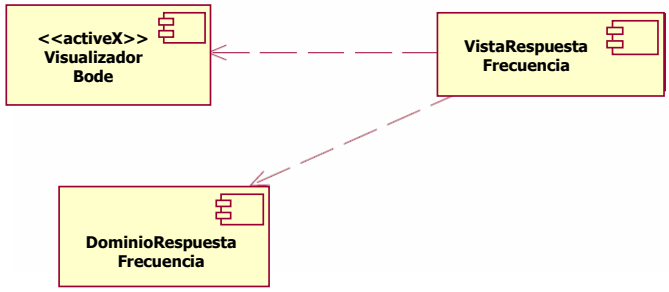
Componentes – Subsistemas

- Un **Subsistema** es una unidad de descomposición de un sistema.
  - En UML 2 se define como un componente con el estereotipo «subsystem».



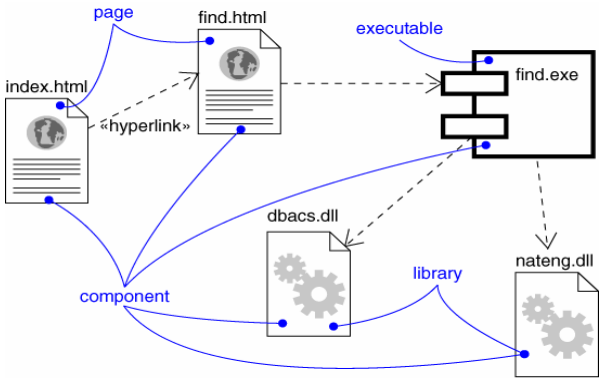
Diagramas de Componentes

- Muestran un conjunto de componentes y sus relaciones.
  - Describen los elementos físicos del sistema y sus relaciones.
- Contienen:
  - Componentes
  - Interfaces
  - Relaciones de dependencia, generalización, asociación y realización.



Diagramas de Componentes

- Capturan la estructura física de la implementación.
  - Las relaciones de dependencia se utilizan para indicar que un componente utiliza los servicios ofrecidos por otro componente.

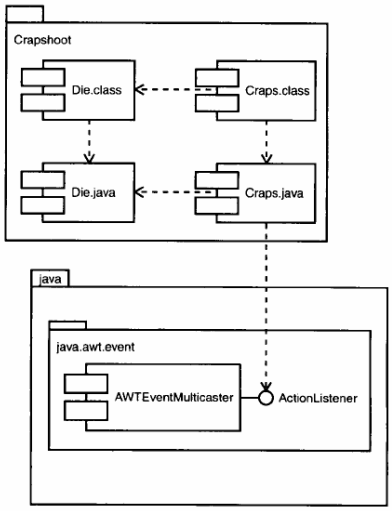


Diagramas de Componentes

Ejemplo. Página Web con un Subprograma Java

Se genera un applet que ejecuta el juego de dados "Craps" en una página Web, y emplea una clase "Die" para crear los dados.
La página Web se llama "Craps.html", el código fuente del applet se encuentra en el archivo "Craps.java" y el código objeto en el "Craps.class". El código fuente de la clase "Die" se encuentra en "Die.java" y el código objeto en "Die.class".Todos los archivos se encuentran en el mismo directorio

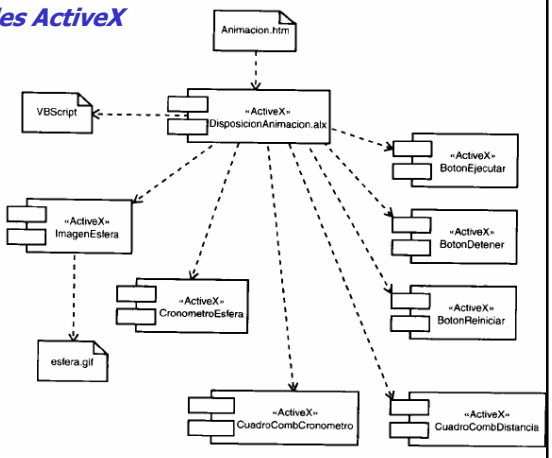
"Craps.html" depende de "Crap.class" y "Die.class", cada uno de estos archivos es un componente. "Craps.java" y "Die.java" importan "java.awt". "Craps.java" es un applet que hereda de la clase "java.appler.Applet". "Craps.java" importa a "java.awt.event" e implementa la interface "ActionListener" (para responder a eventos generados por el usuario como el clic de ratón); esta interfaz proporciona un botón para que se tiren los dados con un click (la clase "AWTEventMulticaster" implementa esta interfaz)



Diagramas de Componentes

Ejemplo. Página Web con controles ActiveX

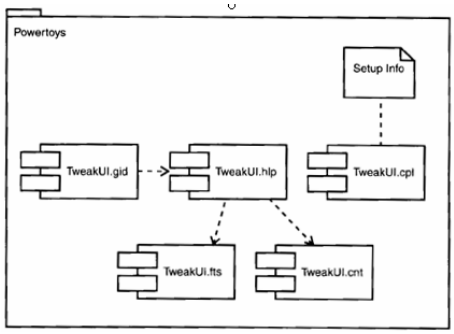
La página Web contiene un control "Timer ActiveX", dos cuadros combinados "ActiveX" y tres botones "ActiveX". La página Web permite al usuario establecer los parámetros para animar el movimiento de una esfera (imagen.gif) por pantalla. Un botón iniciará el movimiento, otro lo detendrá y el tercero restaurara la esfera a su posición inicial. El cronómetro moverá la esfera cuando pase la cantidad de milisegundos elegida por el usuario. Los controles "ActiveX" se encuentran en un componente separado conocido como "Disposición(Layout)". La página HTML y la disposición se encuentran en el mismo directorio



Diagramas de Componentes

Ejemplo. PowerToys

"PowerToys" es un paquete de Microsoft que permite eliminar las horribles flechas que se encuentran en la esquina inferior izquierda de cada icono de acceso directo en Win32 y hacer varias otras cosas con la GUI mediante la aplicación "TweakUI" que es parte del paquete.
La descompresión del paquete contiene varios archivos ".dll", así como uno de ayuda y otro ".CNT". Haciendo clic en el de ayuda se generará un archivo ".GID", utilizando la característica "Buscar" creará un archivo ".FTS".
En la figura se puede ver el diagrama de componentes que modela a "TweakUI" en el paquete "PowerToys", así como las dependencias entre sus componentes



Diagramas de Componentes - Consejos

- Un buen componente es aquel que:
  - Encapsula un servicio que tiene una interfaz y una frontera bien definidas.
  - Tiene suficiente estructura interna para que merezca la pena describirla.
  - No combina funcionalidades que no estén relacionadas en una única pieza.
  - Organiza su comportamiento externo utilizando unas cuantas interfaces y puertos.
  - Interactúa sólo a través de los puertos que ha declarado.

Diagramas de Componentes - Consejos

- Para mostrar la implementación de un componente utilizando subcomponentes anidados:
  - Utilizar un número pequeño de subcomponentes.
    - Si hay demasiados para caber en una página, es mejor utilizar niveles adicionales de descomposición en algunos subcomponentes.
  - Asegurarse de que los subcomponentes interactúan sólo a través de los puertos y conectores definidos.
  - Determinar los subcomponentes que interactúan directamente con el exterior modelando sus conectores de delegación.

Diagramas de Componentes - Consejos

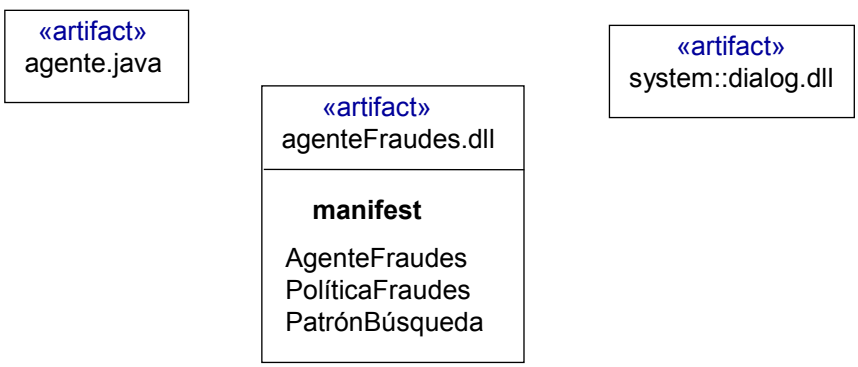
- Al **dibujar** un componente en UML 2:
  - Darle un nombre que indique claramente su propósito.
    - Igual debe hacerse con las interfaces.
  - Dar nombres a los subcomponentes y a los puertos si su significado no está claro a partir de sus tipos o si hay varias partes del mismo tipo.
  - Ocultar los detalles innecesarios.
    - No mostrar todos los detalles de implementación.
  - No pretender mostrar la dinámica.
    - Para ello se deben emplear diagramas de interacción.

Artefactos

- Un **artefacto** es una **parte física** de un sistema que existe a nivel de la plataforma de implementación.
- Es una implementación física de un conjunto de elementos lógicos tales como clases y componentes:
  - Relación de dependencia estereotipada con «manifest».
- Representan el empaquetamiento físico de bits sobre la plataforma de implementación.
- Residen en **nodos**.

Artefactos

- Notación.
  - Se representa con el estereotipo «artifact».
  - El nombre puede ser simple o cualificado.
  - Se pueden adornar con valores etiquetados o compartimentos adicionales.

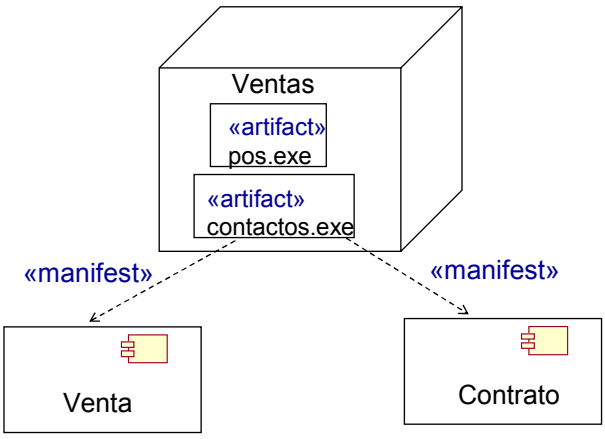


Artefactos

- Aunque clases y artefactos son clasificadores, hay diferencias significativas entre ellos:
  - Las clases representan abstracciones lógicas mientras que los artefactos representan elementos físicos formados por bits.
    - =>Los artefactos pueden estar en nodos; las clases no.
  - Los clases pueden tener atributos y operaciones; los artefactos no.
    - Un artefacto es la implementación física de un conjunto de elementos lógicos (clases, …).
    - La relación entre un artefacto y las clases que implementa se puede representar mediante una relación de manifestación («manifest»).

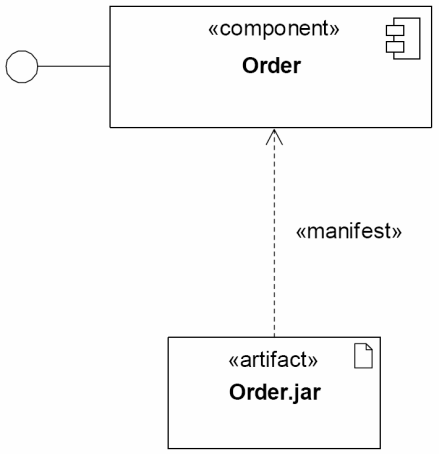
Artefactos

- Ejemplo. Un nodo con artefactos y las clases que implementan (manifiestan).



Artefactos

- Un artefacto puede ser una manifestación (implementación) de un componente.



Artefactos - Tipos

- Despliegue
  - Necesarios y suficientes para formar un sistema ejecutable.
    - DLL, EXE, .NET, CORBA, EJB, scripts, ..
- Producto del trabajo
  - Permanecen al final del proceso de desarrollo.
    - Archivos de código fuente y ficheros de datos a partir de los cuales se crean los artefactos de despliegue.
    - No participan directamente en un sistema ejecutable.
- Ejecución
  - Se crean durante la ejecución.
    - Objeto .NET instanciado a partir de una DLL.

Artefactos - Estereotipos

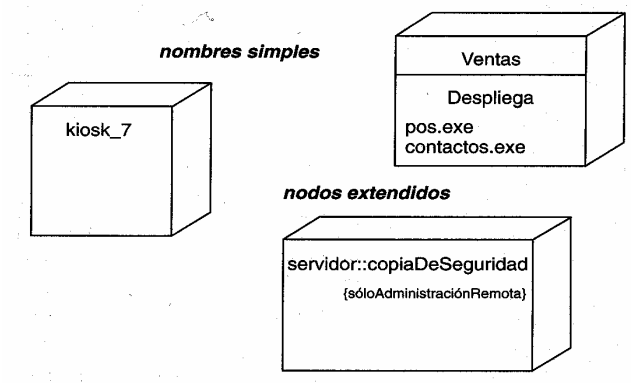
- Todos los mecanismos de extensibilidad de UML se pueden aplicar a los artefactos.
- UML 2 ofrece varios estereotipos predefinidos para artefactos:
  - Document
    - Fichero genérico que no es código fuente o ejecutable. Subclase de File.
  - Executable
    - Artefacto que se puede ejecutar en un nodo. Subclase de File.
  - File
    - Archivo físico en el contexto del sistema desarrollado.
  - Library
    - Fichero de una biblioteca de objetos estática o dinámica. Subclase de File.
  - Source
    - Fichero de código fuente.

Nodos

- Un **nodo** es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional con memoria y capacidad de procesamiento.
  - Se utilizan para modelar la topología del hardware sobre el que se ejecuta el sistema.
  - Representa típicamente un procesador o un dispositivo sobre el que se pueden desplegar componentes.
  - Se pueden estereotipar y se pueden agrupar en paquetes.
  - Se parecen a las clases en que pueden tener atributos (*velocidadProcesador*) y operaciones (*encender*).

Nodos

- Notación.
  - Un **nodo** se representa como un cubo.

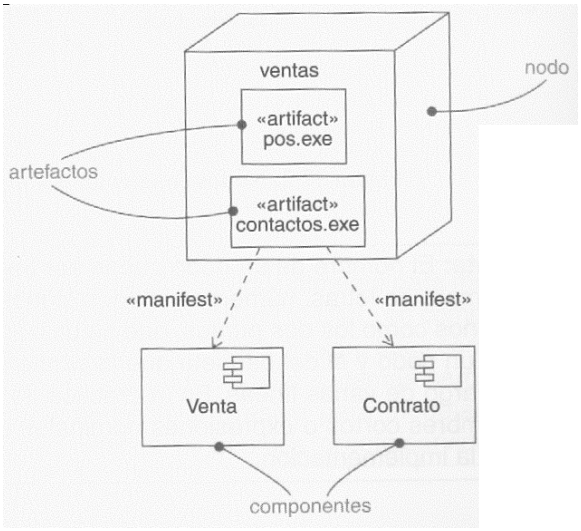


Nodos

- **Nodos vs Artefactos**
  - Parecido:
    - Tienen nombres, pueden participar en relaciones de dependencia, generalización y asociación, pueden anidarse, pueden tener instancias, pueden participar de interacciones.
  - Diferencias:
    - Los artefactos son los elementos que participan en la ejecución de un sistema;
    - los nodos son los elementos donde se ejecutan los artefactos.
    - Los artefactos representan el empaquetamiento físico de los elementos lógicos;
    - los nodos representan el despliegue físico de los componentes.

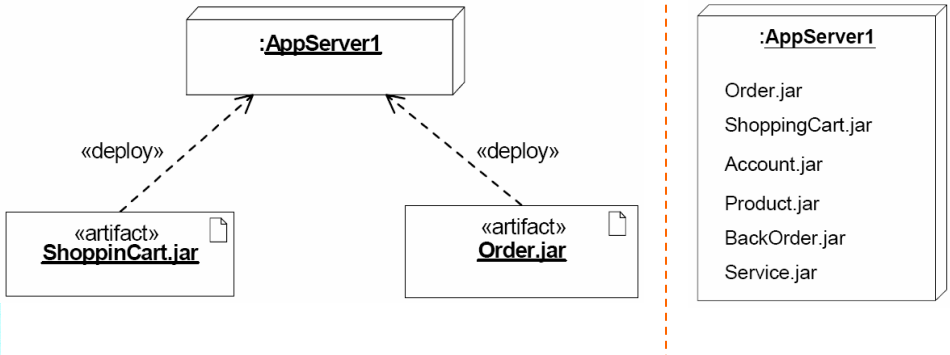
Nodos

- **Nodos vs Artefactos**



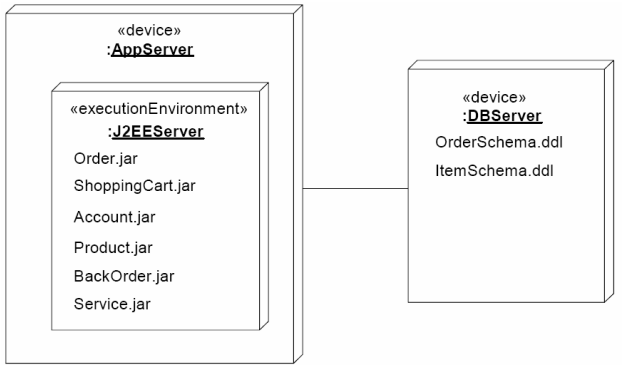
Nodos

- Otras maneras de mostrar los artefactos desplegados en un nodo.



Nodos

- Hay dos estereotipos predefinidos de Nodo.
  - **Unidad** («device»). Recurso computacional físico sobre el cual pueden ser desplegados artefactos para su ejecución.
  - **Entorno de Ejecución** («executionEnvironment»). Nodo que ofrece un entorno para ejecutar un tipo específico de artefactos ejecutables.

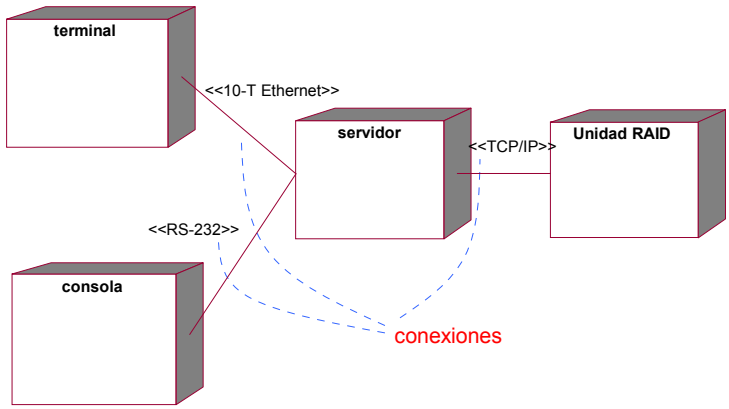


Nodos - Organización

- Un conjunto de objetos o componentes asignados a un nodo como un grupo se denomina unidad de distribución.
- Los nodos se pueden **organizar**:
  - Agrupándolos en paquetes.
  - Especificando relaciones de dependencia, generalización y asociación (incluyendo agregación) entre ellos.
  - Una asociación entre nodos representa una **conexión** física entre nodos (relación más frecuente).
    - Puede incluir roles, multiplicidad y restricciones.

Nodos - Organización

- Conexiones entre Nodos.
  - Se pueden usar estereotipos para indicar el tipo de conexión física.



Diagramas de Despliegue

- El **despliegue** es el proceso de asignar artefactos a nodos o instancias de artefactos a instancias de nodos
- Los diagramas de despliegue muestran:
  - El hardware sobre el que se ejecutará el sistema y cómo el software se despliega en ese hardware.
  - La configuración de los nodos que participan en la ejecución y de los artefactos que residen en los nodos.
- En UML 2 se utilizan para visualizar los aspectos estáticos de los nodos físicos y sus relaciones y para especificar sus detalles para la construcción.

Diagramas de Despliegue

- No son necesarios cuando
  - Se desarrolla un software que reside en **una** máquina e interactúa sólo con dispositivos estándar en esa máquina, que ya son gestionados por el SO (teclado, pantalla, etc..).
- Son necesarios cuando
  - Se desarrolla un software que interactúa con dispositivos que normalmente no gestiona el SO, o
  - 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.

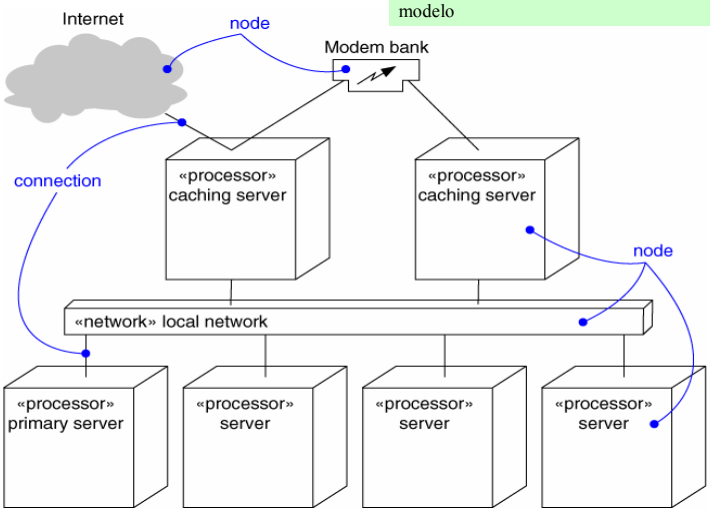
Diagramas de Despliegue

- Contenido:
  - Nodos.
  - Relaciones de dependencia y asociación entre nodos.
  - Opcionalmente, también pueden contener:
  - Notas y restricciones.
  - Artefactos, cada uno dentro de un nodo.
  - Paquetes y subsistemas, utilizados para agrupar en bloques más grandes.
  - Instancias (para visualizar un caso concreto de una familia de topologías hardware).

Diagramas de Despliegue

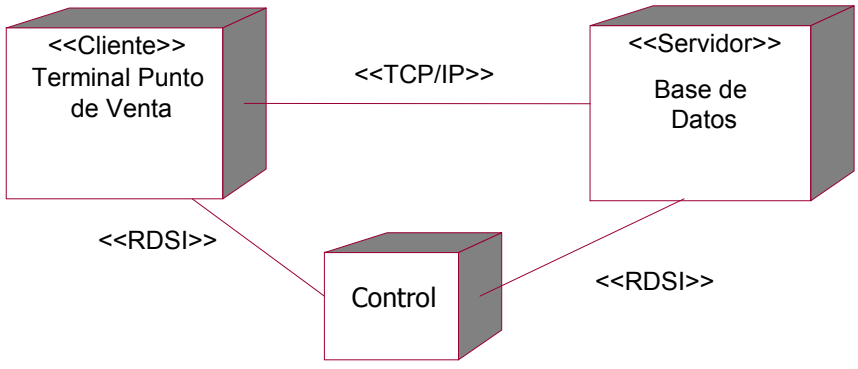
- Notación.

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



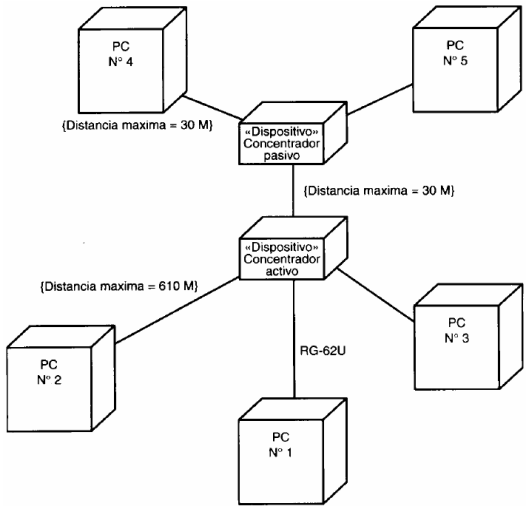
Diagramas de Despliegue

- Notación. Uso de estereotipos para distinguir nodos y conexiones.



Diagramas de Despliegue: Ejemplo

Ejemplo. Red ARCnet.



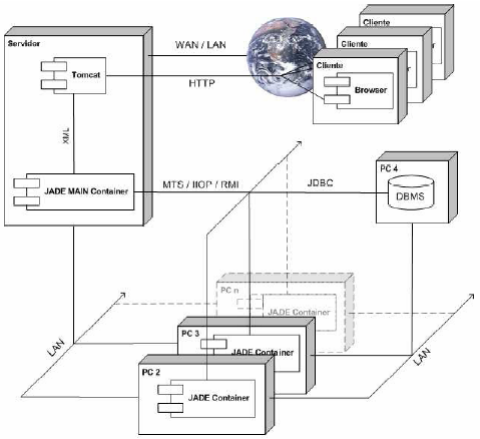
La red ARCnet (Red de Cómputo de Recursos Adjuntos) implica pasar un token o señal de un equipo a otro. La diferencia es que en esta red cada equipo tiene asignado un número. El orden numérico determina el equipo que obtendrá el token. Cada equipo se conecta a un concentrador o hub que podrá ser activo (amplifica la información que llega antes de transmitirla) o pasivo (trasmite la información sin amplificarla). Los concentradores ARCnet no mueven el token en anillo.

Diagramas de Despliegue: Ejemplo

Ejemplo. Sistema Tutorial Inteligente.

Se desarrolla mediante el paradigma de sistemas multi-agentes, utilizando la plataforma JADE (Java Agent Develpment Framework), dicha plataforma será distribuida en una red LAN de la siguiente forma:

- En un servidor principal se alojara un servidor Web Tomcat y se instanciará el contenedor principal, en este se ejecutaran los agentes provistos por JADE, como son: AMS, DF, y RMA, además de algunos agentes del sistema, como son: Planificador, Evaluador, (STI) Organizador,Supervisor (ACAC), ya que estos agentes tienen actividad constante.
- Por otro lado se tendrán contenedores adicionales distribuidos en diferentes computadores en la misma red LAN en los que se repartirán los demás agentes, como son: Recuperador, Filtro (responsables del la búsqueda de OA en la Web), Estudiante (que representa al usuario estudiante conectado), Profesor (que representa el usuario profesor conectado).
- Por último se tiene un computador en que se aloja el sistema manejado de base de datos



Diagramas de Despliegue - Consejos

- Un **nodo** está bien estructurado si:
  - Proporciona una abstracción bien definida de algo extraído del vocabulario del hardware empleado en la solución.
  - Se descompone sólo hasta el nivel necesario para comunicar la intención al lector.
  - Sólo muestra aquellos atributos y operaciones relevantes para el dominio modelado.
  - Despliega directamente un conjunto de artefactos que residen en el nodo.
  - Está conectado con otros nodos de forma que se refleja la topología de un sistema del mundo real.

Diagramas de Despliegue - Consejos

- Al dibujar un **nodo** en UML 2:
  - Definir un conjunto de estereotipos e iconos apropiados, a nivel de proyecto u organización.
    - Deben proporcionar mensajes visuales claros para el lector.
  - Mostrar, si los hay, sólo los atributos y operaciones necesarios para comprender el significado del nodo en el contexto dado.

Diagramas de Despliegue - Consejos

- Un **diagrama de despliegue** está bien estructurado si:
  - Modela un aspecto de la vida de despliegue estática de un sistema.
    - Un único diagrama de despliegue no necesita capturarlo todo sobre la vista de despliegue del sistema.
    - Los aspectos dinámicos se modelan con diagramas de comportamiento.
  - Contiene sólo aquellos elementos esenciales para comprender el aspecto modelado.
  - Proporciona detalles de forma consistente con el nivel de abstracción, mostrando sólo los adornos esenciales para su comprensión.
  - No olvida alguna información importante.

Diagramas de Despliegue - Consejos

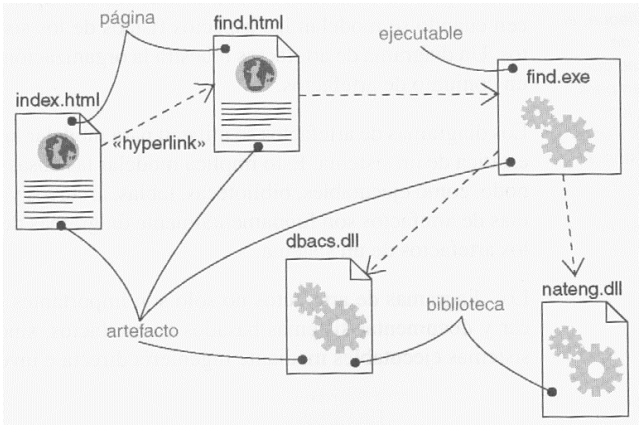
- Al dibujar un **diagrama de despliegue**:
  - Darle un nombre que comunique su propósito.
  - Distribuir sus elementos minimizando los cruces de líneas.
  - Organizar los elementos espacialmente para que los que estén cercanos semánticamente también lo estén en el diagrama.
  - Usar notas y colores como señales visuales para llamar la atención sobre características importantes.
  - Usar con cuidado los elementos estereotipados.
    - Elegir un pequeño conjunto de iconos del proyecto u organización y usarlos de forma consistente.

Diagramas de Artefactos

- Son un tipo especial de diagramas de despliegue.
  - Muestran un conjunto de **artefactos y sus relaciones**, pero sin especificar los nodos.
- Contienen
  - Artefactos
  - Relaciones de dependencia, generalización, asociación y realización.
  - Notas y restricciones (opcionalmente).
- Se emplean para modelar la vista de implementación estática de un sistema y, en particular:
  - Código fuente,
  - Versiones ejecutables,
  - Bases de datos físicas, y
  - Sistemas adaptables.

Diagramas de Artefactos

- Los diagramas de artefactos proveen detalles para la construcción.



Diagramas de Artefactos - Consejos

- Al modelar artefactos en UML 2:
  - Recordar que se está modelando en la dimensión física.
- Un **artefacto** está bien estructurado si:
  - Implementa directamente un conjunto de clases que colaboran entre sí para llevar a cabo la semántica de las interfaces correspondientes.
  - Está débilmente acoplado en relación con otros artefactos.

Diagramas de Artefactos - Consejos

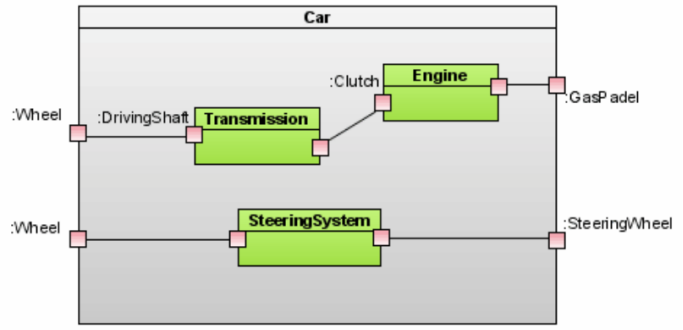
- Un **diagrama de artefactos** está bien estructurado si:
  - Comunica **un** aspecto de la vista de implementación estática del sistema.
  - Contiene sólo aquellos elementos esenciales para comprender dicho aspecto.
  - Proporciona detalles de forma consistente con el nivel de abstracción, mostrando sólo los adornos necesarios para su comprensión.
  - No olvida alguna información relevante.
- Al dibujar un **diagrama de artefactos**:
  - Seguir los mismos consejos que para un diagrama de despliegue.

MODELADO

- Los Componentes, Nodos y Artefactos, y sus diagramas UML 2 (de componentes, de despliegue, de artefactos) se pueden emplear para modelar:
  - Una Clase Estructurada
  - Una API (Interfaz de proggramación de aplicaciones)
  - Dispositivos Físicos
  - Distribución de Artefactos
  - Ejecutables y Bibliotecas
  - Tablas, Archivos y Documentos
  - Código Fuente
  - Un Sistema Embebido
  - Un Sistema Cliente/Servidor
  - Un Sistema Distribuido

Modelado – Clase Estructurada

- Un clasificador (clase, componente) con estructura interna puede modelarse con partes, conectores y puertos.
  - En UML 2 se han incorporado para este fin los nuevos diagramas de estructura compuesta (figura).

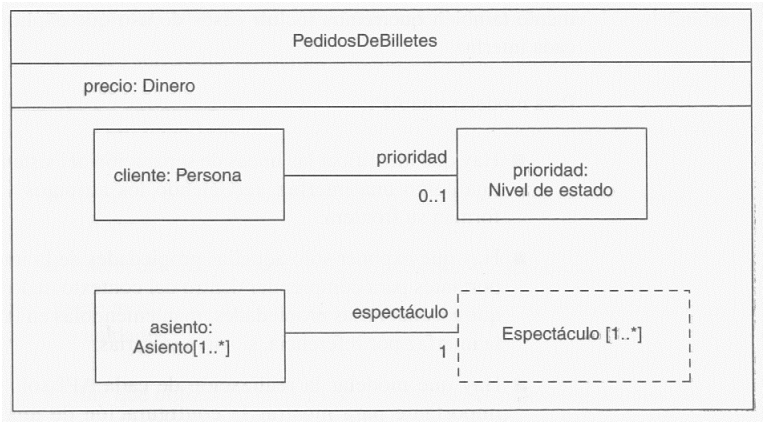


Modelado – Clase Estructurada

- Para modelar una **clase estructurada**:
  - Identificar las partes internas y sus tipos.
  - Dar a cada parte un nombre que indique su propósito dentro de la clase, no su tipo genérico.
  - Dibujar conectores entre las partes que se comunican o que tienen relaciones contextuales.
  - Si hace falta, usar otras clases estructuradas como tipos, pero recordando que las conexiones a las partes dentro de una clase no pueden ser directas, sino a través de sus puertos.

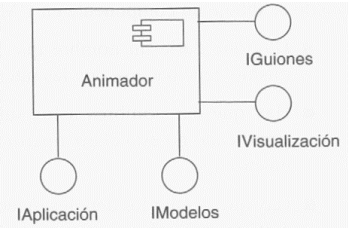
Modelado – Clase Estructurada

- Ejemplo.



Modelado – API

- Para modelar una **API**:
  1. Identificar las líneas de separación del sistema y modelar cada una como una interfaz, recogiendo los atributos y operaciones que forman su frontera.
  2. Exponer sólo aquellas propiedades de la interfaz que son importantes para comprender el contexto dado.
     - En otro caso, esconder las propiedades.
  3. Modelar la realización de la API sólo cuando sea importante para mostrar la configuración de una implementación específica.

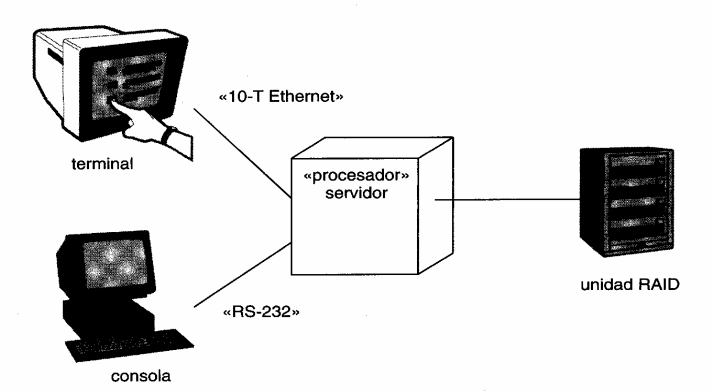


Modelado – Dispositivos Físicos

- La mayoría de las veces los **nodos** se utilizan para el modelado de los **procesadores** y los **dispositivos** que conforman la topología de los sistemas.
- Para modelar dispositivos físicos:
  - Identificar los elementos computacionales de la vista de despliegue del sistema y modelar cada uno como un nodo.
  - Si estos elementos representan procesadores y dispositivos genéricos, hay que estereotiparlos con los estereotipos estándar.
  - Considerar los atributos y operaciones aplicables a cada uno.

Modelado – Dispositivos Físicos

- Ejemplo. Topología de Dispositivos Físicos.

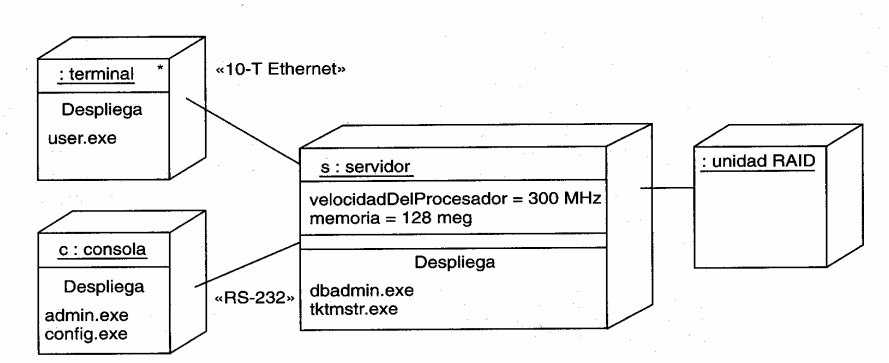


Modelado – Distribución de Artefactos

- Para modelar la distribución de artefactos entre los dispositivos físicos del sistema:
  1. Ubicar cada artefacto significativo del sistema en un determinado nodo.
  2. Tener en cuenta la duplicación de localizaciones.
     1. El mismo tipo de artefactos puede residir simultáneamente en varios nodos.
  3. Representar cada localización de una de las tres formas posibles:
     a) No haciéndola visible (pero dejándola en la especificación de cada nodo).
     b) Conectando cada nodo con el artefacto que despliega mediante relaciones de dependencia.
     c) Listando los componentes de cada nodo en un compartimiento adicional.

Modelado – Distribución de Artefactos

- Ejemplo. Distribución de los artefactos entre nodos.

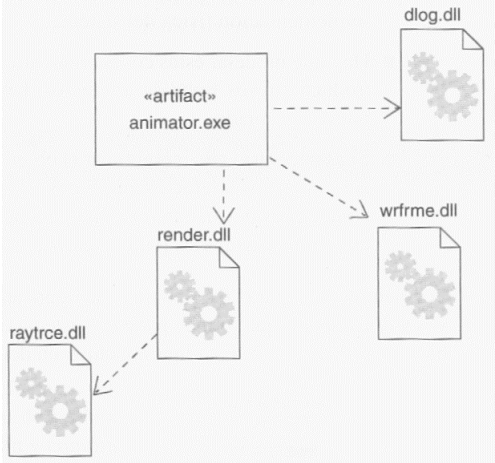


Modelado – Ejecutables y Bibliotecas

- Para modelar ejecutables y bibliotecas mediante artefactos:
  1. Identificar la partición del sistema físico.
  2. Modelar como artefactos cualquier ejecutable y biblioteca, utilizando los elementos estándar apropiados.
     - Incorporar un estereotipo nuevo para los nuevos tipos de artefactos.
  3. Si es importante manejar las líneas de separación del sistema, modelar las interfaces importantes que algunos artefactos utilizan y otros realizan.
  4. Si es necesario para comunicar el objetivo, modelar las relaciones entre los ejecutables, bibliotecas e interfaces.
     - Habitualmente se representan las dependencias.

Modelado – Ejecutables y Bibliotecas

- Ejemplo. Ejecutables y bibliotecas.

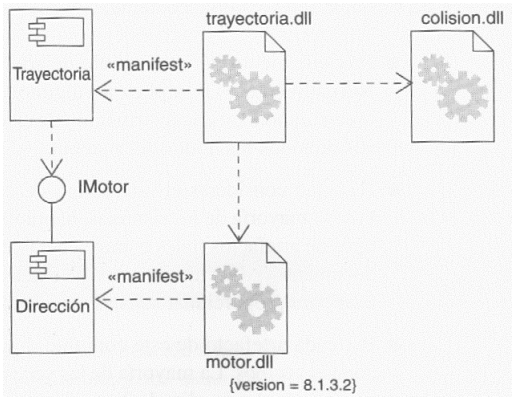


Modelado – Versiones Ejecutables

- Los diagramas de artefactos sirven también para modelar las versiones ejecutables, incluyendo los artefactos de despliegue que forman cada versión y las relaciones entre dichos artefactos.
- Para modelar una versión ejecutable:
  1. Identificar el conjunto de artefactos de despliegue que forman la versión, y el nodo o nodos donde están desplegados.
  2. Establecer el estereotipo de cada artefacto del conjunto.
  3. Considerar las relaciones entre los artefactos.
     - Interfaces exportadas (realizadas) e importadas (utilizadas).
     - Modelar explícitamente dichas interfaces si interesa representar las líneas de separación del sistema.

Modelado – Versiones Ejecutables

- Ejemplo. Versión ejecutable del software de control de un robot móvil.

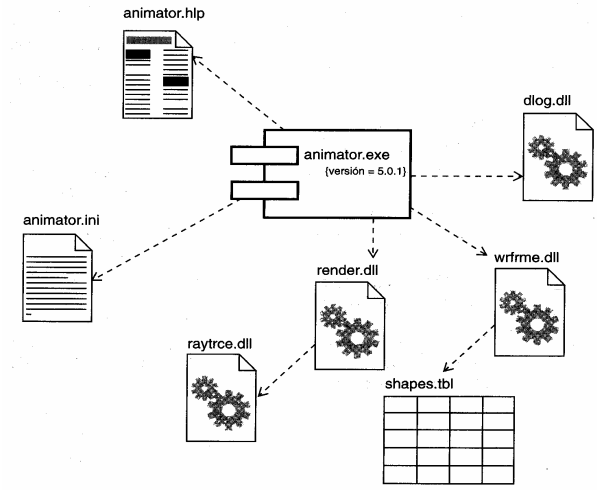


Modelado – Tablas, Archivos y Documentos

- A veces existen archivos importantes que no son ejecutables ni bibliotecas: ficheros de datos, documentos de ayuda, scripts, logs, etc.
- Para modelar este tipo de tablas, archivos y documentos:
  1. Identificar los artefactos auxiliares que forman parte de la implementación física del sistema.
  2. Modelar estas "cosas" como artefactos.
  3. Si es necesario para comunicar el objetivo, modelar las relaciones entre dichos artefactos auxiliares y los demás ejecutables, bibliotecas e interfaces del sistema.
     - Normalmente se representan las dependencias (para control de cambios).

Modelado – Tablas, Archivos y Documentos

- Ejemplo. Modelado de los artefactos auxiliares (tablas, archivos no ejecutables y documentos).

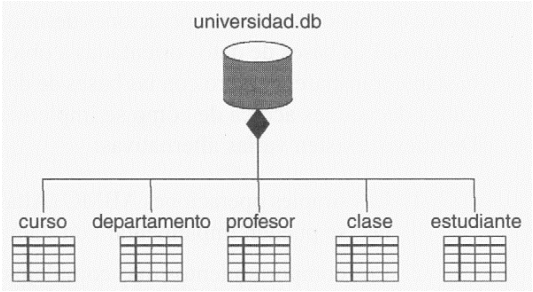


Modelado – Base de Datos Física

- Los diagramas de artefactos también permiten representar una base de datos física, indicando los contenedores de datos (tablas normalmente) existentes.
- Para modelar una base de datos física:
  1. Identificar las clases del modelo que representan el esquema lógico de la BD.
  2. Elegir un patrón de transformación para asociar dichas clases con tablas.
  3. Establecer la distribución física de la base de datos.
  4. Crear un diagrama de artefactos con artefactos estereotipados como tablas.
     - Cada artefacto puede indicar como atributos las columnas de la tabla.
     - Las operaciones pueden utilizarse para denotar procedimientos almacenados.

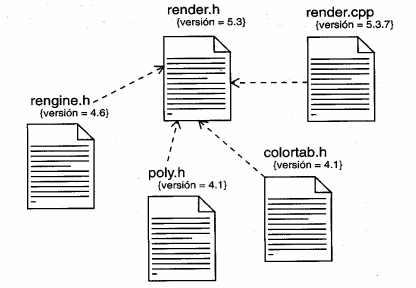
Modelado – Base de Datos Física

- Ejemplo. Modelado de una base de datos física (tablas relacionales).



Modelado – Código Fuente

- El modelado gráfico del código fuente es útil para:
  - Visualizar las dependencias de compilación, y
  - Gestionar la división y combinación de grupos de archivos de código fuente.
  - Visualizar las relaciones y propiedades manejadas por las herramientas de Gestión de Configuración.



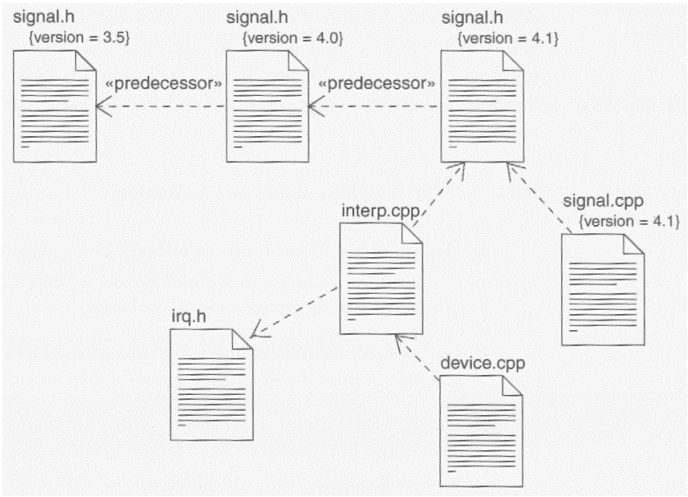
Modelado – Código Fuente

- Para modelar el código fuente de un sistema:
  1. Identificar (por ingeniería inversa o directa), el conjunto de archivos de código fuente de interés.
  2. Modelar dichos archivos como artefactos estereotipados.
  3. En sistemas grandes, usar paquetes para mostrar los grupos de archivos.
  4. Considerar el mostrar un valor etiquetado para indicar la versión, autor y fecha.
  5. Modelar las dependencias de compilación entre dichos archivos.

  - CONSEJO: Usar herramientas de SCM (*Software Configuration Management*) para los dos últimos puntos.

Modelado – Código Fuente

- Ejemplo. Modelo de código fuente con dependencias de compilación.

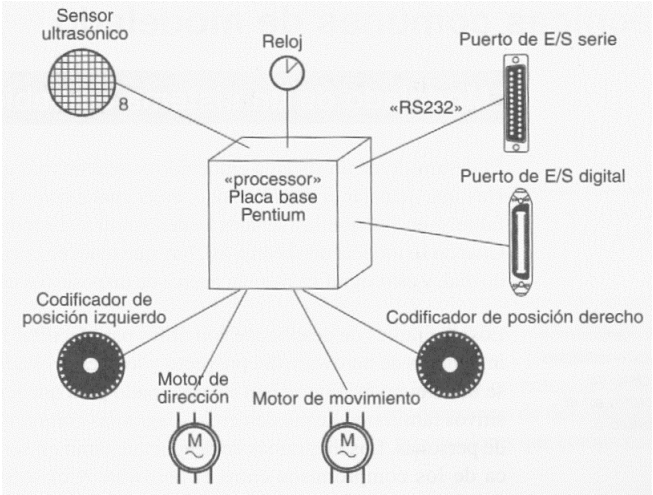


Modelado – Sistema Embebido

- Al desarrollar **sistemas embebidos** los diagramas de despliegue facilitan la comprensión entre los ingenieros hardware y los ingenieros software.
- Para modelar un sistema embebido:
  1. Identificar los dispositivos y nodos propios del sistema.
  2. Proporcionar señales visuales mediante los mecanismos de extensibilidad de UML para los dispositivos especiales.
     - Elegir estereotipos e iconos adecuados.
     - Distinguir entre procesadores (que contienen artefactos software) y los dispositivos (no tienen software de nuestro sistema).
  3. Modelar las relaciones entre procesadores y dispositivos mediante un diagrama de despliegue.
     - Igualmente especificar las relaciones entre los artefactos y los nodos.
  4. En los dispositivos en que sea necesario, modelar su estructura con un diagrama de despliegue adicional más pormenorizado.

Modelado – Sistema Embebido

- Ejemplo. Modelado de un sistema embebido.

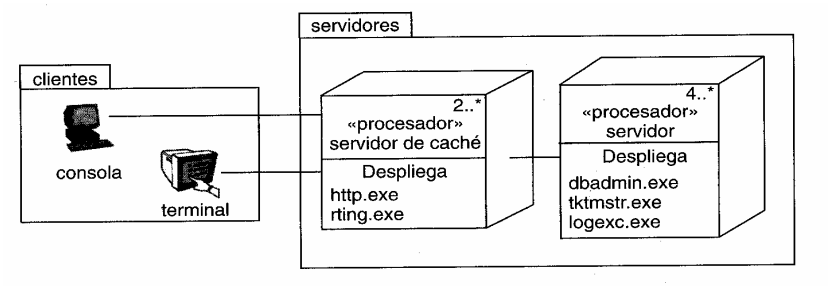


Modelado – Sistema Cliente/Servidor

- Los diagramas de despliegue de UML 2 se pueden emplear para visualizar, especificar y documentar las decisiones sobre la topología de un sistema cliente/servidor.
  - Para ello, es preferible emplear varios diagramas:
    - Uno general para el sistema.
    - Otros más detallados para las partes del sistema.
- Para modelar un sistema cliente/servidor:
  1. Identificar los nodos que representan los procesadores cliente y servidor.
  2. Destacar los dispositivos relacionados con el comportamiento del sistema.
     - Ej: lectores de input, visualizadores de información, ..
  3. Incluir señales visuales para los citados procesadores y dispositivos mediante estereotipos.
  4. Modelar la topología de los nodos en un diagrama de despliegue.
     - Incluir las relaciones entre artefactos y nodos.

Modelado – Sistema Cliente/Servidor

- Ejemplo. Modelado de un sistema cliente/servidor para gestión de recursos humanos.



Modelado – Sistema Distribuido

- Los diagramas de despliegue de UML 2 son útiles para razonar acerca de la topología de un sistema distribuido.
- Para modelar un sistema distribuido:
  1. Identificar los dispositivos y procesadores del sistema.
  2. Modelar los dispositivos de comunicaciones al nivel de detalle necesario.
     - Para razonar sobre el rendimiento de la red o el impacto en el sistema de cambios en ella.
  3. Especificar mediante paquetes las agrupaciones lógicas de nodos.
  4. Modelar los dispositivos y procesadores mediante uno o varios diagramas de despliegue.
  5. Si es necesario modelar dinámica del sistema, usar diagramas de casos de uso para los tipos de comportamientos del sistema y, después, extender dichos casos de uso mediante diagramas de interacción.

Modelado – Sistema Distribuido

- Ejemplo. Modelado de un sistema distribuido.