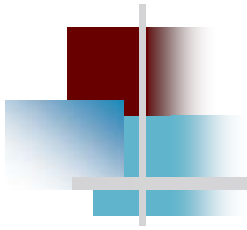
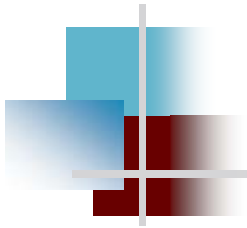


Tema 4
Ingeniería del Software basada en Componentes

Miguel A. Laguna

Desarrollo del tema

- 4.1 Especificación de componentes con UML
  - Partición (subsistemas como componentes)
- 4.2 Diseño de componentes con JEE
  - Diseño en capas
  - Proceso de desarrollo: tareas de diseño
- 4.3 Diseño de las capas
  - Patrones de las Capas de Persistencia, Negocio y Presentación
- 4.4 CBSE: desarrollo para y con reutilización
- 4.5 Diseño basado en componentes
- 4.6 Despliegue de componentes

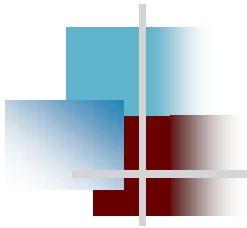


Bibliografía (General)

- Sommerville, I. "Ingeniería del software" Pearson, 2005 (7ª ed.)
- John Cheesman & John Daniels. UML components: a simple process for specifying component-based software. Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc., 2000

Lecturas complementarias

- Arlow, Jim, Neustadt, Ila. "UML 2", Anaya Multimedia, 2006
- OMG, "OMG Unified Modeling Language Specification. Version 2.4, 2011.



Bibliografía (JEE, .NET)

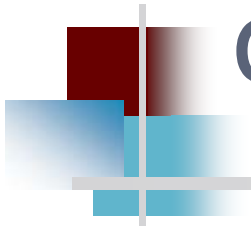
- Derek C. Ashmore. The J2EE Architect's Handbook. DVT Press. ISBN: 0972954899. 2004.
- Martin Fowler. Patterns of Enterprise Application Architecture. Addison Wesley, 2003.

Lecturas complementarias (JEE)

- Oracle/Sun (D. Alur et al.). Core J2EE patterns. 2003.
- Adam Bien. Real World Java EE Patterns Rethinking Best Practices. ISBN 978-0-557-07832-5, 2009.

Lecturas complementarias (.NET)

- César de la Torre Llorente y otros. Guía de Arquitectura N-Capas Orientada al Dominio con .NET 4.0. Microsoft Ibérica, 2010.



4.1. Especificación de componentes con UML

- Composición y adaptación
- Partición y subsistemas

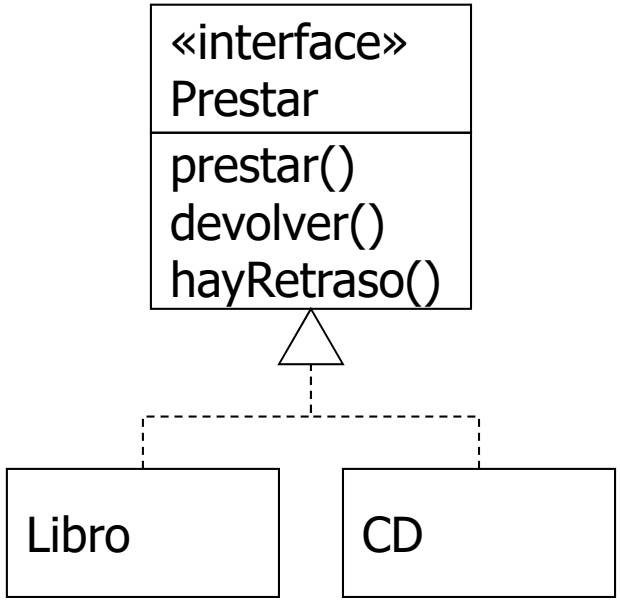
UML: componente

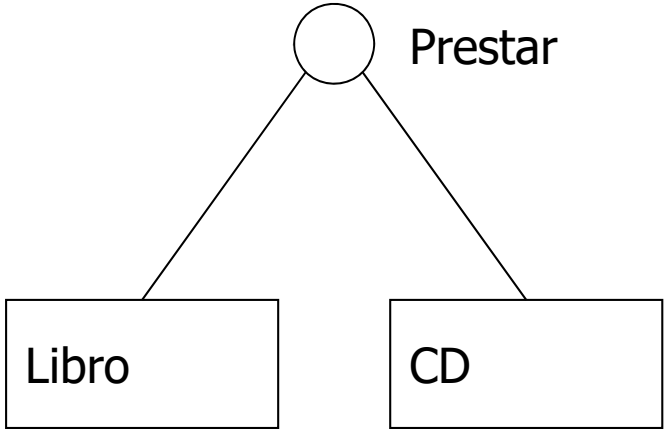
- "Un componente representa una parte modular de un sistema que encapsula su contenido y cuya manifestación es reemplazable dentro de su entorno"
  - Una caja negra cuyo comportamiento está completamente definido por sus interfaces proporcionadas y requeridas
  - Pueden ser sustituidos por otros componentes que proporcionen el mismo protocolo

- Los componentes pueden ser:
  - Físicos - puede ser directamente instanciado en tiempo de ejecución, por ejemplo, un Enterprise JavaBean (EJB)
  - Lógicos - una construcción lógica, por ejemplo, un subsistema. Sólo instanciado indirectamente cuando se crean instancias de sus partes

Interfaz proporcionada

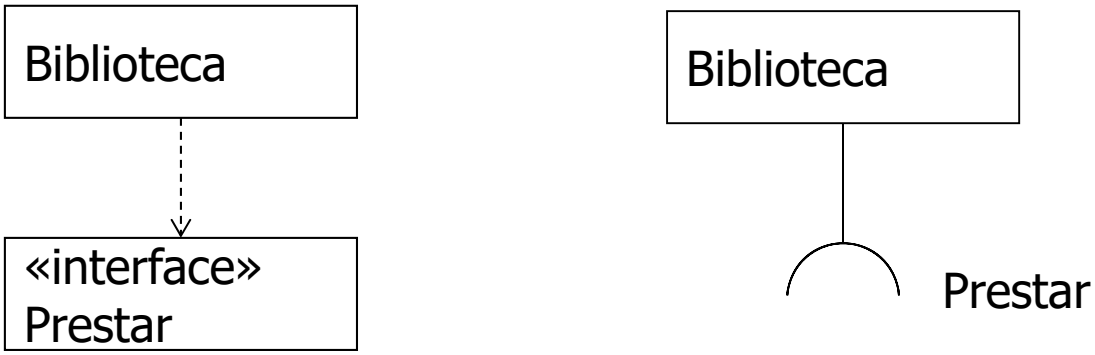
- Un clasificador implementa una interfaz y por tanto proporciona sus servicios





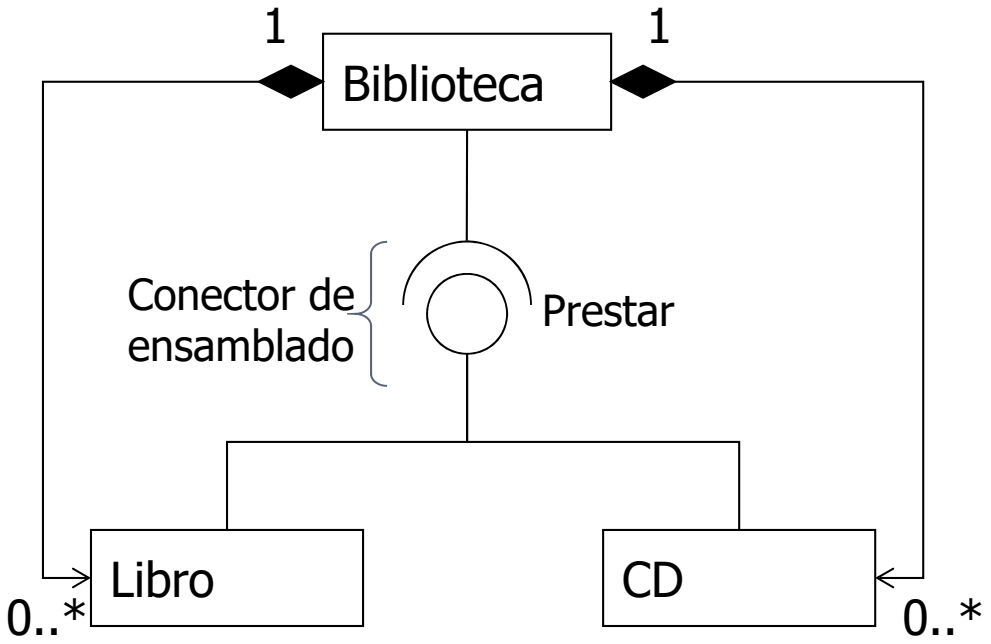
Interfaz requerida

- Una interfaz requerida indica que un clasificador utiliza los servicios definidos por la interfaz

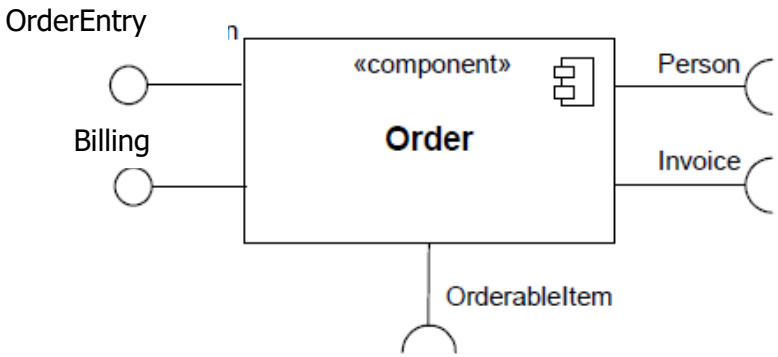


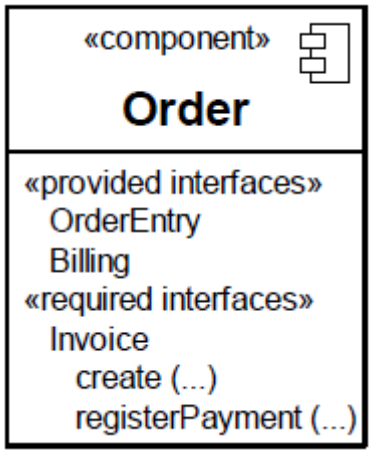
Conectores de ensamblado

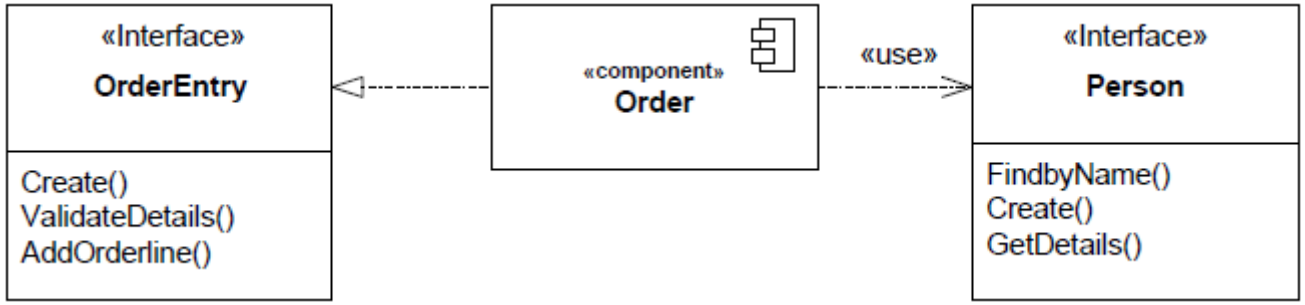
- Se pueden conectar ambos tipos de interfaz



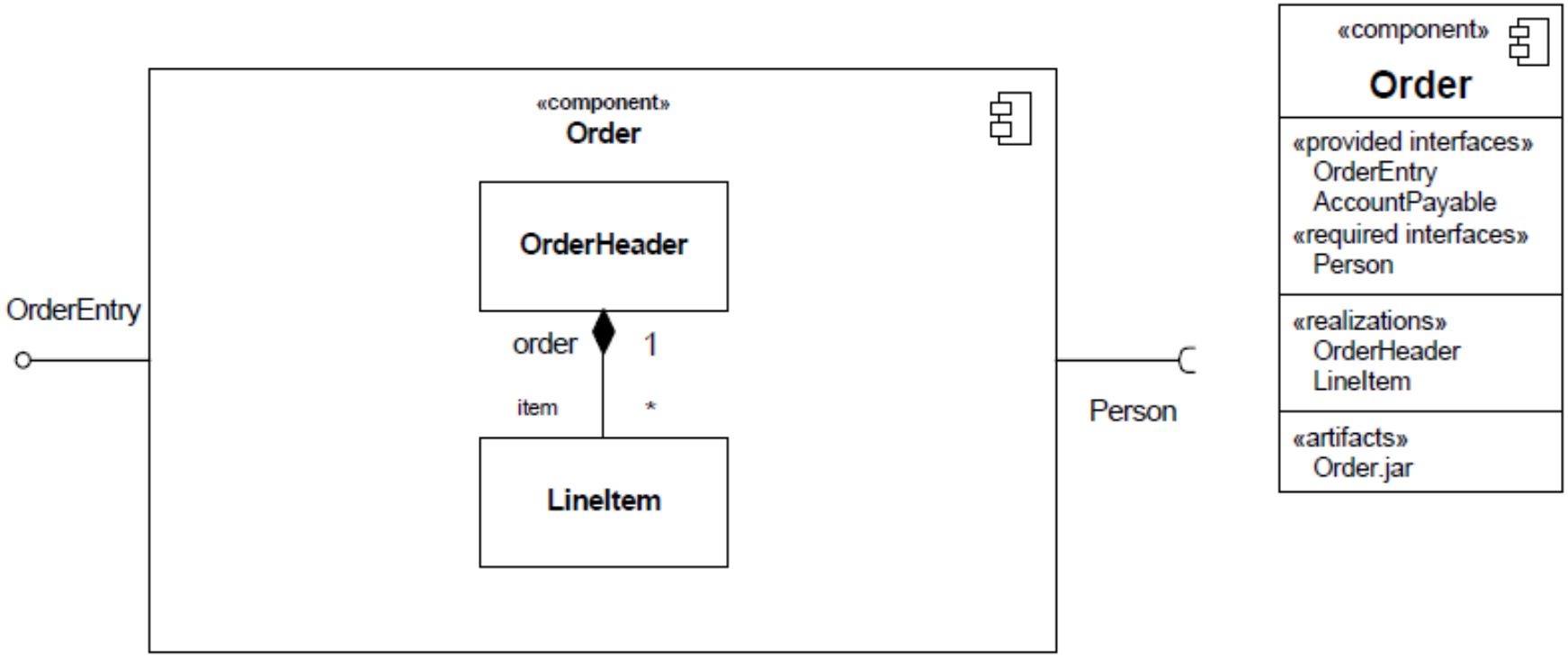
Representaciones (caja negra)





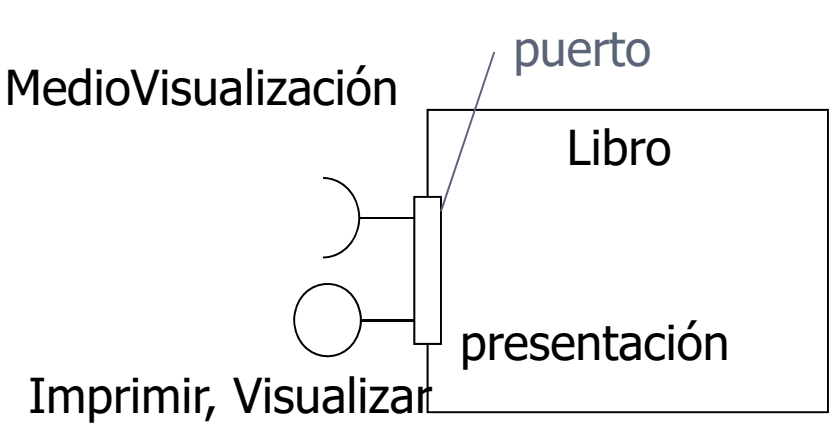


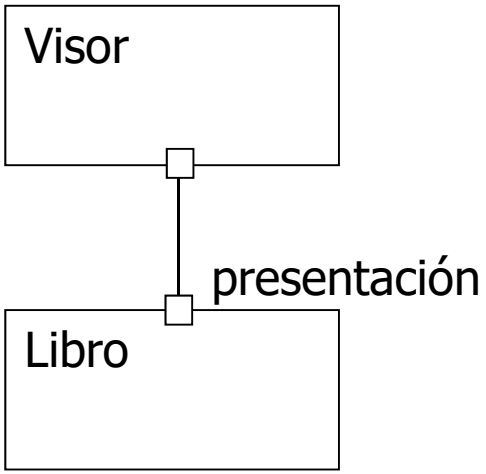
Representación de caja blanca: Detalle del componente



Puertos

- Un puerto especifica un punto de interacción entre un clasificador y su entorno
- Un puerto se describe por sus interfaces proporcionadas y requeridas:
  - Es un conjunto semánticamente cohesivo de interfaces
  - Puede tener un nombre



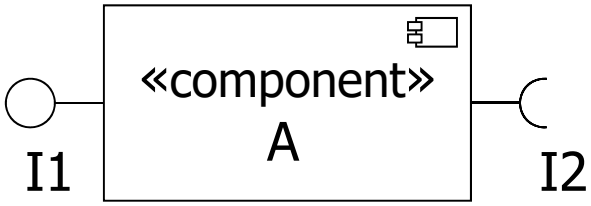


Estructura interna

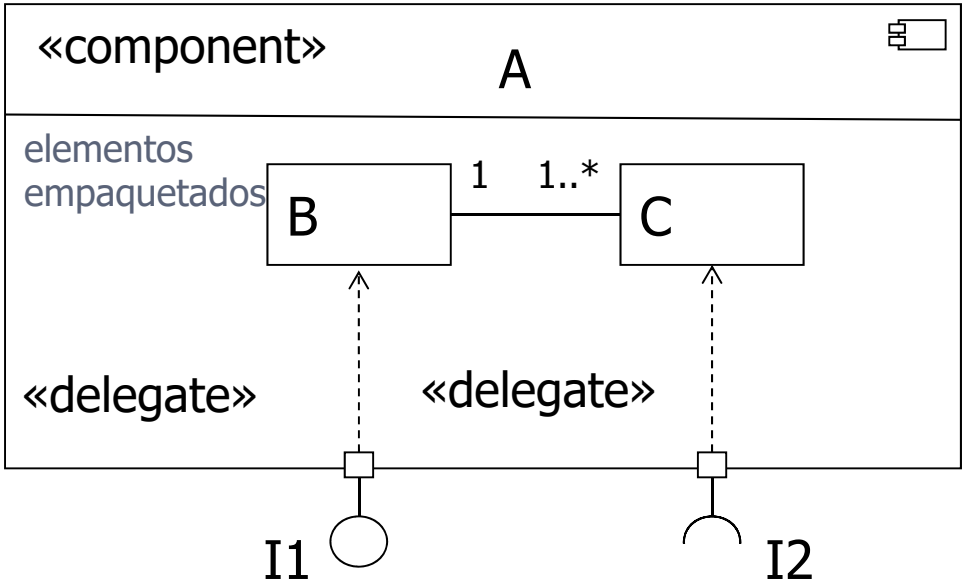
- Un componente puede realizarse con clases o puede descomponerse en otros componentes
- Las interfaces delegan en las partes internas
  - Se pueden mostrar los elementos anidados, conectados a él por relaciones de dependencia (<<delegación>>)
- Se pueden mostrar
  - Clases que realizan el componente (“Compartimento de elementos empaquetados”)
  - Componentes internos (“Compartimento de estructura interna”)
    - Partes y conectores (visión simplificada)

Caja negra/Caja blanca

Vista de caja negra

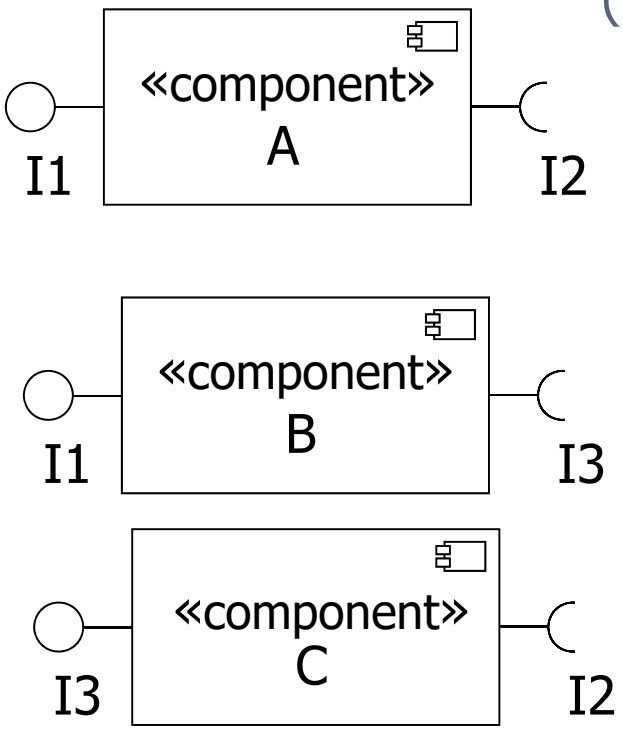


Vista de caja blanca: B y C pueden ser clases (Compartimento de elementos empaquetados)

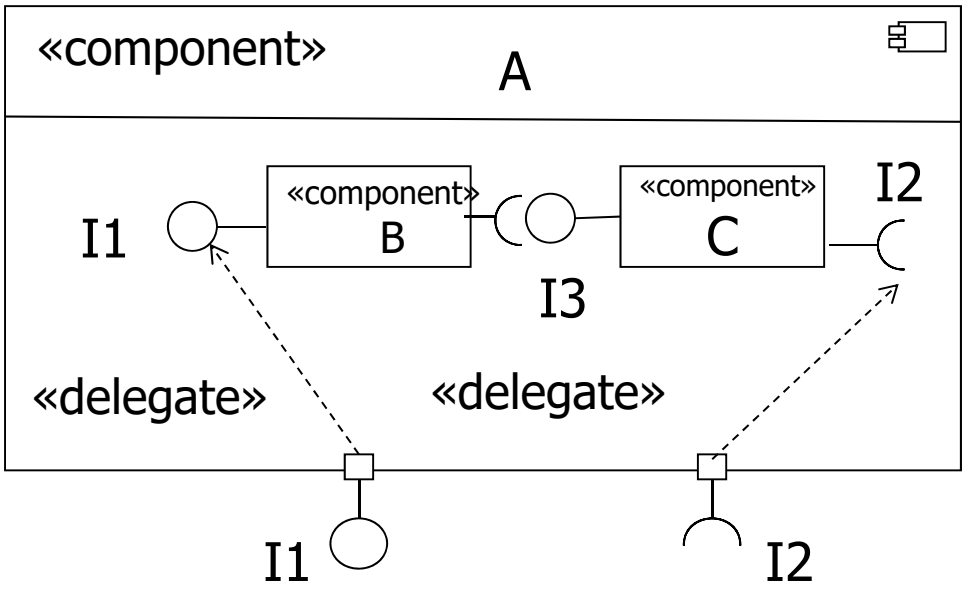


Caja negra/Caja blanca

Vista de caja negra

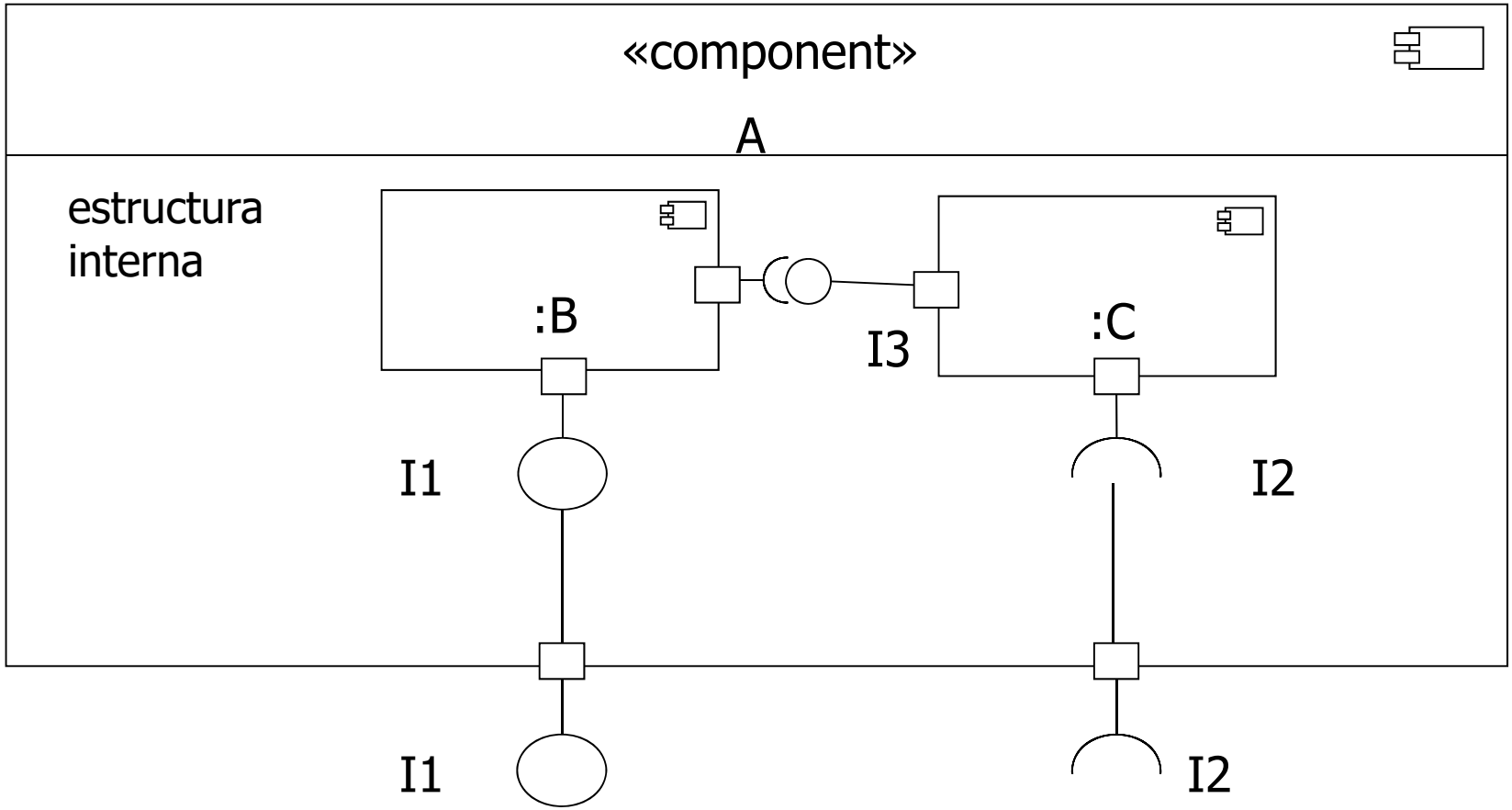


Vista de caja blanca:
¡ B y C pueden ser a su vez componentes!
(elementos empaquetados)

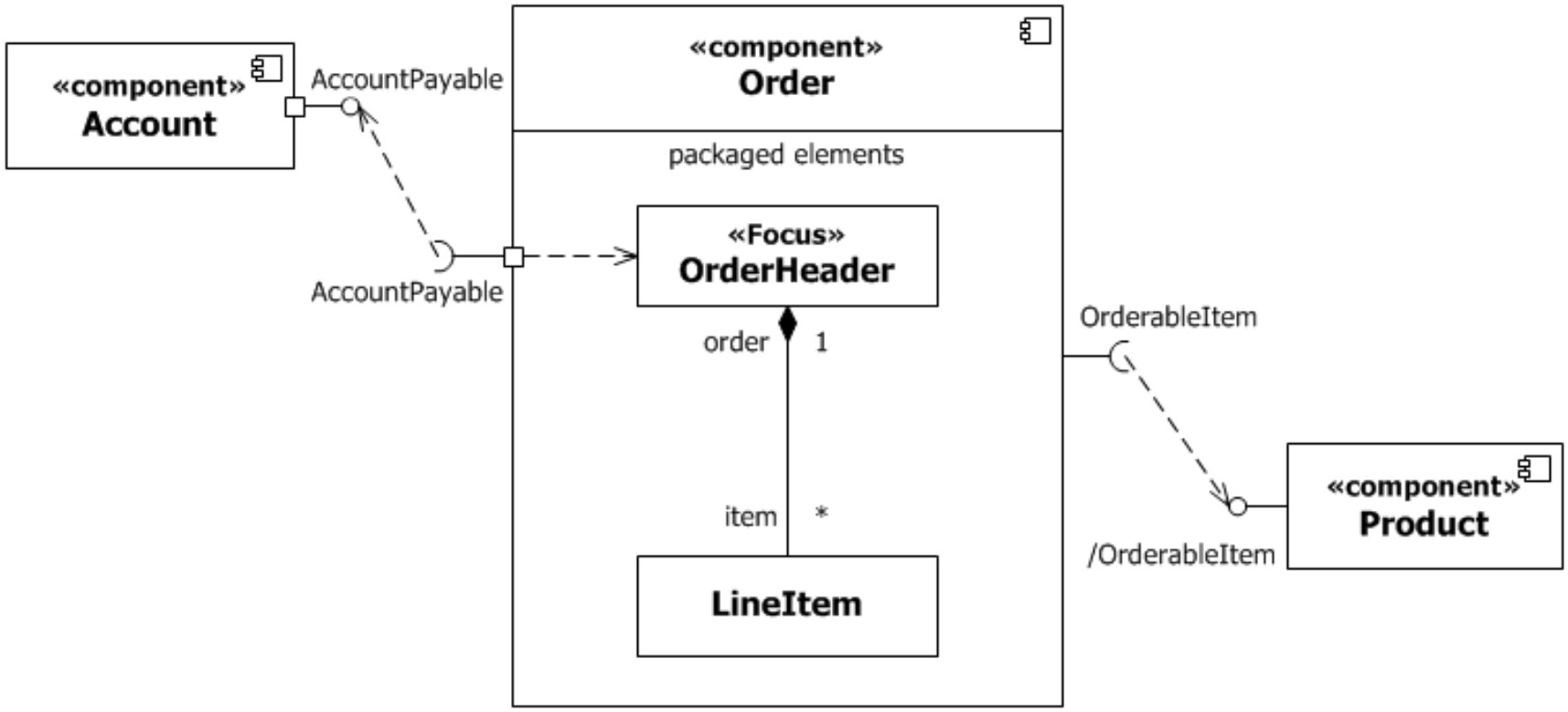


Partes y conectores

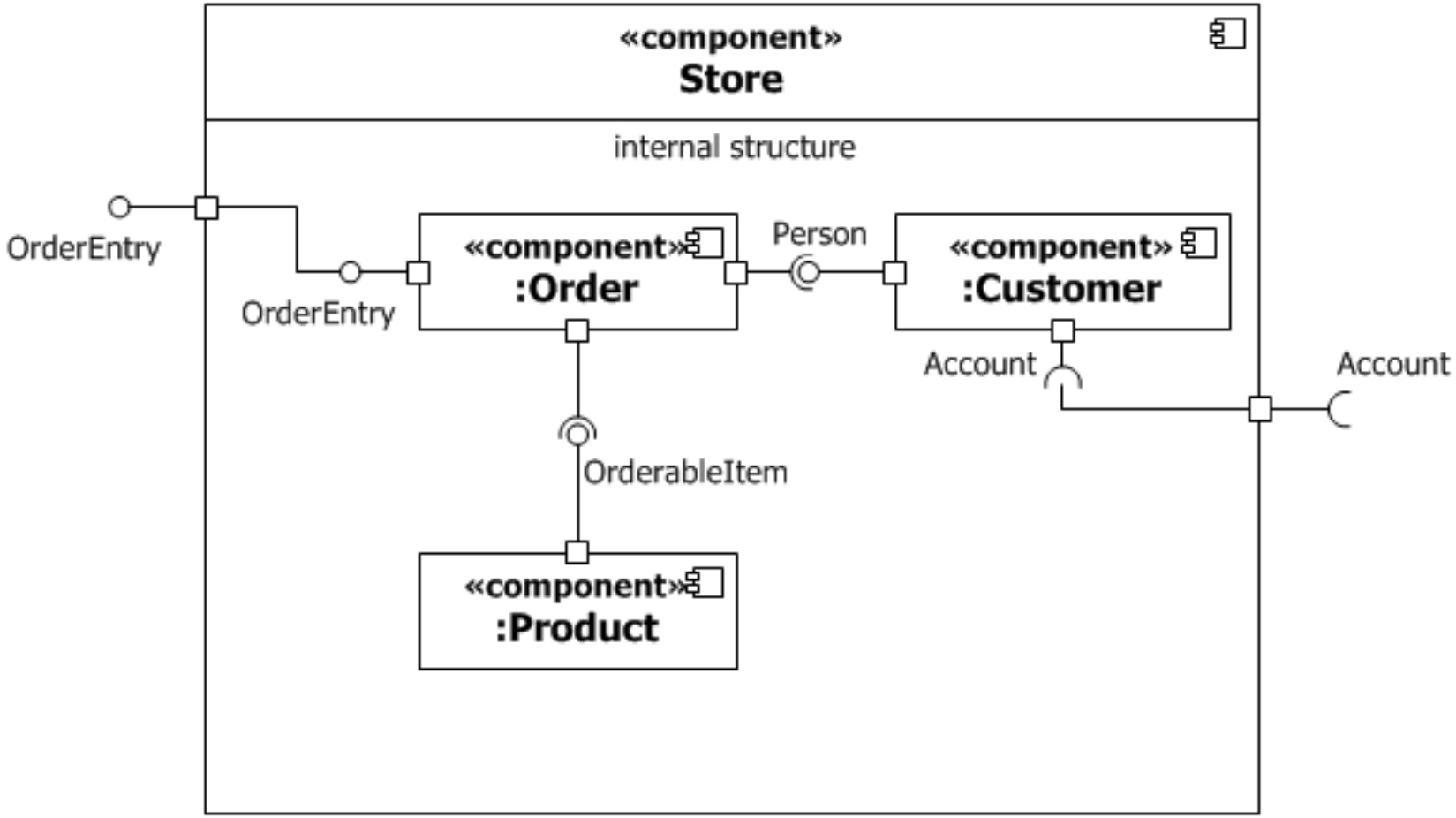
Vista de caja blanca:
Como ejecutables B y C son (instancias) de componentes



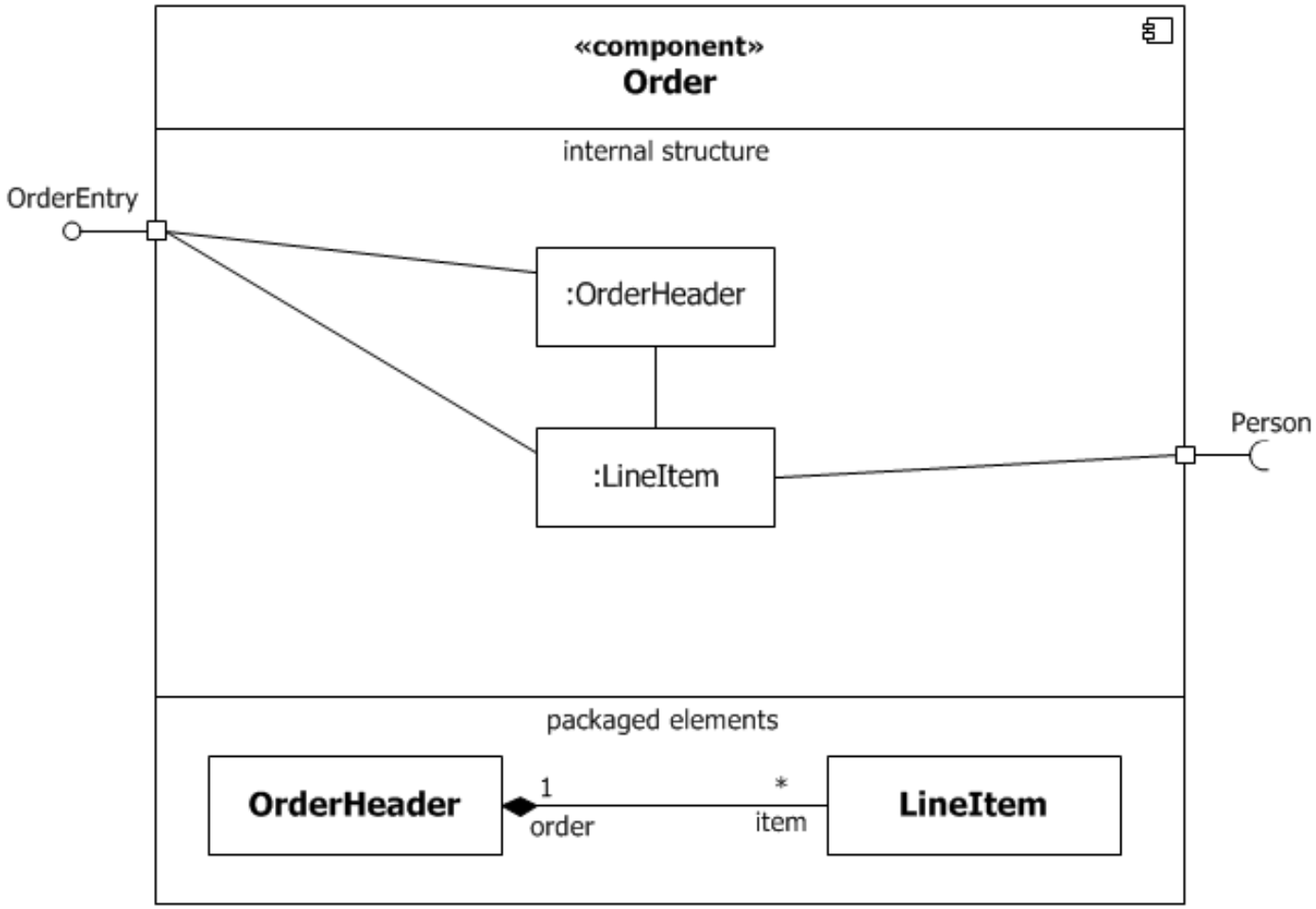
Compartimento de elementos empaquetados



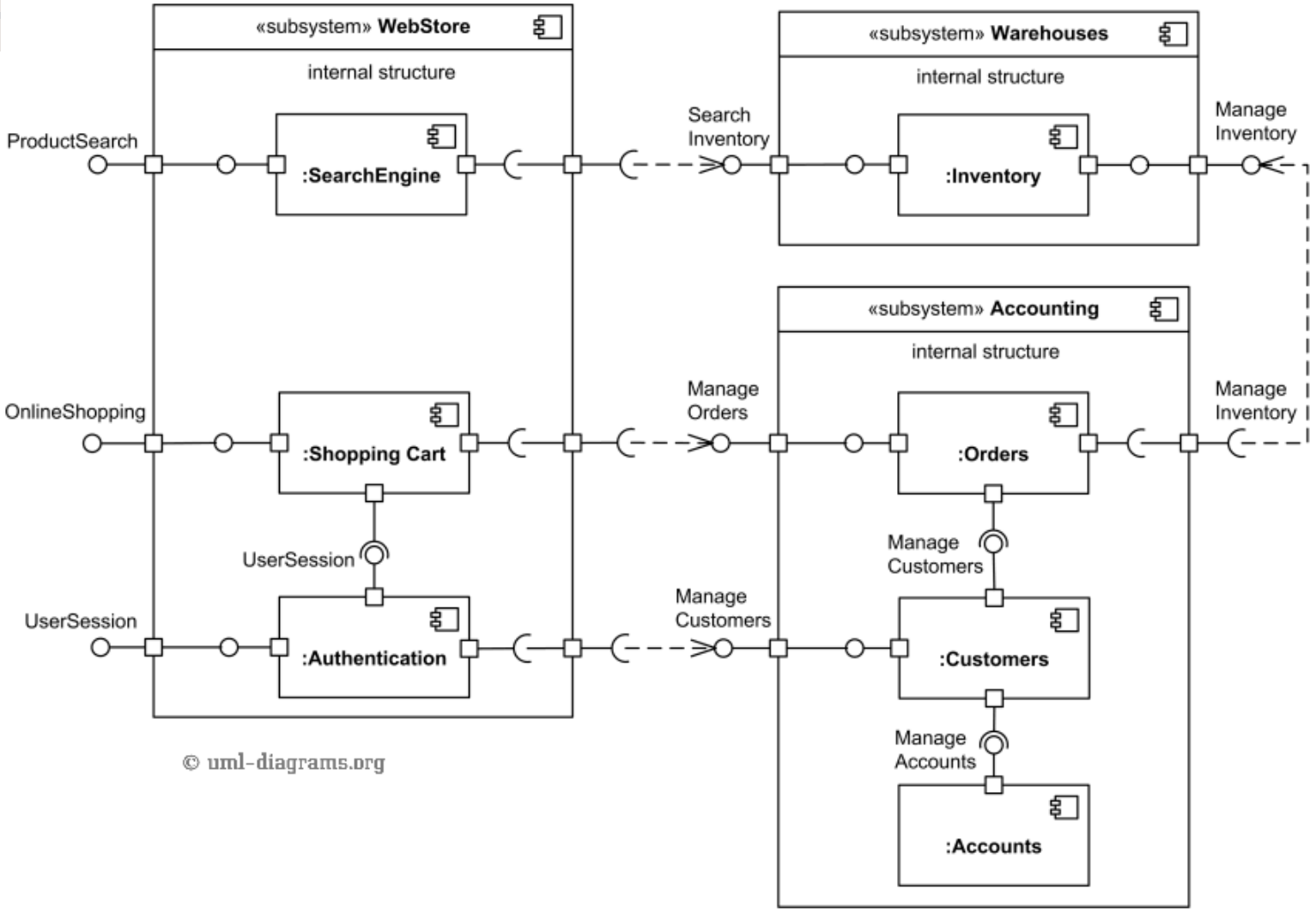
Compartimento de estructura interna



Los dos compartimentos

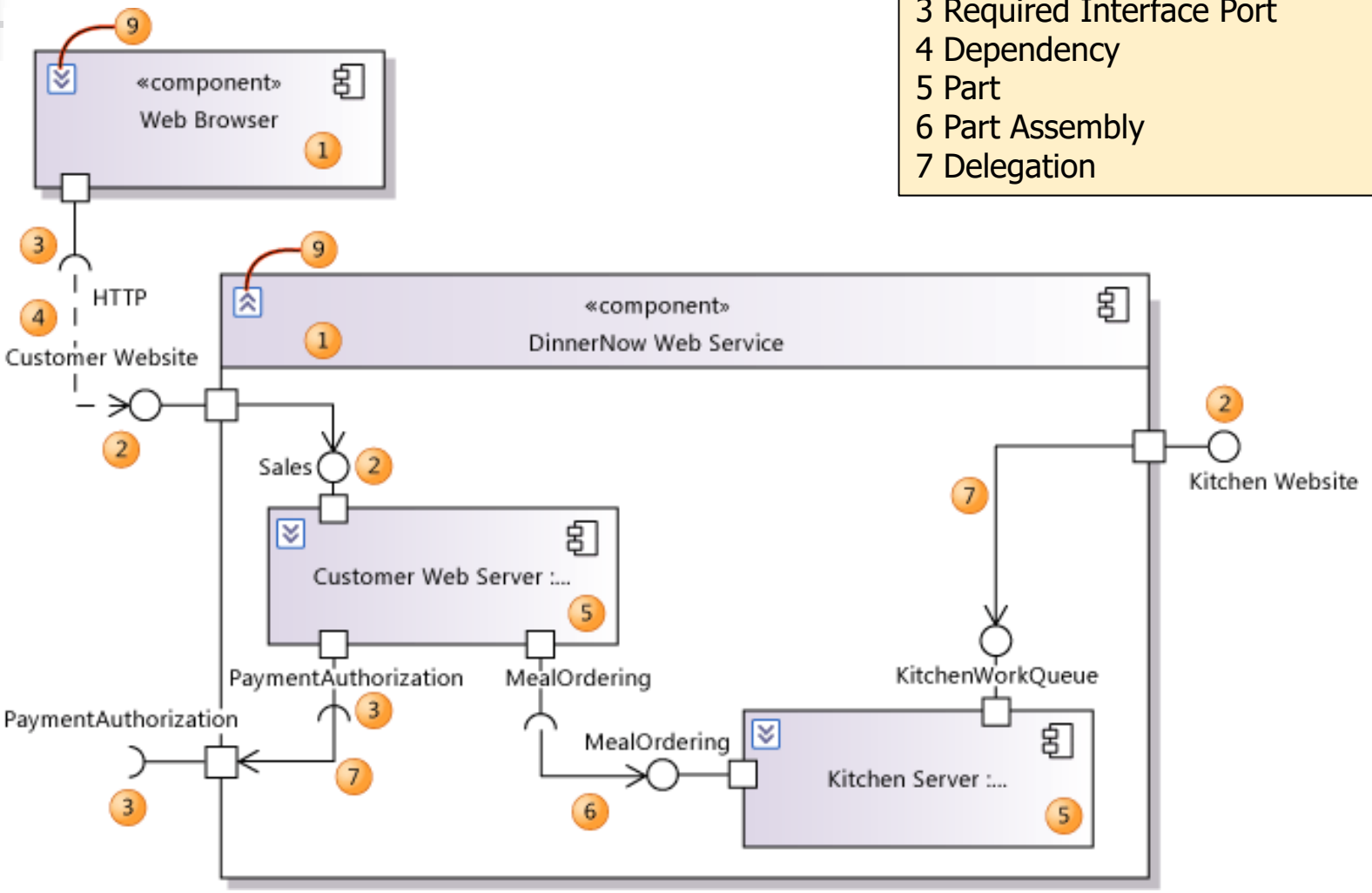


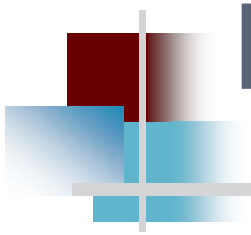
Interconexión y subsistemas



Microsoft

1 Component
2 Provided Interface Port
3 Required Interface Port
4 Dependency
5 Part
6 Part Assembly
7 Delegation

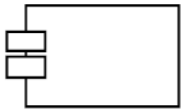
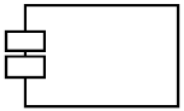
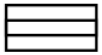
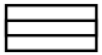
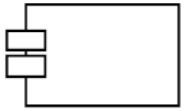
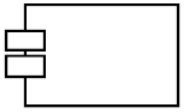
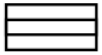
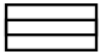
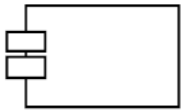
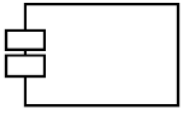
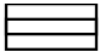
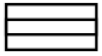
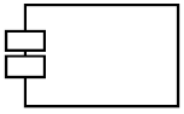
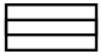
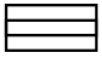
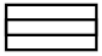


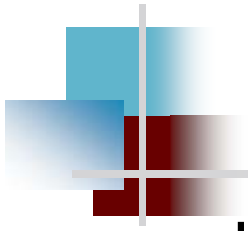


4.1.1. Partición en componentes

- Subsistemas como componentes

Descomposición

Level 1 Sub-system	Level m Module	(...)	Level n-1 Component	Level n Class / Object
				
				
				
				
				
				
				
				
				

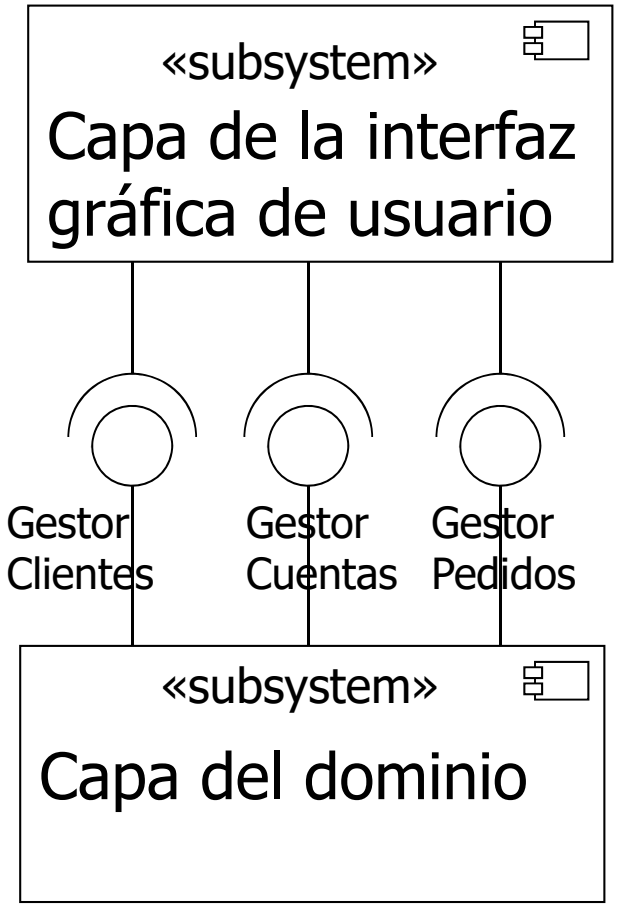


Subsistemas

- Un subsistema es un componente que actúa como unidad de descomposición de un sistema para dividirlo en porciones manejables
- Es básicamente un paquete UML
- Hay que poner en el mismo subsistema aquellas clases/componentes que:
  - Se encuentran en la misma área de interés (mismos objetivos o conceptos)
  - Pertenecen a la misma jerarquía
  - Comparten casos de uso!
  - Están asociadas fuertemente

Subsistemas

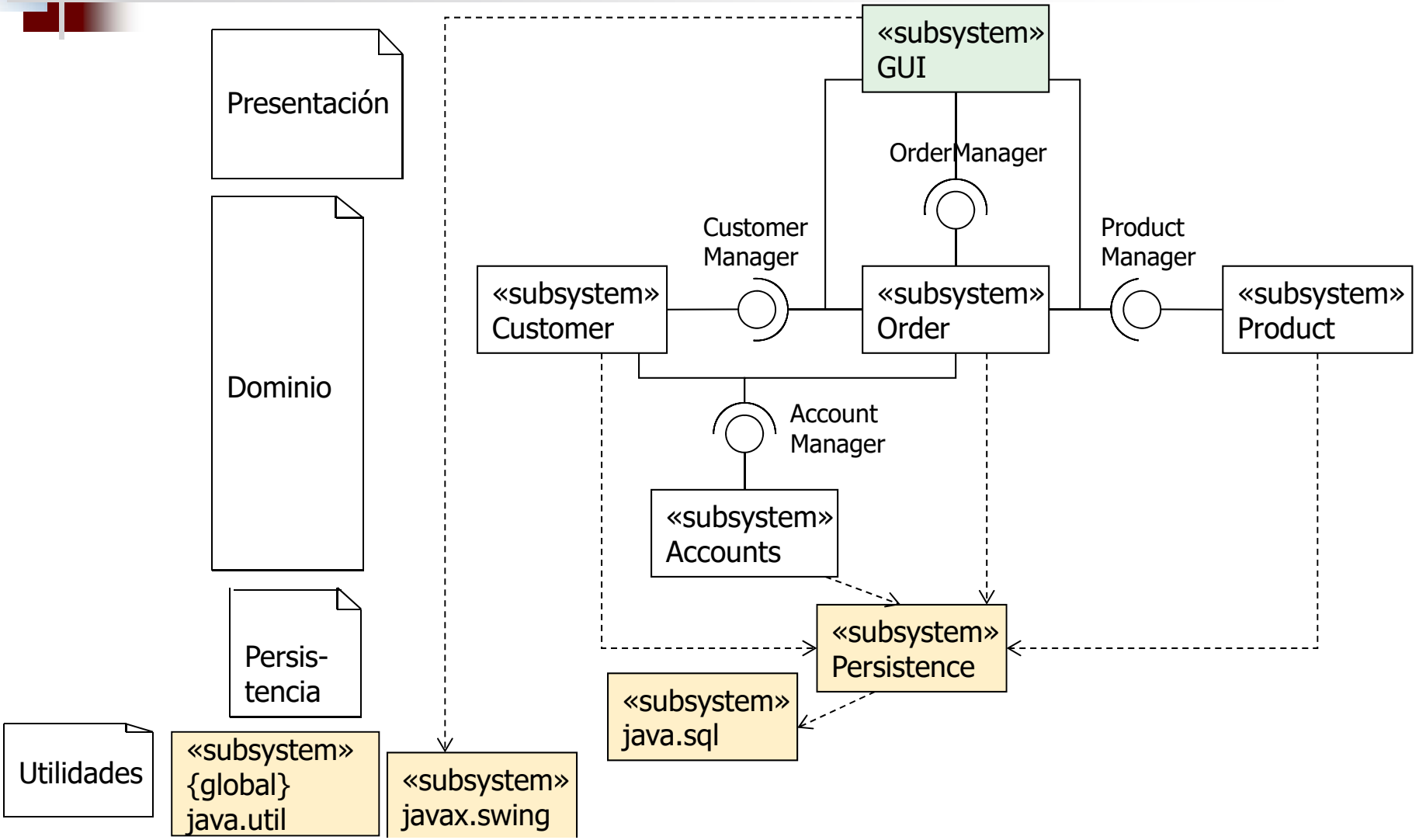
- No se pueden crear instancias en tiempo de ejecución pero sí de sus partes contenidas
- Pueden conectarse a otros subsistemas a través de sus interfaces para crear una arquitectura



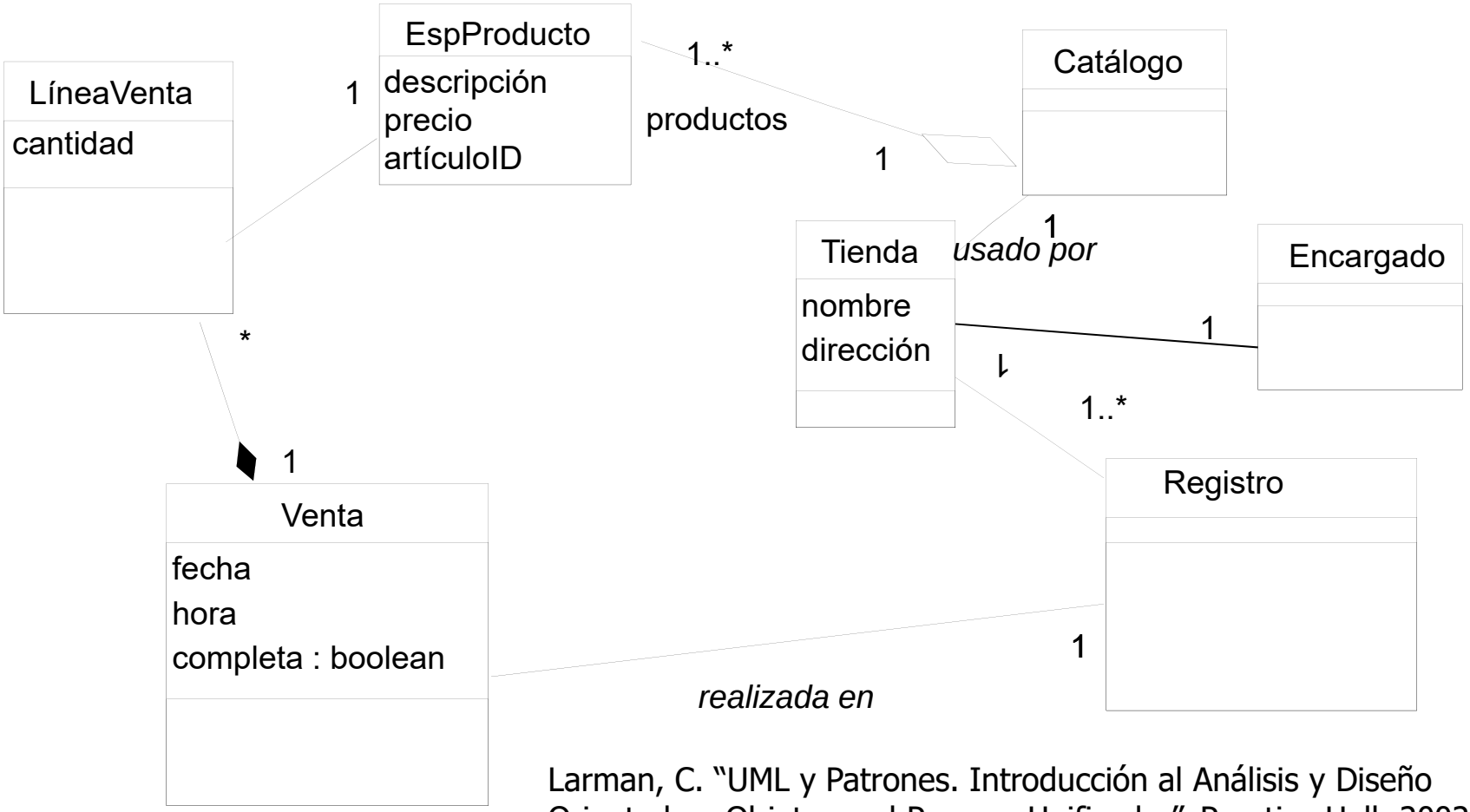
Arquitectura del sistema

- Subsistemas e interfaces forman la arquitectura física
- Se organizan las interfaces y subsistemas para crear la arquitectura:
  - Los subsistemas están dispuestos según el patrón arquitectónico capas
  - Cada capa contiene subsistemas de diseño: Capa de presentación, capa de Dominio (Lógica de negocio), capa de Servicios, capa de Utilidades …
  - Las dependencias entre capas se definen cuidadosamente
  - Las dependencias van en una sola dirección y a través de interfaces

Arquitectura en capas



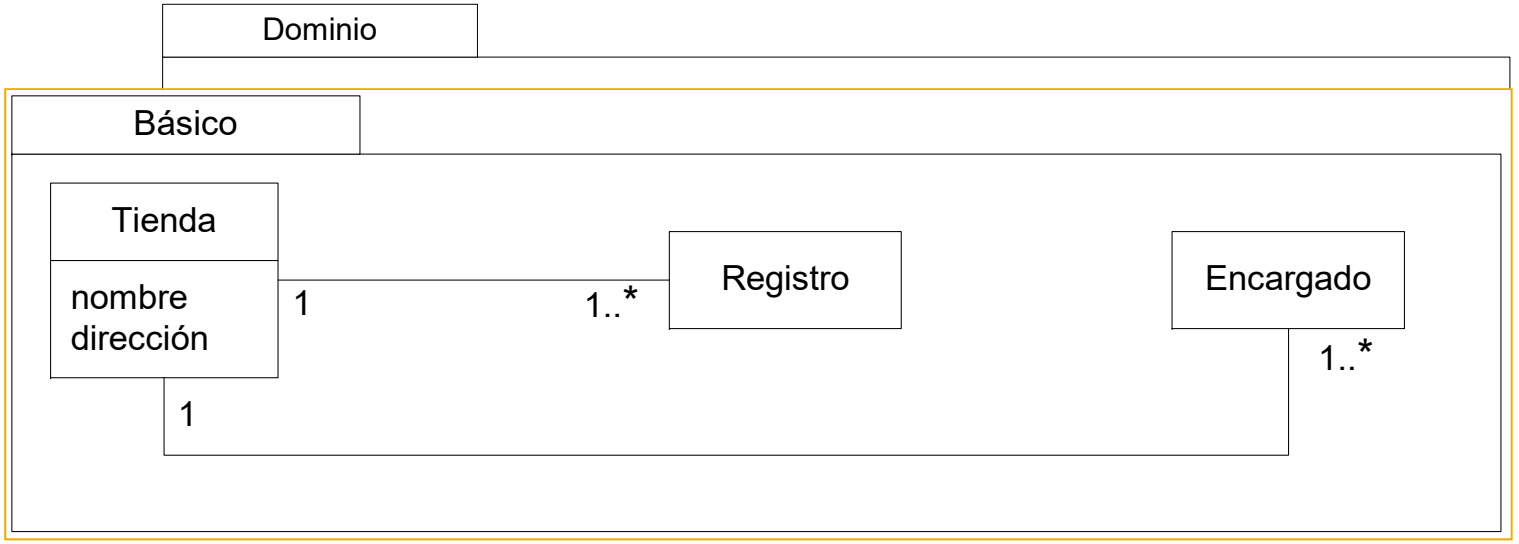
Ejemplo de Tienda (Larman)



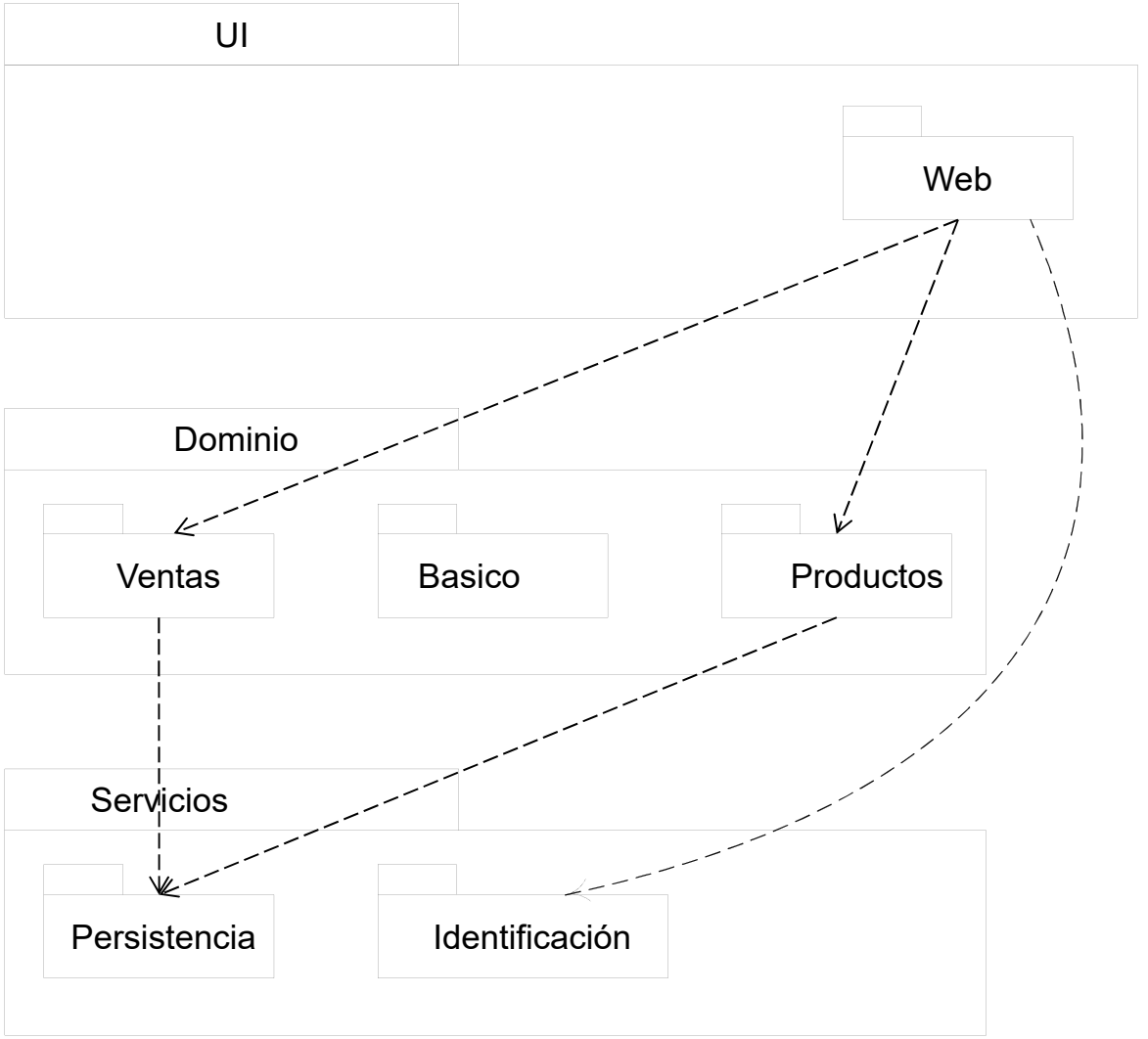
Larman, C. "UML y Patrones. Introducción al Análisis y Diseño Orientado a Objetos y al Proceso Unificado ". Prentice Hall, 2002

¿Cómo se organizan los subsistemas?

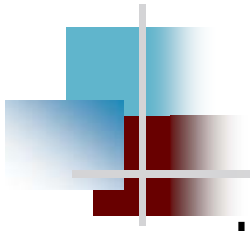
- Es conveniente dividir el modelo que representa todo el dominio
  - Un paquete con los elementos comunes -> subsistema
  - El resto de los paquetes -> subsistemas



Subsistemas de diseño



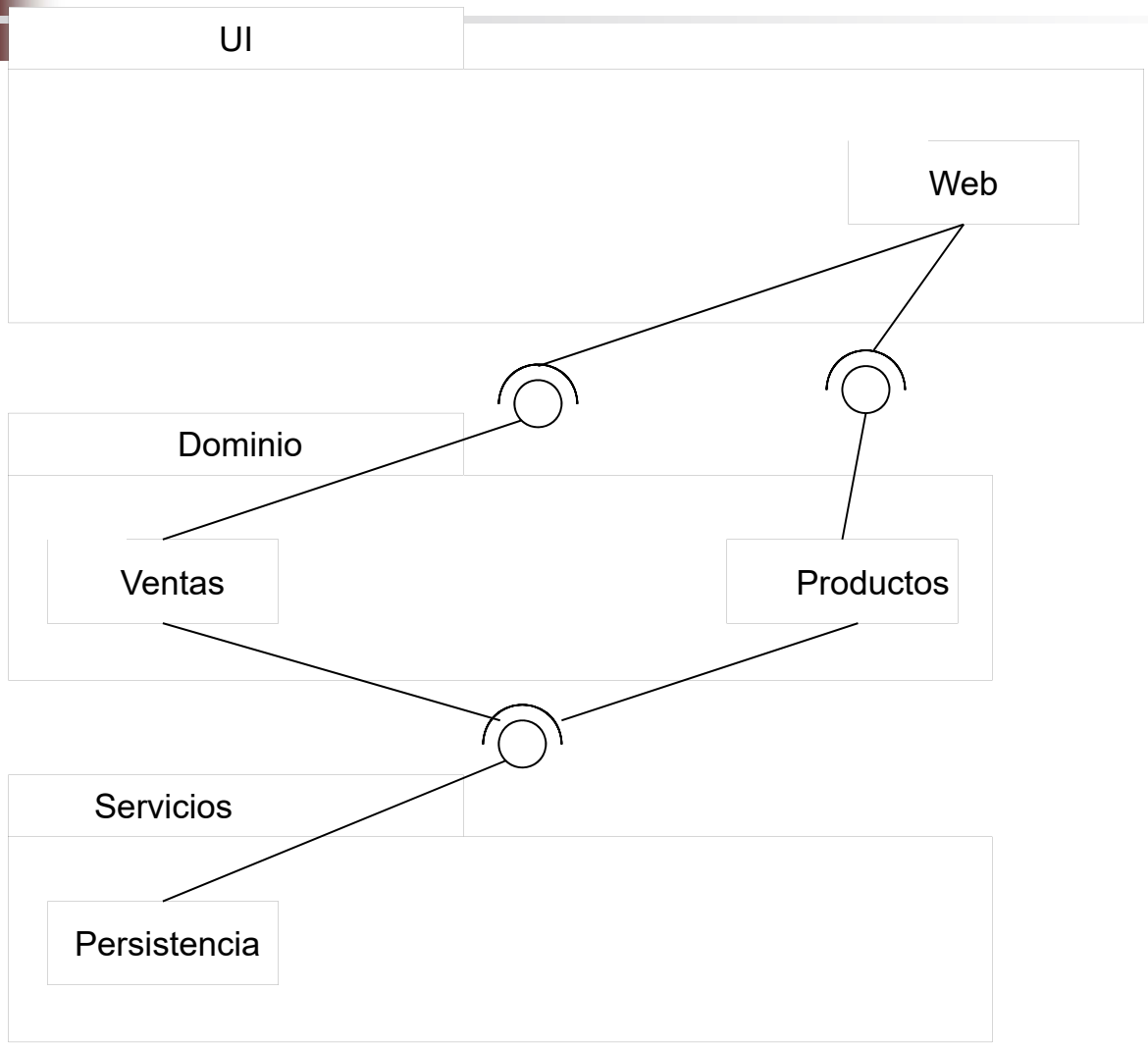
La Interfaz de usuario y el controlador (de fachada)



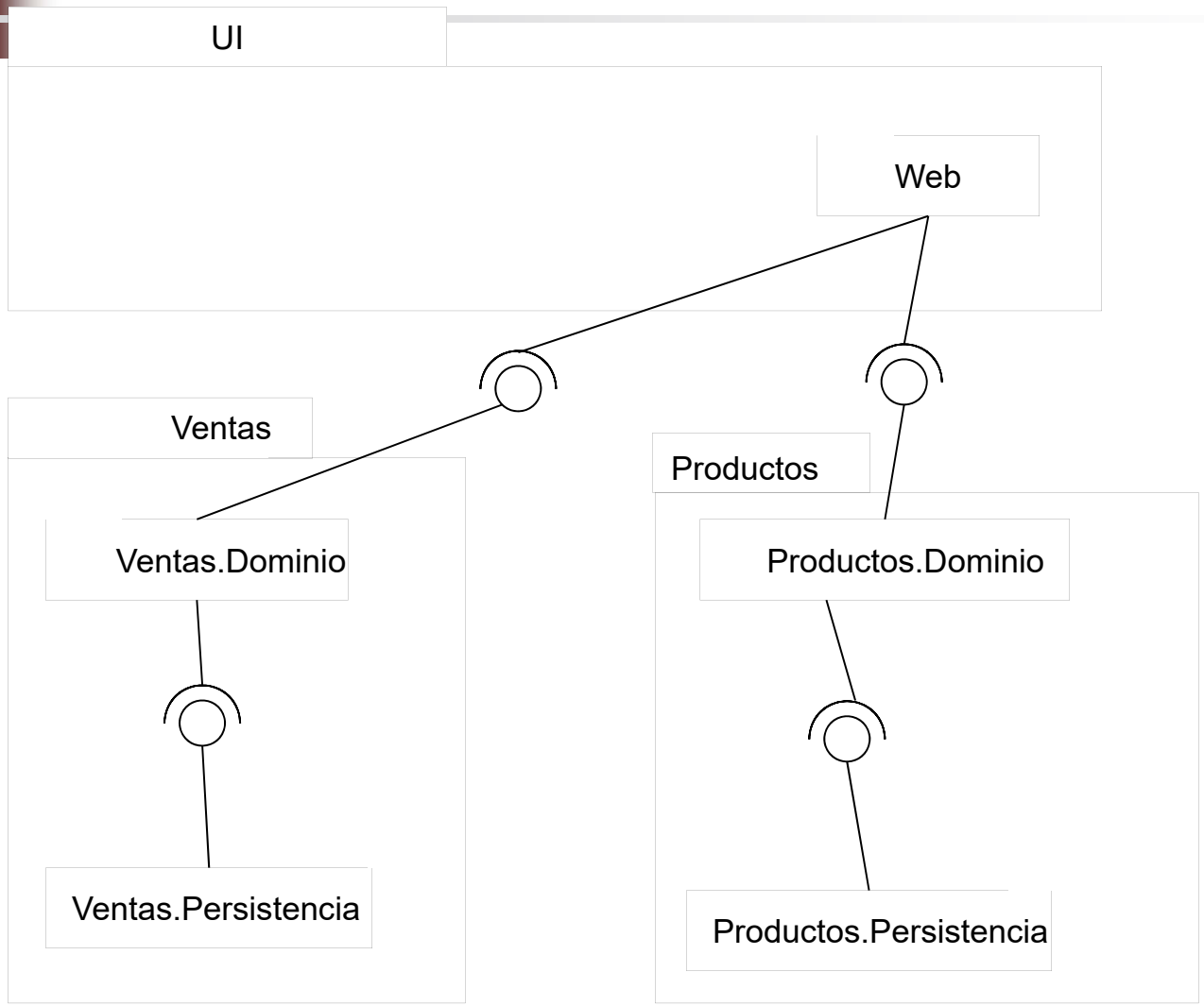
Transformar en componentes

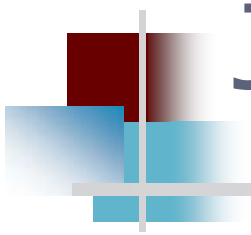
- Las dependencias entre subsistemas se pueden implementar con conectores
- Utilizar el patrón Fachada!
  - Las interfaces se pueden utilizar para crear las "costuras" de un sistema:
  - Identificar las partes cohesivas del sistema
  - Empaquetar éstas en un subsistema
  - Definir una interfaz para cada subsistema

Dependencias -> interfaces



Dependencias -> interfaces



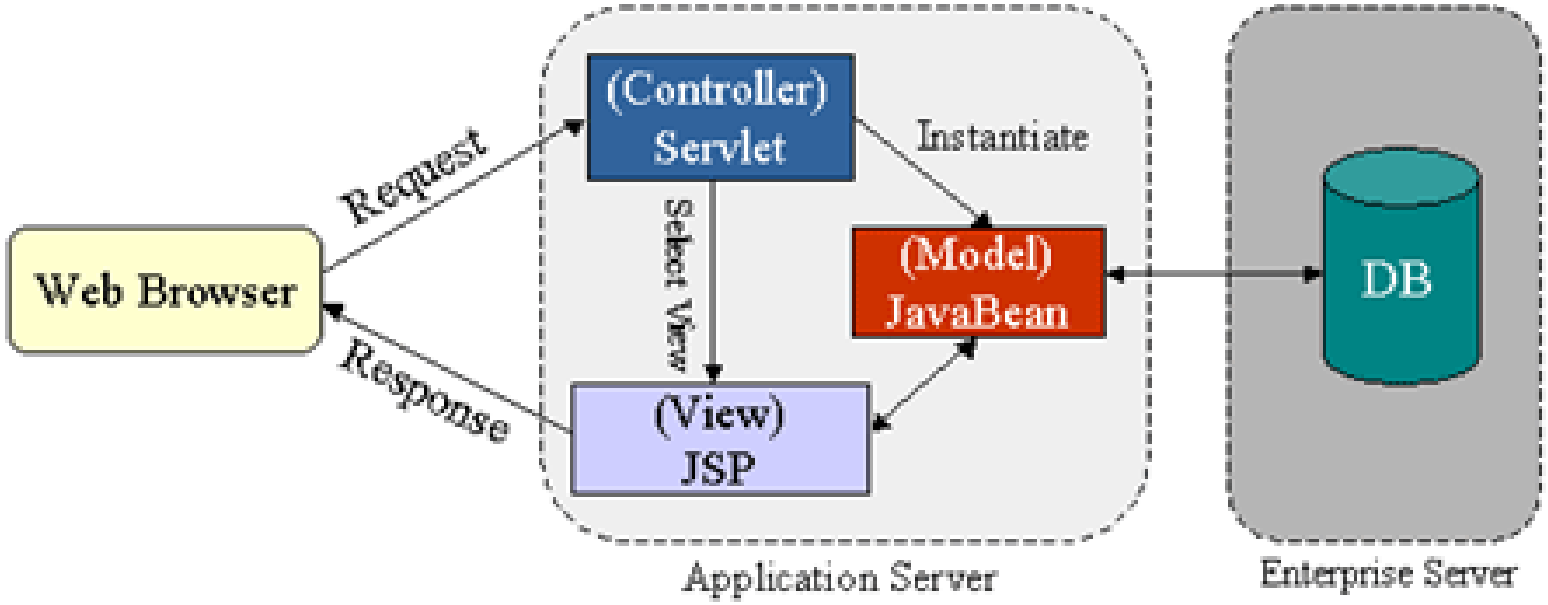


4.2 Diseño de componentes con JEE

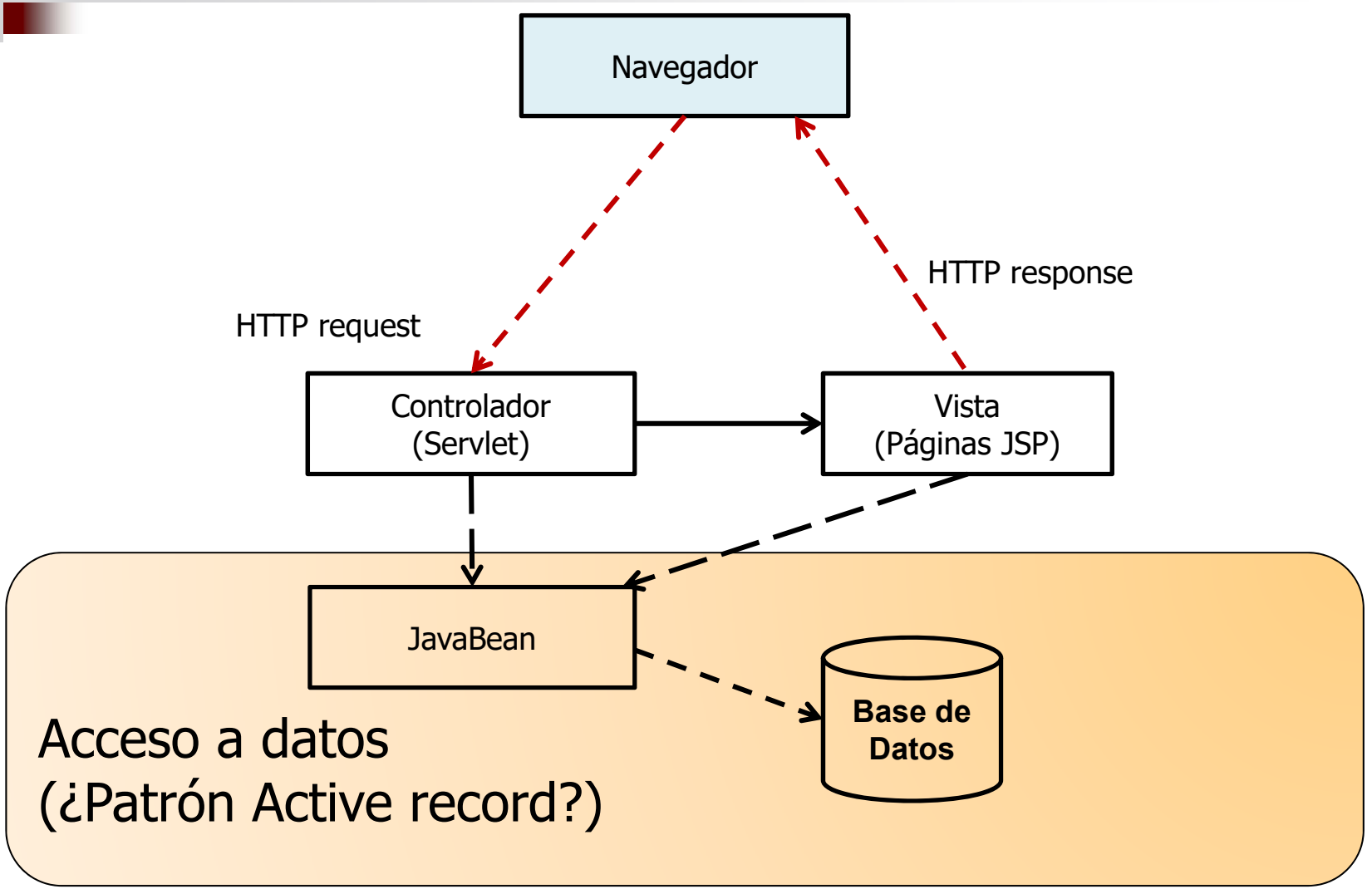
- [MVC/JavaBean frente a MVC/JEE]
- Diseño en capas
- Proceso de desarrollo: tareas de diseño

El modelo 2 de JSP

- MVC y JavaBean como modelo

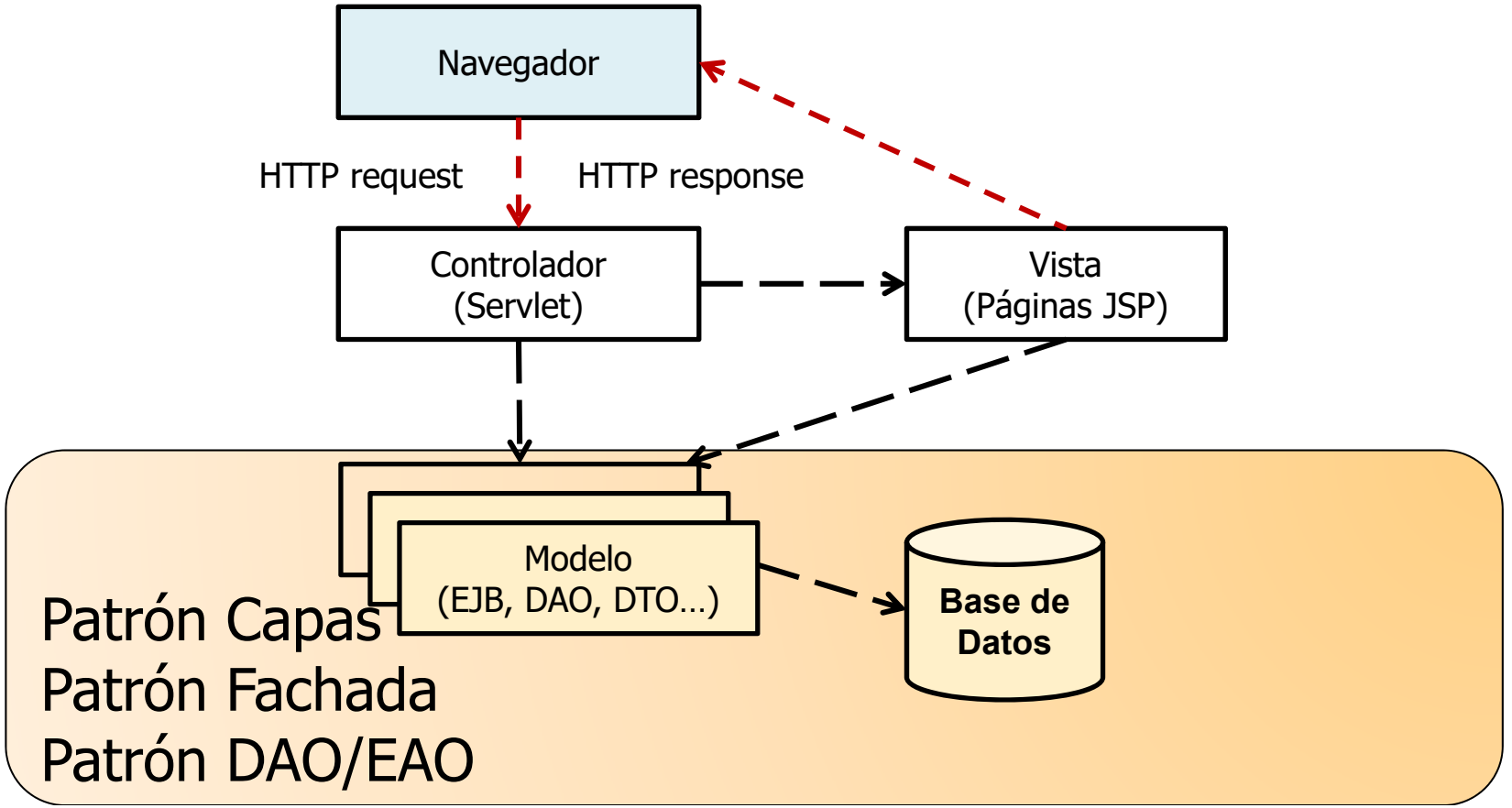


Modelo Vista Controlador y JavaBeans



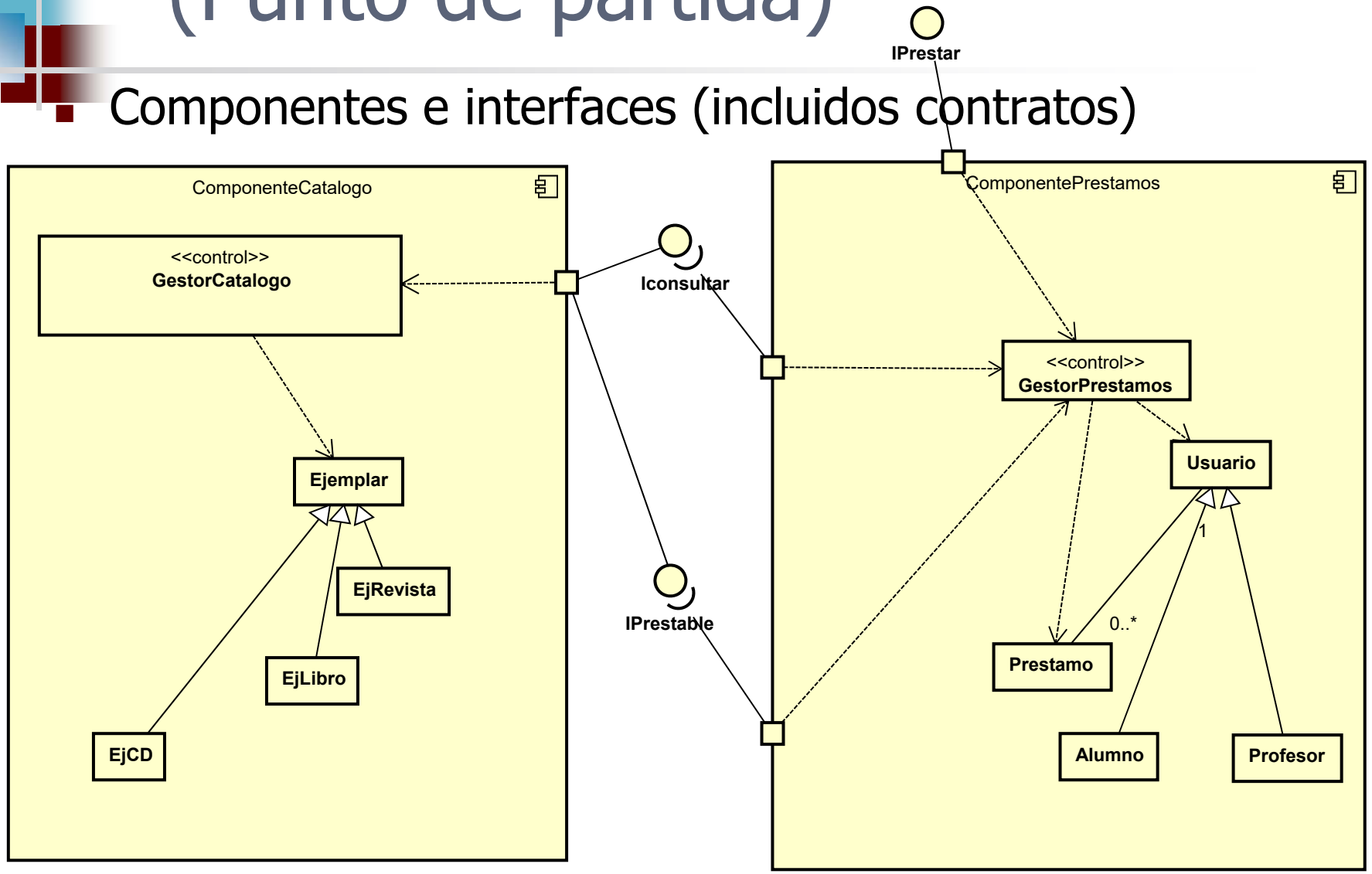
Modelo Vista Controlador/JEE

- Cada componente especifícado se diseña en varias capas



Especificación de componentes (Punto de partida)

- Componentes e interfaces (incluidos contratos)

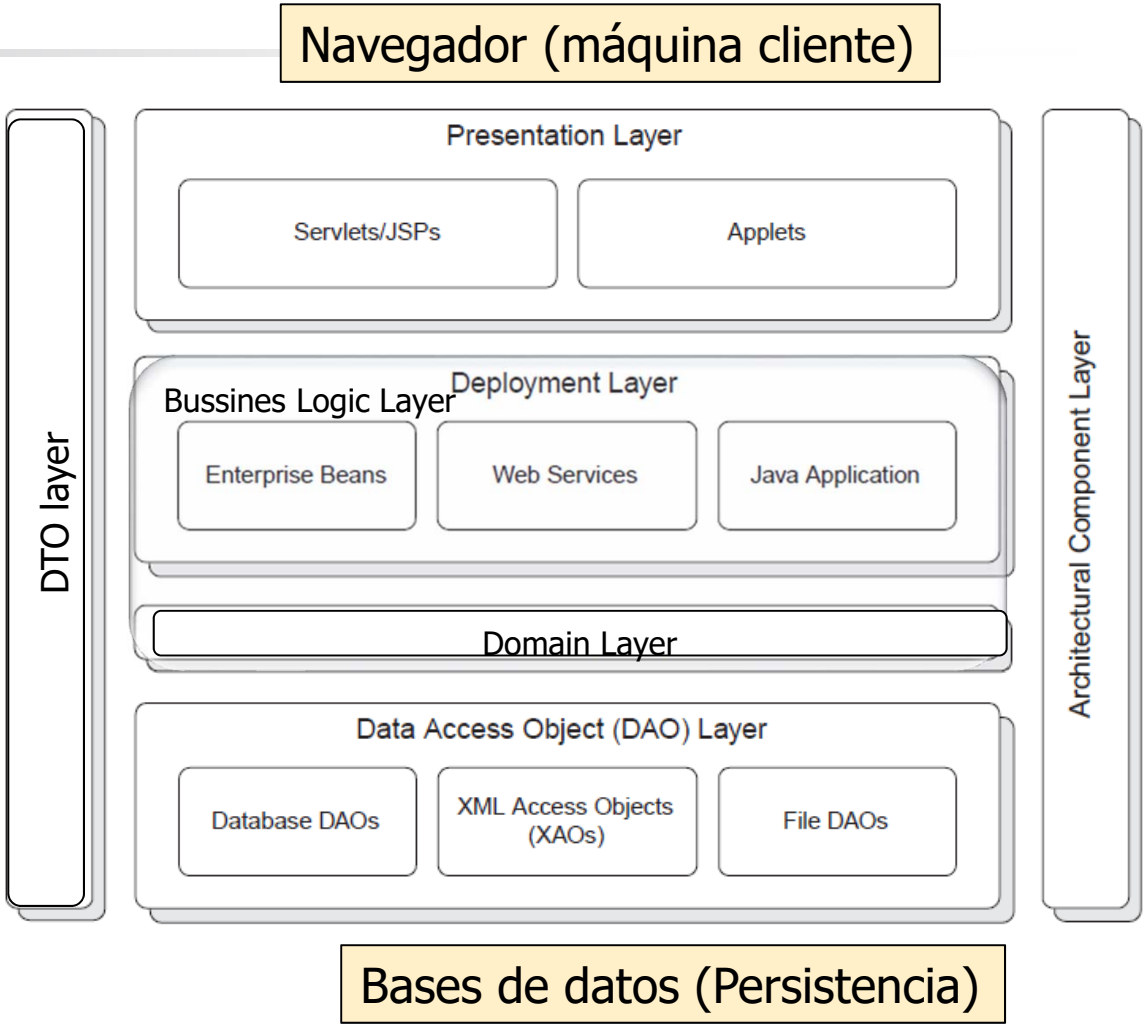


Tareas de desarrollo: Diseño de componentes JEE

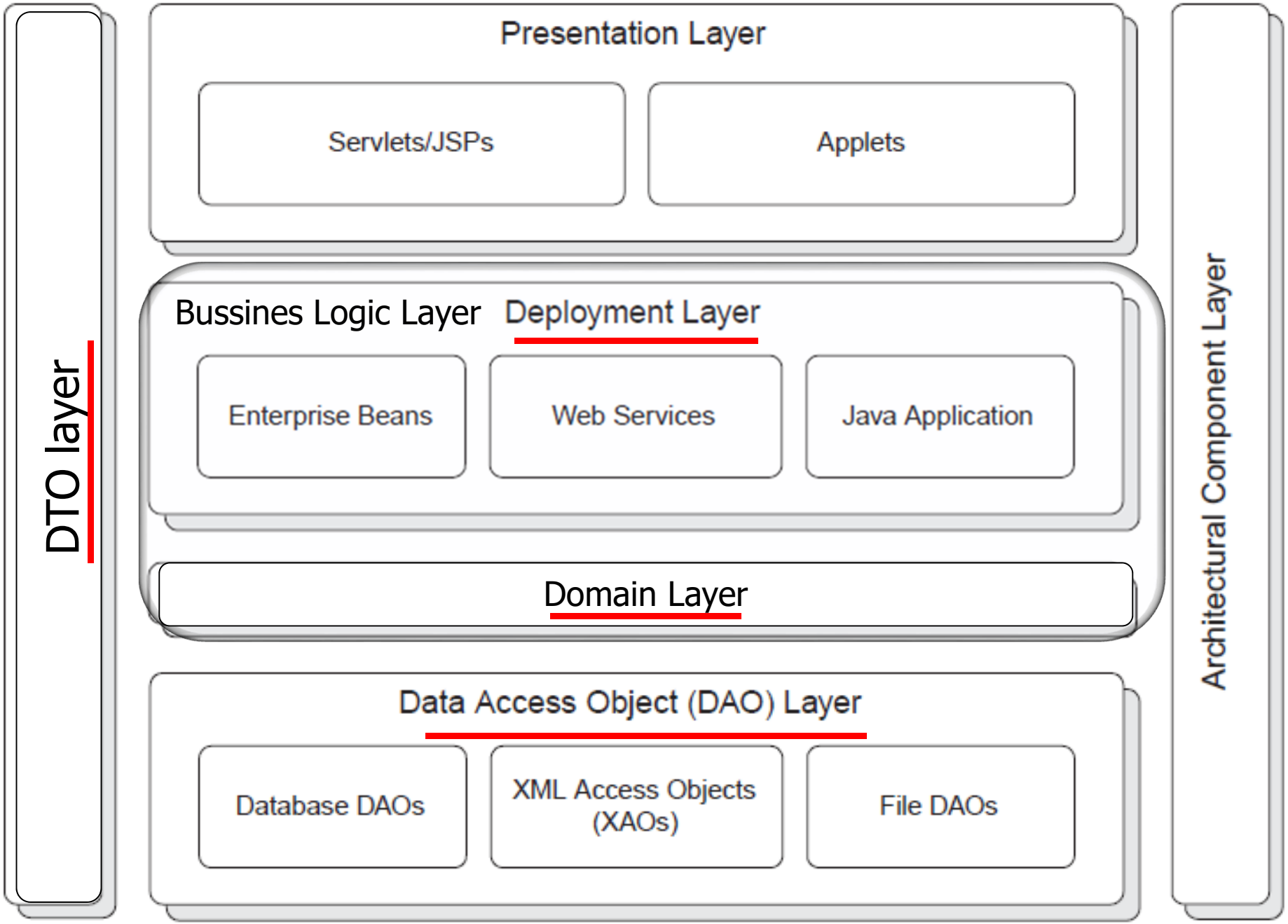
- Requisitos, Modelado del dominio, Análisis de casos de uso (Diagramas de secuencia)…
- Diseño de la Base de Datos
- Especificación (UML) de los componentes
- Diseño de los componentes (capas)
- Diseño de la arquitectura de red
- Codificación y pruebas unitarias
  - Clases DAO [EAO] y DTO [”Entity”]
  - Clases de dominio (con clases de prueba) y controlador (EJB de sesión, implementación e interfaces)
  - Capa de Presentación
- Pruebas de Sistema
- Pruebas de aceptación del usuario
- Actividades de implantación

Capas JEE

- Basado en C/S pero…
- Cambian los nombres, aparecen más capas



- Fuente: Ashmore, J2EE Architects Handbook y Fowler,Patterns of EAA


DTO layer
Presentation Layer
Servlets/JSPs
Applets
Bussines Logic Layer
Deployment Layer
Enterprise Beans
Web Services
Java Application
Domain Layer
Data Access Object (DAO) Layer
Database DAOs
XML Access Objects (XAOs)
File DAOs
Architectural Component Layer

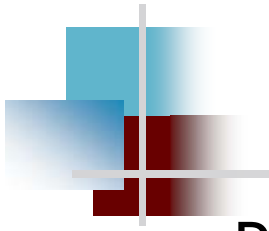
Capas JEE

Capa	Rol
Capa de presentación	La capa que controla lo que se mostrará al usuario final.
Lógica de negocio: **Capa de Despliegue	Publica las operaciones de las clases de dominio
**Capa de Dominio	Procesa la lógica y las reglas del negocio: clases de dominio u objetos de negocio (business objects)
Capa de acceso a datos	Gestiona la lectura, escritura, actualización y borrado de datos almacenados. Contiene código JDBC/JPA, pero también podría ser utilizado con XML o archivos.
Capa de componentes arquitectónicos	Componentes genéricos de la capa de aplicaciones genéricas. Candidatos para su uso en toda la empresa.
Capa de objetos de transferencia de datos	Estructuras de información ligeras (DTO, data transfer objects) relacionadas con las clases de dominio

Capa de acceso a datos

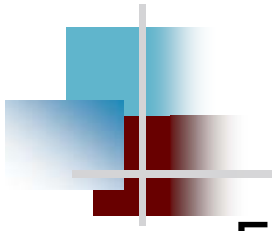
- Mediante clases de tipo Data Access Objects (DAO) u otro patrón equivalente
- Gestiona el acceso al almacenamiento persistente de algún tipo.
  - base de datos relacional, archivos, documentos XML
- El acceso de datos independiente permite cambiar las fuentes de datos y compartir el acceso a datos entre distintas aplicaciones.
  - Es probable que las empresas tengan varias aplicaciones que utilizan los mismos datos

Lógica de negocio: Capa de Dominio



- Proviene de las clases del dominio y las reglas de negocio
  - Las clases del dominio deberían estar separadas de las capas DAO y despliegue, para maximizar la posibilidad de reutilización.
- Suelen utilizar y coordinar varios DAOs
- Son comunes patrones como adaptador, compuesto, estrategia, etc.

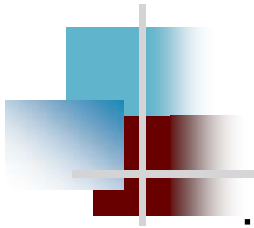
Lógica de negocio: Capa de despliegue



- Específica de la arquitectura JEE
- Incluyen EJBs (mensaje y sesión) y/o servicios Web
- Se puede tener un servicio Web y un EJB para la misma capa de dominio.

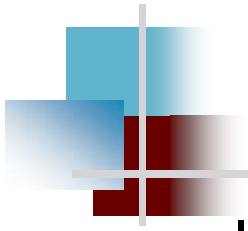
Tipos de despliegue

Feature	EJB	Web Services	Messaging/ JMS	RMI	HTTP	CORBA
Caller platform requirements	Java-compliant only	Any	Any	Java-compliant only	Any	Any
Communication method supported	Synch. only	Both	Both	Synch. only	Synch. only	Synch. only
Coupling	Tight	Loose	Loose	Tight	Loose	Loose
Transaction support	Local and JTA	Local and JTA	Local	Local	Local	Local
Requires J2EE container?	Yes	No	No	No	No	No
Supports clustering for scalability and availability?	Yes	Yes	Yes	No	Yes	Yes



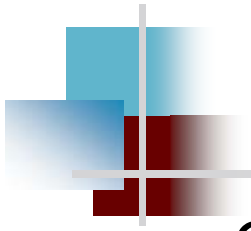
Tipos de despliegue

- ¿La clase de dominio es utilizada por páginas HTML dinámicas, servlets, JSP?
  - Pueden aislarse con algún tipo de despliegue en lugar de ser utilizados directamente por la capa de presentación
  - Suele implementarse con un EJB de sesión.
- ¿La clase de dominio requiere soporte de transacciones?
  - Requiere JTA, proporcionado por contenedores JEE
  - Puede implementarse con un EJB de sesión tipo "Stateful"
- ¿La clase de dominio recibe y procesa mensajes JMS?
  - Implementado con un EJB de mensaje
- ¿La clase de dominio es invocada desde aplicaciones no Java?
  - Se puede implementar como un servicio Web.



Capa de presentación

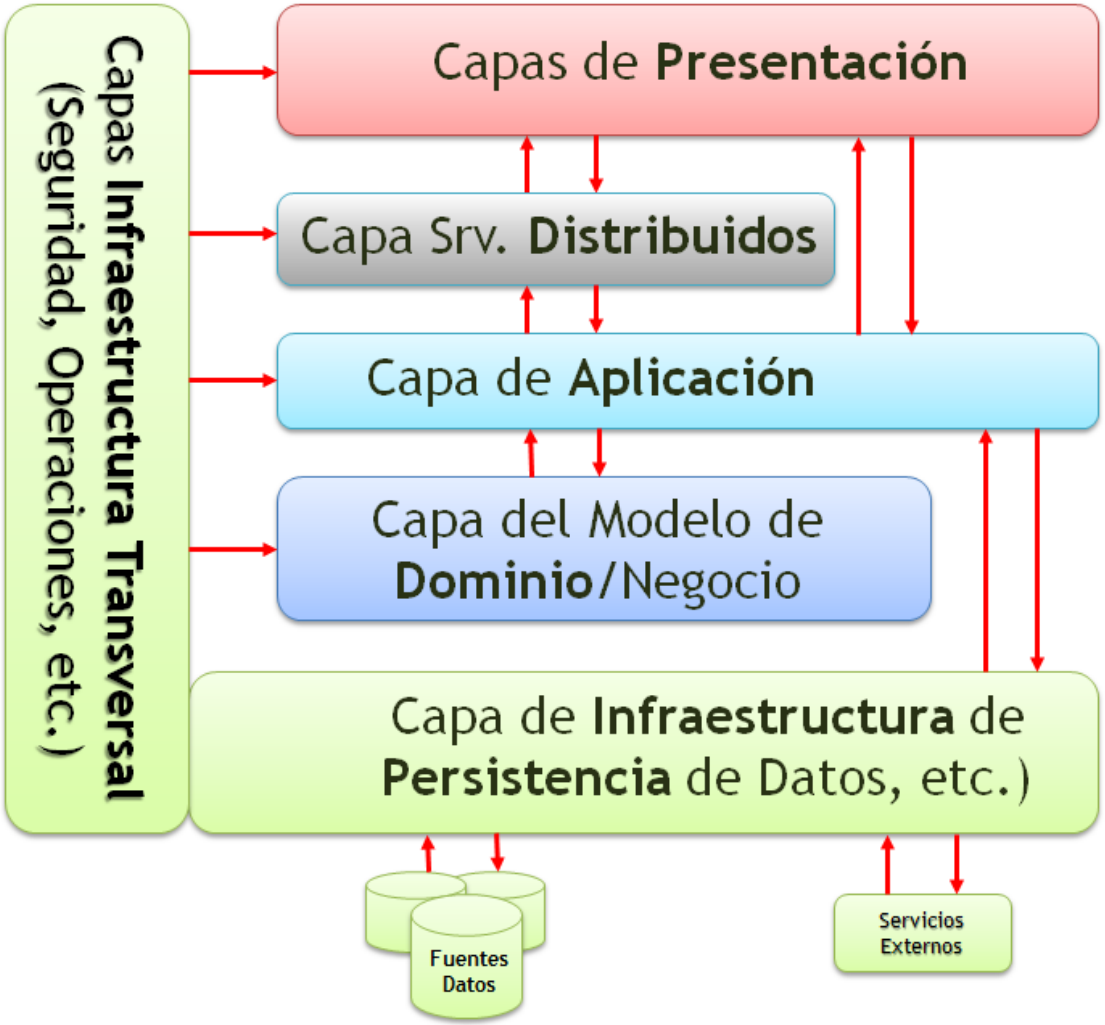
- la sección de la aplicación responsable de todo lo que los usuarios finales ven físicamente en la interfaz de usuario
  - JEE soporta interfaces HTML/Javascript.
  - JEE produce interfaces HTML usando una combinación de páginas HTML estáticas y dinámicas, generadas por servlets y JSP.
  - Puede estar en una máquina distinta del contenedor JEE (interfaz remota)



Capas auxiliares

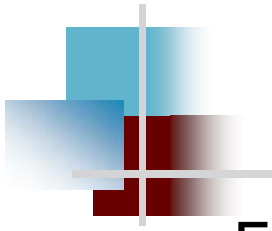
- Capa de objetos de transferencia de datos (data transfer objects)
  - Estructuras de datos útiles (nombre y dirección de un cliente para imprimir etiquetas)
  - UML <<DataType >>
  - Patrón DTO “data transfer object”
- Capa de componentes arquitectónicos
  - Componentes de terceros, utilidades comunes, etc.

Alternativa Microsoft



Tareas de desarrollo: Diseño de componentes JEE

- Requisitos, Modelado del dominio, Análisis de casos de uso (Diagramas de secuencia)…
- Diseño de la Base de Datos
- Especificación (UML) de los componentes
- Diseño de los componentes (capas)
- Diseño de la arquitectura de red
- Codificación y pruebas unitarias
  - Clases DAO y DTO
  - Clases de dominio (con clases de prueba) y controlador (EJB de sesión, implementación e interfaces)
  - Capa de Presentación
- Pruebas de Sistema
- Pruebas de aceptación del usuario
- Actividades de implantación

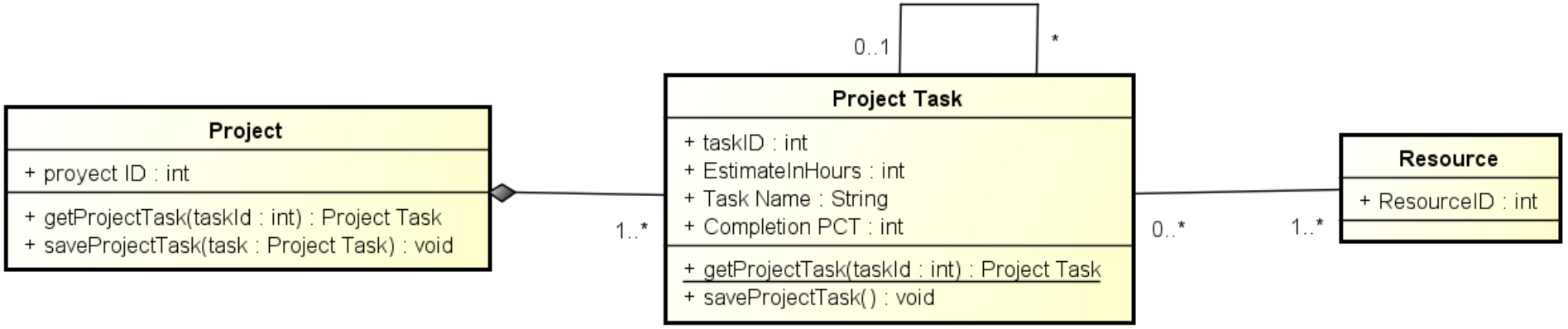


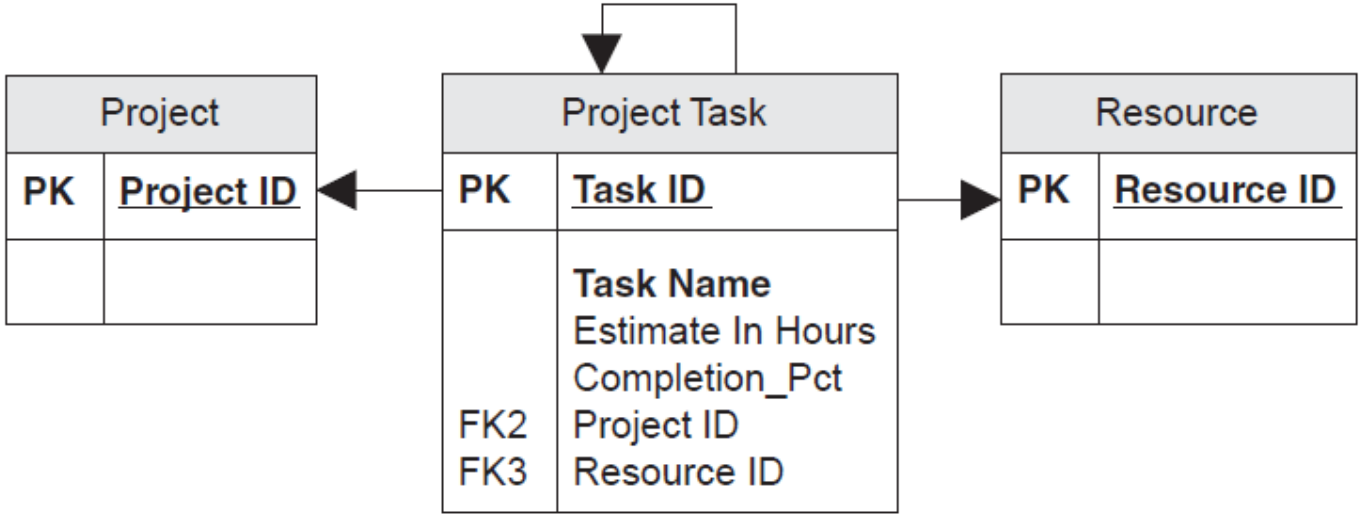
Ejemplo: Seguimiento de proyectos

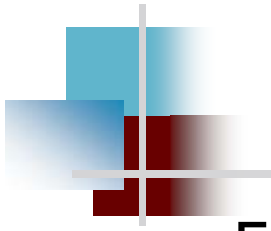
- El sistema permitirá a los usuarios definir, editar y mostrar las tareas de un proyecto.
- Cada tarea de proyecto tiene un nombre, una estimación (en horas), el porcentaje en el que está completada, un recurso de personal asignado, cualquier número de tareas dependientes, y una prioridad (alta / media / baja).

- (Se asume que el framework de presentación es MVC, mediante Servlets y JSPs)

Modelos de dominio/datos







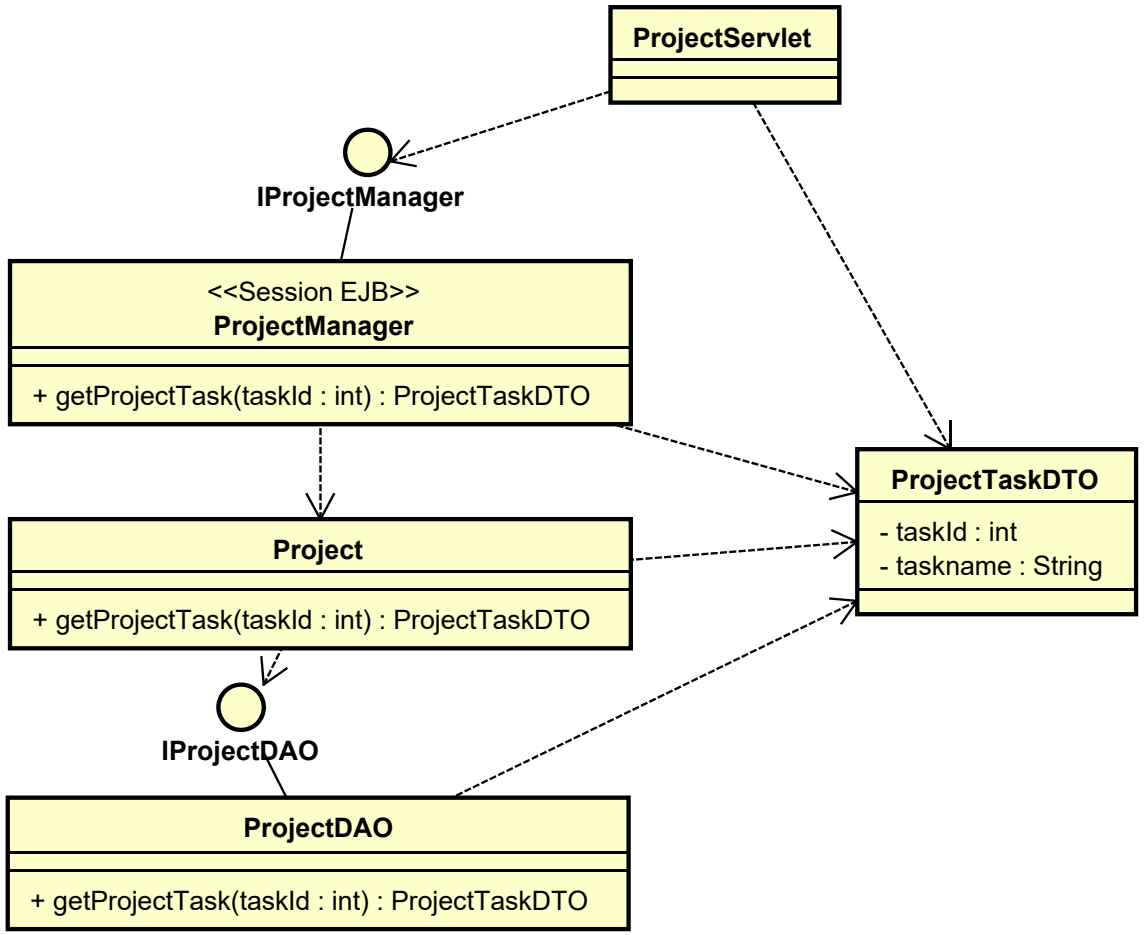
Proceso de diseño

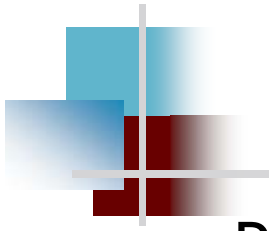
- Fases previas:
  - Casos de uso y Modelo de domino: Project
  - Diagramas de secuencia (y máquinas de estados): aparecen interfaces y clases de tipo controlador como ProjectManager Diagrama de componentes
- Capas y clases de diseño de cada componente ("Arquitectura estándar JEE")
  - De cada clase del dominio se derivan varias, una por cada capa: ProjectManager, Project, ProjectDAO, ProjectDTO
  - Se utilizan interfaces, especialmente para los EJB
  - Los métodos derivan de las operaciones del dominio y se incorporan a todas o parte de las clases de diseño

Clase Project, Operación getProjectTask()

Capas:

- Presentación
- Despliegue (EJB)
- Dominio
- Acceso a datos





Pistas

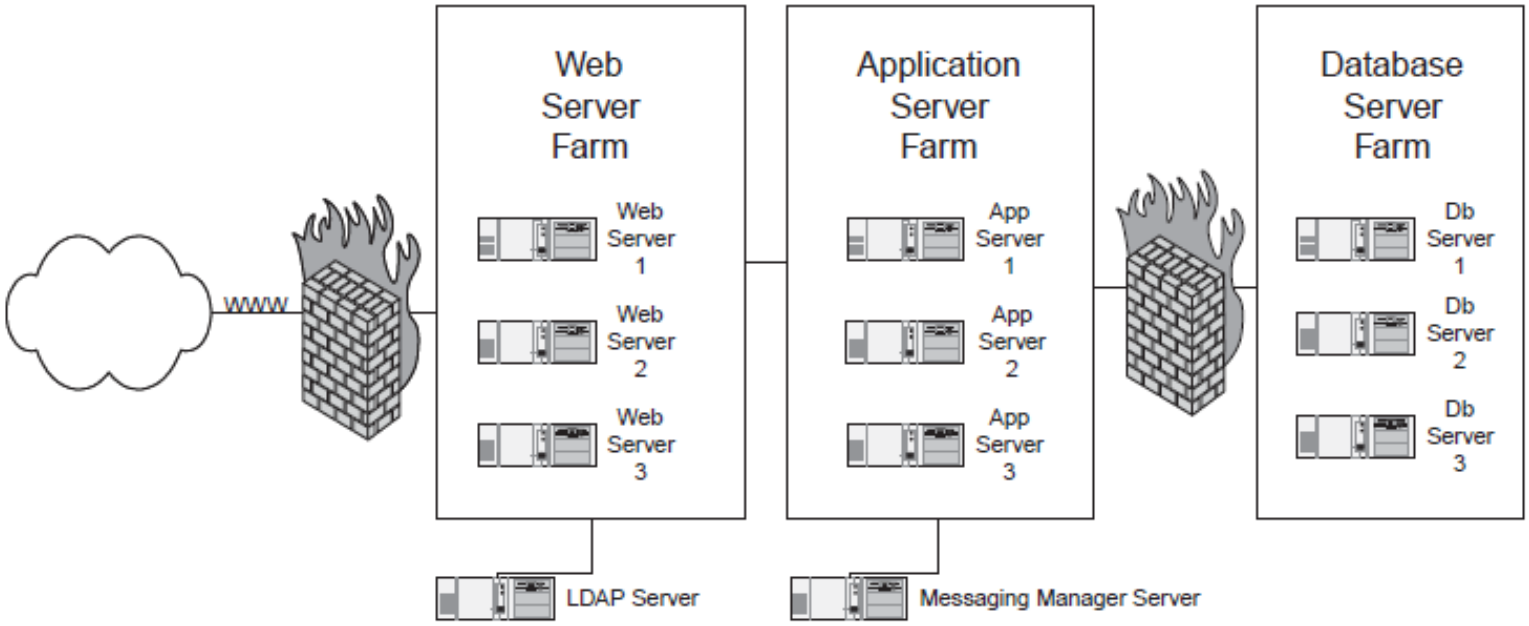
- Pedir y devolver estructuras de datos (DTO) en lugar de atributos individuales para aumentar el rendimiento
- Documentar cada Clase del dominio como un bloque en lugar de dividirlo en sus tres o cuatro componentes (EJB, DAO, DTO, etc.)
- Todos los atributos documentados tendrán implementados los métodos get y set
- No es necesario detallar los modelos completos con UML (las herramientas lo hacen mejor!)
- Se puede automatizar (Acceso a Datos mediante JPA) y simplificar las capas de Dominio y DTO

Arquitectura de la red

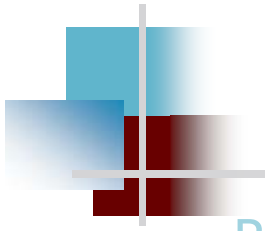
- El arquitecto es responsable de asegurarse de que las aplicaciones cumplan con la infraestructura de seguridad de la empresa.
- El arquitecto es responsable de la escalabilidad y la disponibilidad de las aplicaciones.
  - Escalabilidad se refiere a la capacidad de la aplicación para manejar un número cada vez mayor de usuarios.
  - El término alta disponibilidad describe una aplicación que está siempre disponible para su uso y tiene un mínimo tiempo de inactividad

Seguridad

- La seguridad en la mayoría de las empresas es una función centralizada y se trata como una cuestión de la infraestructura
  - Hay que aprovechar la infraestructura de seguridad tanto como sea posible
  - Hay que auditar el uso de identificadores genéricos (acceso a BD).



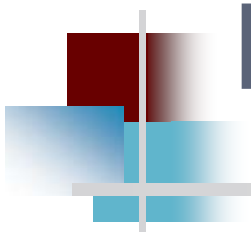
Implementación de componentes JEE



- Requisitos, Modelado del dominio, Análisis de casos de uso (Diagramas de secuencia)…
- Diseño de la Base de Datos
- Especificación (UML) de los componentes
- Diseño de los componentes (capas)
- Diseño de la arquitectura de red
- Codificación y pruebas unitarias
  - Clases DAO y DTO
  - Clases de dominio (con clases de prueba) y controlador (EJB de sesión, implementación e interfaces)
  - Capa de Presentación
- Pruebas de Sistema
- Pruebas de aceptación del usuario
- Actividades de implantación

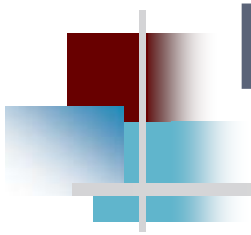
Tareas de desarrollo

ID	Task Name	Duration	Start	Finish	Resource Names	% Complete
13	**Coding**	46.75 days	Wed 7/2/03	Thu 9/4/03		0%
14	**DTO Objects**	30 days	Wed 7/2/03	Tue 8/12/03		0%
15	BaselineVO	2 hrs	Wed 7/9/03	Wed 7/9/03	Developer 1	0%
16	ProjectVO	2 hrs	Wed 7/2/03	Wed 7/2/03	Developer 1	0%
17	ProjectTaskVO	2 hrs	Wed 7/9/03	Wed 7/9/03	Developer 1	0%
21	**Data Access Layer with Test Classes**	15.5 days	Wed 7/2/03	Wed 7/23/03		0%
26	ResourceDAO	32 hrs	Tue 7/8/03	Mon 7/14/03	Developer2	0%
27	**Business Logic Layer with Test Classes**	25.75 days	Wed 7/2/03	Wed 8/6/03		0%

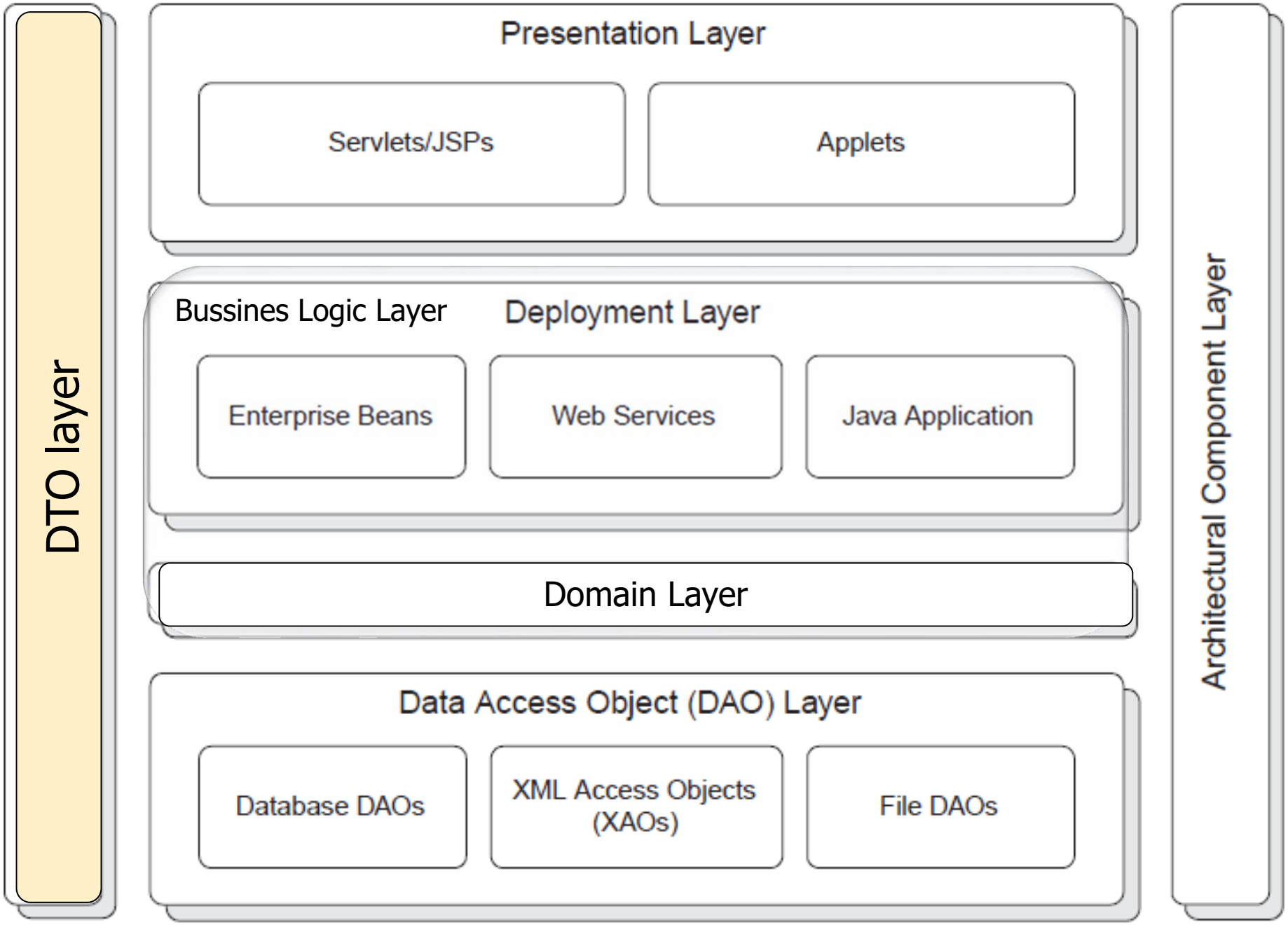


4.3 Diseño de las capas

- Patrones de diseño

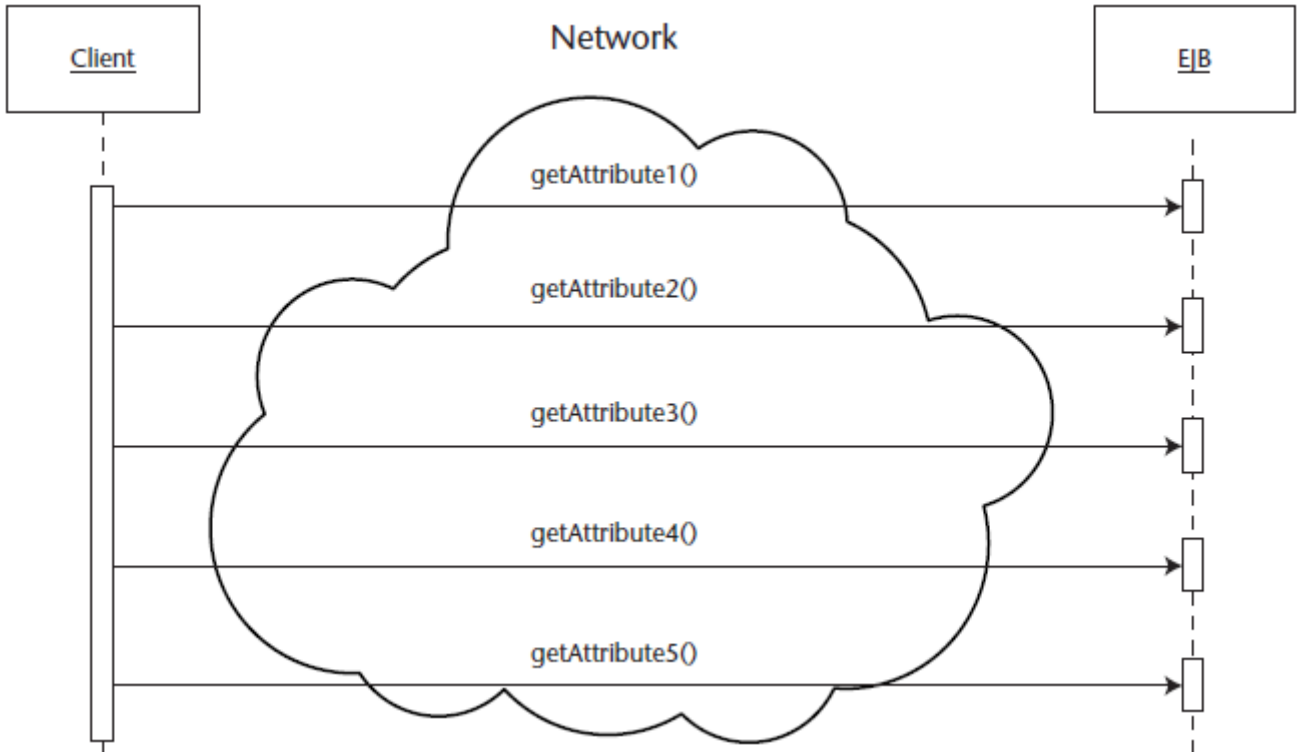


4.3.1 Patrón DTO

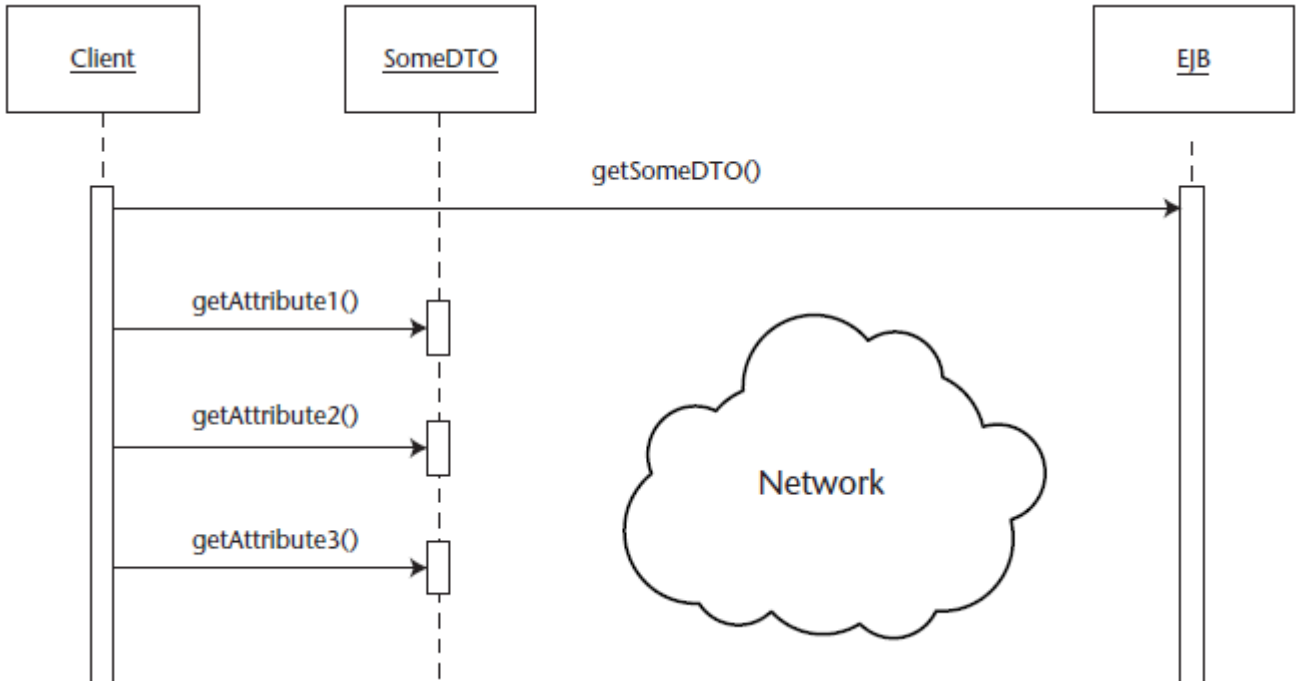

DTO layer
Presentation Layer
Servlets/JSPs
Applets
Bussines Logic Layer
Deployment Layer
Enterprise Beans
Web Services
Java Application
Domain Layer
Data Access Object (DAO) Layer
Database DAOs
XML Access Objects (XAOs)
File DAOs
Architectural Component Layer

Data Transfer Object , DTO
(Objeto de transferencia de datos)

- ¿Cómo puede un cliente intercambiar datos en bloque con el servidor sin hacer múltiples llamadas de red de grano fino ?



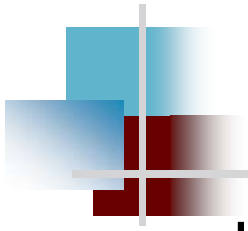
Patrón DTO: Solución



Patrón DTO: Solución

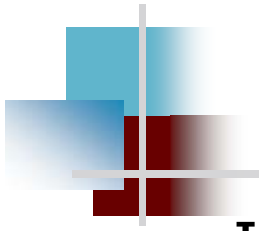
- Crear clases Java que encapsulan los datos en un paquete transportable por la red

```
import java.io.Serializable;
public class ClaseDTO implements Serializable {
    private long attribute1;
    private String attribute2;
    private String attribute3;
    ...
    public long setAttribute1();
    public String setAttribute2();
    public String setAttribute3();
    public long getAttribute1();
    public String getAttribute2();
    public String getAttribute3();
...}
```



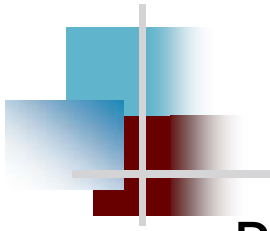
DTO

- Una clase ligera, que representa una estructura de datos (UML <<datatype>>) e implementa java.io.Serializable
  - EmpleadoDTO contiene el apellido, nombre, identificación de empleado, puesto de trabajo.
  - EmpleadoDTO se utiliza en todas las capas de la aplicación, a partir de la de presentación para el acceso a datos de la clase Empleado.
  - Es una cuestión de eficiencia/comodidad:
    - crearEmpleado(e: EmpleadoDTO);
  - En lugar de:
    - crearEmpleado(apellido: String, nombre: String, identificacionEmpleado: String, puestoTrabajo: String);



DTO: Consejos y Técnicas

- Implementar siempre java.io.Serializable
- Rellenar siempre todos los campos de un DTO
  - si sólo se rellena un subconjunto de los datos inevitablemente se provocarán errores (excepciones NullPointerException)
- Hacer que los DTOs sean auto-descriptivos
- También se pueden usar arrays o colecciones de DTOs
- Considerar métodos que redefinan **`equals()`**



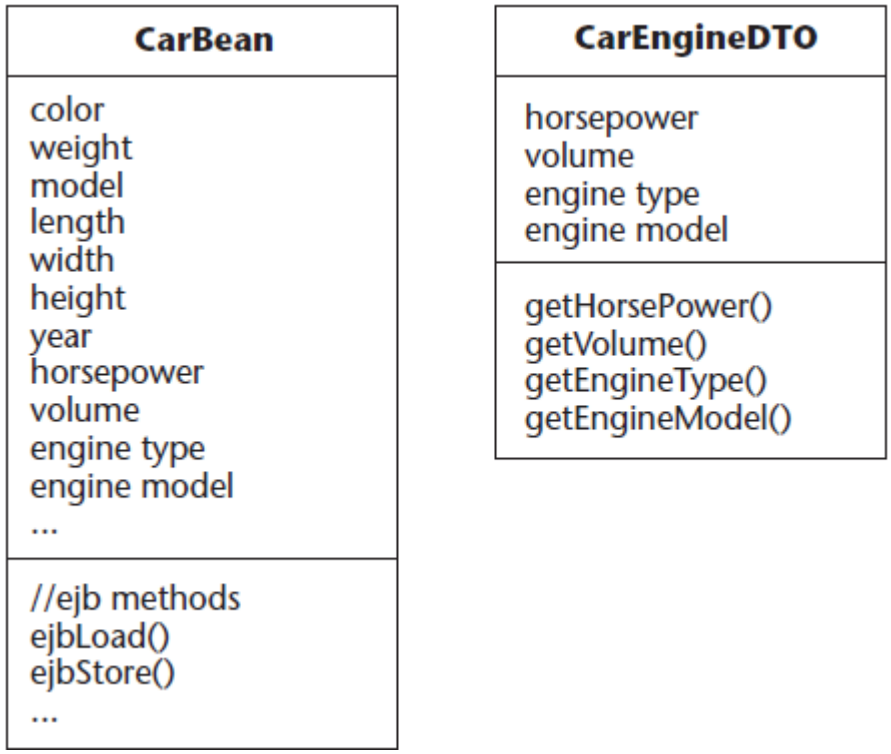
Errores comunes

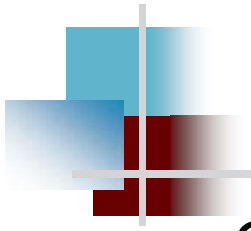
- Rellenar DTOs de forma inconsistente.
  - Poblados con conjuntos de valores diferentes según el uso
- Usar una cadena en blanco para evitar excepciones NullPointerException.

- Dos variantes:
  - DTO de dominio
  - DTO personalizados

DTO personalizado

- Siempre se pueden diseñar DTOs que representen parte de un bean o incorporen varios en uno...





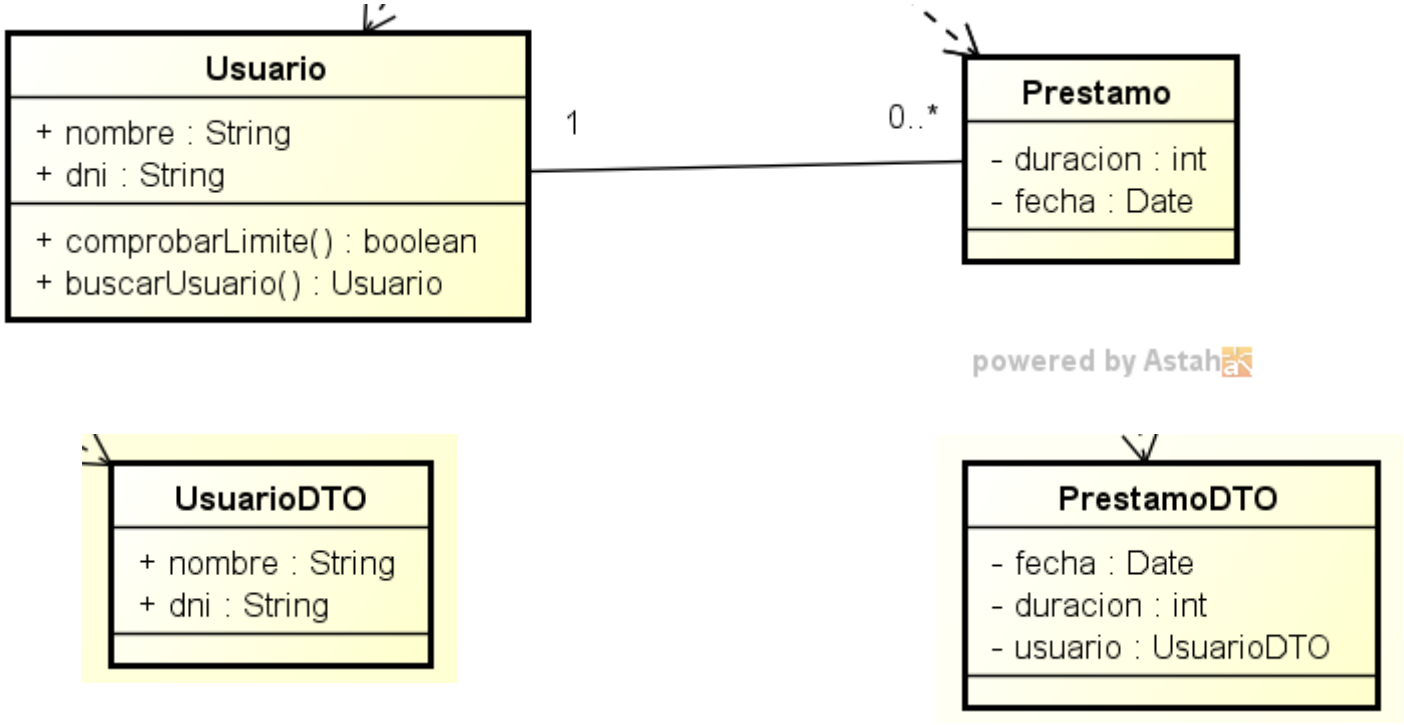
DTO de dominio: “Entities”

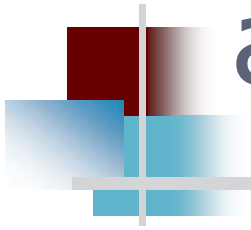
- Caso particular de DTO:
  - Cómo puede un cliente acceder y manipular los objetos de la capa de dominio en el servidor (clases “Entity”) sin la sobrecarga de rendimiento de llamadas remotas?
  - Con el patrón fachada, una clase del dominio no es accesible directamente

- Solución:
  - Diseñar copias DTO de los objetos de dominio del servidor (“Entity”) . Los clientes pueden operar sobre copias locales, mejorando el rendimiento de las lecturas y actualizaciones.

DTO de dominio: Solución

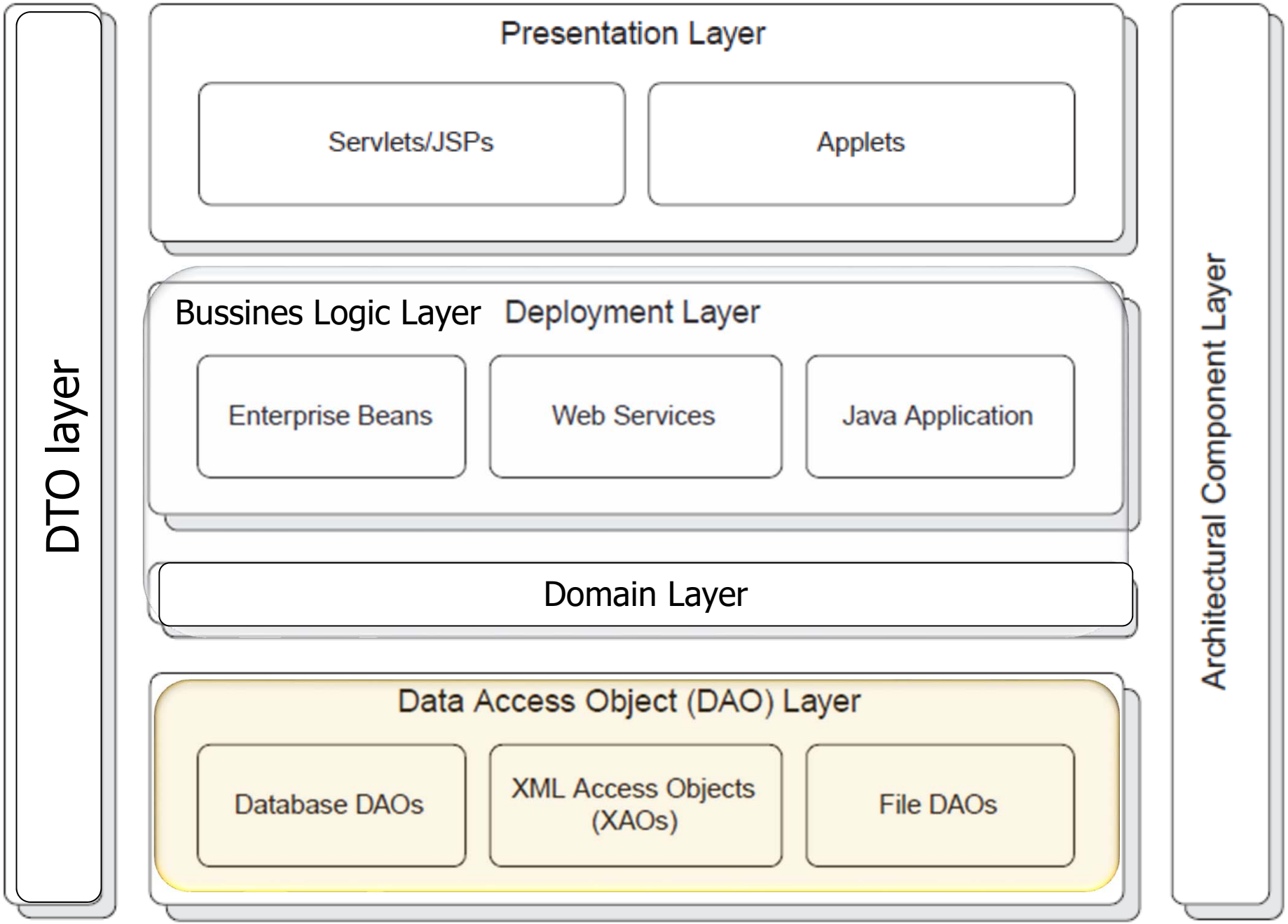
- DTO por cada clase de dominio
  - **`PrestamoDTO`** tendrá una ref. a un **`UsuarioDTO`**
  - Con paquetes diferentes, se puede usar el nombre original, **`Prestamo,Usuario`**

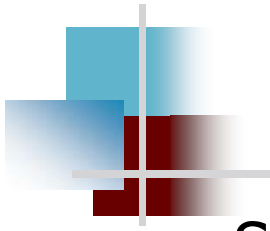




4.3.2 Patrones de la Capa de acceso a datos

- Patrón DAO/EAO
- [Mapper]


DTO layer
Presentation Layer
Servlets/JSPs
Applets
Bussines Logic Layer
Deployment Layer
Enterprise Beans
Web Services
Java Application
Domain Layer
Data Access Object (DAO) Layer
Database DAOs
XML Access Objects (XAOs)
File DAOs
Architectural Component Layer

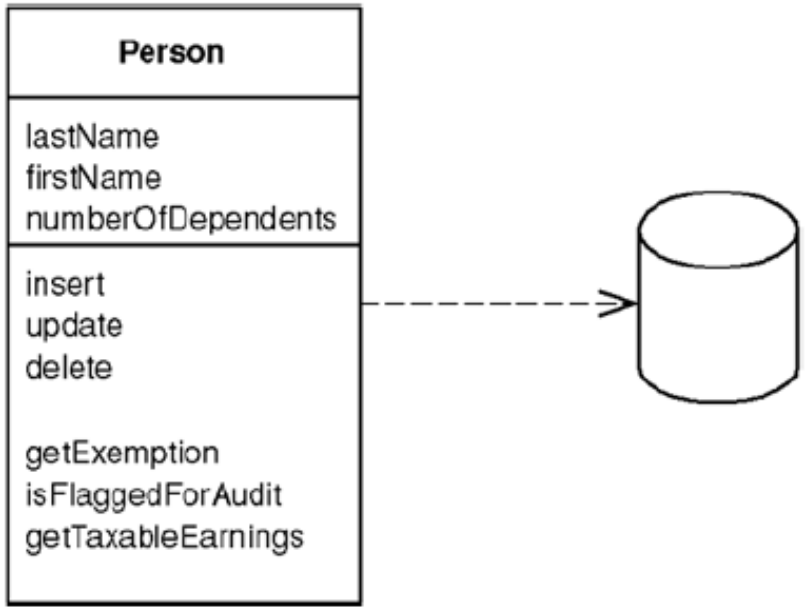


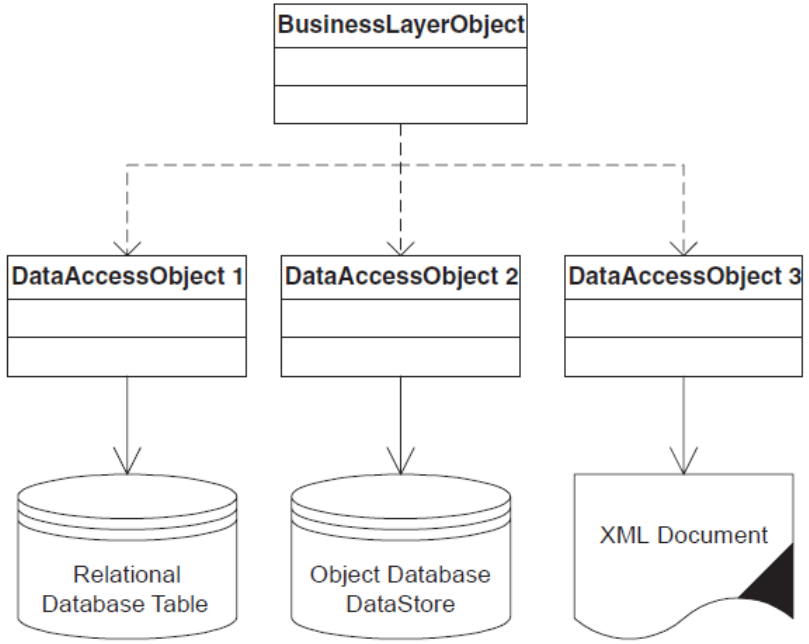
Acceso a datos

- Sin capa de Acceso a Datos
  - Active Record: La clase de dominio maneja directamente el Acceso a Datos
- Sin embargo, conviene separar el acceso a datos en un paquete independiente
- Patrones más comunes
  - Patrón DAO
  - Patrón Mapper

(Patrón) simple de acceso a datos

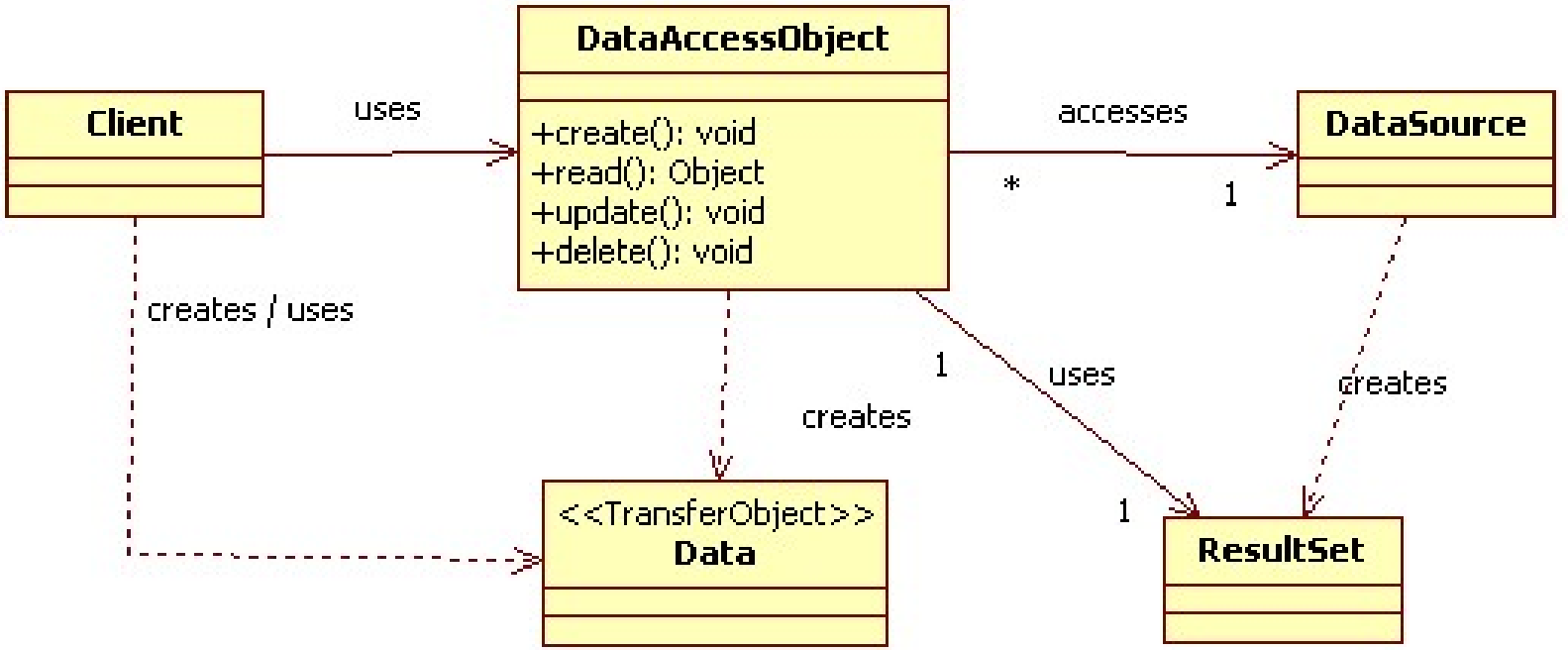
- Active Record: El código de acceso (crear, borrar, buscar, etc.) está incluido en la clase
- Alternativa: clase auxiliar especializada en BD, archivos, XML



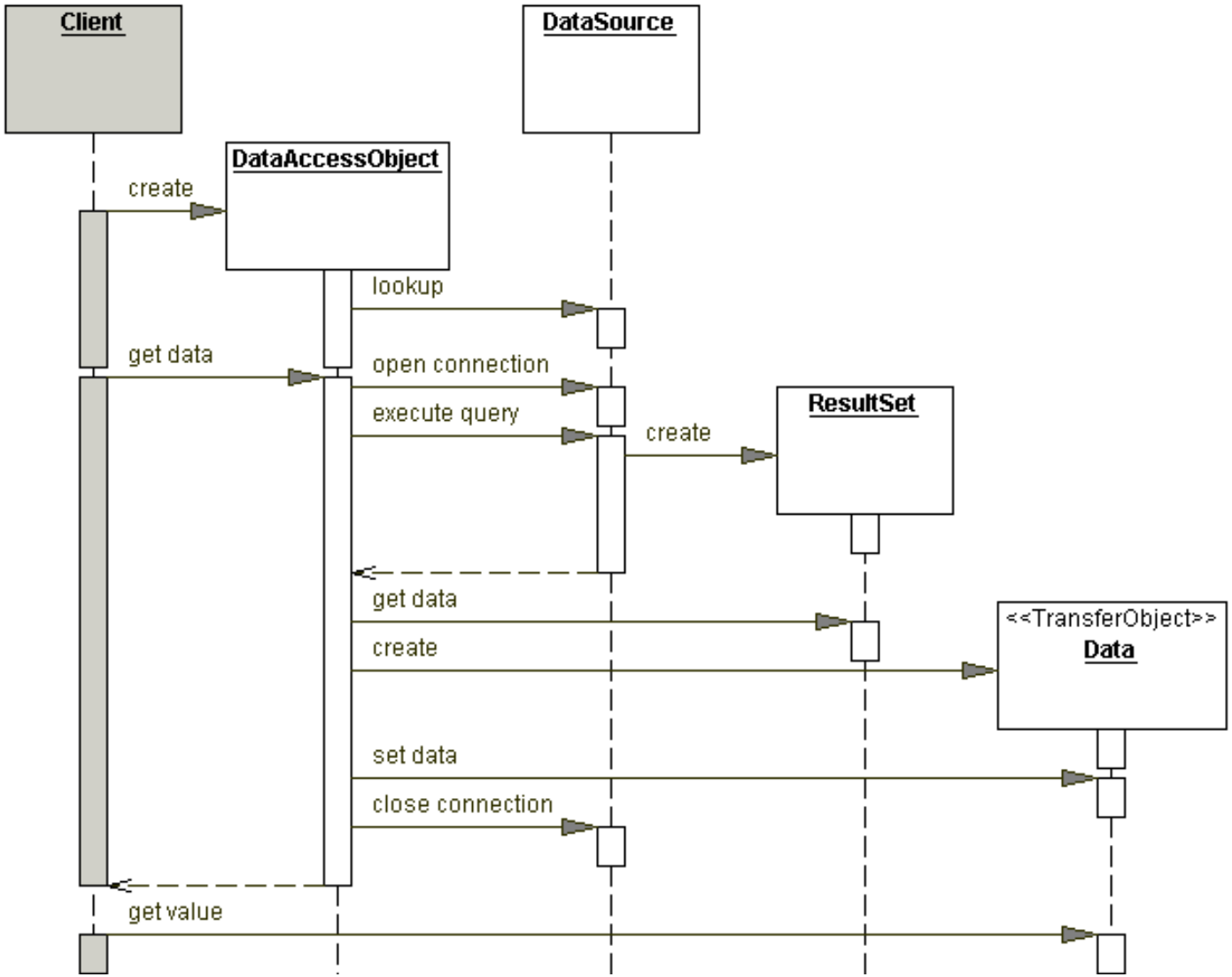


Patrón Data Access Objects

- DataSource: una base de datos (via JDBC/SQL), XML o archivo.
- ResultSet es lo que devuelve DataSource (SQL)
- El DAO manipula el resultado para devolver un DTO serializable

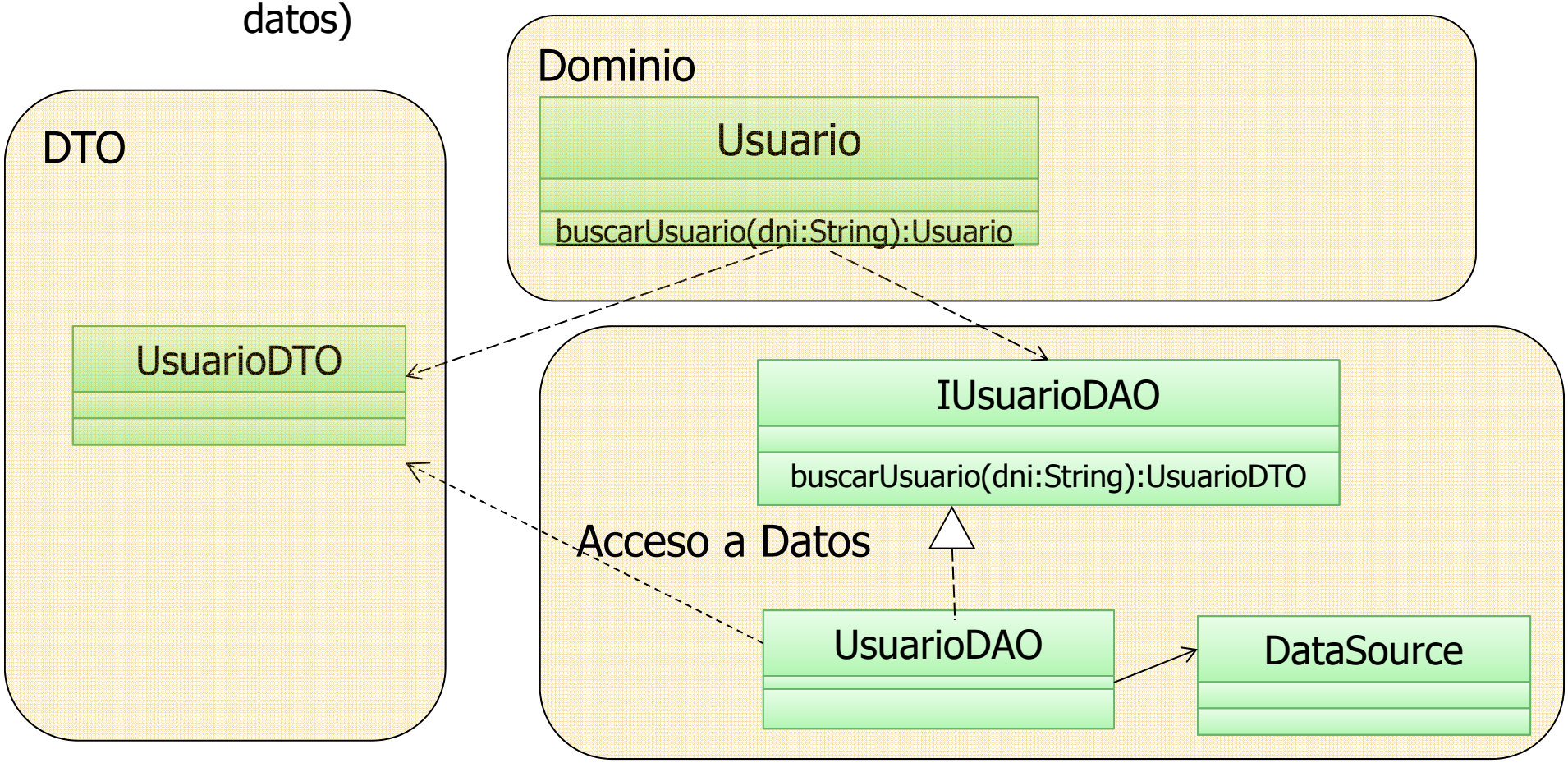


Patrón DAO


Client
DataSource
DataAccessObject
create
lookup
get data
open connection
execute query
create
ResultSet
get data
create
<<TransferObject>>
Data
set data
close connection
get value

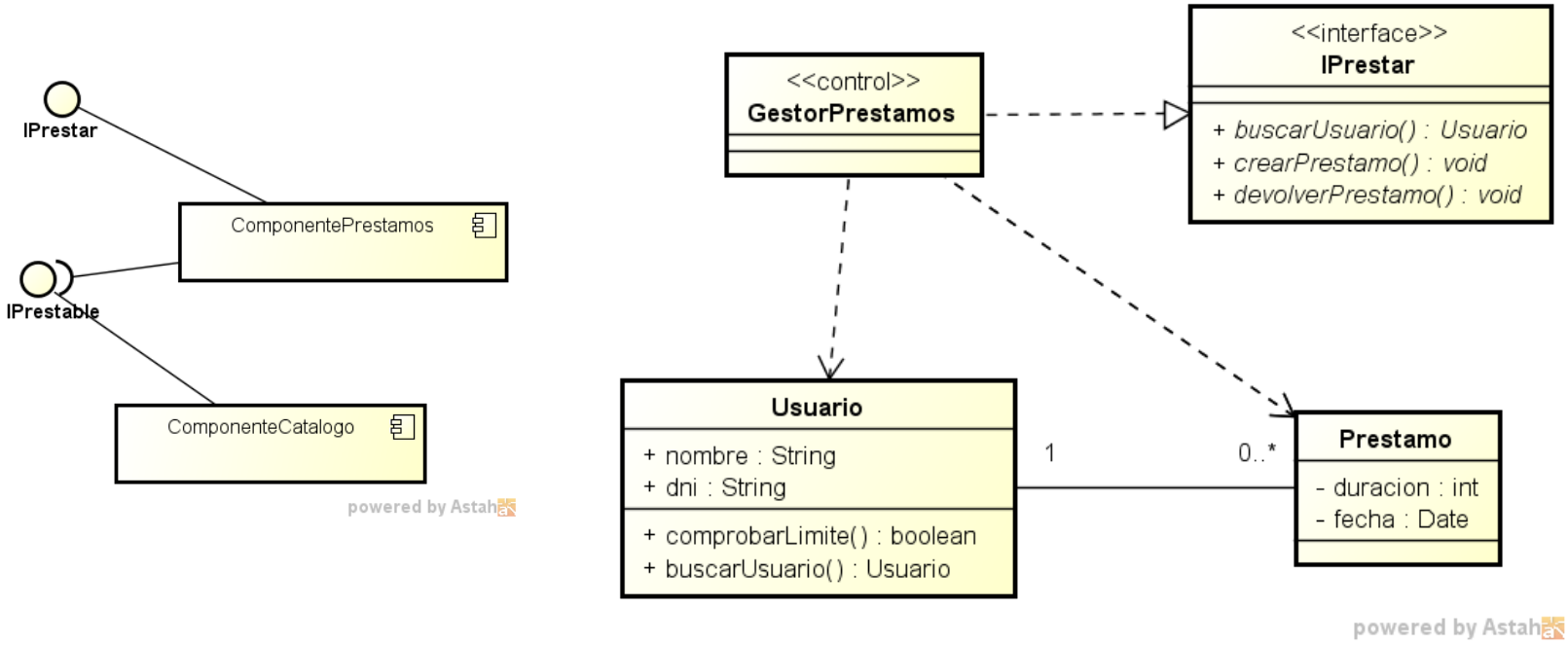
Patrón Data Access Objects

- Se puede añadir una interfaz que aisle las capas y puede ser implementada de varias maneras (accediendo a distintas fuentes de datos)

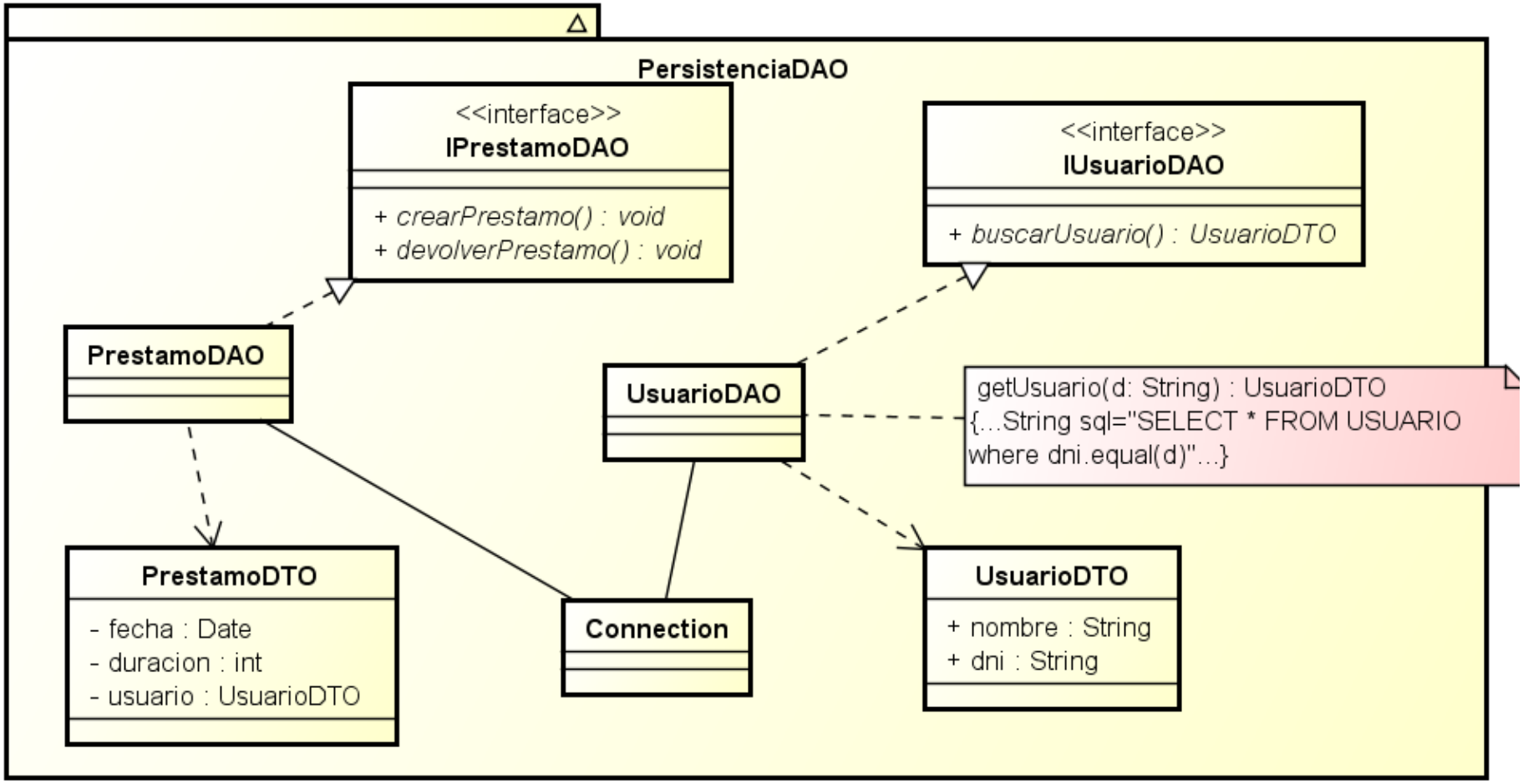


Ejemplo Biblioteca

- ComponentePrestamos, capas de lógica de negocio (despliegue y dominio)



Ejemplo Biblioteca



Patrón DAO: más ejemplos…

- CompraDAO, un DAO en una aplicación de compra, lee la información de una base de datos y lo convierte en DTO (por ejemplo, CompraDTO)

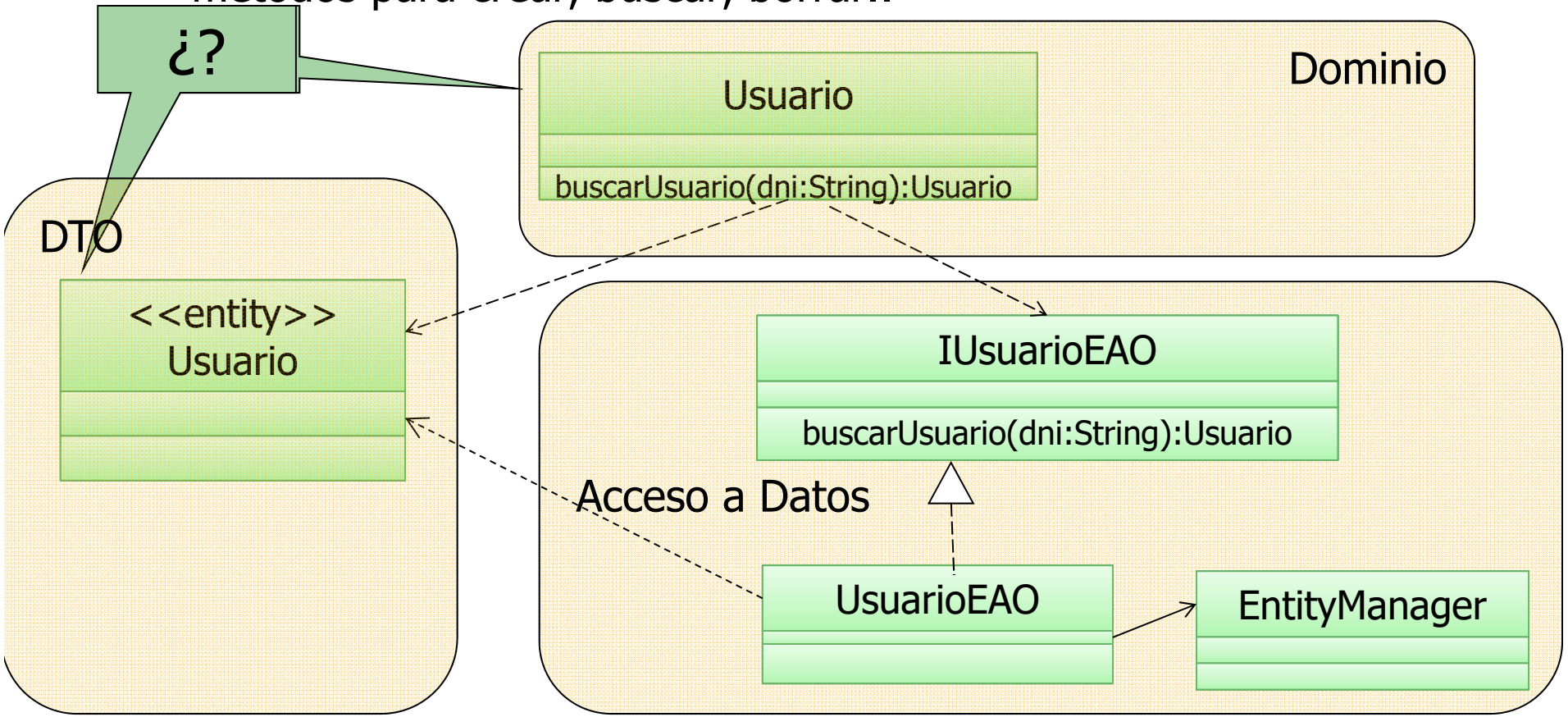
```
public CompraDTO getCompra(int numCompra);
public void saveCompra(CompraDTO compra);
public CompraDTO[] getComprasCliente(String
  clienteId);
public CompraDTO[] getComprasPendientes();
public CompraDTO[] getComprasEntregadas();
```

- Internamente se usa JDBC:

```
String SELECT_SQL = "select CLIENTEID,…
…
pStmt.executeQuery();
```

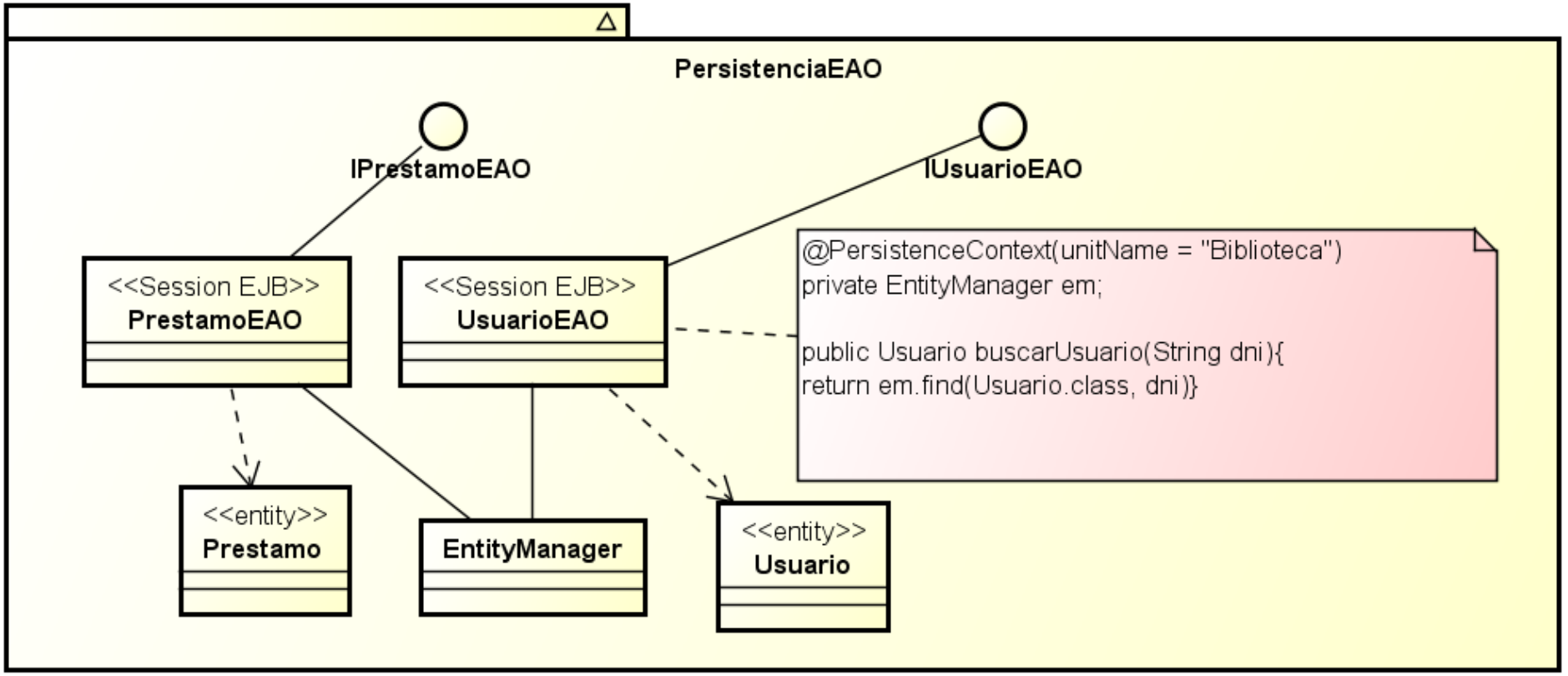
Patrón DAO/JPA o EAO (Entity Access Objects)

- El DTO es una clase de tipo "Entity" JPA
- La interfaz (y su implementación) del EAO nos proporciona los métodos para crear, buscar, borrar..



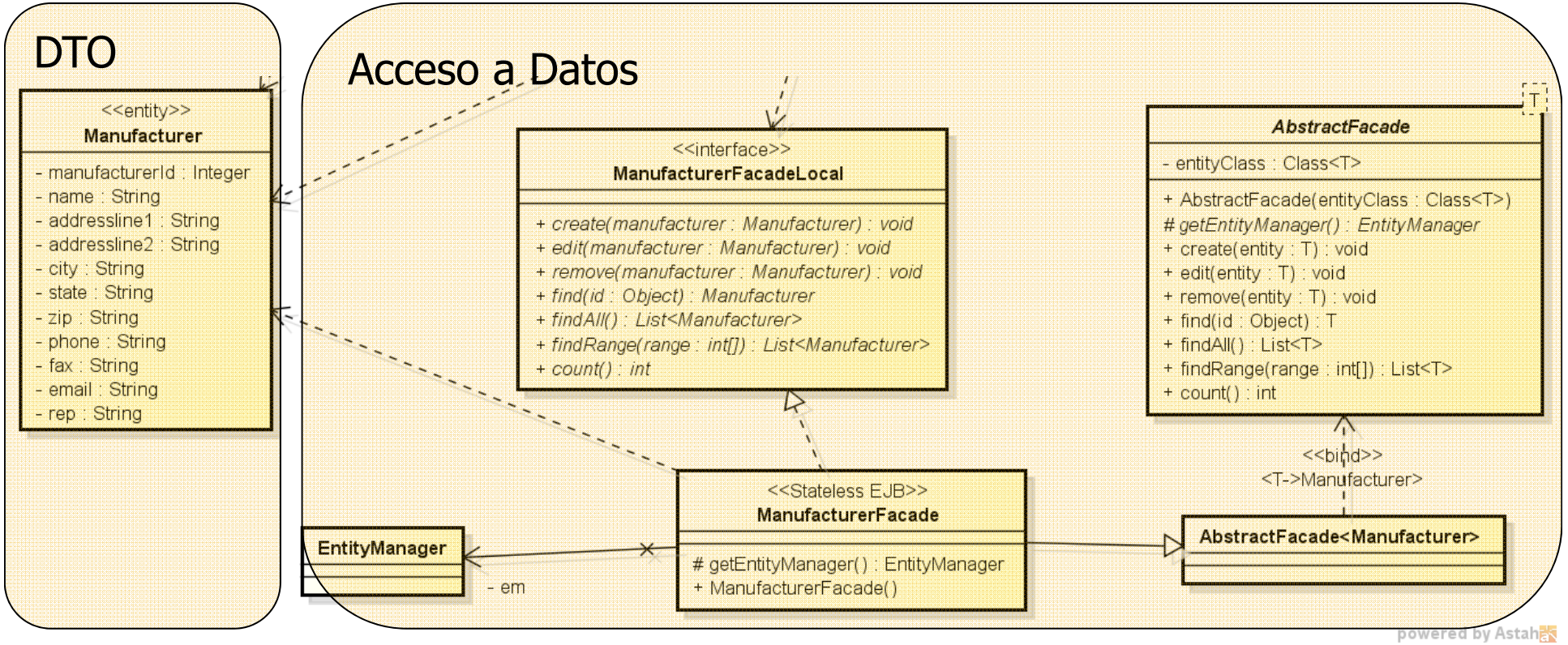
Ejemplo Biblioteca

- Conexión JDBC cambia por EntityManager/JPA
- Devuelve instancias de clases entity JPA



Patrón Fachada (de Acceso a Datos)

- Una clase Entity es un POJO, no un EJB.
  - Con un EJB que actua de fachada se puede acceder a la instancia de la clase entity desde las capas de dominio o despliegue



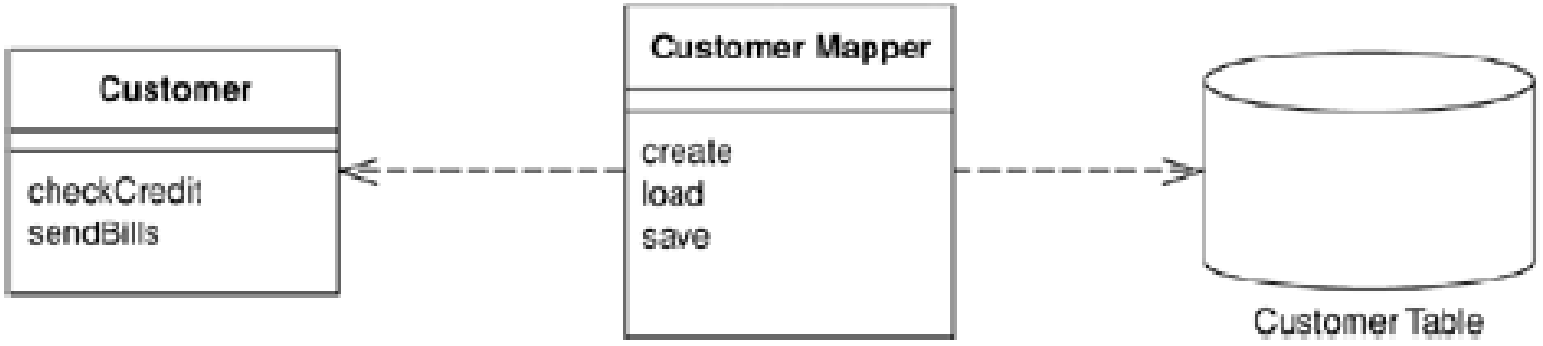
Automatización (NetBeans)

- Adaptación del patrón DAO
- No solo genera la clase **`@Entity`** sino también un EJB de sesión y su interfaz básica CRUD de Acceso a Datos que actúan como fachada de Acceso a Datos

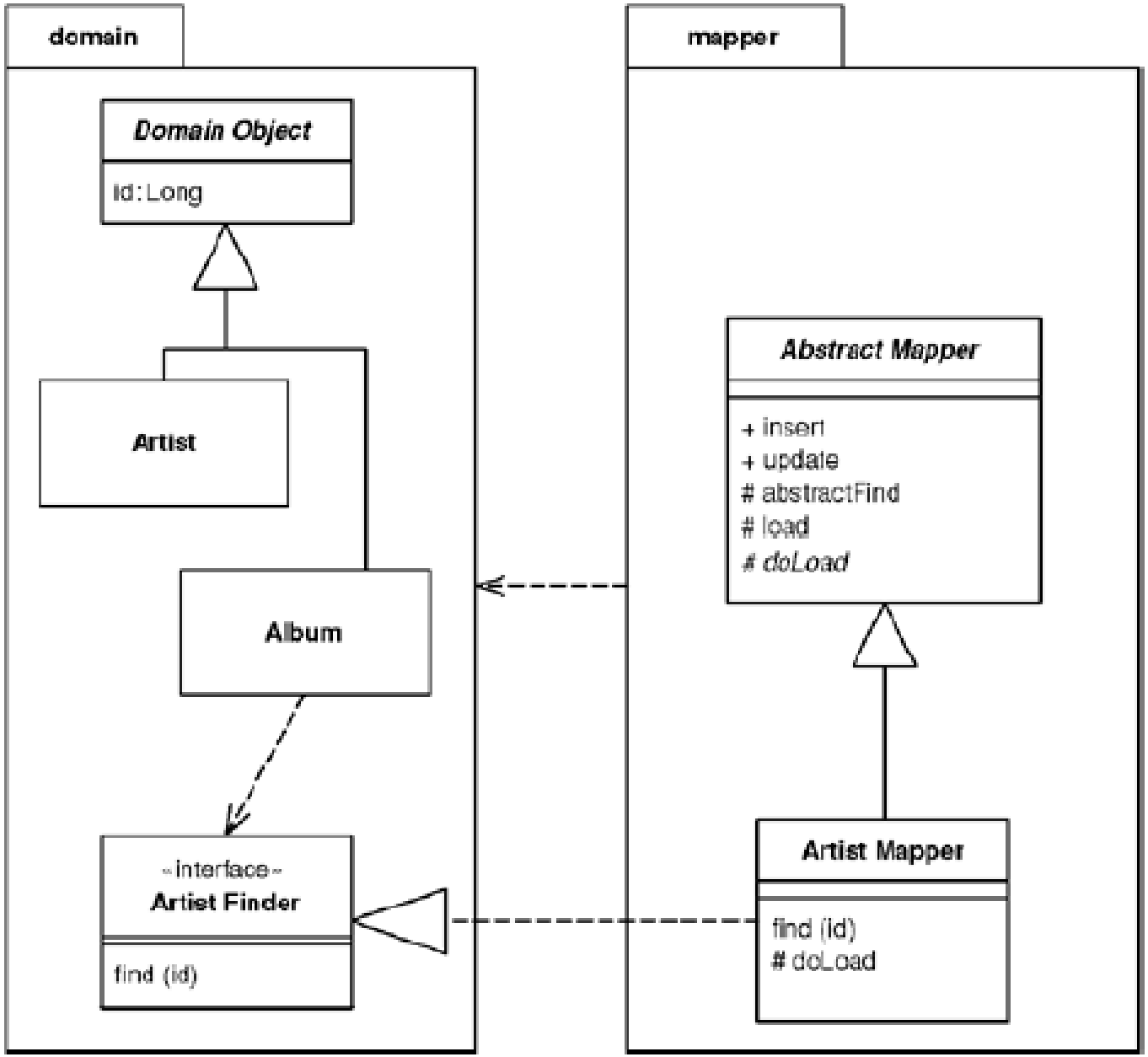
```
@Local
public interface ManufacturerFacadeLocal {
    void create(Manufacturer manufacturer);
    void edit(Manufacturer manufacturer);
    void remove(Manufacturer manufacturer);
    Manufacturer find(Object id);
    List<Manufacturer> findAll();
}
```

Alternativa: Patrón Mapper OR (Fowler)

- Un opción más directa de traducción objeto relacional…
  - ¿es posible que el EJB devuelva instancias de clases del dominio?
  - Entity y clase del dominio como equivalentes
  - Inversión de dependencia!

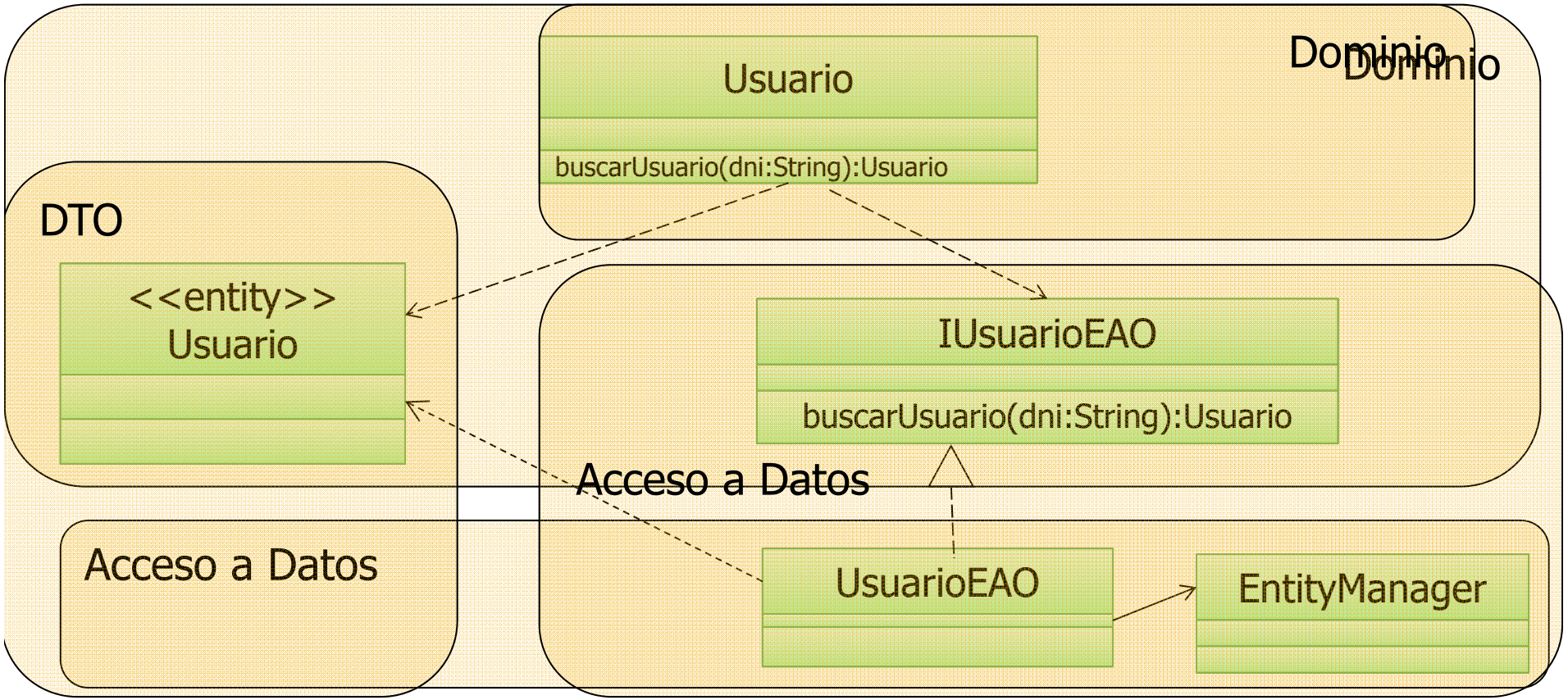


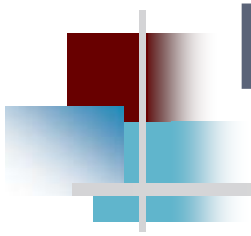
- Cambian las dependencias (interfaz)



Patrón Mapper/JPA

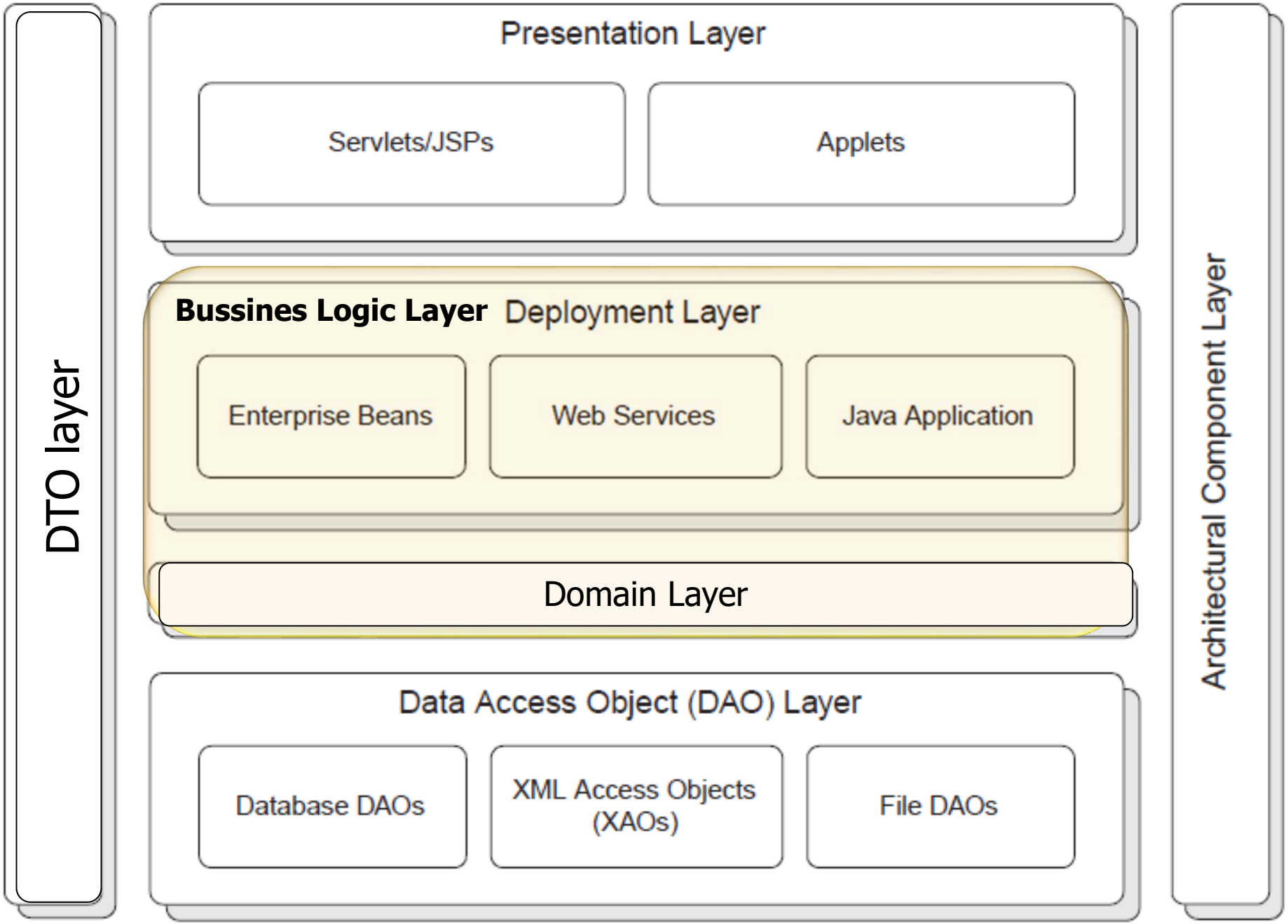
- La interfaz del mapper nos proporciona los métodos CRUD
- Dificultad: Usuario es una clase Java (POJO), no un EJB

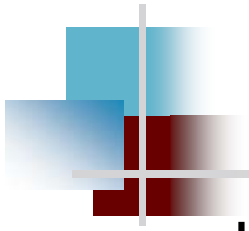




4.3.3 Patrones de la Capa de Negocio

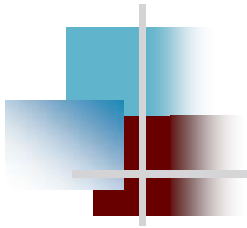
- Despliegue: Patrón Fachada de sesión
- Dominio: Patrones Domain Model y Transaction Script


DTO layer
Presentation Layer
Servlets/JSPs
Applets
Bussines Logic Layer
Deployment Layer
Enterprise Beans
Web Services
Java Application
Domain Layer
Data Access Object (DAO) Layer
Database DAOs
XML Access Objects (XAOs)
File DAOs
Architectural Component Layer



Capa de despliegue

- Los EJB de sesión pueden ser utilizados desde distintos clientes (servlets, aplicaciones C/S de escritorio…), reduciendo el acoplamiento
- Corresponde al Controlador (de Fachada) del modelo de análisis
  - Los métodos de los EJB de sesión corresponden a las operaciones del sistema de los DSS
- Usar EJB de sesión sin estado (Stateless) mejora el rendimiento.

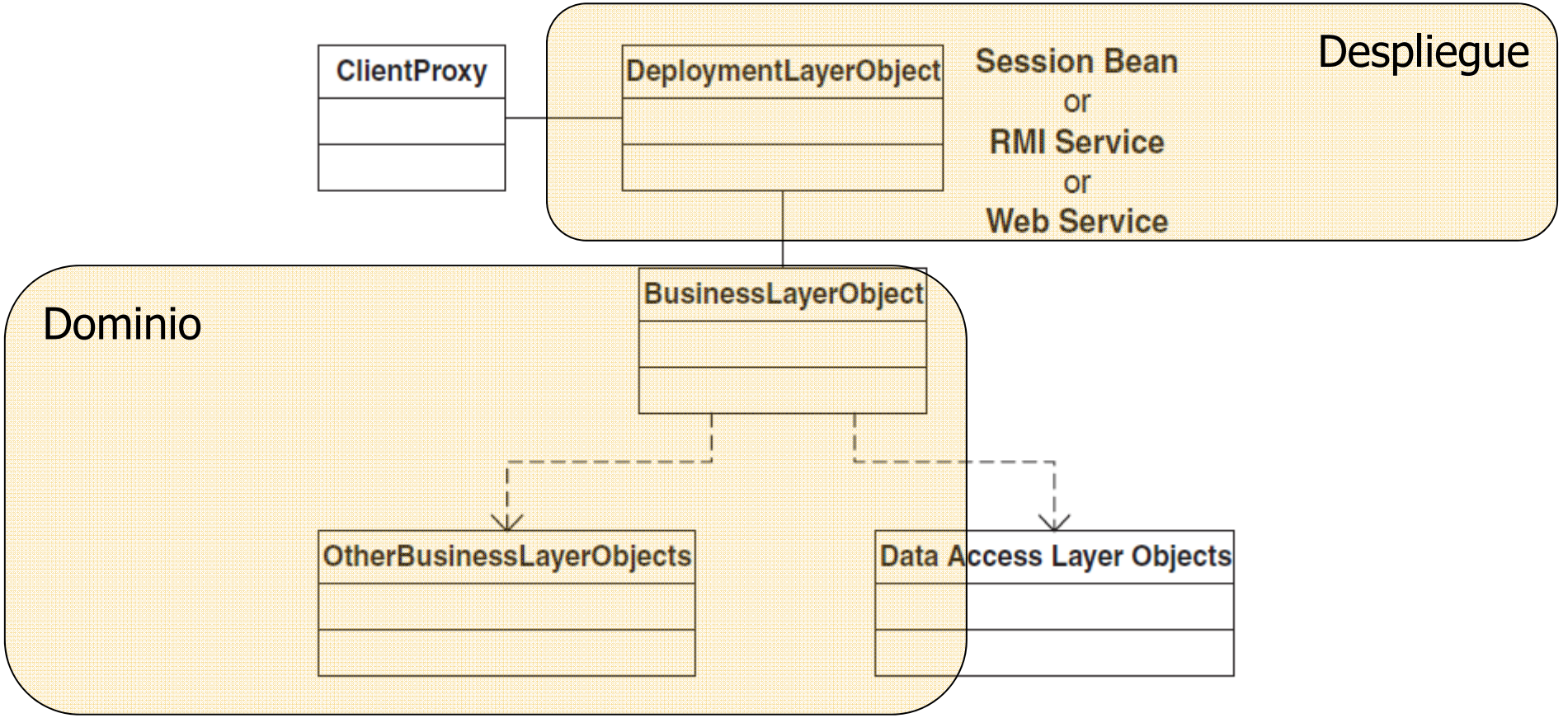


Capa de dominio

- Además de las clases de tipo “entity” que ya tengo…
- Si necesito una transacción que involucre a varias clases (creación de un pedido con líneas de detalle, etc.), la lógica no puede limitarse a una única clase/EJB
- Otras situaciones:
  - Comprobaciones de crédito suficiente o de stock disponible, transferencia entre dos cuentas bancarias, etc.

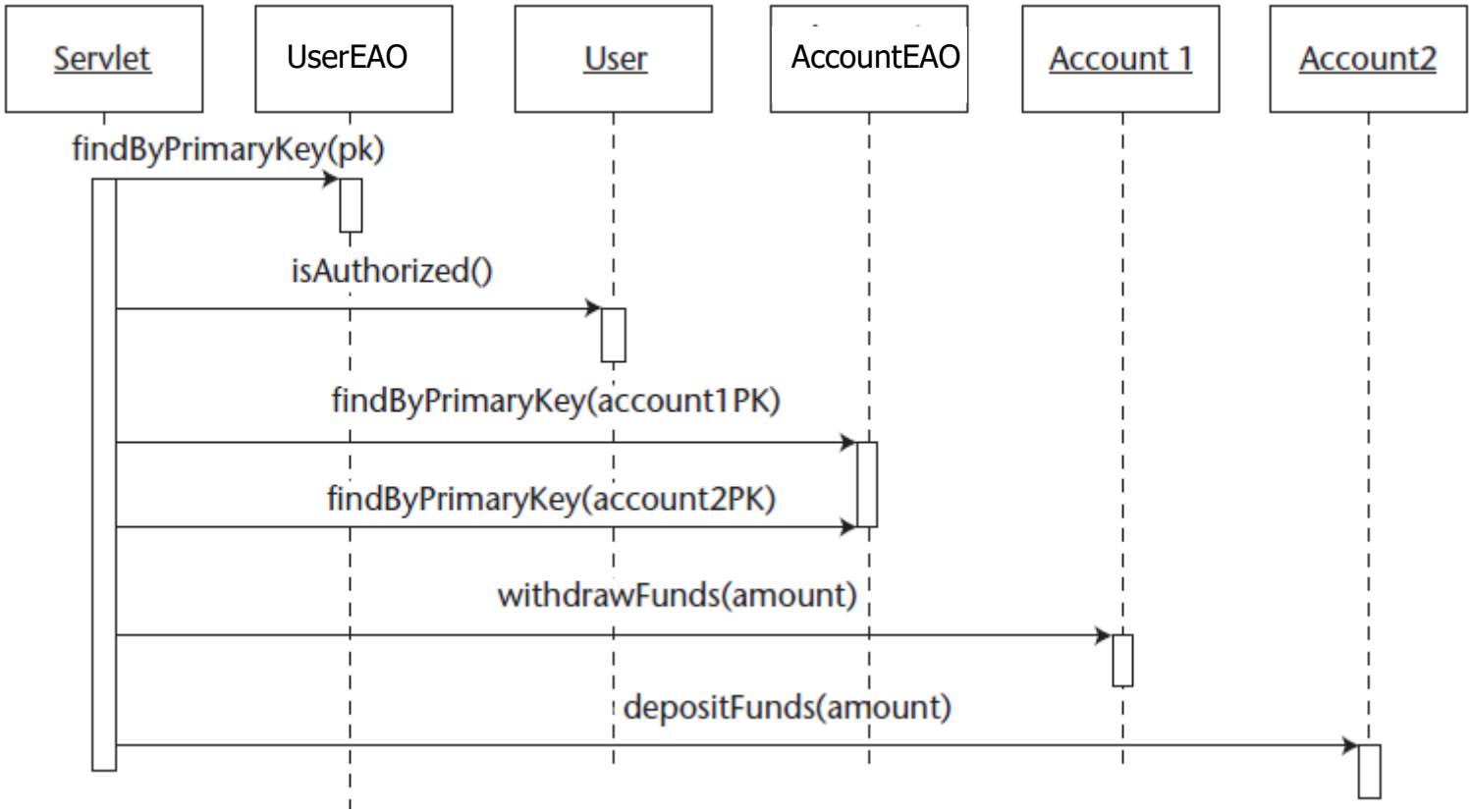
Patrones

- Capa de despliegue: Fachada (Fachada de sesión)
- Capa de domino: Domain Model o Transaction Script



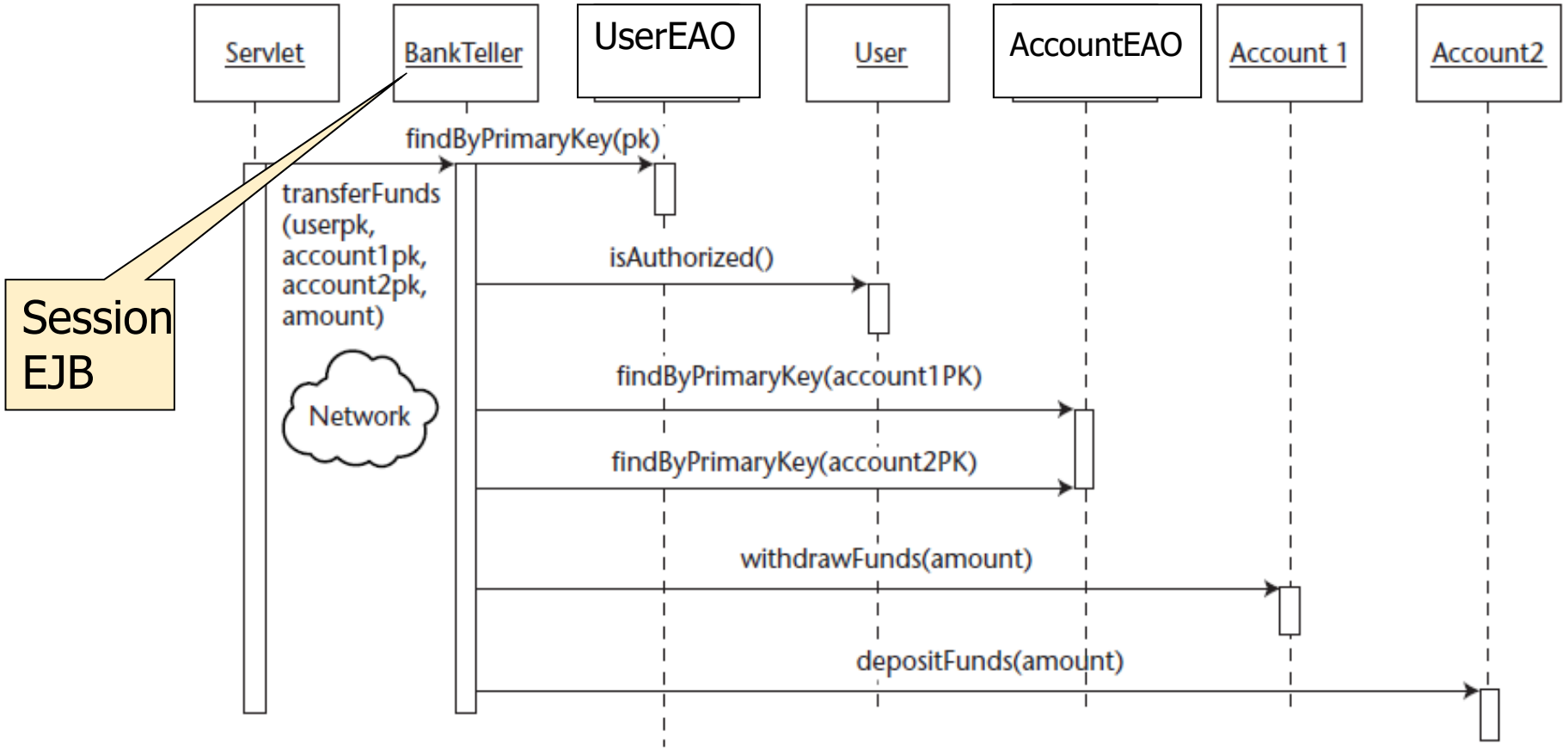
Patrón Fachada de sesión

- ¿Cómo puede un cliente EJB ejecutar la lógica de un caso de uso del negocio en una transacción y una llamada de red en bloque?



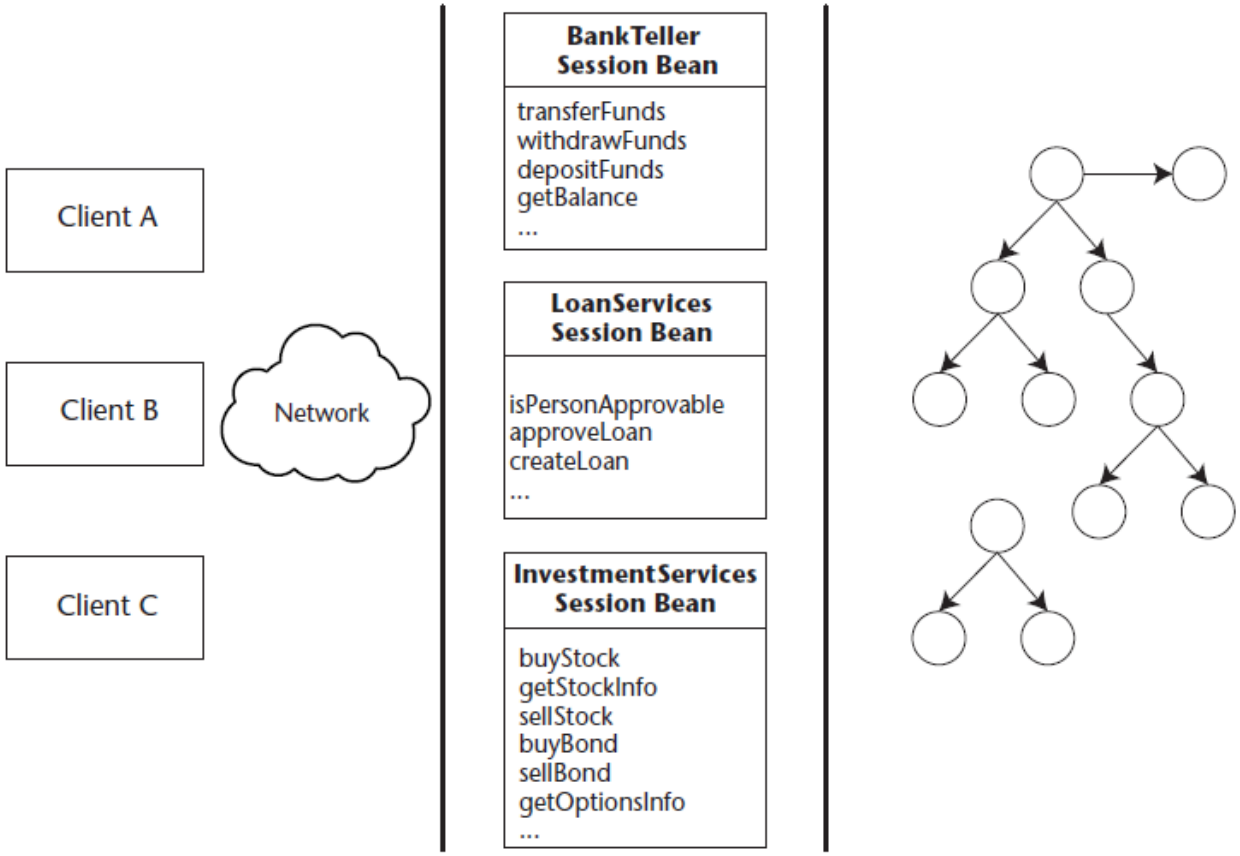
Patrón Fachada de sesión

- Solución: reducir la carga de peticiones a través de la red utilizando un EJB de sesión

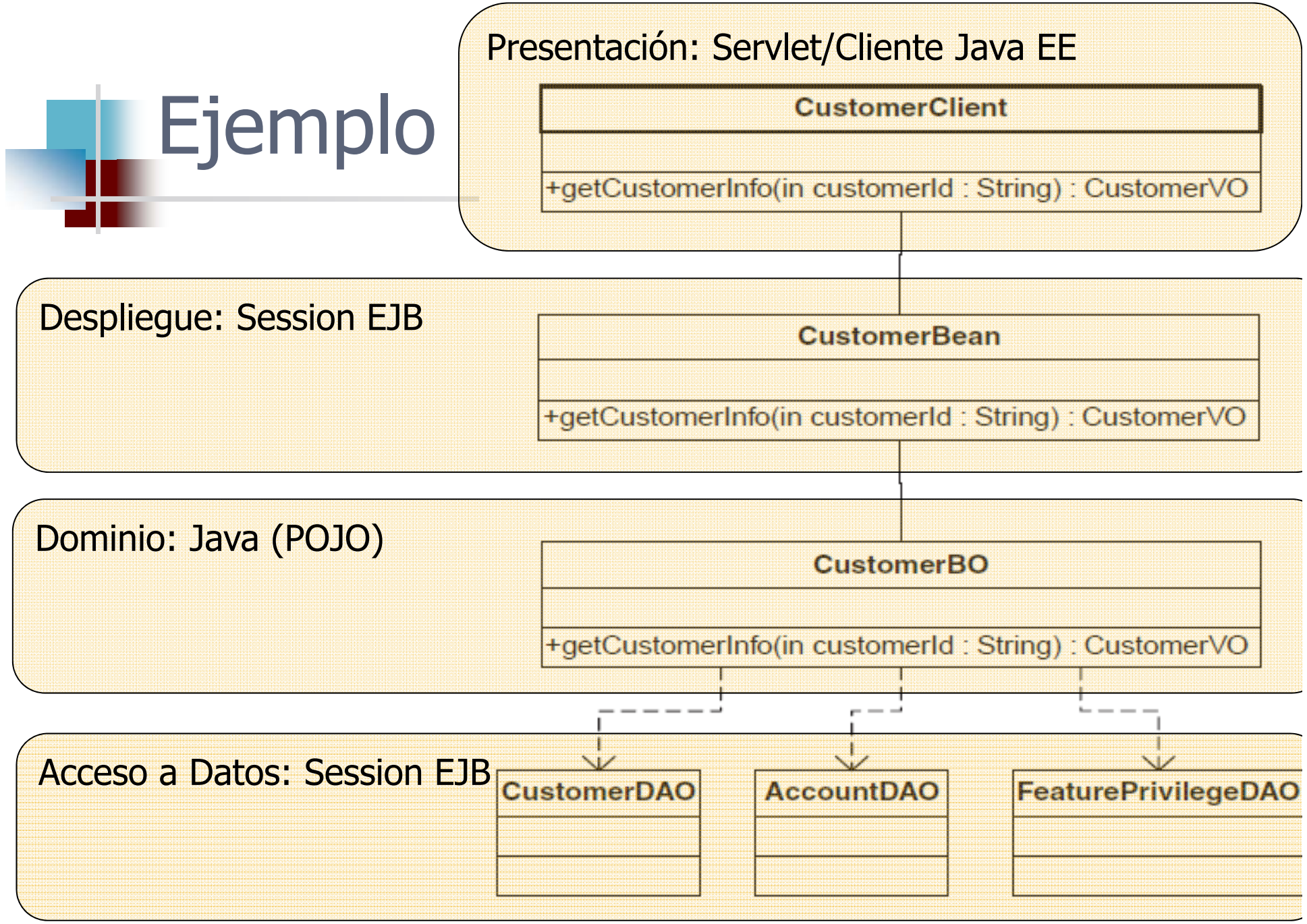


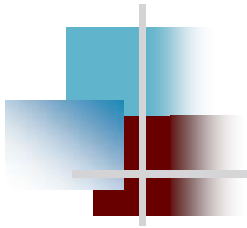
Arquitectura JEE estándar

- Se agrupan las interfaces de uno o varios casos de uso (controlador de análisis) en una fachada de sesión (EJB)



Ejemplo

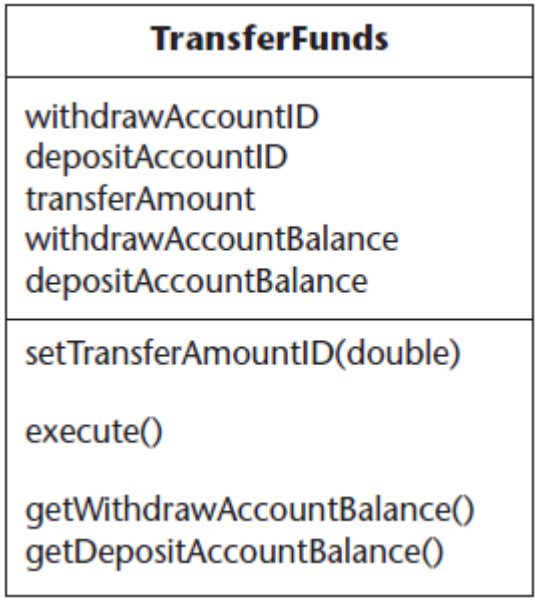

Presentación: Servlet/Cliente Java EE
CustomerClient
+getCustomerInfo(in customerId : String) : CustomerVO
Despliegue: Session EJB
CustomerBean
+getCustomerInfo(in customerId : String) : CustomerVO
Dominio: Java (POJO)
CustomerBO
+getCustomerInfo(in customerId : String) : CustomerVO
Acceso a Datos: Session EJB
CustomerDAO
AccountDAO
FeaturePrivilegeDAO

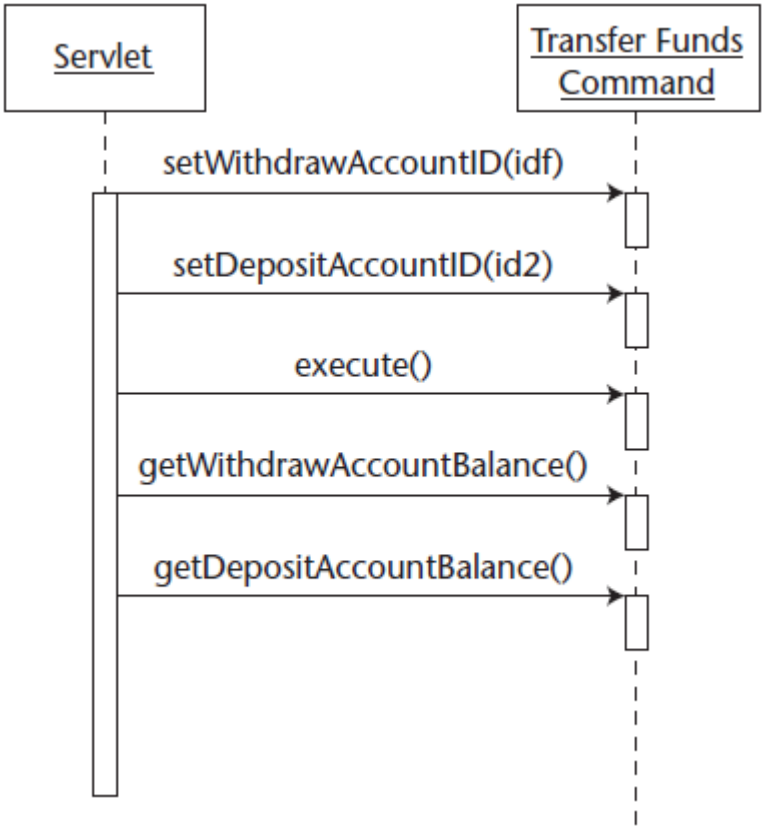


Alternativa: Patrón Comando EJB

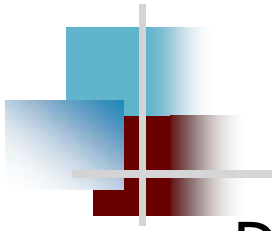
- ¿Cómo se puede implementar la lógica de un caso de uso de manera ligera, desacoplando el cliente del EJB y ejecutando el caso de uso en una transacción y una llamada de red?
- Una solución más ligera que Fachada
- Suele aparecer en frameworks

Patrón Comando EJB





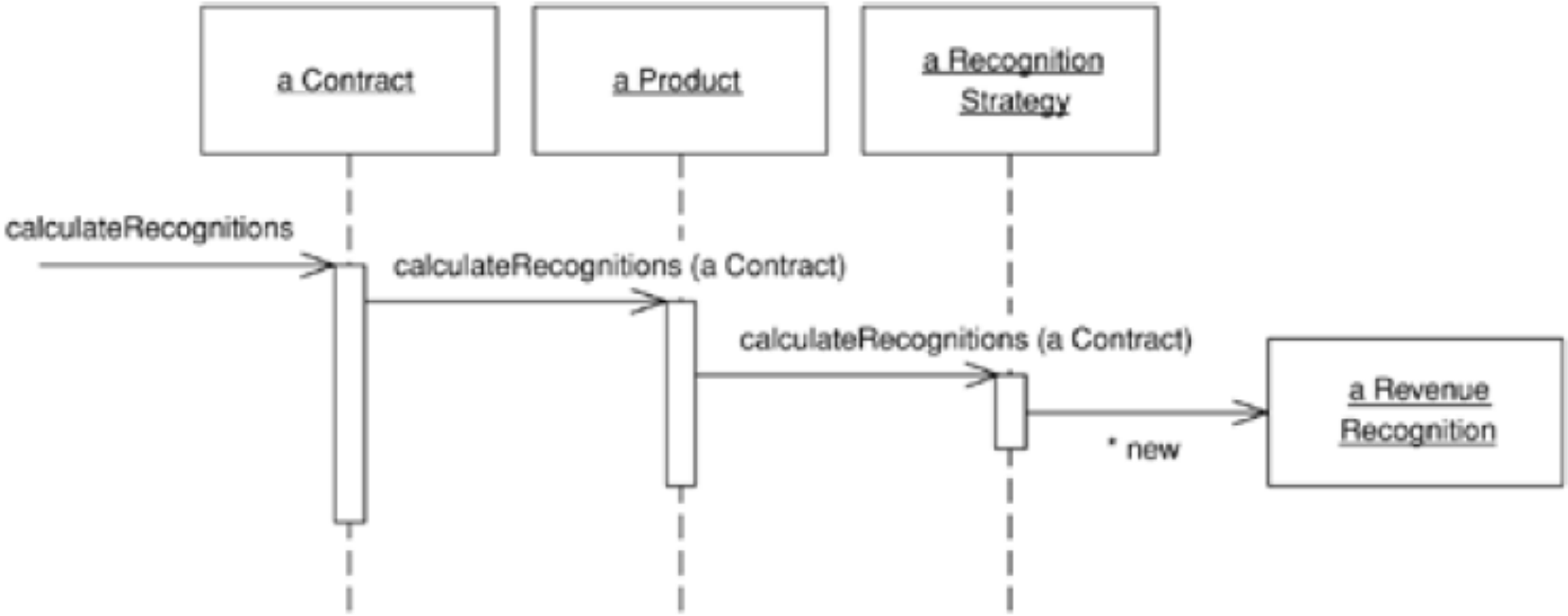
Patrones de la capa de dominio



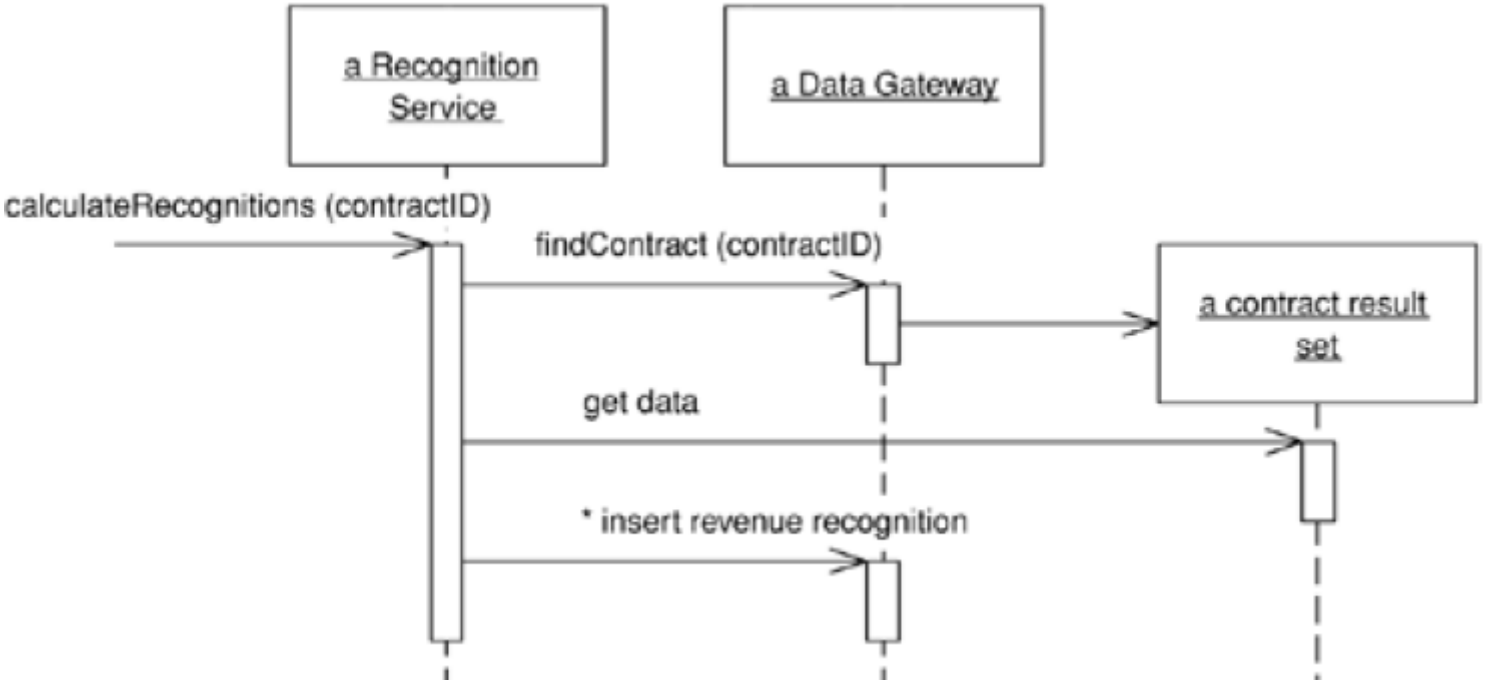
- Domain Model
  - El modelo de análisis se reparte entre la capa de despliegue (EJBs fachada de sesión) y la capa de dominio (clases del dominio)
  - Cuando las clases tienen un comportamiento complejo
  - Refleja directamente el modelo de análisis (Clases de análisis y diagramas de secuencia)

- Transaction Script
  - Cuando las clases tienen un comportamiento (operaciones) muy sencillo
  - El comportamiento se traslada a los EJB de sesión
  - La capa de dominio se integra en la de despliegue

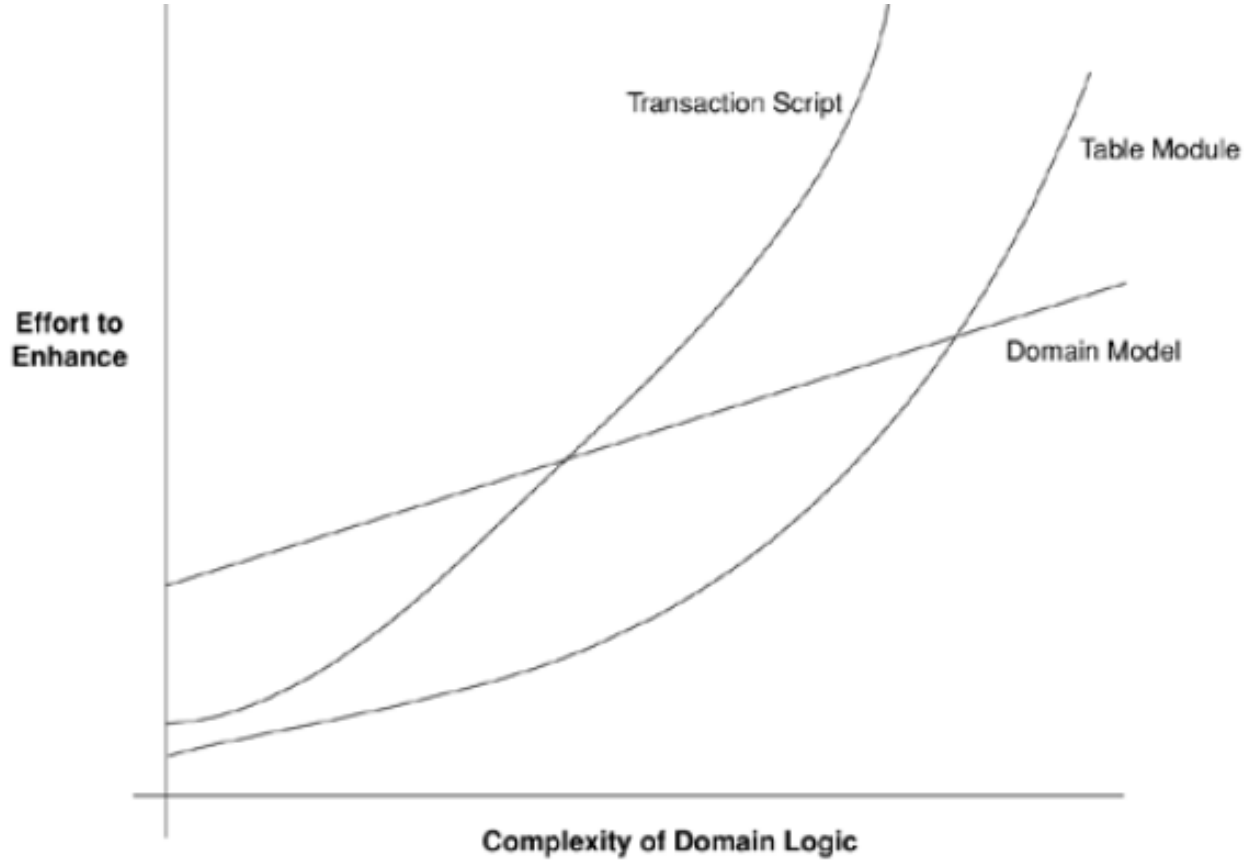
Domain Model



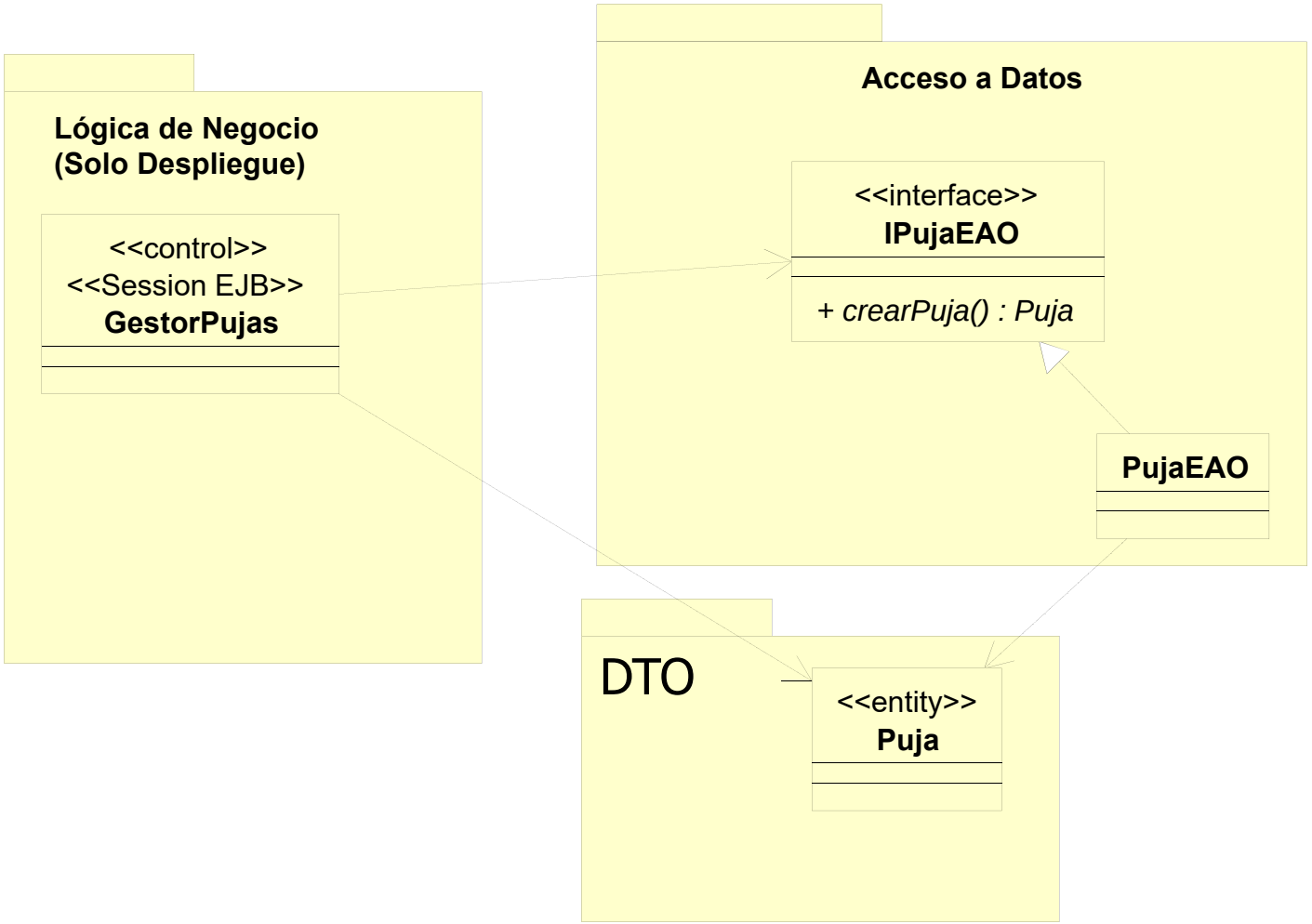
Transaction Script



Esfuerzo y complejidad



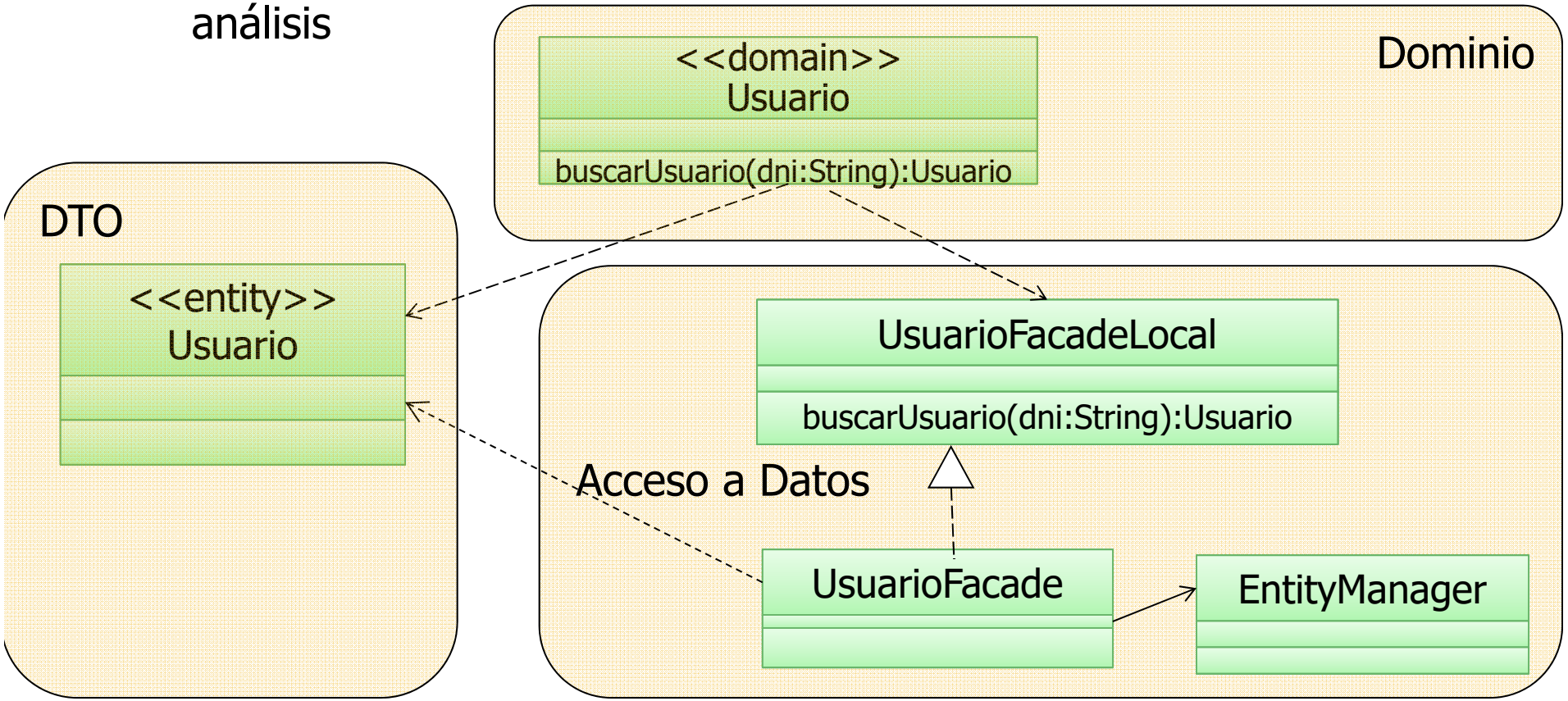
Transaction Script y JEE

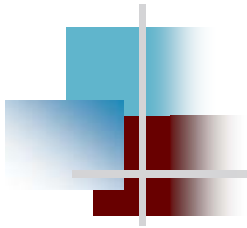


(Panda)

Domain Model (JEE)

- Usuario <<entity>> no tiene operaciones (solo get/set) ni puede "autogestionar" el Acceso a Datos
- Usuario <<domain>> tiene las operaciones del modelo de análisis

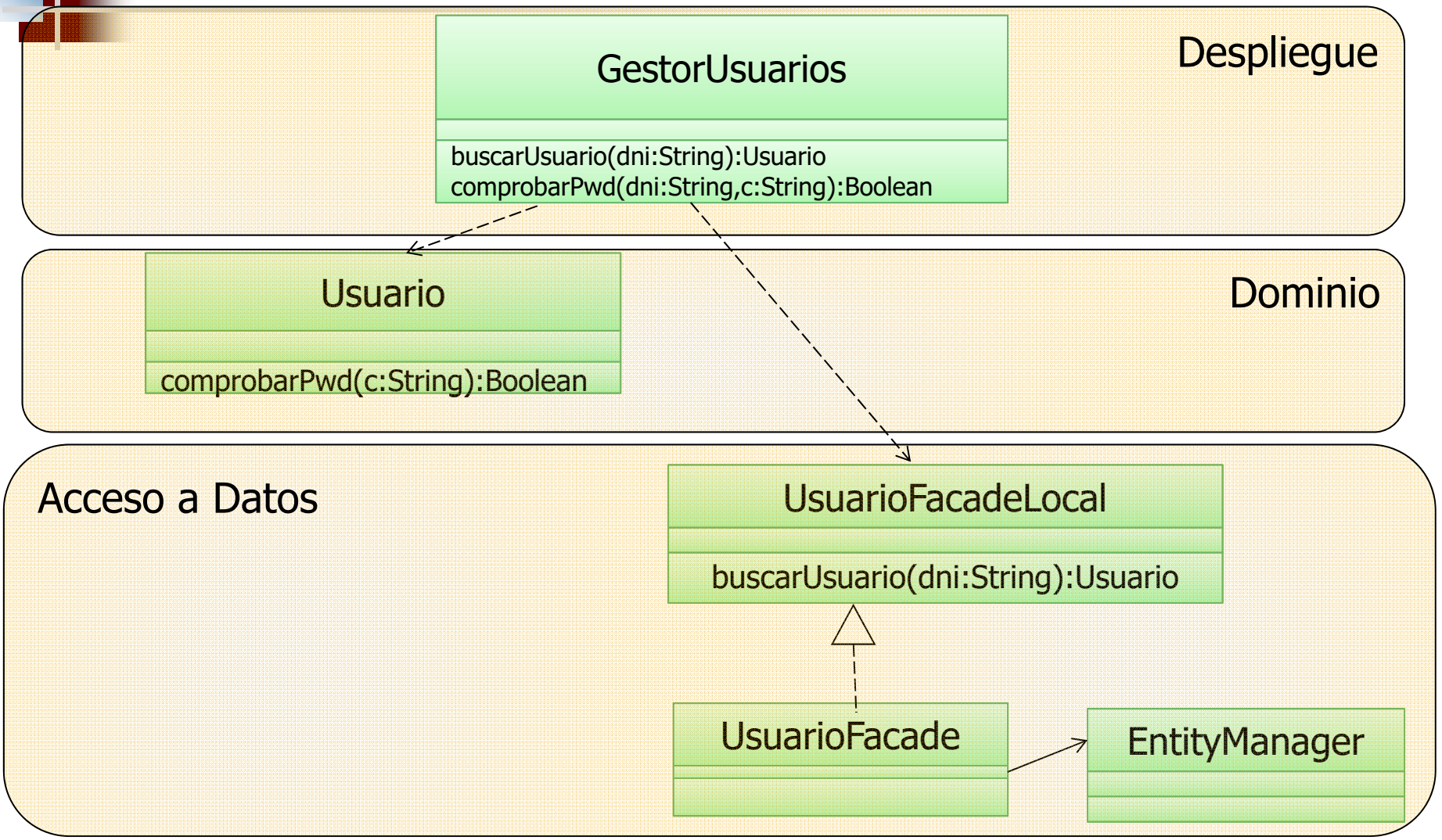




Problemas

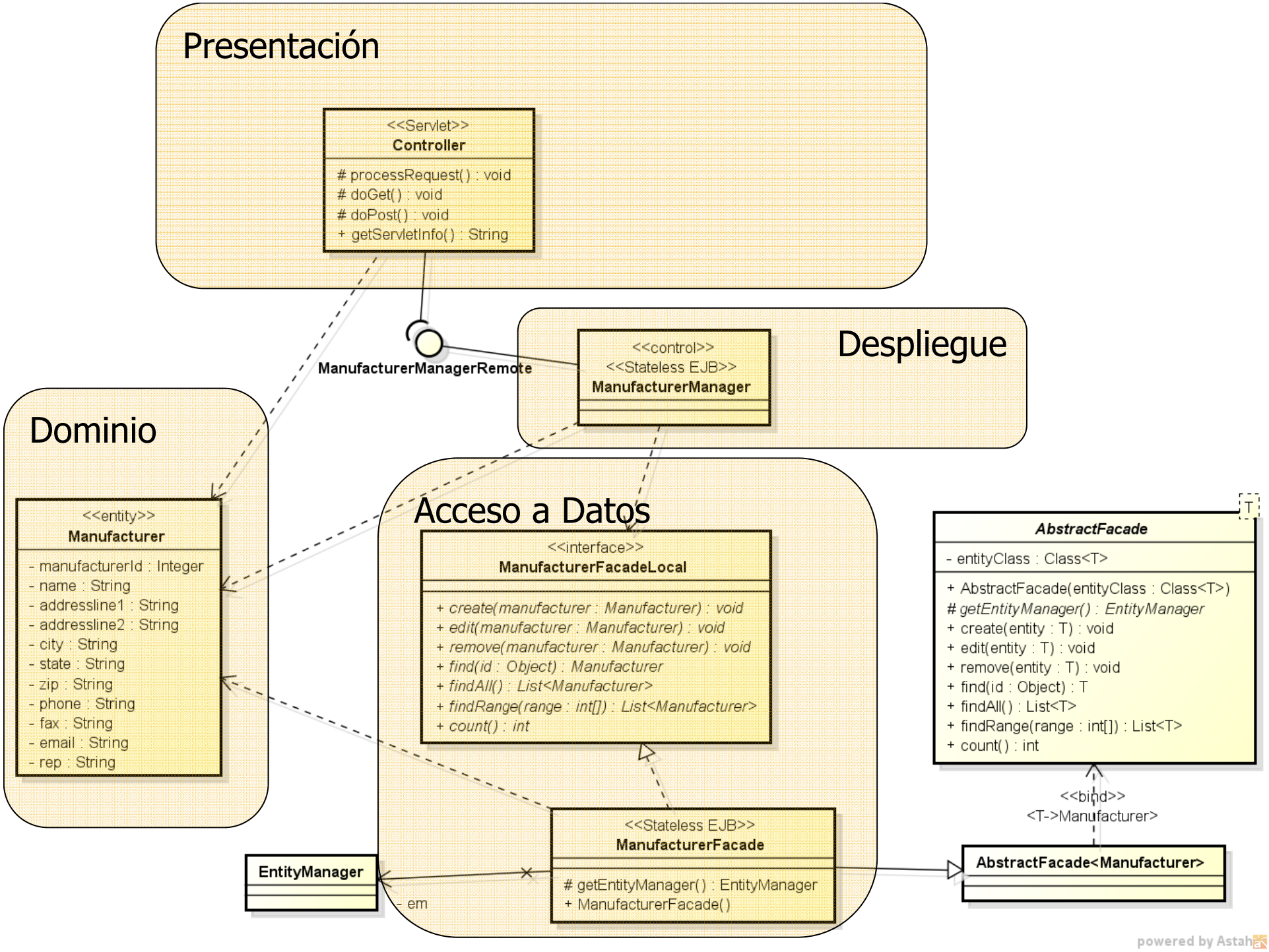
- ¿Cómo inyectar el EJB de fachada en una clase Java normal?
  - Es posible pero complicado
- ¿Cómo conservar la sincronización de una instancia <<entity>> con la Base de Datos?
  - Cada llamada al EJB de Acceso a Datos (Stateless) libera la conexión al devolver control a Usuario
- Solución intermedia:
  - Clase Entity (Usuario): añadir las operaciones "normales" del dominio
  - Clase controlador (GestorUsuario): operaciones de Acceso a Datos

Entities como clases de dominio

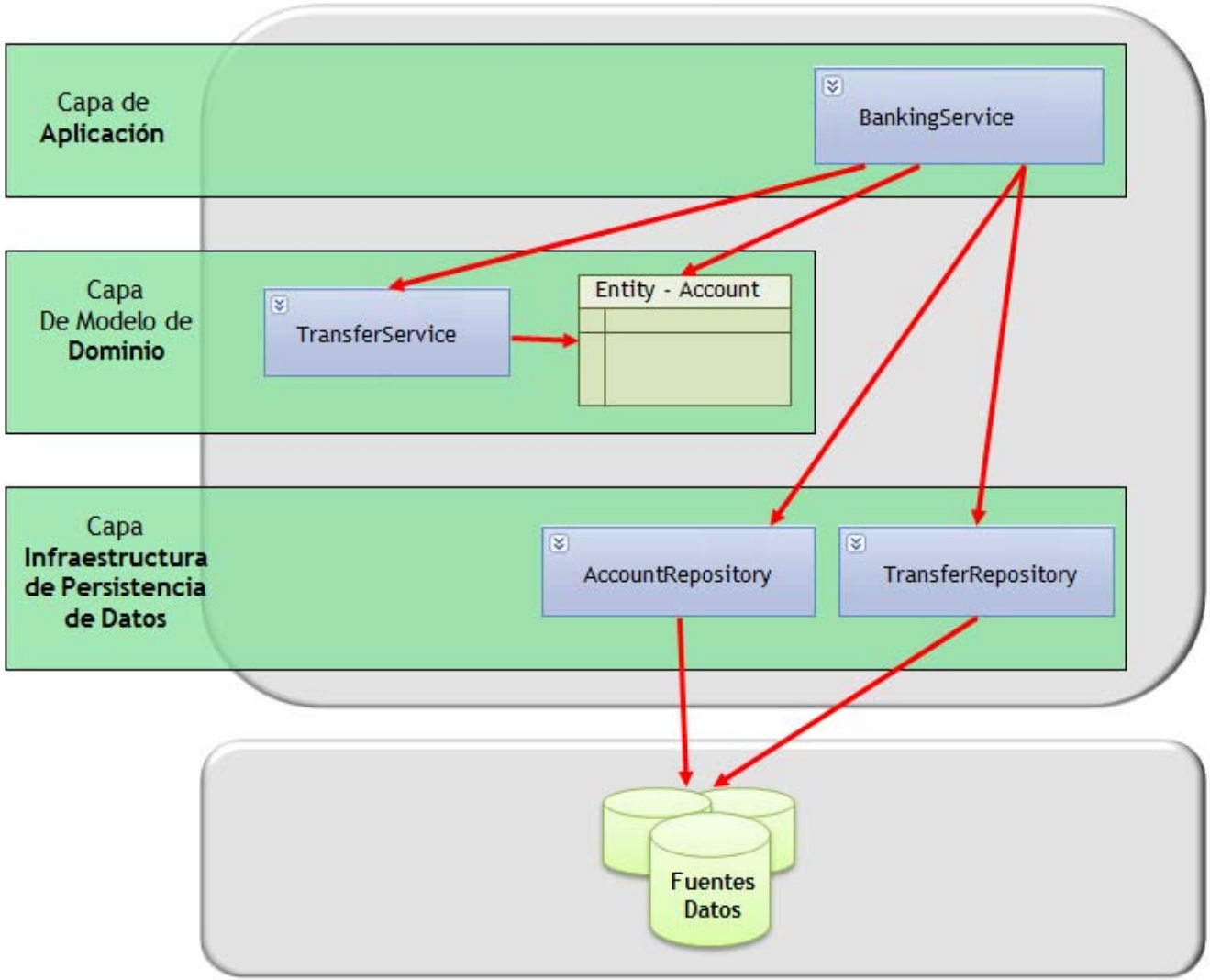


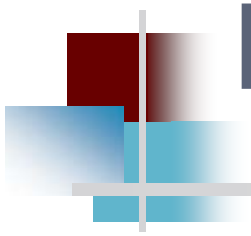
Implementación

```
public class GestorUsuarios implements Iusuarios{
   @EJB
   private UsuarioFacadeLocal ufl;
   ...
   comprobarPwd(dni:String,c:String):Boolean {
      Usuario u=ufl.buscarUsuario(dni:String);
      return u.comprobarPwd(c:String);
   }
```

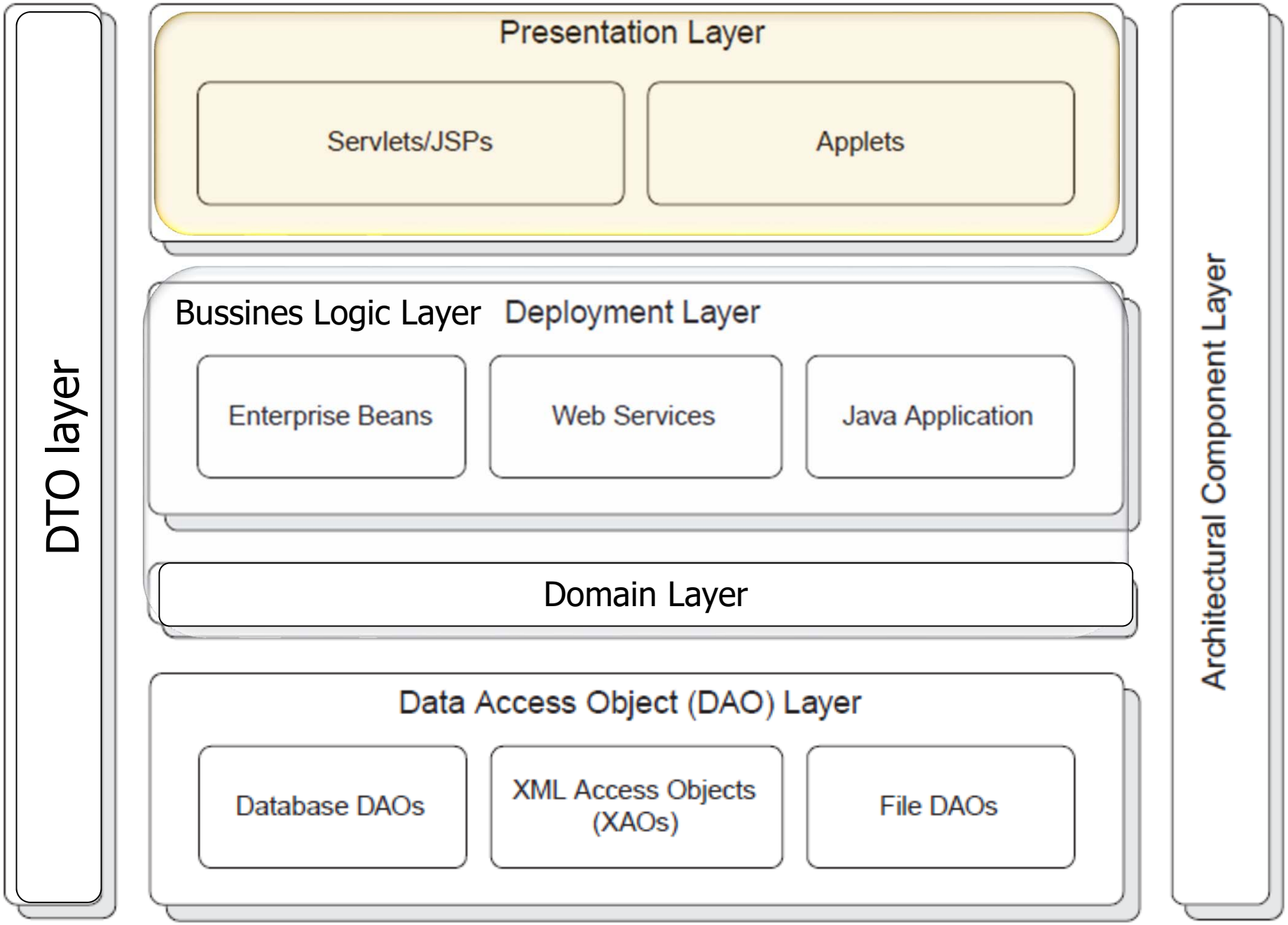

Presentación
<<Servlet>>
Controller
processRequest( ) : void
doGet( ) : void
doPost( ) : void
+ getServletInfo( ) : String
ManufacturerManagerRemote
Despliegue
<<control>>
<<Stateless EJB>>
ManufacturerManager
Dominio
<<entity>>
Manufacturer
- manufacturerId : Integer
- name : String
- addressline1 : String
- addressline2 : String
- city : String
- state : String
- zip : String
- phone : String
- fax : String
- email : String
- rep : String
Acceso a Datos
<<interface>>
ManufacturerFacadeLocal
+ create(manufacturer : Manufacturer) : void
+ edit(manufacturer : Manufacturer) : void
+ remove(manufacturer : Manufacturer) : void
+ find(id : Object) : Manufacturer
+ findAll() : List<Manufacturer>
+ findRange(range : int[]) : List<Manufacturer>
+ count() : int
AbstractFacade
T
- entityClass : Class<T>
+ AbstractFacade(entityClass : Class<T>)
# getEntityManager() : EntityManager
+ create(entity : T) : void
+ edit(entity : T) : void
+ remove(entity : T) : void
+ find(id : Object) : T
+ findAll() : List<T>
+ findRange(range : int[]) : List<T>
+ count() : int
<<bind>>
<T->Manufacturer>
<<Stateless EJB>>
ManufacturerFacade
getEntityManager() : EntityManager
+ ManufacturerFacade( )
EntityManager
- em
AbstractFacade<Manufacturer>
powered by Astah

Alternativa Microsoft

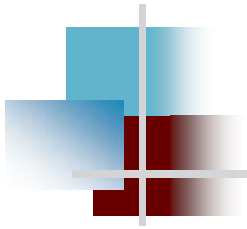




4.3.4 Patrones de la Capa de Presentación: MVC

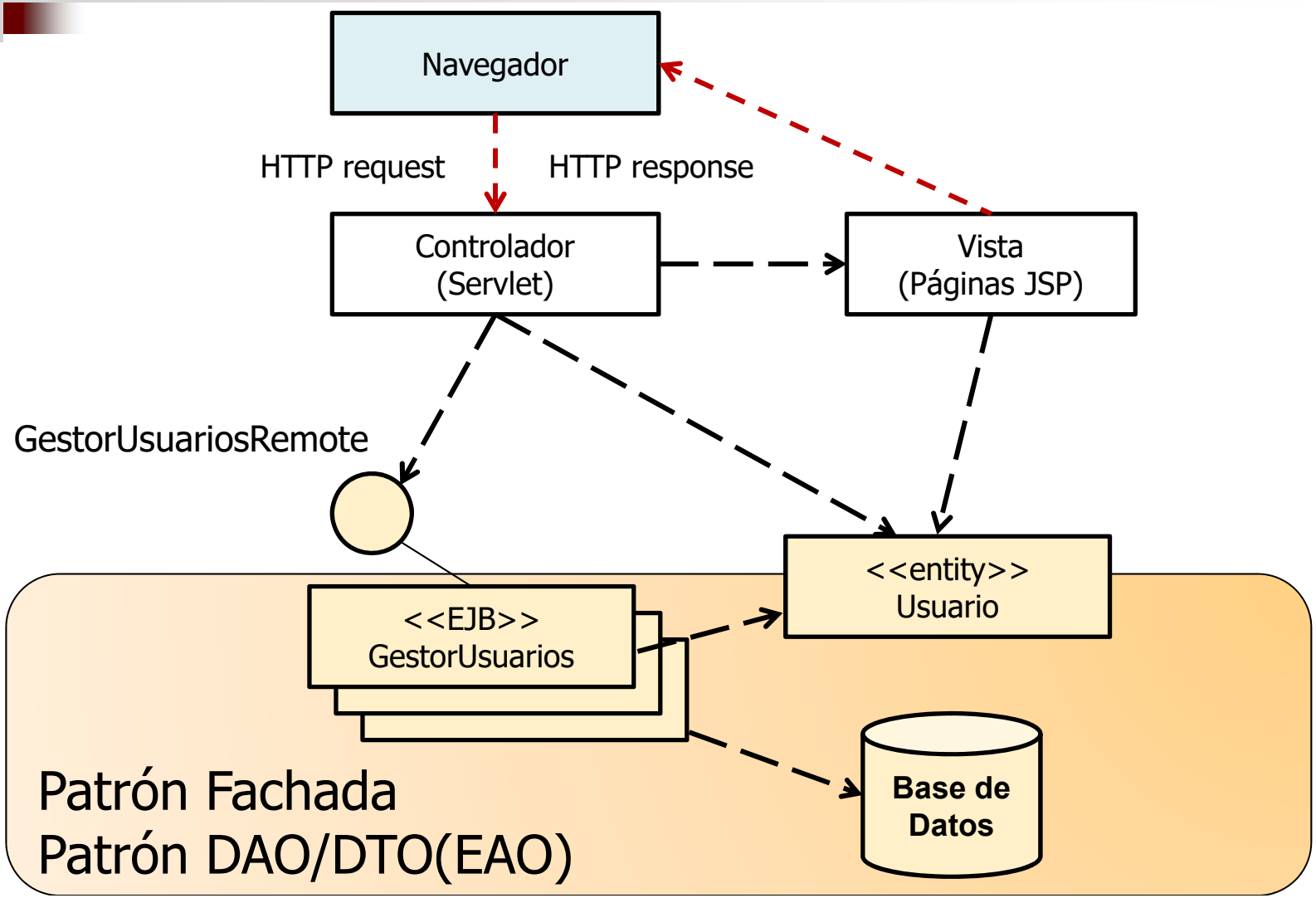

DTO layer
Presentation Layer
Servlets/JSPs
Applets
Bussines Logic Layer
Deployment Layer
Enterprise Beans
Web Services
Java Application
Domain Layer
Data Access Object (DAO) Layer
Database DAOs
XML Access Objects (XAOs)
File DAOs
Architectural Component Layer

Patrones de la Capa de presentación



- Patrón: MVC en alguna de sus variantes
  - Modelo: EJBs, servicios Web
  - Controlador: servlet
  - Vista: JSP
  - Las clases "Entity" son DTOs y también JavaBeans
  - Simplifican la presentación en las JSPs
- El Framework Struts proporciona un controlador genérico (servlet configurable) que soporta vistas y modelos JEE.

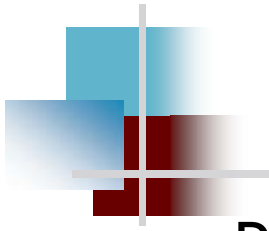
Modelo Vista Controlador/JEE



Implementación

```
public class UsuariosServlet extends HttpServlet {
   @EJB
   private GestorUsuariosRemote gu;
   ...
   Usuario u=gu.buscarUsuario(dni:String);
   Boolean b=gu.comprobarPwd(dni:String,c:String); //?
   ...
}
```

- Pero la conexión con el **`Usuario u`** se pierde entre llamadas porque los Servlets son compartidos por naturaleza
- No se debe inyectar un EJB Stateful en un Servlet!



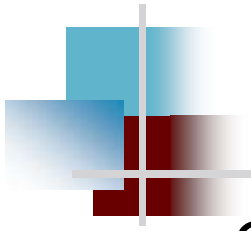
EJB Stateful y Servlets

- Para usar EJBs de sesión con estado en la aplicación web:
  - Hay que hacer una búsqueda JNDI (método lookup()) y almacenar la instancia EJB devuelta en HttpSession (session o request)
  - De esta manera, el mismo usuario utiliza la misma instancia del EJB en llamadas consecutivas
  - La instancia EJB queda bloqueada para la sesión del usuario Web evitando la posibilidad de otras sesiones de adquirirla o borrarla

EJB, Servlets y clases auxiliares

- No se puede inyectar un EJB en una clase normal
  - Anotar la clase Servlet con @EJB y la referencia al EJB
  - Hacer una búsqueda JNDI (lookup()) en la clase auxiliar

```
@EJB (name = “GestorUsuarios", beanInterface =
IUsuarios.class)
public class UsuariosServlet extends HttpServlet {
…}
public class UsuarioAuxiliar{
…
GestorUsuarios gu=(GestorUsuarios)
    context.lookup("java:comp/env/ejb/GestorUsuarios");
Boolean b=gu.comprobarPwd(dni:String,c:String);
…}
```

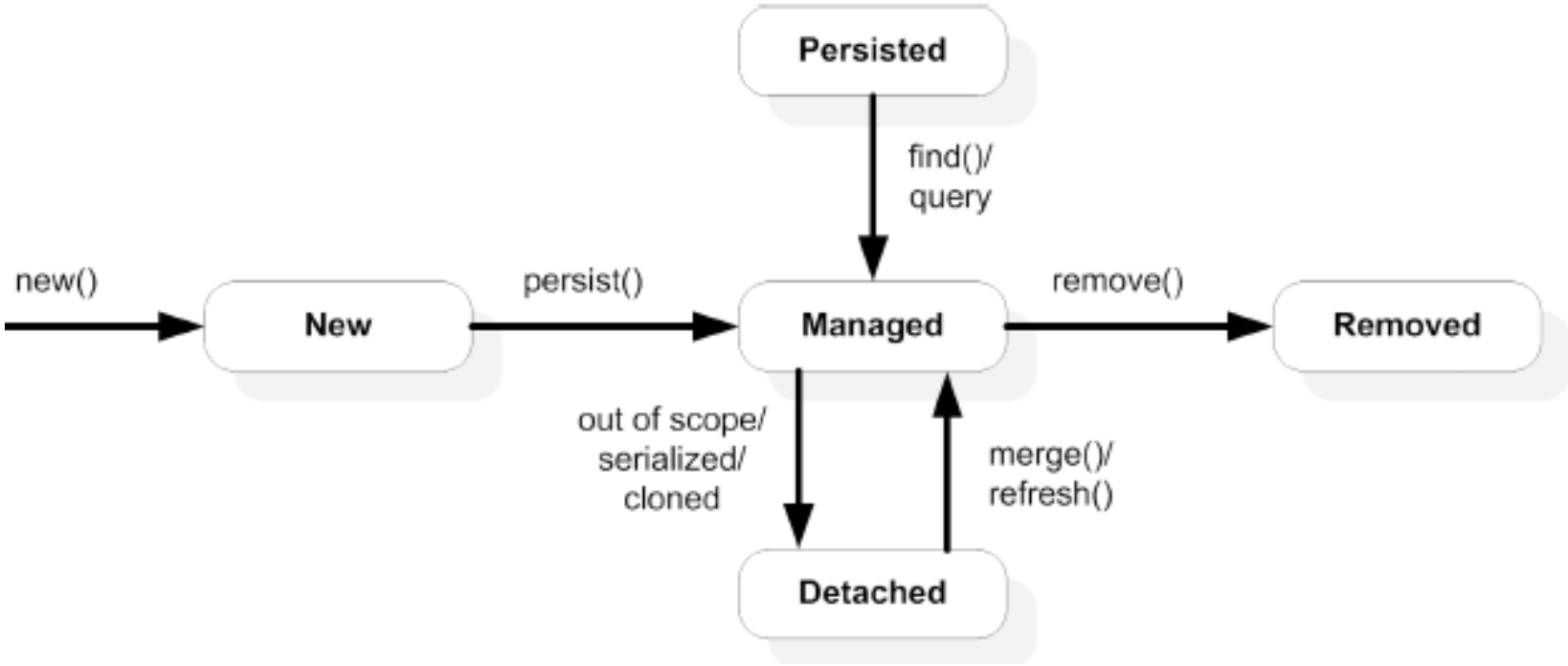


Interfaces remotas

- Cuando se usa más de un servidor, hay que definir interfaces remotas
- Eso influye en la forma de diseñar el Acceso a Datos:
  - perdemos la referencia directa a la instancia
  - No hay acceso directo a la clases entity (hay que colocarlas en una biblioteca compartida)
- El rendimiento disminuye con interfaces remotas y con los EJB Stateful

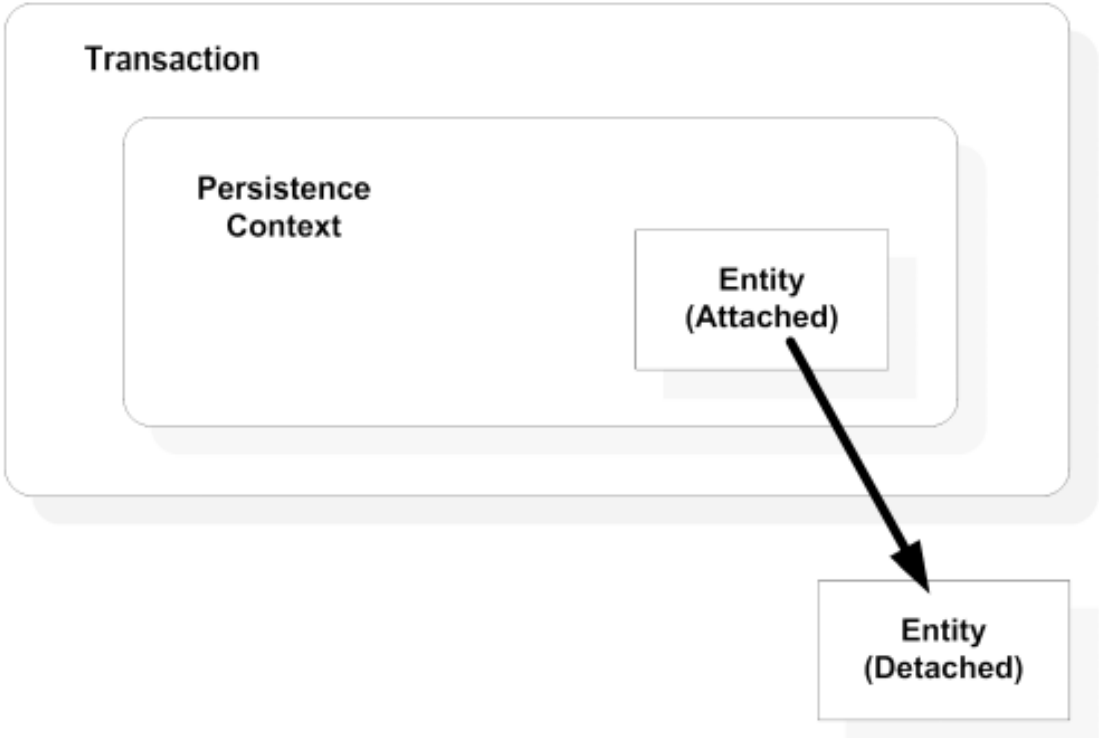
JPA y JEE: sincronización

- Estado Managed (persist(), find(), merge())
  - significa que el cambio de la entity es grabado en la BD de forma automática.



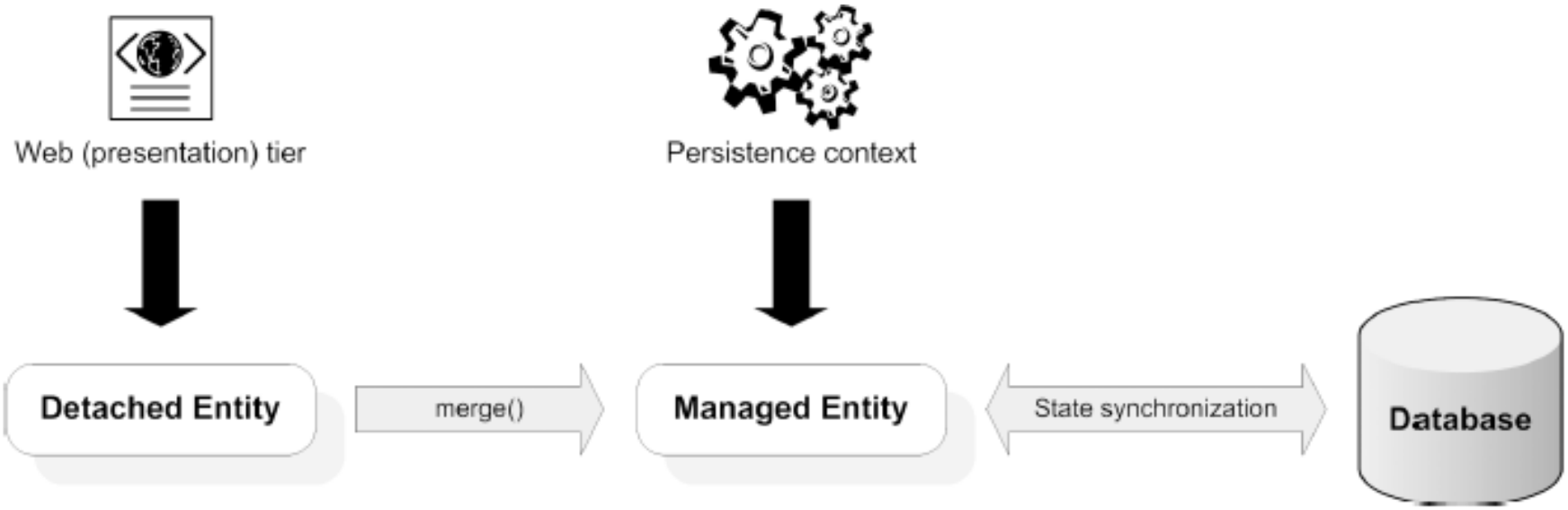
JPA y JEE: sincronización

- Estado Detached (fuera de ámbito, serializada, borrada, clonada) significa que los cambios no se guardan.
- Los EJB Stateless terminan la transacción en cada invocación
- Una interfaz remota implica serialización



JPA y JEE/Web: sincronización

- Para volver a sincronizar el estado de una instancia de una clase entity hay que utilizar merge() (o un método update() de la fachada de Acceso a Datos que lo encapsule)



	Session EJB	
Inyección de dependencia	Stateless	Stateful
EJB Stateless	SI	NO se debe (perdemos el estado del EJB)
EJB Stateful	SI	SI
Servlet	SI	NO: se debe utilizar JDNI lookup() y asignar a un atributo de sesión/request
Clase Java auxiliar llamada desde un Servlet	NO en ningún caso, JDNI lookup() y anotación EJB en el Servlet	
Cliente Java **JEE**	SI	SI
	Session EJB Interface	
Acceso a Datos (Sincronizada)	Local	Remote
EJB Stateful	Entity (Referencia)	No (copia: paso de parámetros por valor)
EJB Stateless	Entity (Referencia) Pero se pierde la conexión con la BD en cada llamada	No (copia)
Servlet	Entity (Referencia) Pero se pierde la conexión con la BD cada vez	No (copia)
Servlet y JSP (a través del lookup() y un atributo de sesión)	Entity (Referencia)	No (copia)
Cliente Java JEE	-	No (copia)