



# INGENIERÍA DEL SOFTWARE I

## Práctica 7, Sesión 1

### *Modelado de Implementación*

*Univ. Cantabria – Fac. de Ciencias*

*María Sierra*



## Introducción a la Implementación

- **Implementamos el sistema** en términos de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares. Se desarrolla la arquitectura y el sistema como un todo



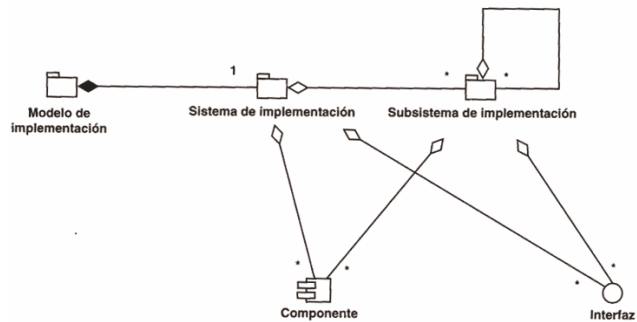
- **Propósitos:**

- Planificar las integraciones del sistema necesarias siguiendo un enfoque incremental
- Distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue
- Implementar las clases y subsistemas encontrados en diseño (las clases como componentes de fichero que contienen código fuente)
- Probar los componentes individualmente, integrarlos compilándolos y enlazándolos en uno o más ejecutables, antes de enviarlos para ser integrados y llevar a cabo las comprobaciones del sistema



# Modelo de Implementación

- Describe **como** los **elementos** del **modelo de diseño** (como las clases) **se implementan** en términos de componentes, como ficheros de código fuente, ejecutables, etc...
- Describe **cómo** se **organizan** los **componentes** de acuerdo con los **mecanismos de estructuración y modularización** disponibles en el **entorno de implementación** y en el **lenguaje o lenguajes de programación** utilizados, y **cómo dependen** los componentes unos de otros
- Se representa con un sistema de implementación que denota el subsistema de nivel superior del modelo. El utilizar otros subsistemas es una forma de organizar el modelo de implementación en trozos más manejables
- Define una **Jerarquía de subsistemas** de implementación que contiene componentes e interfaces



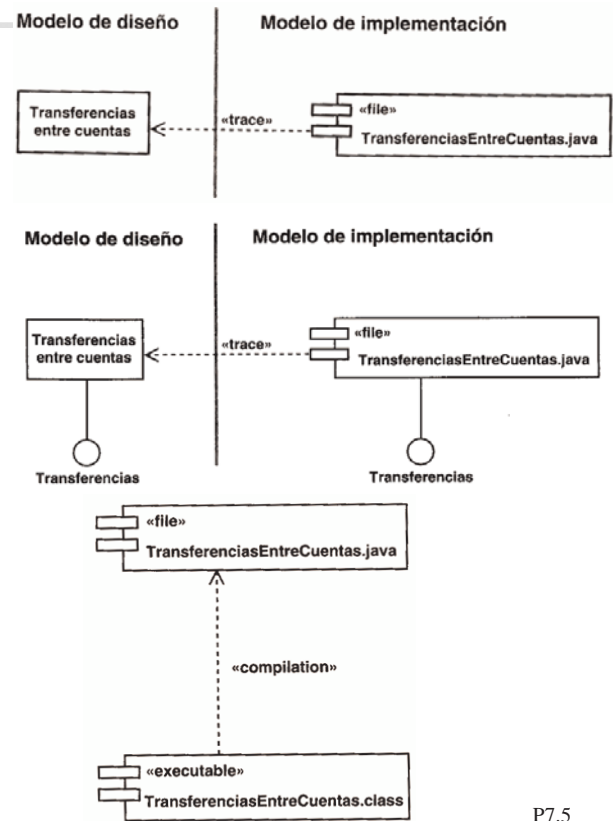
# Componente

- Empaquetamiento físico** de los **elementos de un modelo** (como son las clases en el modelo de diseño)
- Estereotipos de Componentes:**
  - `<<executable>>`, `<<file>>`, `<<library>>`, `<<table>>`, `<<document>>`
  - Durante la **creación de componentes** en un **entorno de implementación particular** **pueden ser modificados** para reflejar el **significado real** de estos componentes
- Características:**
  - Tienen **relaciones de traza** con los **elementos del modelo que implementan**
  - En normal que **un componente implemente varios elementos**, por ejemplo varias clases. Esta traza depende de cómo van a ser estructurados y modularizados los ficheros de código fuente, dado el L. de Programación que se use
  - Proporcionan las **mismas interfaces** que los **elementos del modelo que implementan**
  - Puede haber **dependencias de compilación entre componentes** (denotan los componentes necesarios para compilar un componente determinado)



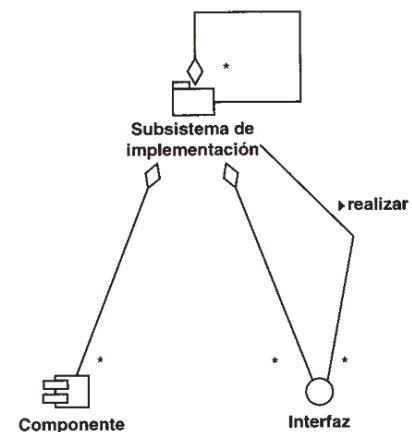
# Componente

- **Ejemplo:** Componente sigue la traza de una clase
  - En el sistema ejemplo, la clase de diseño "Transferencias entre Cuentas" se implementa en el componente de código fuente "TransferenciasEntreCuentas.java" (comúnmente en Java se crea un fichero de código fuente para cada clase). Dependencia de traza entre el modelo de diseño y de implementación
- **Ejemplo:** Interfaces en el diseño y en la implementación
  - En el sistema ejemplo, la clase de diseño "Transferencias entre Cuentas" proporciona una interfaz "Transferencias", esta interfaz es proporcionada también por el componente "TransferenciasEntreCuentas.java"
- **Ejemplo:** Dependencias de compilación entre componentes
  - En el sistema ejemplo, el componente (fichero) "TransferenciasEntreCuentas.java" se compila a un componente (ejecutable) "TransferenciasEntreCuentas.class"



# Subsistema de Implementación

- Organiza los artefactos del modelo de implementación en trozos manejables.
- Se manifiesta a través de un mecanismo de empaquetamiento concreto en un entorno de implementación determinado:
  - "Paquete" en Java
  - "Proyecto" en Visual Basic
  - "Directorio" de ficheros en un proyecto C++
  - "Subsistema" en un entorno de desarrollo integrado como Rational Apex
  - "Paquete" en herramienta de modelado visual como Rational Rose

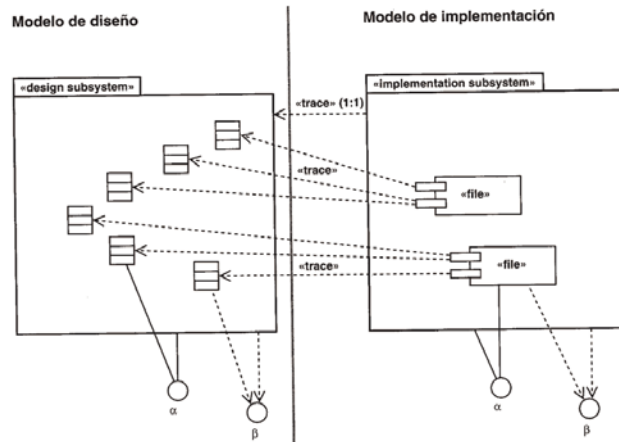


Formado por componentes, interfaces y otros subsistemas (recursivamente). Además puede implementar las interfaces que representan la funcionalidad que exportan en forma de operaciones



# Subsistema de Implementación

- Relacionados con los subsistemas de diseño, deberían seguir la traza uno a uno (también en subsistemas de servicio). Los aspectos definidos en los subsistemas de diseño son importantes y los subsistemas de implementación deberían:
  - Definir dependencias análogas hacia otros subsistemas de implementación o interfaces
  - Proporcionar las mismas interfaces
  - Definir qué componentes, o recursivamente, qué otros subsistemas de implementación dentro del subsistema deberían proporcionar las interfaces proporcionadas por el subsistema. Siguiendo la traza de las clases correspondientes en el subsistema de diseño que implementan
  - Los cambios en el modo en que los subsistemas proporcionan y usan interfaces, o cambios en las interfaces mismas, están descritos en los flujos de trabajo de diseño, luego no es necesario tratarlos en los flujos de trabajo de la implementación, aunque afectan al modelo de implementación



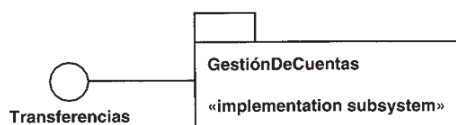
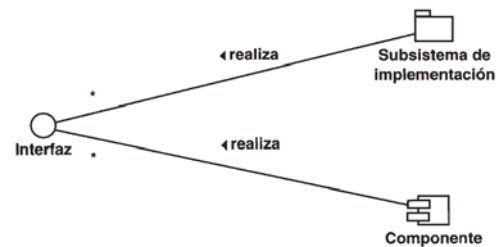
María Sierra - IS1

P7.7



# Interfaz

- Se utilizan para especificar las operaciones implementadas por componentes y subsistemas de implementación. Estos a su vez pueden tener "dependencias de uso" sobre interfaces
  - Un componente que implementa una interfaz ha de implementar correctamente todas las operaciones definidas por la interfaz
  - Un subsistema de implementación que proporciona una interfaz tiene también que contener componentes que proporcionen la interfaz u otros subsistemas (recursivamente) que proporcionen la interfaz
- **Ejemplo:** Un subsistema de implementación que proporciona una interfaz
  - En el sistema ejemplo, tiene un subsistema de implementación llamado "GestiónDeCuentas" (un paquete de Java), el cual proporciona la interfaz Transferencias



```
package GestionDeCuentas;
// interfaces proporcionados:
public interface Transferencias{
    public Cuenta Crear (Cliente propietario, Dinero saldo,
                        NumeroCuenta id_cuenta);
    public void Depositar (Dinero cantidad, Cadena razon);
}
```

María Sierra - IS1

P7.8



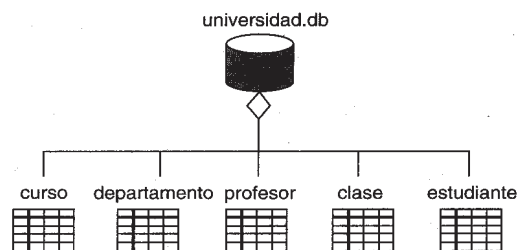
# Introducción: Modelado Arquitectónico

- 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 elementos para modelar la arquitectura física de un sistema:
  - Componentes
  - Nodos



# Diagrama de Componentes: Introducción

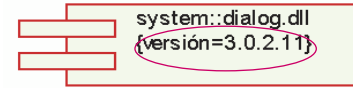
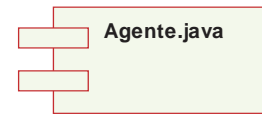
- Muestran un conjunto de **componentes** y sus **relaciones**
- **Contienen:**
  - Componentes
  - Interfaces
  - Relaciones de dependencia, generalización, asociación y realización
- **Técnicas Comunes de Modelado**
  - Modelado de código fuente
  - Modelado de una versión ejecutable
  - Modelado de una BBDD física
  - Modelado de sistemas adaptables





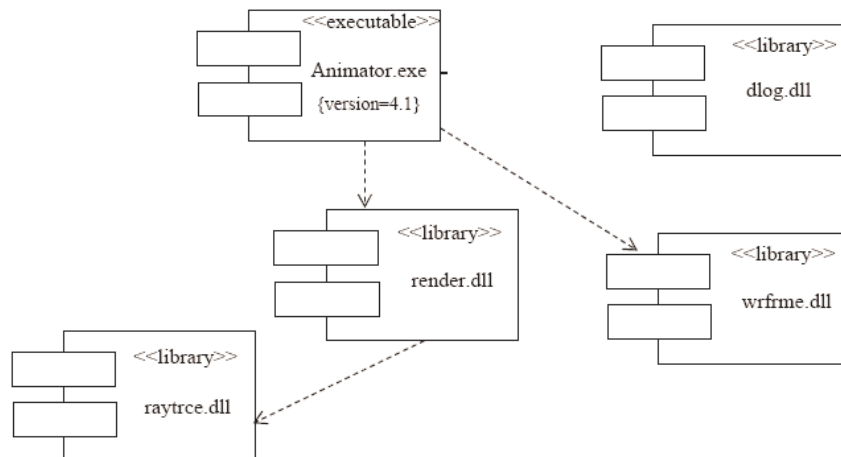
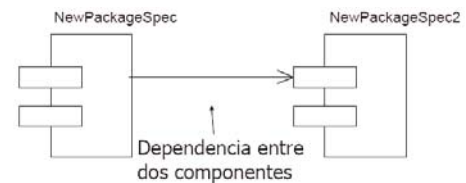
# Diagrama de Componentes: Elementos

- **Componente:** Parte física y reemplazable de un sistema que implementa y suministra ciertos conjuntos de interfaces
- Se utiliza para modelar elementos físicos que pueden hallarse en un nodo, como **ejecutables**, **bibliotecas**, **tablas**, **archivos**, **documentos**, etc
- Gráficamente:
  - Se representa como un **rectángulo con pestañas**
  - Normalmente se dibujan mostrando sólo su nombre
  - Se pueden adornar con valores etiquetados o con compartimentos adicionales
  - Pueden ser estereotipados y tener valores etiquetados



# Diagrama de Componentes: Elementos

- **Relaciones de Dependencia:** se utilizan en los para indicar que un componente se refiere a los servicios ofrecidos por otro componente
- Ejemplo: Modelado de ejecutables y bibliotecas

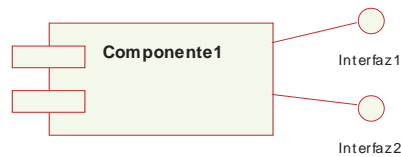
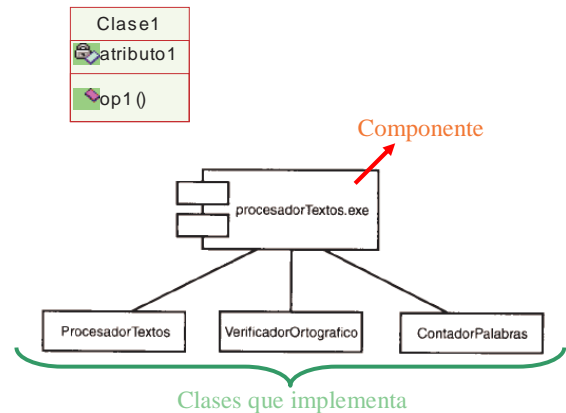
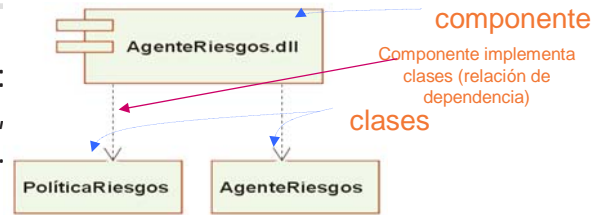




# Componentes y Clases

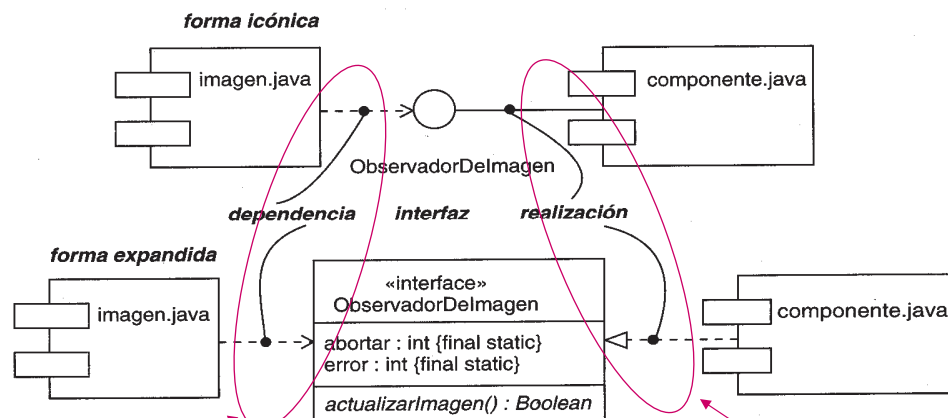
- Los **componentes** son como las **clases**: tienen nombres, realizan interfaces, pueden participar en relaciones, etc. Sin embargo,

- Clases = abstracciones lógicas
- Componentes = fragmentos físicos del sistema
- Los componentes son "paquetes físicos" de otros elementos lógicos
- Las clases tienen operaciones y atributos
- Los componentes tienen interfaces



# Componentes e Interfaces

- Una **interfaz** es una colección de operaciones que se utiliza para especificar un servicio de una clase o un componente
- La relación entre componente e interfaz es importante
- Para mostrar estas relaciones hay dos formas: **canónica** y **expandida**

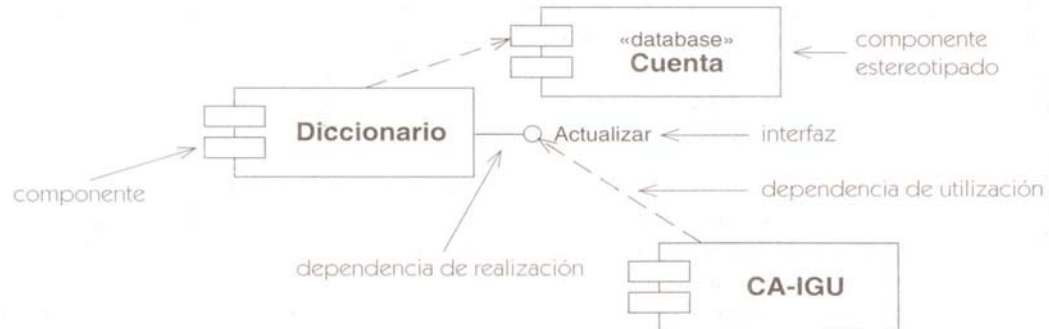


Componente usa el interfaz

Componente realiza el interfaz



## Componentes e Interfaces



## Diagramas de Componentes: Tipos

- **Componentes de despliegue:** Son los que se necesitan para formar un sistema ejecutable, como bibliotecas dinámicas (DLL) y ejecutables (EXE)
  - Se incluyen modelos de objetos clásicos (COM+, CORBA y EJB)
- **Componentes producto del trabajo:** Son productos que quedan al final del proceso de desarrollo (archivos de código fuente y de datos)
  - Estos componentes no participan directamente en un sistema ejecutable pero son los productos del trabajo de desarrollo que se utilizan para crear el sistema ejecutable.
- **Componentes de ejecución:** Se crean como consecuencia de un sistema en ejecución
  - Como un objeto COM+ que se instancia a partir de un DLL



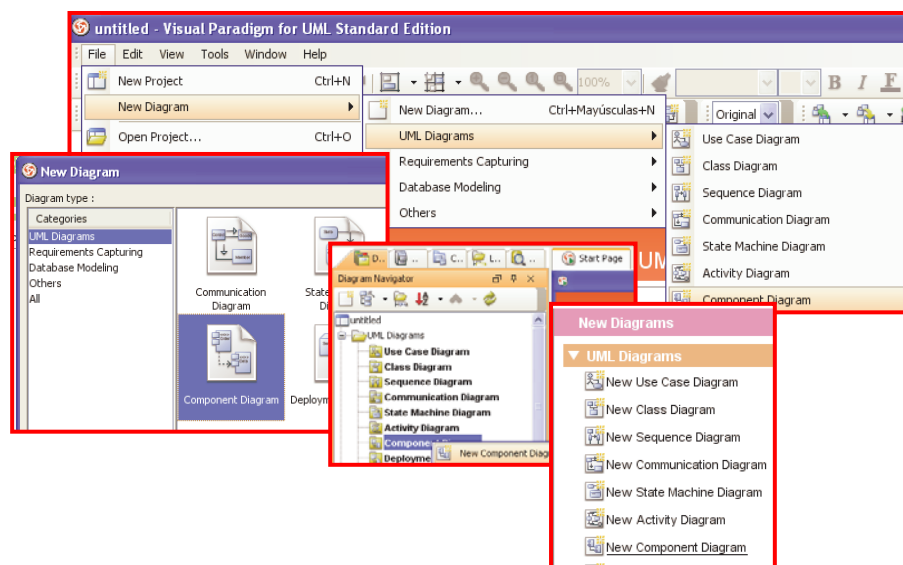
# Diagramas de Componentes: Elementos Estándar

- Los **mecanismos de extensibilidad** de UML se pueden utilizar libremente para definir nuevos tipos de componentes (mediante estereotipos), agregando nuevas propiedades (valores etiquetados), y dándole nueva semántica (restricciones), incluso nueva apariencia (iconos)
- UML ofrece varios estereotipos predefinidos:
  - *Executable*
    - Componente que se puede ejecutar en un nodo
  - *Library*
    - Biblioteca de Objetos estática o dinámica
  - *Table*
    - Tabla de una BBDD
  - *File*
    - Documento que contiene código fuente o datos
  - *Document*



# Diagramas de Componentes con VP

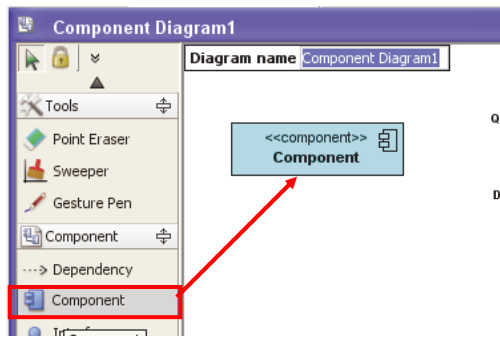
## Crear Diagrama



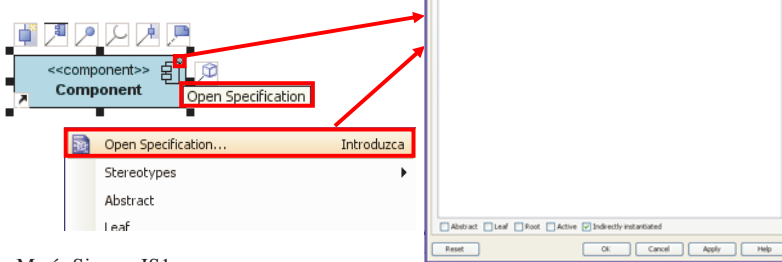


# Diagramas de Componentes con VP: Elementos

## Componente



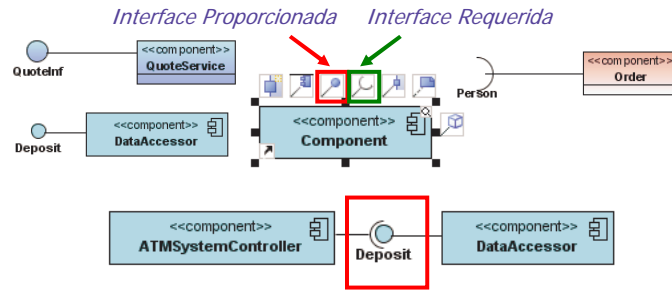
## Especificar Componente



María Sierra - IS1

## Interfaces

Colecciones de uno o más métodos que pueden o no contener atributos



## Conector "Esamble"

Une la interfaz requerida del componente (Componente1) con la interfaz proporcionada de otro componente (Component2); esto permite que un componente provea los servicios que otro componente requiere

P7.19



# Diagramas de Componentes con VP: Elementos

## Componentes con Puertos

- Usar puertos con Diagramas de Componentes permite que se especifique un servicio o comportamiento a su entorno así como también un servicio o comportamiento que un componente requiere o proporciona. Los puertos pueden especificar entradas, salidas así como también operar bi-direccionalmente (puertos complejos)
- Las interfaces asociadas con un puerto especifican la naturaleza de las intercciones que ocurren sobre el puerto

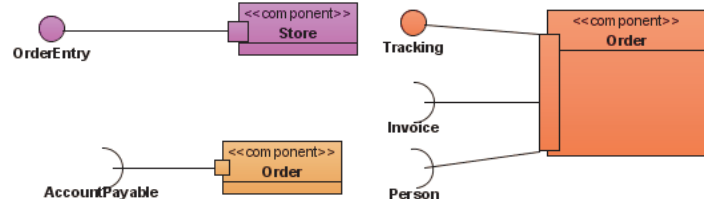
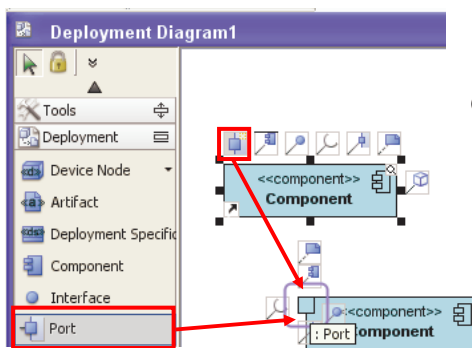
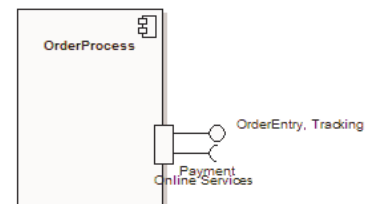


Diagrama que detalla un componente con un puerto para servicios "En Línea" conjuntamente con dos interfaces proporcionadas "Ordenar Entrada" y "Seguimiento" así como también una interfaz requerida "Pago"

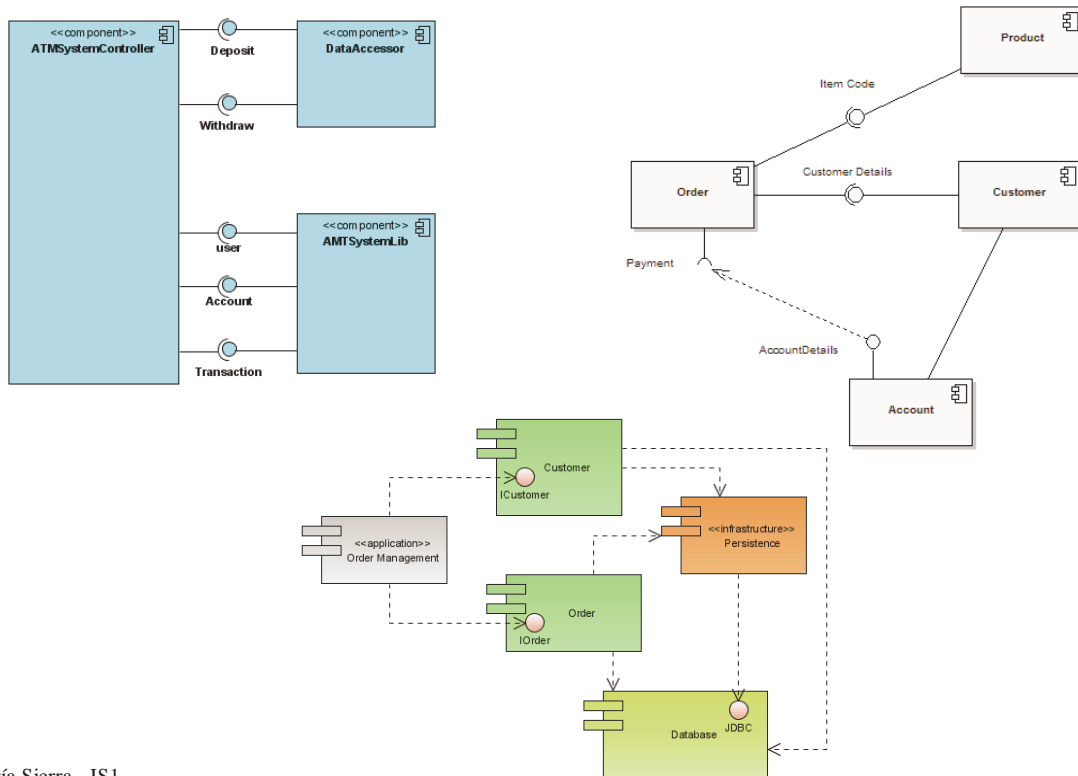


María Sierra - IS1

P7.20



# Diagramas de Componentes con VP: Ejemplos



María Sierra - IS1

P7.21



# Diagramas de Despliegue: Introducción

- Se utilizan para modelar los aspectos físicos de los sistemas orientados a objetos
- Muestran la configuración de nodos que participan en la ejecución de los componentes que residen en ellos
- En UML se utilizan para visualizar los aspectos estáticos de los nodos físicos y sus relaciones y para especificar sus detalles para la construcción
- No siempre es necesario utilizar diagramas de despliegue
  - **No son necesarios** si 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 de un PC, etc..)
  - **Son necesarios** si:
    - Se desarrolla un software que interactúa con dispositivos que normalmente no gestiona el SO
    - El sistema está distribuido físicamente sobre varios procesadores

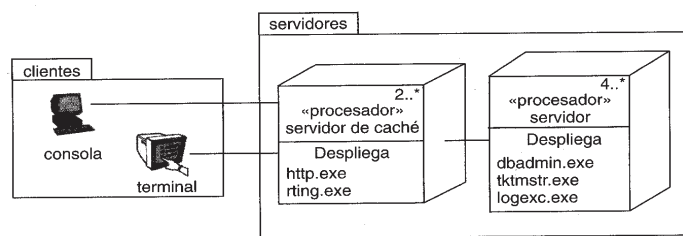
María Sierra - IS1

P7.22



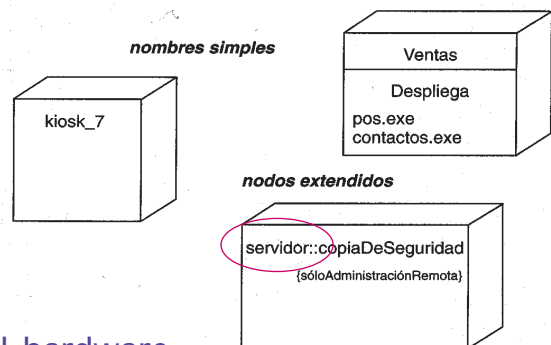
# Diagramas de Despliegue: **Introducción**

- Los diagramas de despliegue **contienen**:
  - Nodos
  - Relaciones de Dependencia y Asociación
  - Pueden contener notas, restricciones y paquetes
- **Técnicas Comunes de Modelado**
  - Modelado de un sistema emputrado
  - Modelado de un sistema cliente/servidor
  - Modelado de un sistema completamente distribuido



# Diagramas de Despliegue: **Elementos**

- **Nodos:**
  - Elemento físico que existe en tiempo de ejecución
  - 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



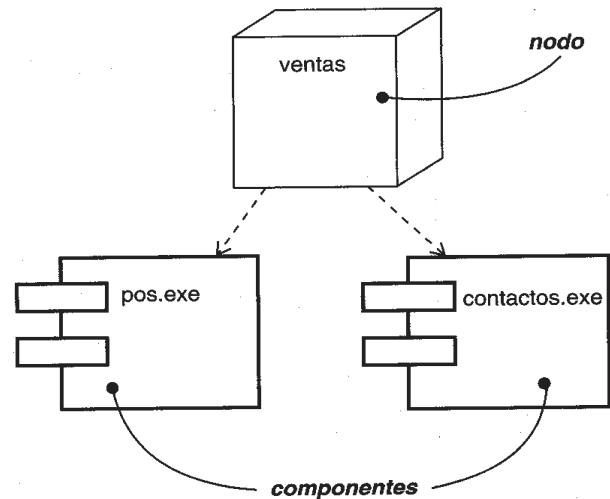


## Diagramas de Despliegue: **Nodos vs. Componentes**

- Ambos tienen nombres, pueden participar en relaciones de dependencia, generalización y asociación, pueden anidarse, pueden tener instancias, pueden participar de interacciones

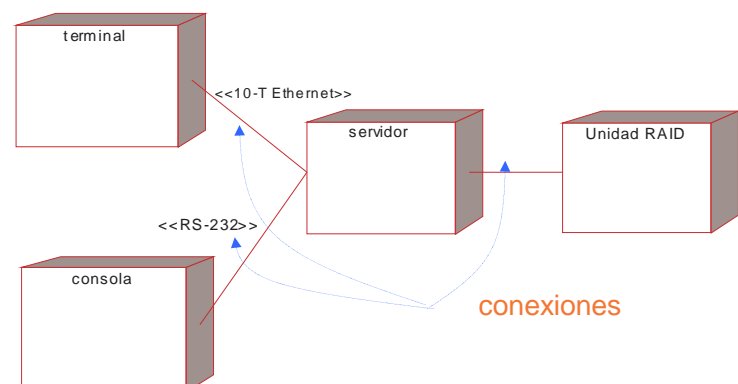
- **Diferencias**

- Los **componentes** son los elementos que participan en la ejecución de un sistema; los **nodos** son los elementos donde se ejecutan los componentes
- Los **componentes** representan el empaquetamiento físico de los elementos lógicos; los **nodos** representan el despliegue físico de los componentes



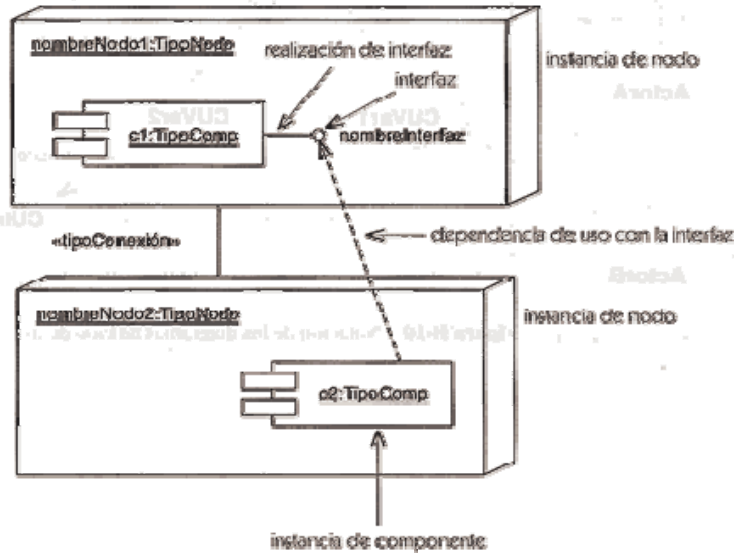
## Diagramas de Despliegue: **Nodos y Conexiones**

- 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)
- Se pueden incluir roles, multiplicidad y restricciones



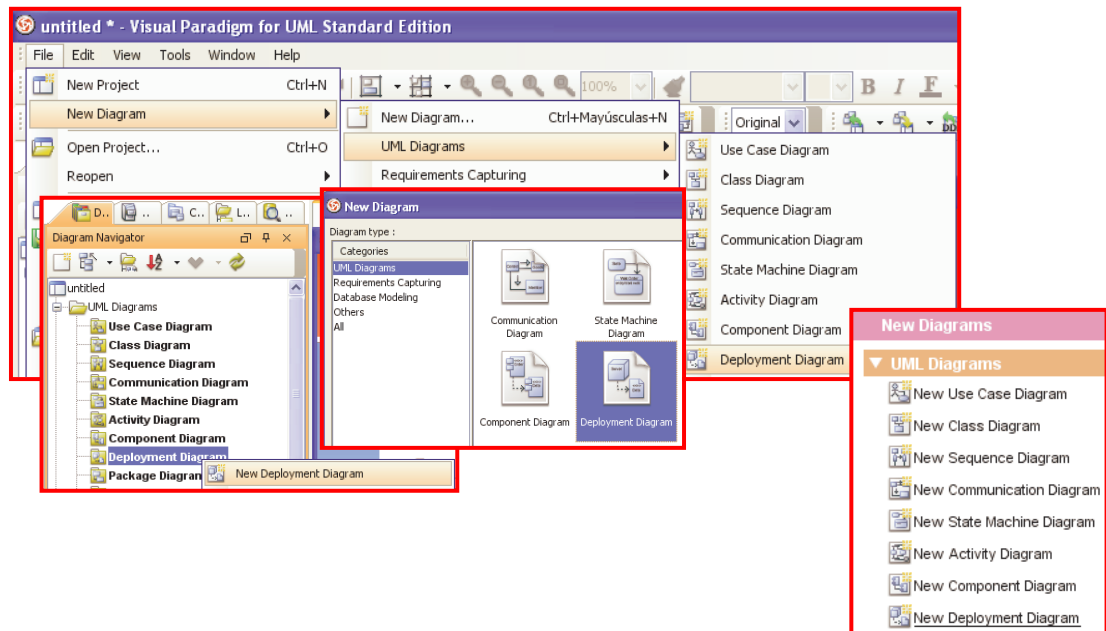


# Resumen de la Notación



# Diagramas de Despliegue con VP

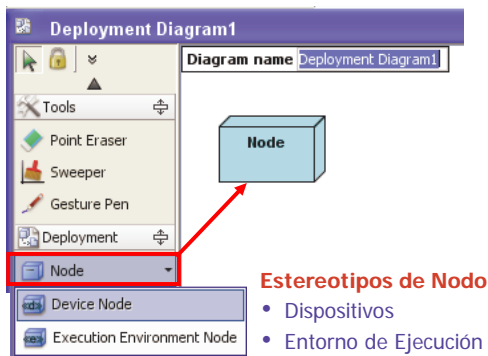
## Crear Diagrama



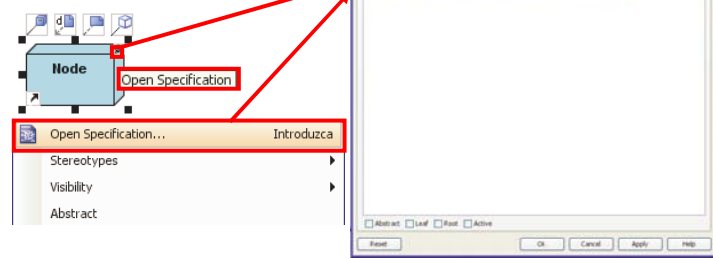


# Diagramas de Despliegue con VP: Elementos

**Nodo:** Elemento HW ó SW

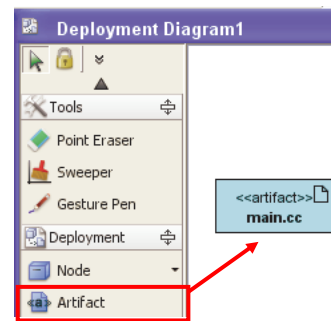


**Especificar Nodo**



## Artefacto

Producto del proceso de desarrollo de software, que puede incluir los modelos del proceso (modelos de Casos de Uso, modelos de Diseño, etc.), archivos fuente, ejecutables, documentos de diseño, reportes de prueba, prototipos, manuales de usuario y más...



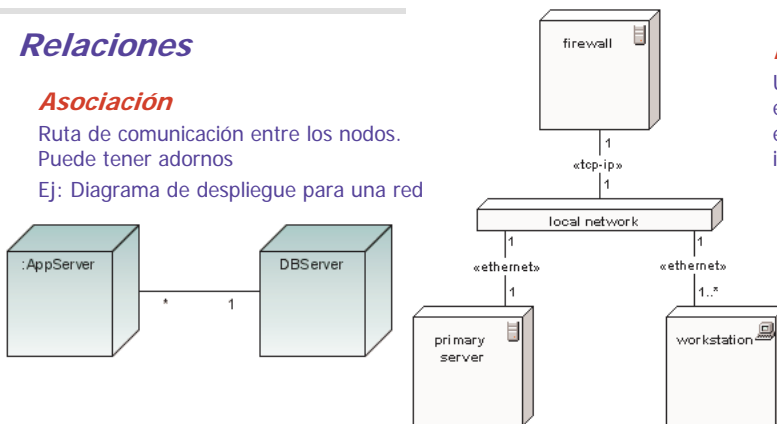
# Diagramas de Despliegue con VP: Elementos

## Relaciones

### Asociación

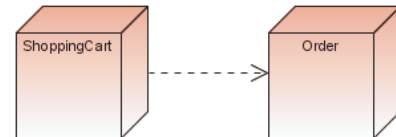
Ruta de comunicación entre los nodos. Puede tener adornos

Ej: Diagrama de despliegue para una red



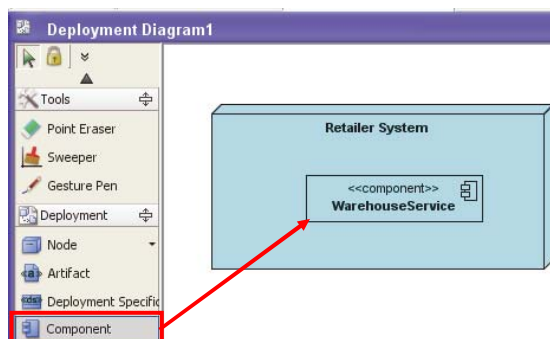
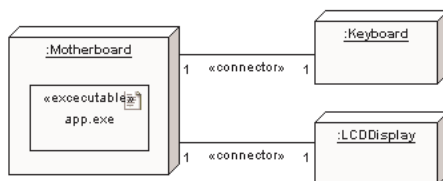
### Dependencia

Un modelo o un conjunto de modelos de elementos requieren de otro modelo de elementos para su especificación o implementación. Ej: Un nodo requiere otro nodo



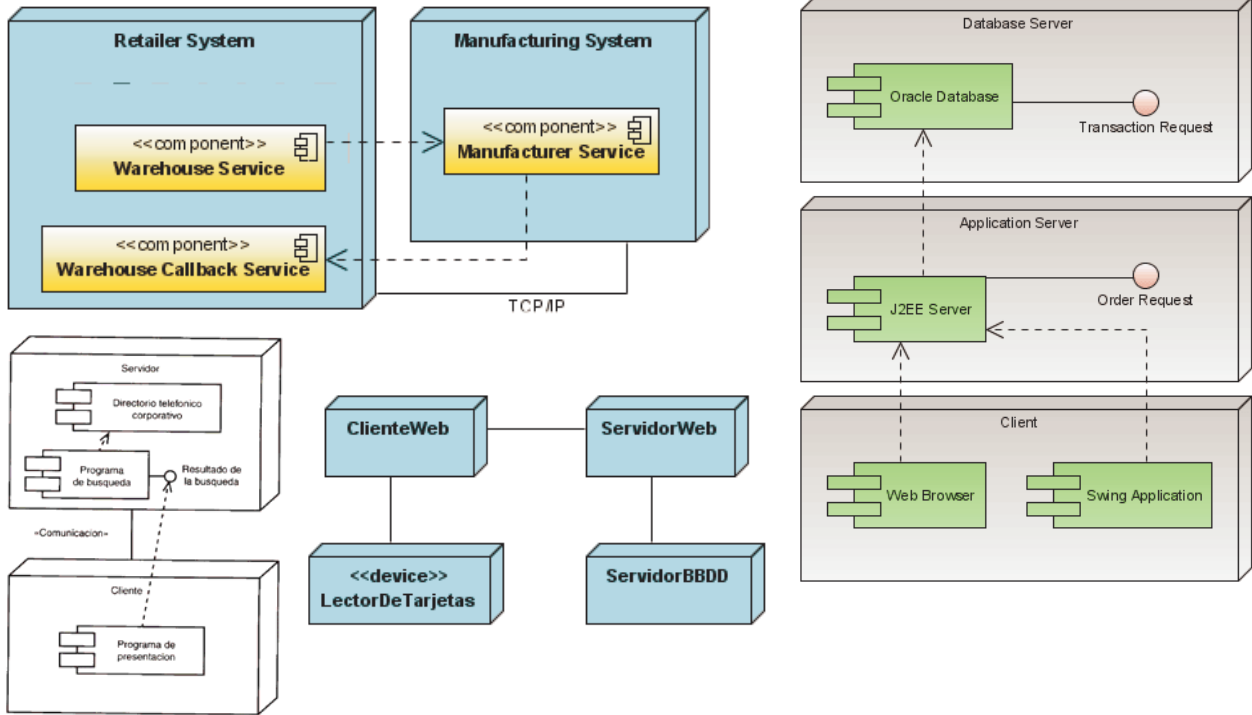
## Nodo como Contenedor

Un nodo puede contener otros elementos, como componentes o artefactos





# Diagramas de Despliegue con VP: Ejemplos



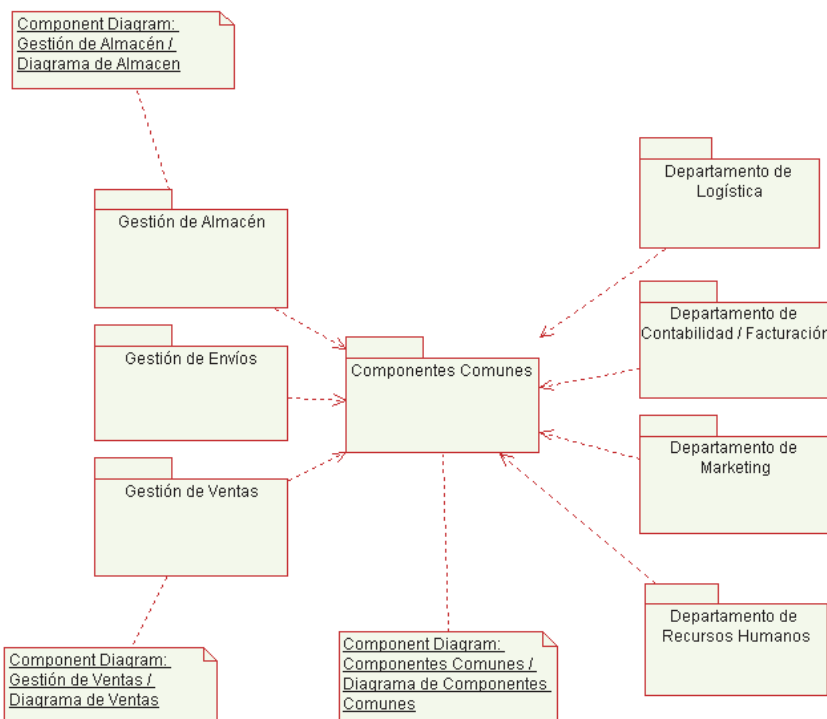
María Sierra - IS1

P7.31



# Modelado de Implementación: Ejemplo

## Diagrama de Paquetes del Sistema: Gestión de Artículos Deportivos



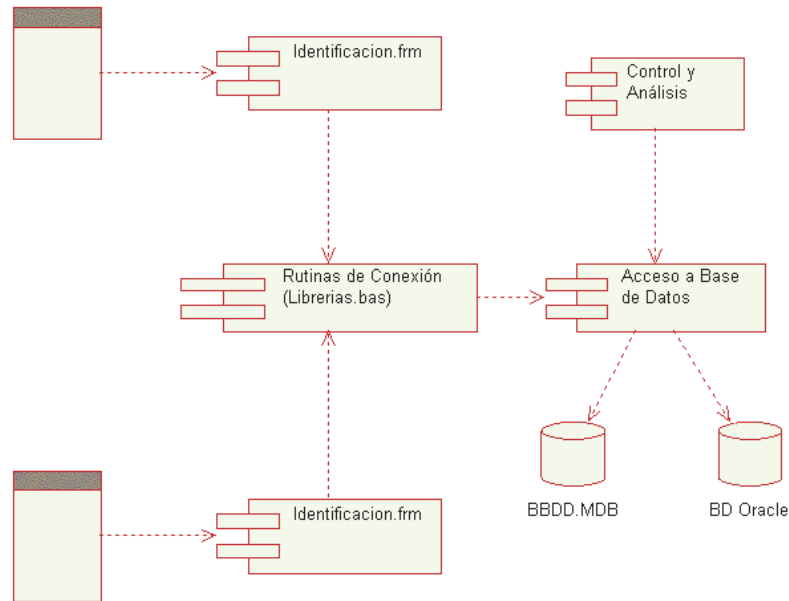
María Sierra - IS1

P7.32



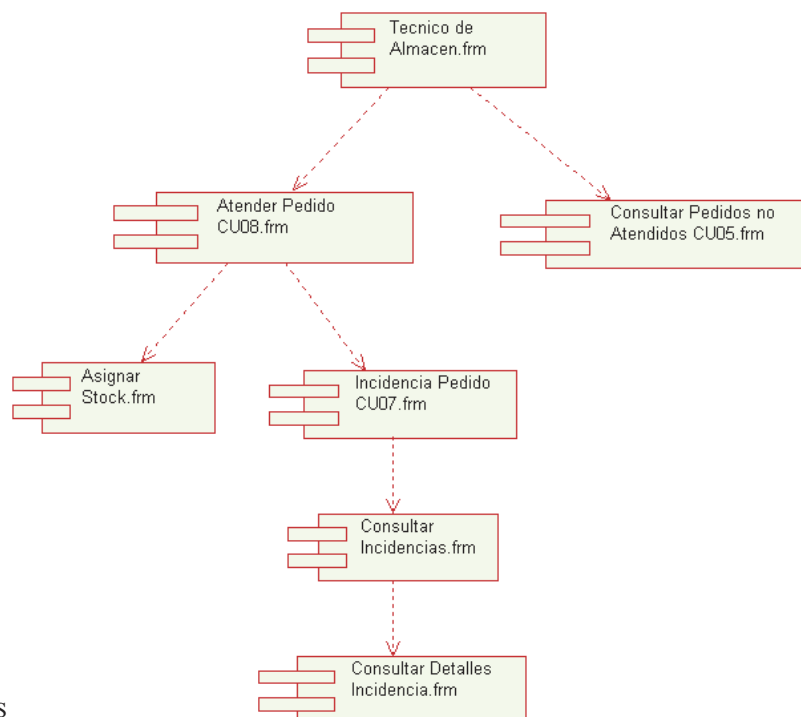
# Modelado de Implementación: Ejemplo

## Diagrama de Componentes Comunes



# Modelado de Implementación: Ejemplo

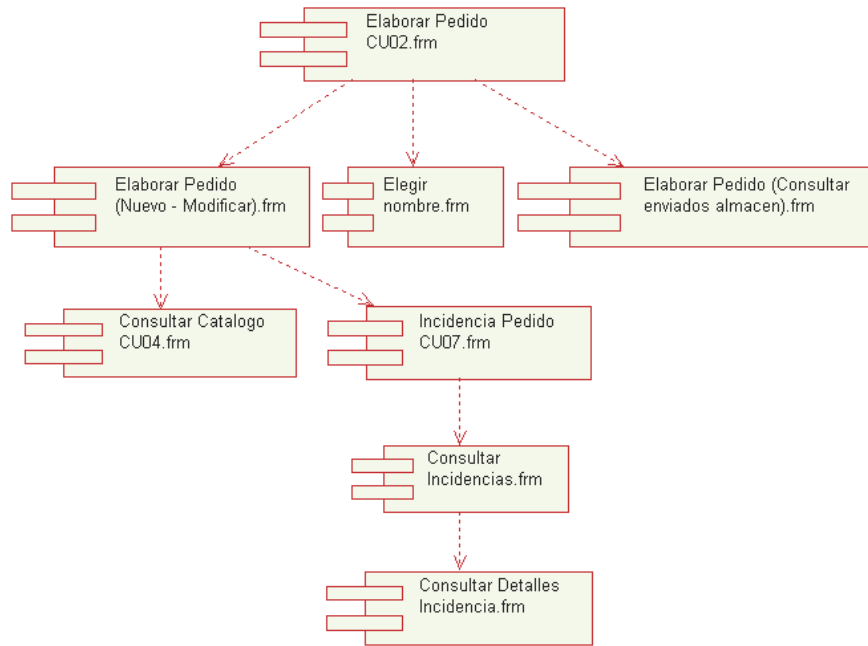
## Diagrama de Componentes Almacén





# Modelado de Implementación: Ejemplo

## Diagrama de Componentes Ventas



# Modelado de Implementación: Ejemplo

## Diagrama de Despliegue

