Aplicaciones Open Source

Semana07

Carlos A. Quinto Cáceres
pcsicqui@upc.edu.pe

Agenda

- Conceptos

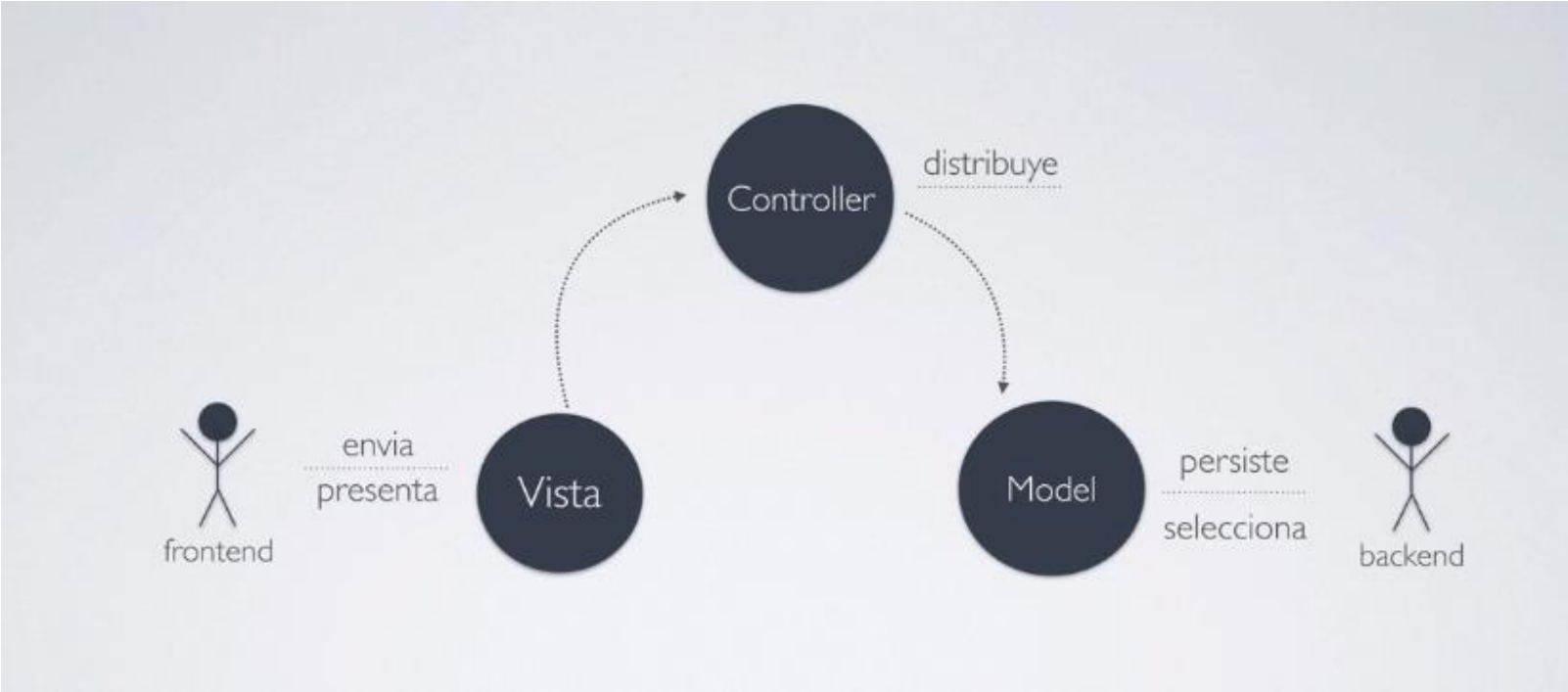
MVC

Mode-View-Controller

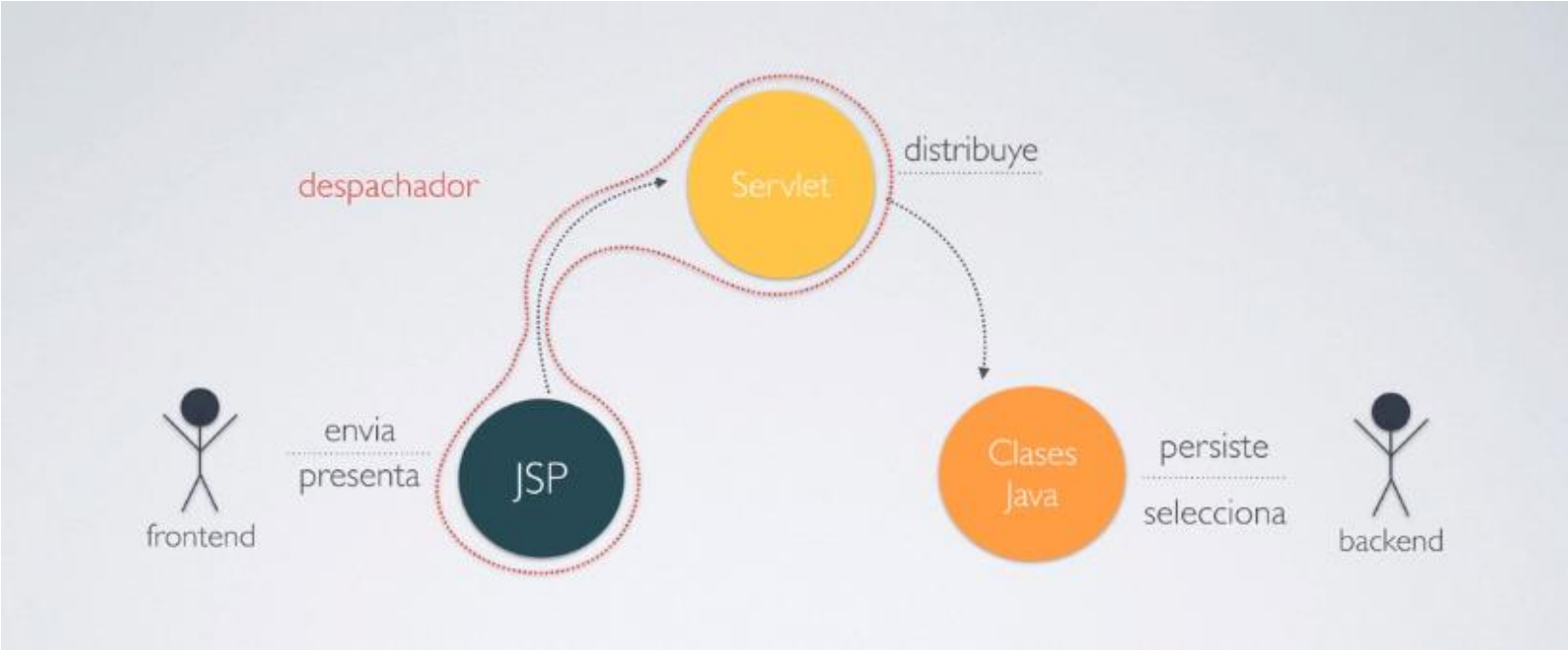
MVC

- Nos permite separar una aplicación Web en tres componentes principales:
  - Modelo, son los objetos que nos permiten implementar la lógica de la aplicación y también obtener y guardar información en un origen de datos.
  - Vista, es el componente que nos permite mostrar en la aplicación los datos, es la interfaz visible de la aplicación con los usuarios.
  - Controlador, es el que se encarga de manejar las peticiones de los usuarios, trabaja con los modelos para acceder a los datos y selecciona la vista que se le mostrará finalmente a los usuarios.

MVC


Controller
distribuye
frontend
envia
presenta
Vista
Model
persiste
selecciona
backend

MVC


despachador
Servlet
distribuye
envia
presenta
JSP
Clases
Java
persiste
selecciona
frontend
backend

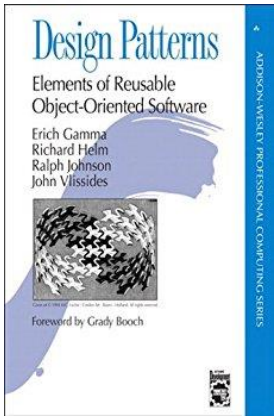
DAO

Data access object

Patrones de diseño

Los patrones de diseño:

- Son soluciones reutilizables a problemas de desarrollo de software comunes.
- Están documentados en catálogos de patrones.
  - *Design Patterns: Elements of Reusable Object-Oriented Software* (Patrones de diseño: elementos del software reutilizable orientado a objetos), de Erich Gamma et al. (conocido como "Gang of Four", la banda de los cuatro, por sus cuatro autores)
- Forman un vocabulario para hablar sobre el diseño.



DAO

Data Access Object

Provee una interfaz común entre la aplicación y componentes de almacenamiento de datos

El patrón de objeto de acceso a datos (DAO) se usa al crear una aplicación que debe mantener información. El patrón DAO:

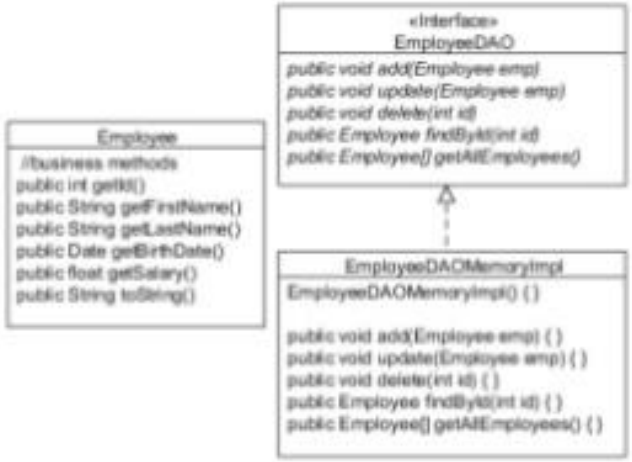
- Separa el dominio de problemas del mecanismo de persistencia.
- Usa una interfaz para definir los métodos usados para la persistencia. Una interfaz permite sustituir la Implementación de la persistencia por:
  - DAO basados en memoria como solución temporal
  - DAO basados en archivos para una versión inicial
  - DAO basados en JDBC para soportar la persistencia de la base de datos
  - DAO basados en la API de persistencia Java (JPA) para soportar la persistencia de la base de datos

DAO

Observe la combinación de métodos de persistencia y métodos de negocio.

Antes del patrón DAO

El patrón DAO extrae la lógica de persistencia de las clases de dominios y las traslada a clases distintas.



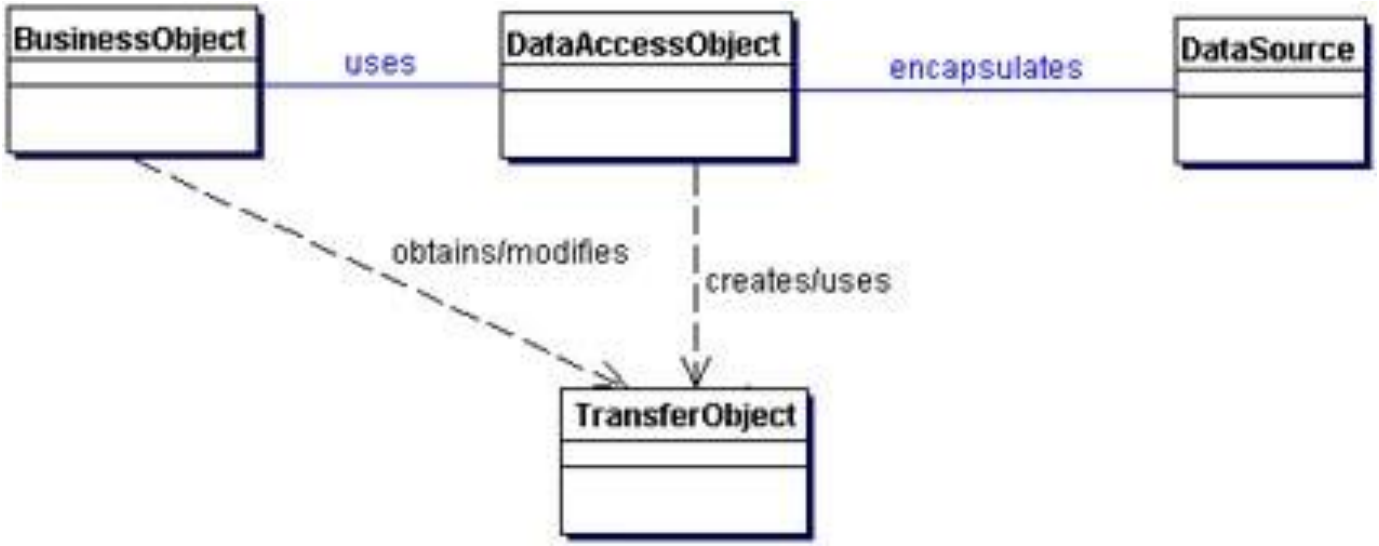
Después de la refactorización del patrón DAO

DAO

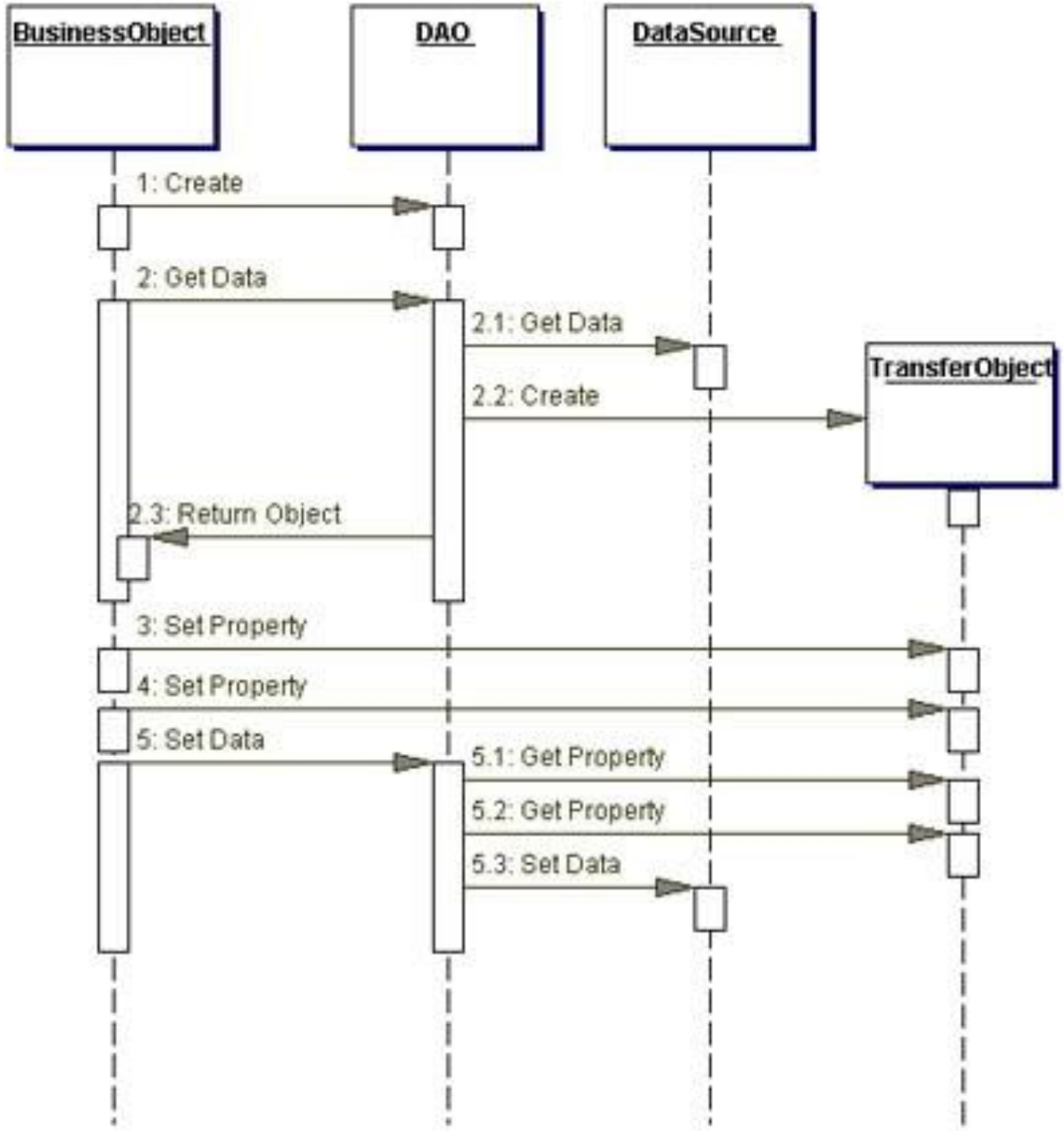
Participantes:

- Data Access Object Interface, donde se definen las operaciones que se pueden o que se ejecutarán en los modelos.
- Data Access Object Class, son las clases que implementan las interfaces, aquí se obtiene o envía la información al origen de datos (ej: base de datos).
- Model Object o Transfer Object (clase), es el objeto que contiene los métodos set y get para almacenar y obtener información a través de las clases de DAO.

DAO


BusinessObject
uses
DataAccessObject
encapsulates
DataSource
obtains/modifies
creates/uses
TransferObject

DAO


BusinessObject
DAO
DataSource
TransferObject
1: Create
2: Get Data
2.1: Get Data
2.2: Create
2.3: Return Object
3: Set Property
4: Set Property
5: Set Data
5.1: Get Property
5.2: Get Property
5.3: Set Data

CDI

Context Dependency Injection

CDI

- Inyección de dependencias es un patrón de diseño orientado a objetos.
- Aquí se establecen los objetos a una clase, en lugar que la propia clase genere los objetos.
- El objetivo es que el desarrollo se vuelva independiente y la comunicación se solo por interfaces, eliminando la instrucción **new**.

CDI

Ejemplo SIN DI

```
public class Robot{

   private BrazoX brazo= new BrazoX();
   private CabezaX cabeza= new CabezaX();

   public void disparar() {
      brazo.fire();
   }
}
```

Ejemplo CON DI - Set

```
public class Robot{

   private BrazoX brazo;
   private CabezaX cabeza;

   public void setBrazo(BrazoX brazo) {
         this.brazo = brazo;
   }
   public void setCabeza(CabezaX cabeza) {
         this.brazo = brazo;
   }
```

Modelo Tradicional

CDI

CDI

Ejemplo CON DI - Constructor

```
public class Robot{

   private BrazoX brazo;
   private CabezaX cabeza;

   public Robot (BrazoX brazo, CabezaX cabeza) {
         this.brazo = brazo;
         this.cabeza = cabeza;
    }

}
```

Ejemplo CON DI - Interfaz

```
public class Robot{

   private IBrazo brazo;
   private ICabeza cabeza;

   public Robot(IBrazo brazo, ICabeza cabeza) {
       this.brazo = brazo;
       this.cabeza = cabeza;
   }
   public setBrazo(IBrazo brazo) {
             this.brazo = brazo;
   }
   public setCabeza(ICabeza cabeza) {
             this.brazo = brazo;
   }
  public void disparar(){
    brazo.disparar();
  }
}
```

```
public class IBrazo{
      void disparar();
}

 public class BrazoX
 implements IBrazo{
       public void disparar(){
          //código marca X
       }
 }

 public class BrazoY
 implements IBrazo{
       public void disparar(){
          //código marca Y
       }
 }
```

CDI

Anotaciones

@Inject | *@Autowired (Spring)*

@EJB

@Resource

@Any

@Named | *@Component (Spring)*

@PostConstruct

@PreDestroy

CDI - Ventajas

- Bajo acoplamiento de código
- Testing sencillo
- Mejor abstracción en capas
- Diseño orientado a objetos
- Favorece al uso de patrones de diseño complementarios

CDI - Desventajas

- Aumento de complejidad del código
- Rendimiento bajo para aplicaciones críticas o que requiera interacción aguda con el hardware.

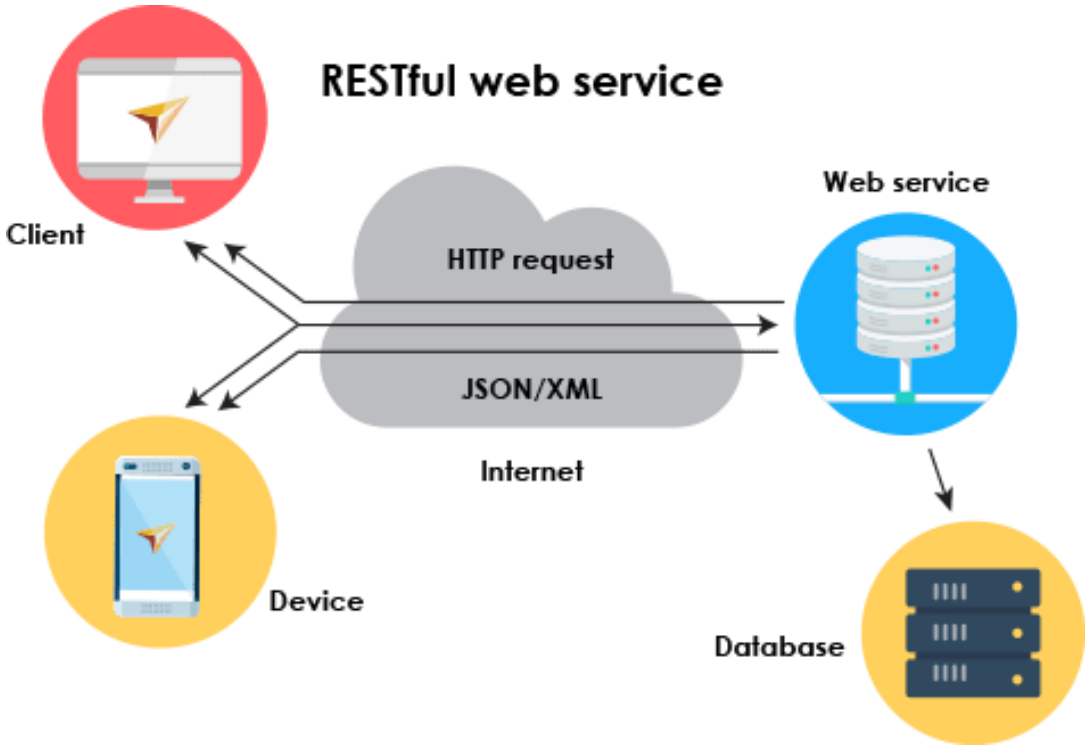
Servicios Web

REST

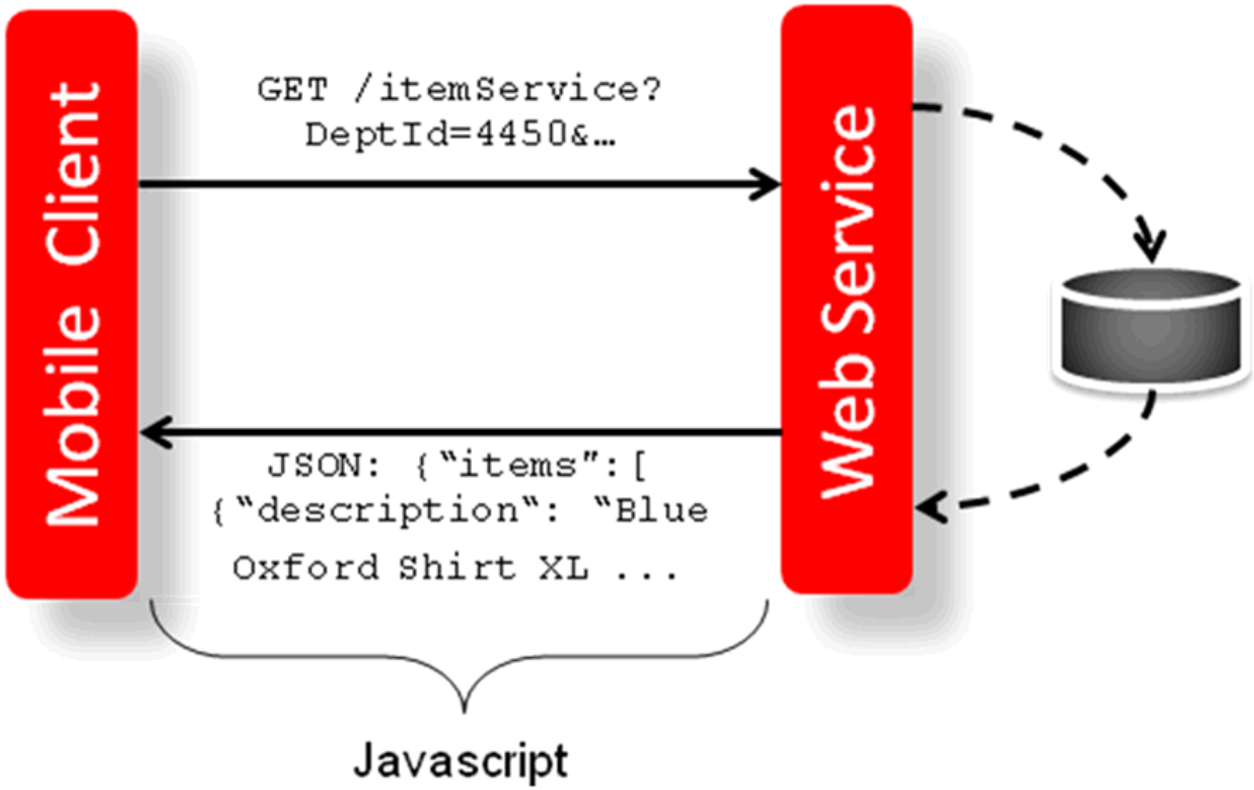
REST

- Representational State Transfer.
- Es un estilo arquitectónico para diseñar sistemas distribuidos.
- Principios:
  - Se exponen a través de URI simples.
  - Transfieren JSON o XML
  - Usa métodos HTTP como GET y POST

Servicios Web



Servicios Web



Anotaciones

Anotación	Descripción
@Path	Especifica como se accede o estará alojada la clase Java
@GET	Es el método usado para la petición, HTTP GET
@POST	Es el método usado para la petición, HTTP POST
@Consumes	Para especificar el tipo de medio que puede consumir
@Produces	Es utilizado para especificar el tipo de medio que produce como respuesta y envía al cliente.

EJB

Enterprise Java Beans

EJB

- Es un componente del lado del servidor.
- Encapsula la lógica de negocio de una aplicación.
- La lógica de negocio es el código que permite cumplir el objetivo de la aplicación.

¿Cuándo usarlos?

- Si la aplicación debe escalar, que permita el acceso de un creciente número de usuarios, se necesitaría distribuir los componentes de la aplicación a través de diversas máquinas.
- Las transacciones deben asegurar la integridad de datos, los EJB soportan transacciones.
- La aplicación tendrá diversos clientes.

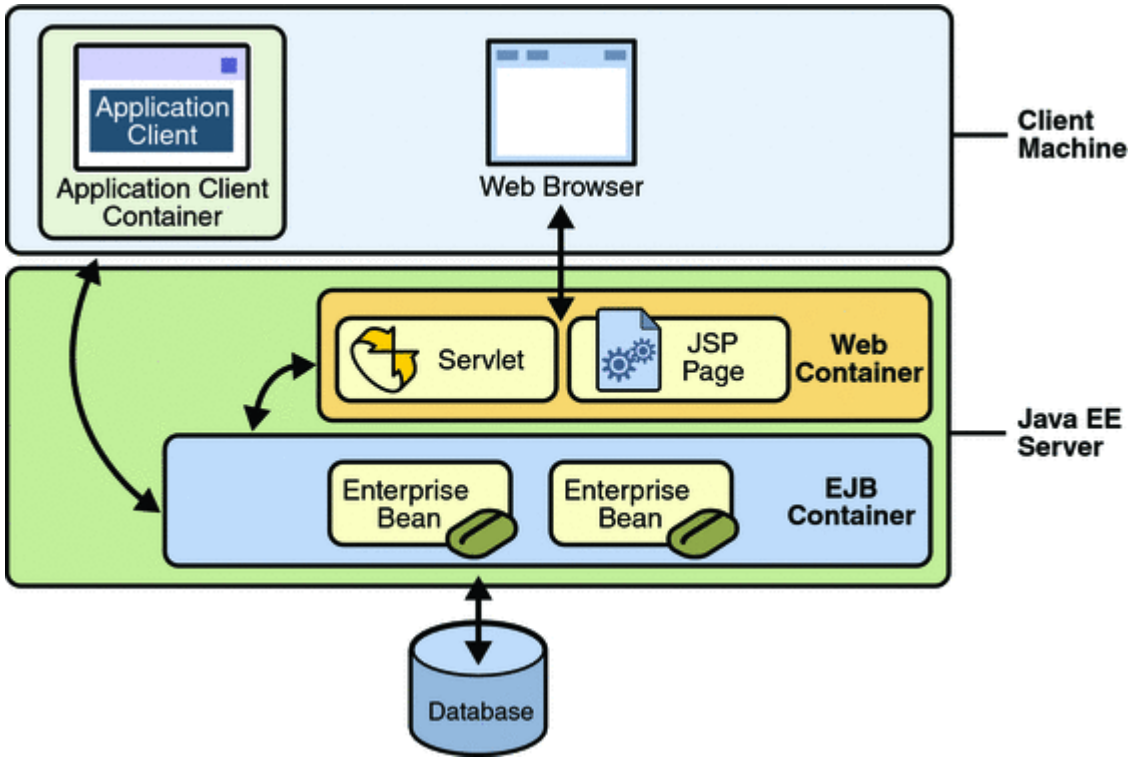
Beneficios

- Manejo de transacciones
- Seguridad, comprobación de acceso a los beans
- Concurrencia
- Persistencia

Desventajas

- Complejidad de implementación
- Rendimiento para aplicaciones cr´ticas

EJB


Application Client
Application Client Container
Web Browser
Client Machine
Servlet
JSP Page
Web Container
Enterprise Bean
Enterprise Bean
EJB Container
Java EE Server
Database