Patrones de diseño

Sesión 2:

Patrones de diseño J2EE para arquitecturas locales sin EJBs

Índice

- **Catálogo básico de patrones para arquitecturas locales sin EJBs**
- **Patrones para la capa de presentación**
- **Patrones para la capa de negocio**
- **Patrones para la capa de integración**

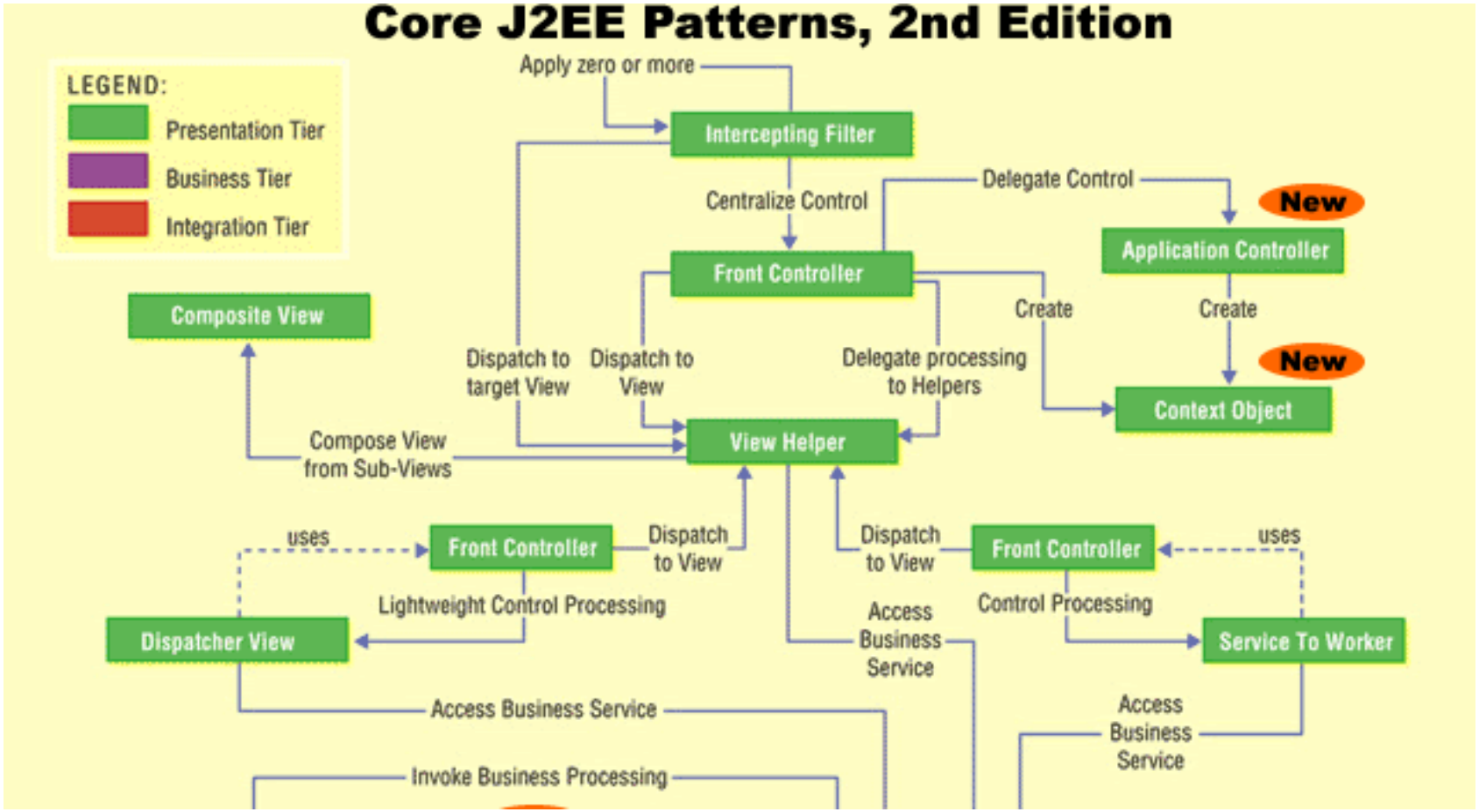
Patrones para aplicaciones J2EE

- Capa de presentación
  - Service to Worker
  - View Helper
- Capa de negocio
  - Business Delegate
  - Business Object + Application Service vs. Service Facade
  - Transfer Object
- Capa de integración
  - DAO vs. Domain Store
  - Service Locator

Índice

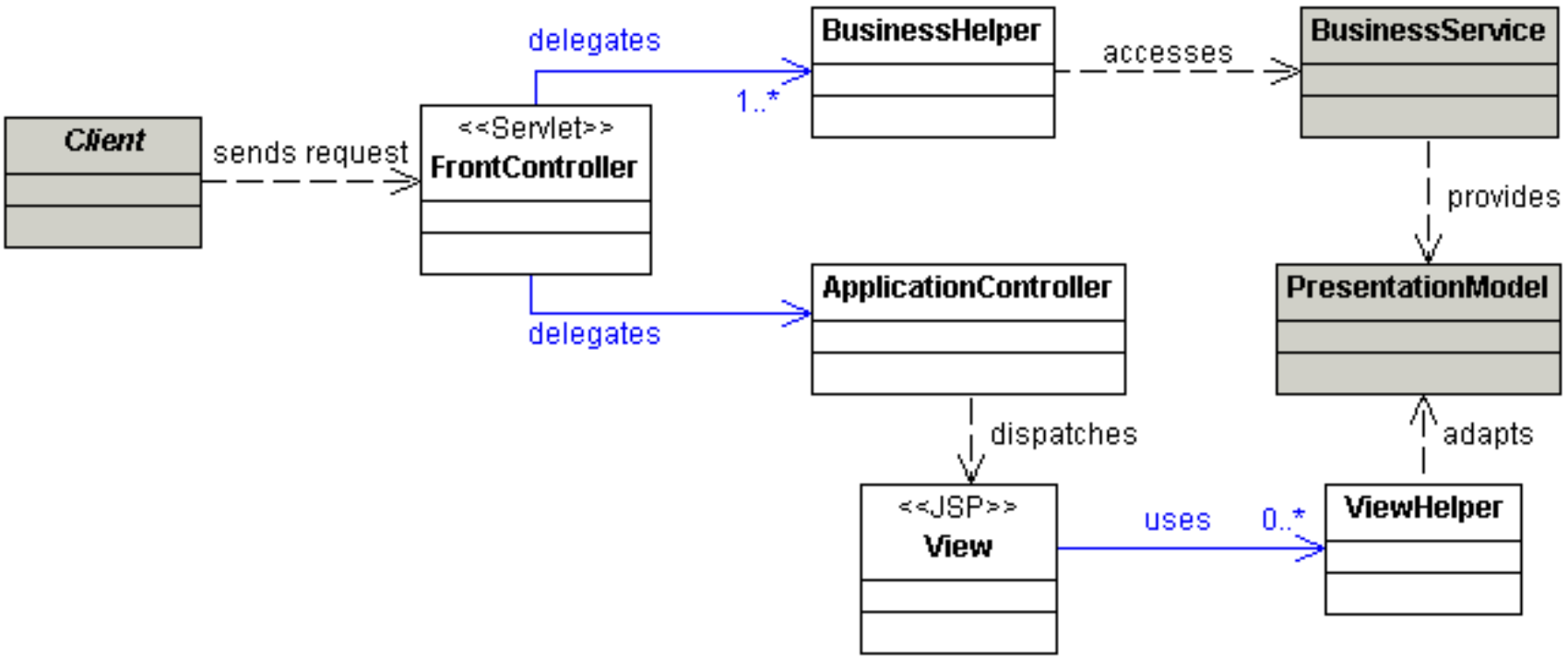
- Catálogo básico de patrones para arquitecturas locales sin EJBs
- Patrones para la capa de presentación
- Patrones para la capa de negocio
- Patrones para la capa de integración

Patrones “Core J2EE” para la capa de presentación



Service to Worker y View Dispatcher

- Básicamente corresponden con el modelo MVC
- Struts implementa Service to Worker



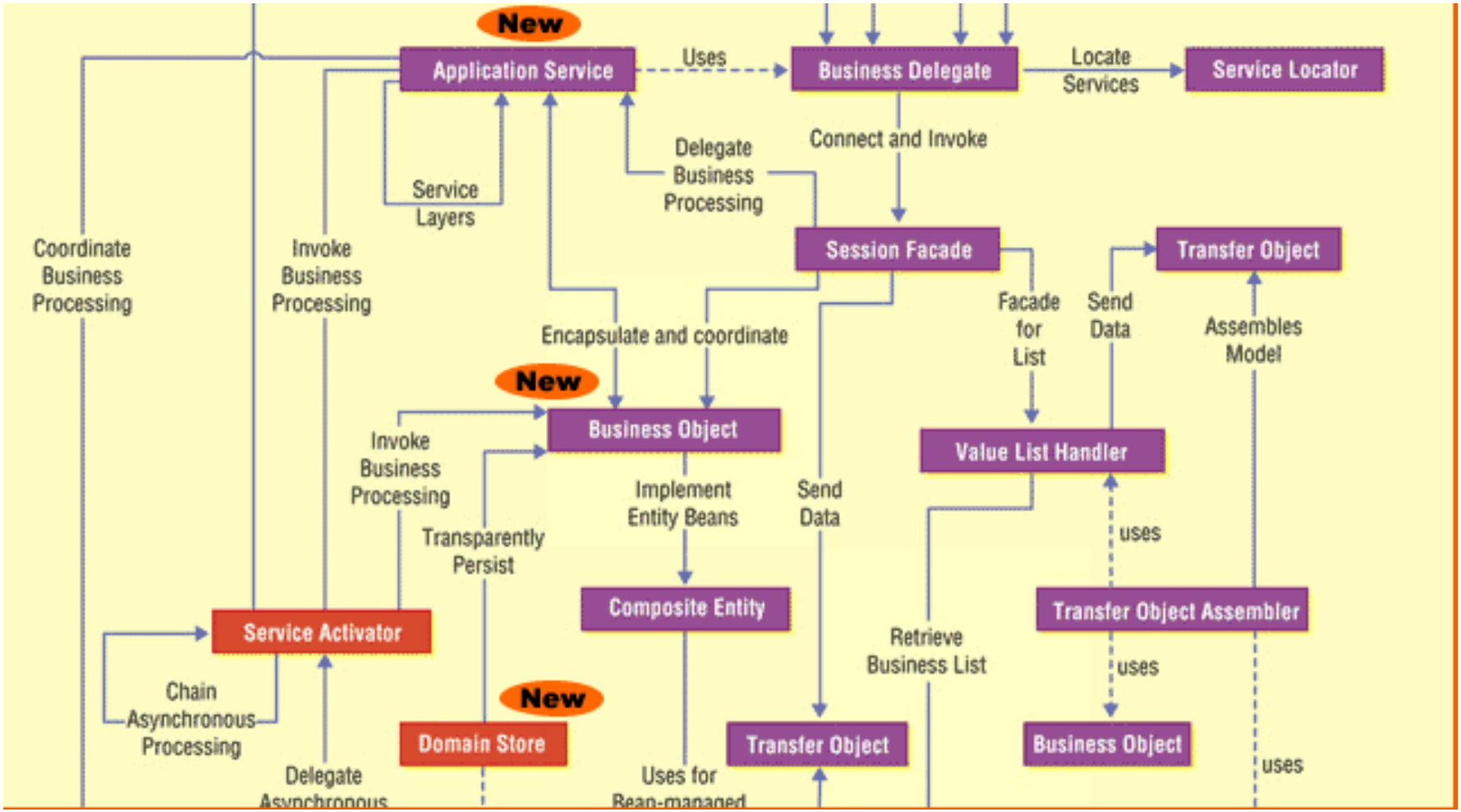
Índice

- Catálogo básico de patrones para arquitecturas locales sin EJBs
- Patrones para la capa de presentación
- Patrones para la capa de negocio
- Patrones para la capa de integración

Índice

- Catálogo básico de patrones para arquitecturas locales sin EJBs
- Patrones para la capa de presentación
- Patrones para la capa de negocio
- Patrones para la capa de integración

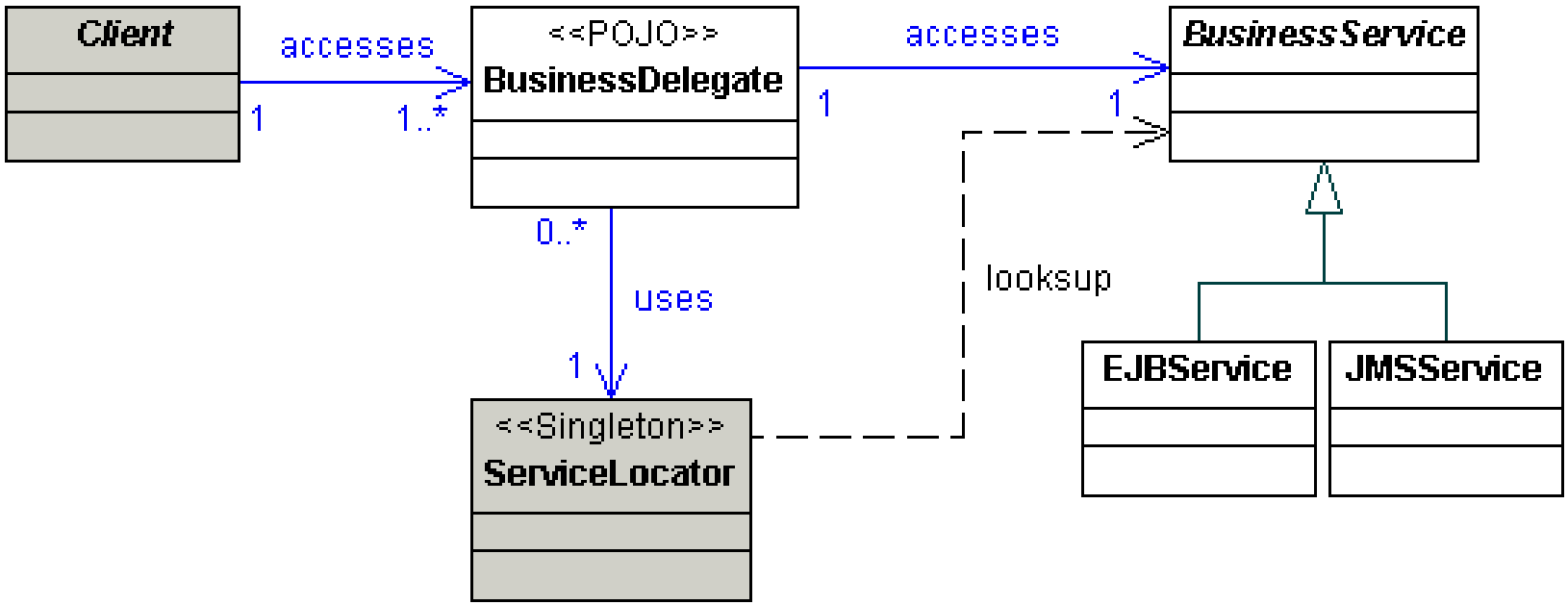
Patrones “Core J2EE” para la capa de negocio



Business Delegate

- **Encapsula la comunicación entre la capa de presentación y la de negocio**
- **Aplicabilidad:**
  - **Los componentes de la capa de presentación necesitan acceder a la de negocio a través de un API estable**
  - **Se desea minimizar el acoplamiento entre presentación y negocio escondiendo los detalles de implementación de esta última capa (¿EJBs? ¿POJOs? ¿a quién le importa?)**
  - **Se quiere esconder las excepciones que los clientes no sabrían manipular (ej. fallo de la base de datos)**

Business Delegate: esquema



Business Delegate: consecuencias

- **Positivas**
  - Se reduce el acoplamiento entre capas
  - Se traducen las excepciones en otras significativas para el cliente
  - Se mejora la disponibilidad de la capa de negocio si reintenta por sí mismo las operaciones

- **Negativas:**
  - Se añade una capa extra que “no hace casi nada” (podría pensarse)

Service Facade vs. Business Objects

- **Hay dos tendencias a la hora de modelar la capa de negocio:**
  - **La capa de negocio proporciona un API de funciones (vale, de métodos, pero como si fueran funciones). Desde fuera veremos una fachada (facade) con este API**
  - **La capa de negocio es un conjunto de objetos. Desde fuera podemos ver el modelo de objetos (sean EJBs o POJOs)**

Service Facade

- Clase que implementa la lógica de negocio, ofreciendo al exterior un API de funciones
- Cuestiones
  - Se suele utilizar cuando no se justifica un modelo OO del dominio por su sencillez. En ese caso el único modelo del dominio que hay son los datos de la B.D.
  - Se suele usar una facade por cada entidad del modelo del dominio (FacadeCliente, FacadePedido, …), pero es una cuestión más de organización que de OO
  - En arquitecturas distribuidas, las facades se justifican sobre todo por cuestiones de eficiencia (lo veremos)
- Si esto nos parece demasiado sencillo, siempre podemos hacer que la lógica de negocio la implemente otra clase más

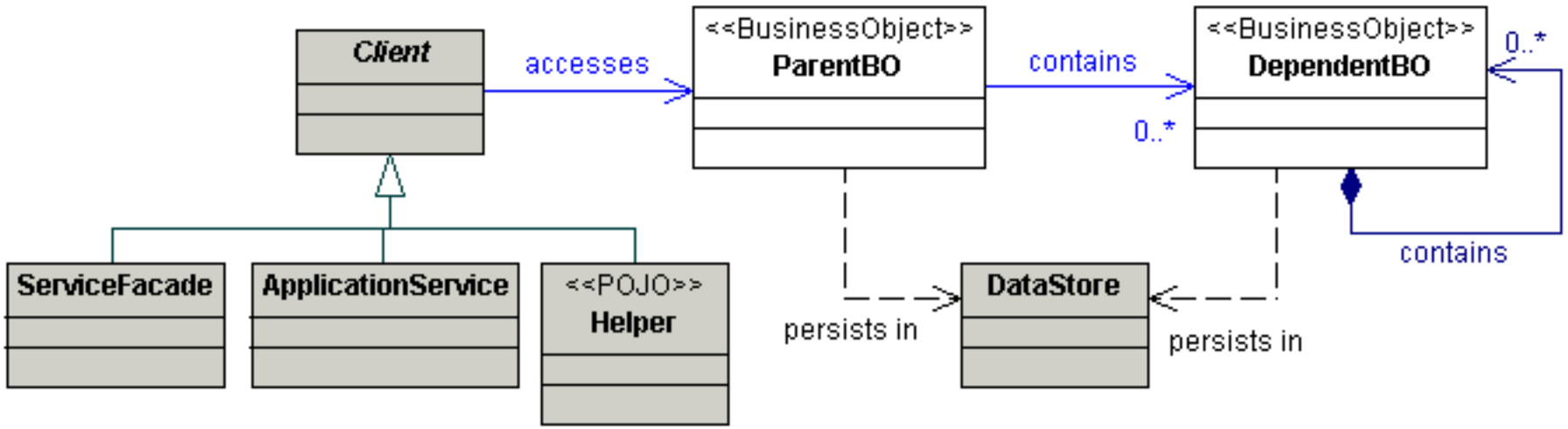
Business Objects

- Objetos que representan el modelo de dominio (clientes, usuarios, pedidos, mensajes, productos,…)
- En teoría, modelar esto es la parte central de la aplicación
- ¿Qué implementa un B.O.?
  - Estado
  - Lógica de negocio propia
- La lógica de negocio "no propia" se saca fuera: ej. un usuario envía un mensaje a un foro.

Implementación de Business Objects

- **Básicamente 2 opciones**
  - **POJOs**
  - **EJBs**
- **Ventajas de POJOs:**
  - **Implementación sencilla**
  - **No hace falta servidor de aplicaciones**
- **Ventajas de EJB**
  - **Transacciones, seguridad, …**

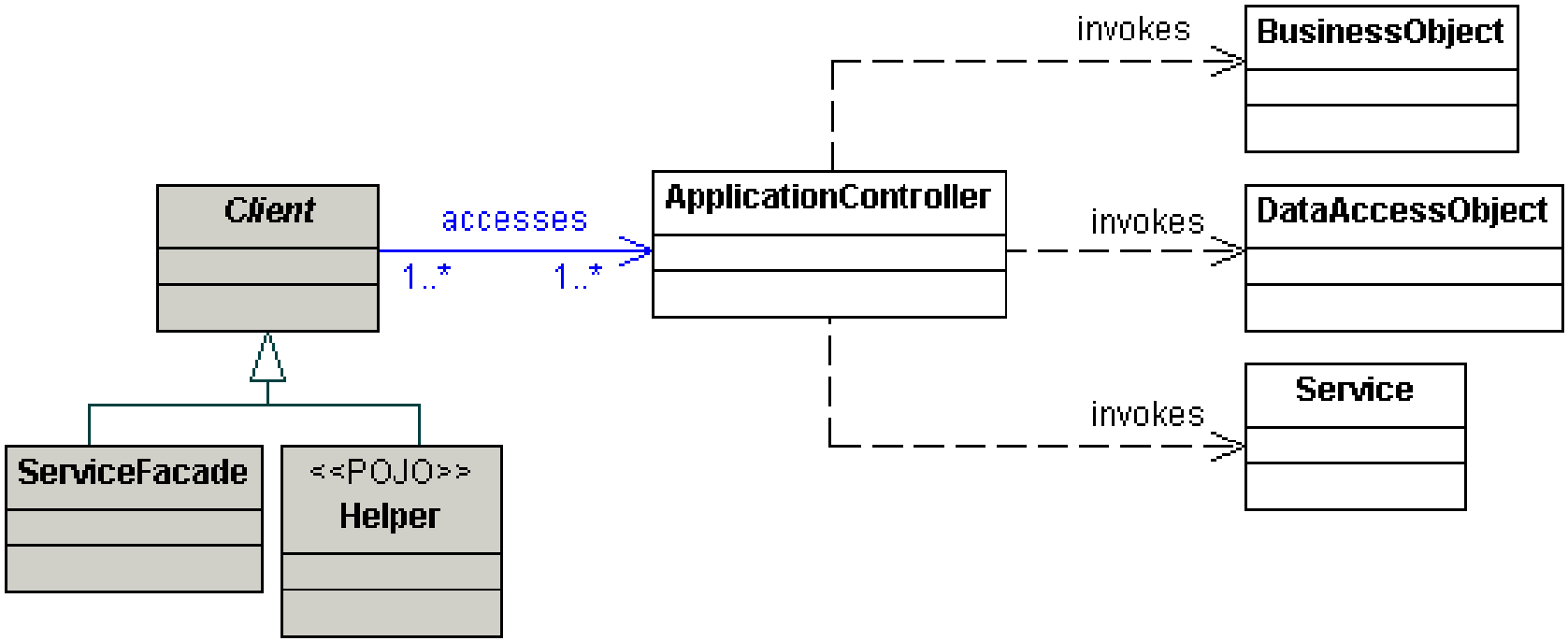
Business Object: esquema



Application Service

- **Encapsula la lógica que implica a varios B.O. no dependientes. Proporciona una capa de servicios, al estilo de una facade (¡vaya!)**
- **Aplicabilidad**
  - **Lógica que implica a varios B.O**
  - **Se desea un API de "grano grueso" de cara al exterior (no getXXX, setXXX,…)**
  - **La lógica cambia según el tipo de cliente u otro factor, aunque siempre implique al mismo B.O.**

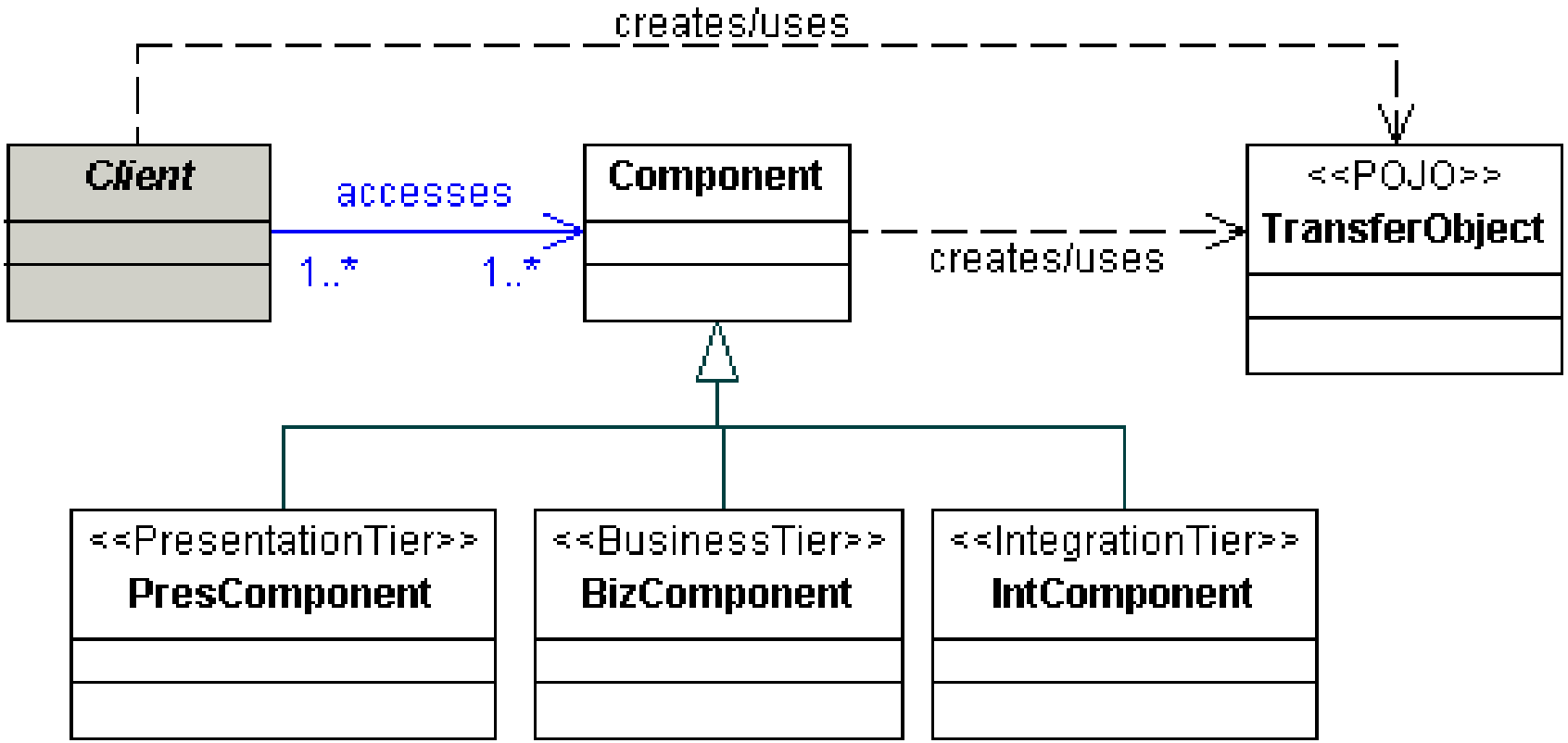
Application Service: diagrama



Data Transfer Object

- Vehículo de transporte de información entre capas
- Evita los getXXX, setXXX
  - Eficiencia en aplicaciones distribuidas
  - En aplicaciones locales, todo queda “más organizado”
- Cuestiones
  - En su forma más simple, un DTO es un JavaBean
  - En teoría, se recomienda un DTO para cada caso de uso. Los DTOs NO son BOs (aunque a veces se utilizan como tales)

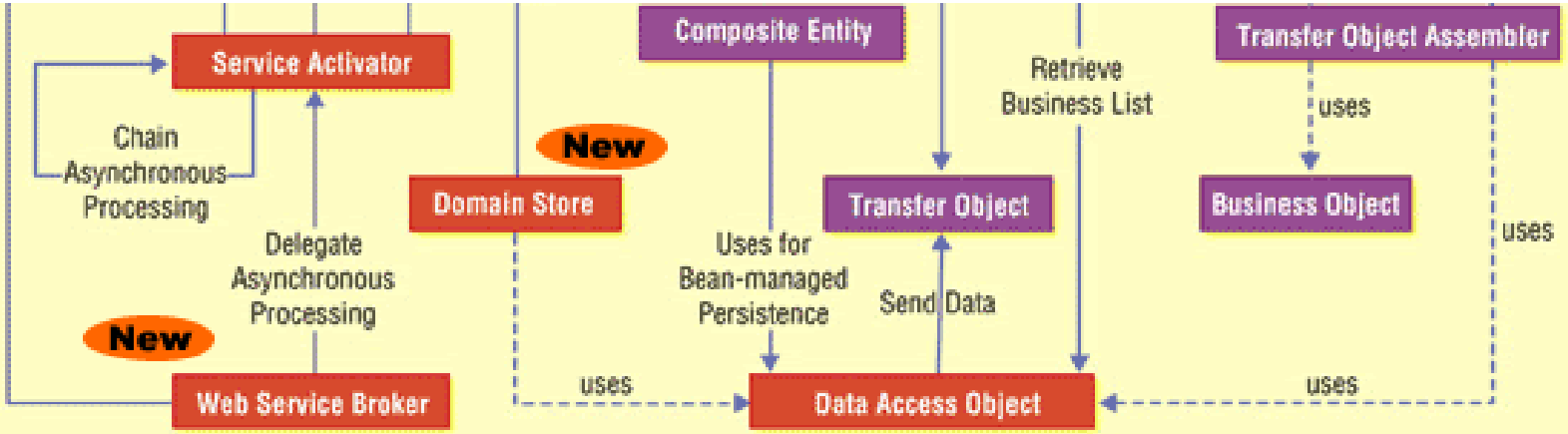
Data Transfer Object: esquema



Índice

- Catálogo básico de patrones para arquitecturas locales sin EJBs
- Patrones para la capa de presentación
- Patrones para la capa de negocio
- Patrones para la capa de integración

Patrones “Core J2EE” para la capa de integración

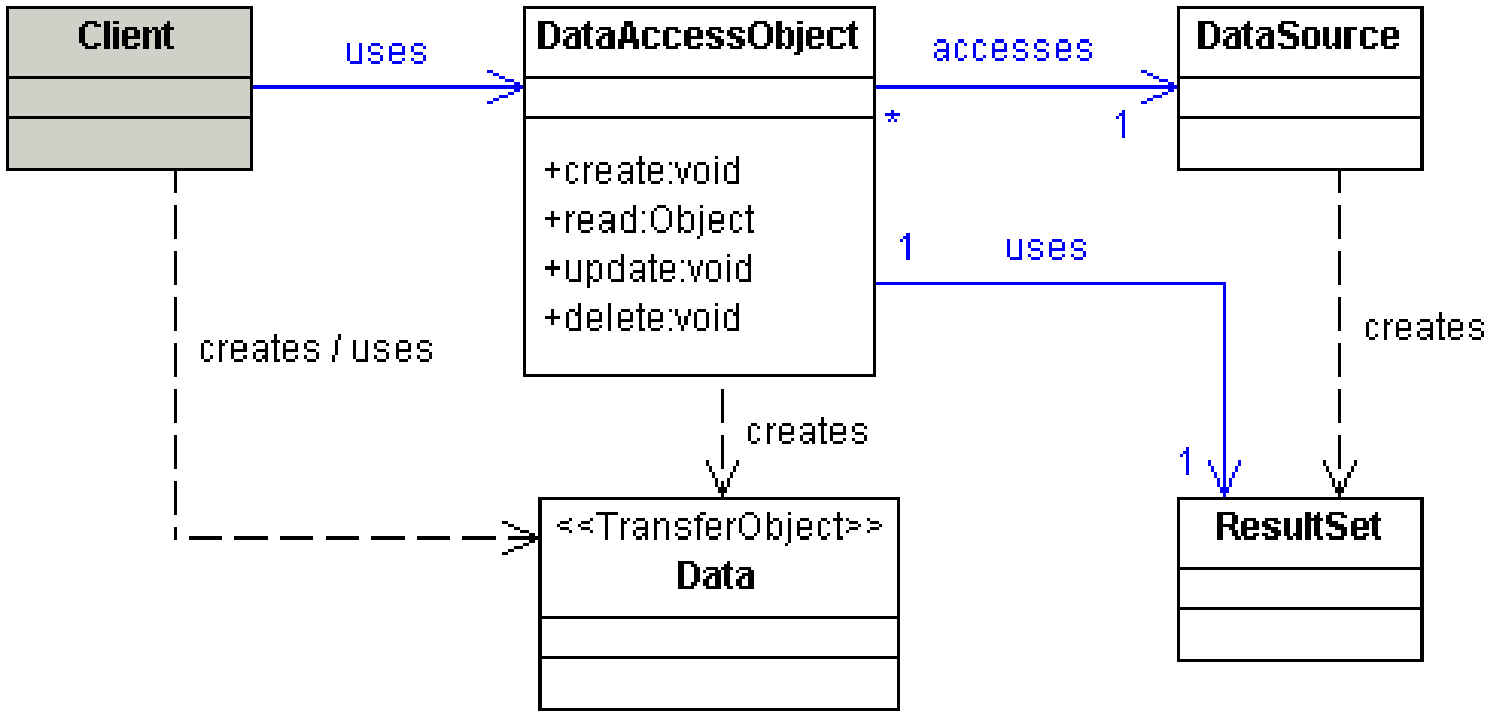


Persistencia de la información

- **Una de las funcionalidades más importantes de la capa de integración (además de la integración con otros sistemas)**
- **Posibilidades**
  - **Los objetos de la capa de negocio acceden directamente a las fuentes de datos (¡puag!)**
  - **El acceso a las fuentes de datos se encapsula en algún patrón (Data Access Object)**
  - **Se utilizan EJBs con persistencia manejada por el contenedor (CMP)**
  - **Se utiliza persistencia transparente (Java Data Objects):**
    - **No impone requerimientos especiales sobre los objetos (POJOs normalitos)**

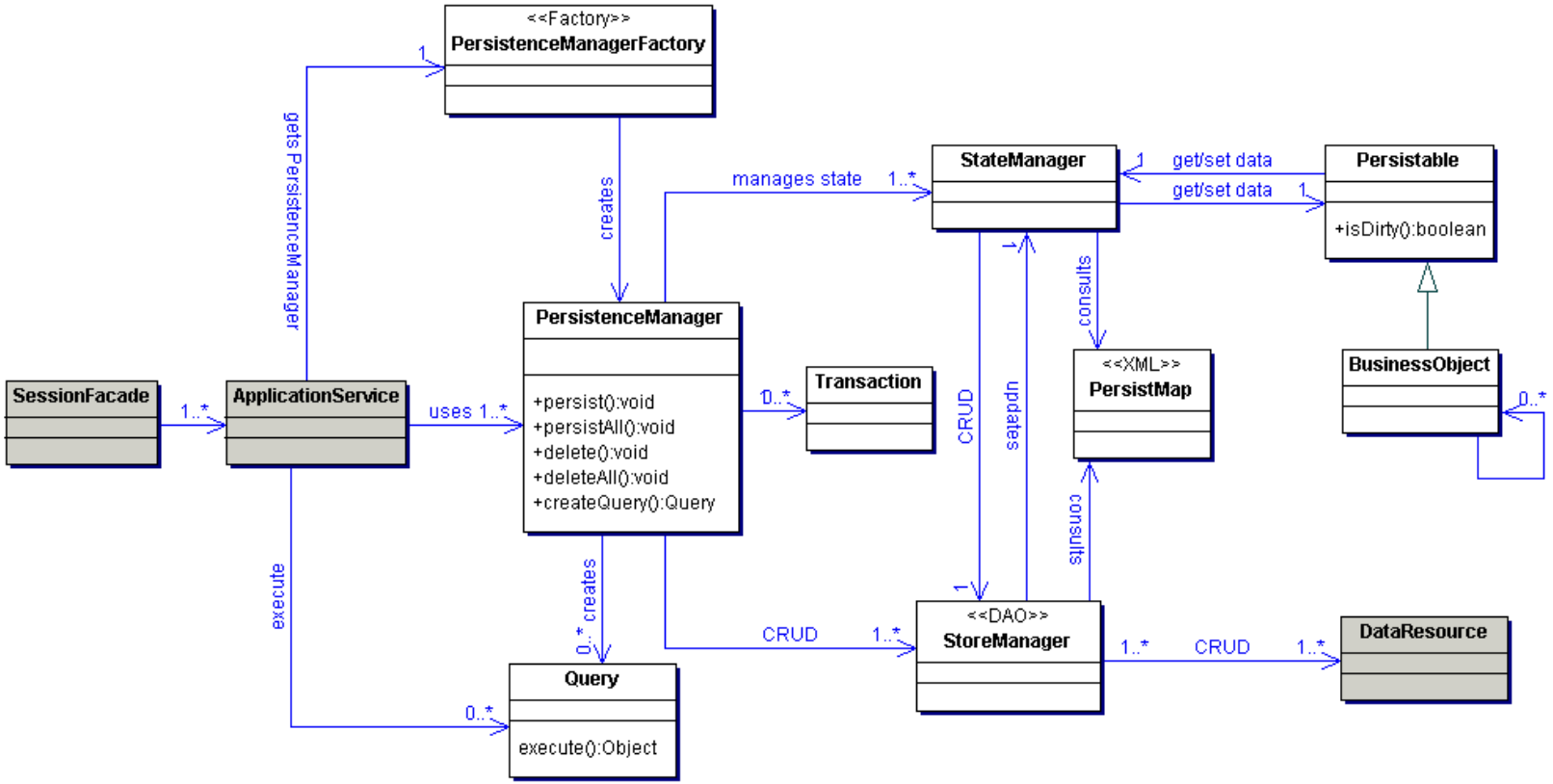
Data Access Object

- Encapsula el acceso a una fuente de datos (normalmente una R.D.)



Domain Store

- **Persistencia transparente**
- **Macro-patrón**



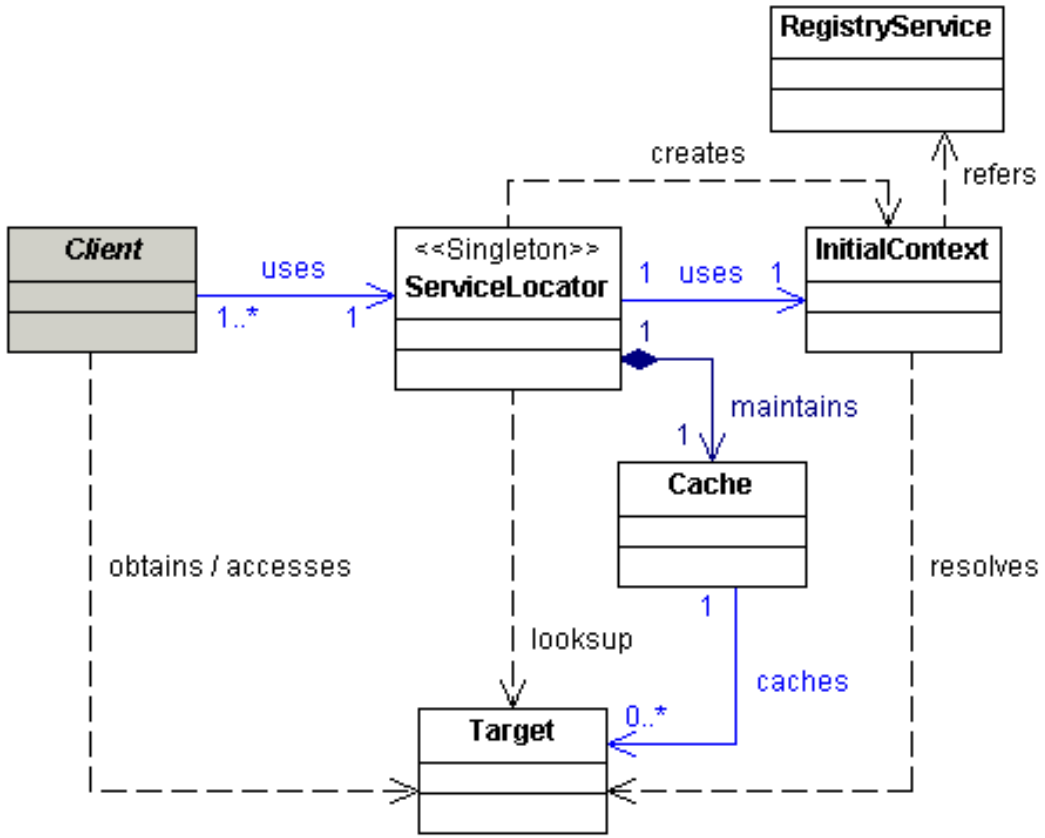
Domain Store

- **Opciones de implementación**
  - **Manual: muy costosa y compleja**
  - **Utilizar una herramienta estandar al estilo de JDO (o Hibernate o similares)**

- **Cuestión básica: ¿el esfuerzo merece la pena?**

Service Locator

- **Localizar servicios (DataSources, EJBs, …) de manera transparente**



Singleton [GoF]

```
public class Singleton {
    private static Singleton elSingleton;

    private Singleton() {
            …
    }

    public static Singleton getInstance() {
            if (elSingleton==null)
                    elSingleton = new Singleton();
            return elSingleton;
    }

}
```