Desarrollo de aplicaciones Java EE con Struts 2, Spring y EJB 3.0

¿Qué es J2EE?

- *Conjunto de especificaciones y prácticas que permiten desarrollar, desplegar y gestionar aplicaciones multicapa*" – Sun Microsystems.

- Lógica dividida en componentes.

- Componentes divididos en capas.

Componentes J2EE

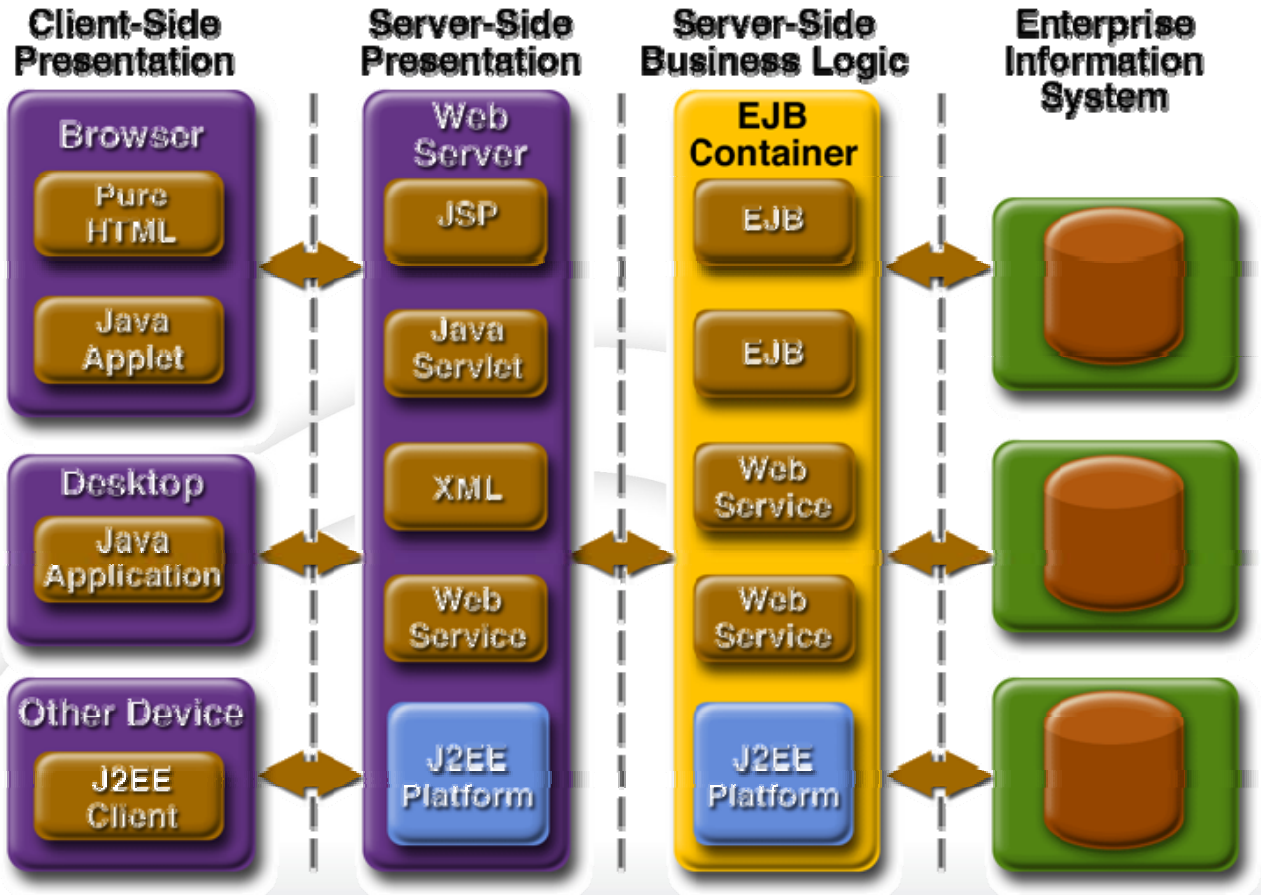
Salenda

- Componentes software autocontenidos.
- Ensamblados en una aplicación J2EE, con sus clases y ficheros relacionados.
- Tipos de componentes:
  - Clientes de la aplicación y applets – se ejecutan en el cliente.
  - Componentes web (JSP & Servlet) – se ejecutan en el servidor.
  - Componentes de negocio (EJB) – se ejecutan en el servidor.

Clientes J2EE

- Clientes web (aka "Clientes ligeros").
  - Páginas web dinámicas – generadas por componentes web de la capa Web.
  - Navegador web – renderiza las páginas recibidas del servidor.
- Aplicaciones cliente (aka "Clientes pesados").
  - Interfaz de usuario (Swing / AWT / SWT).
  - Se ejecutan en el PC del cliente.
  - Acceden directamente a los componentes de negocio

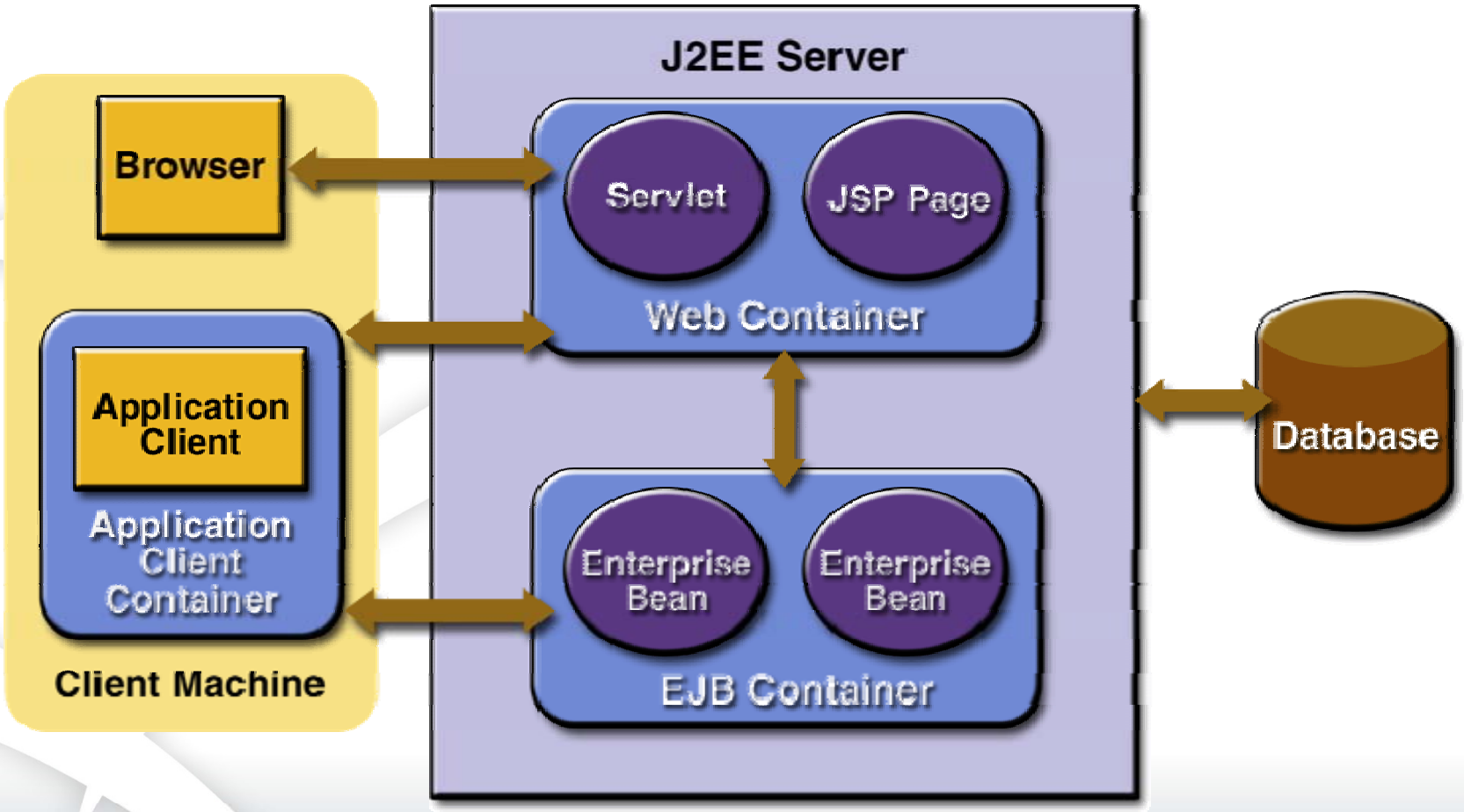
Capas de J2EE



Contenedores J2EE

- Interfaz entre un componente y la funcionalidad de bajo nivel de la plataforma que soporta ese componente.
- Tipos:
  - Servidor J2EE – proporciona contenedores web y EJB.
  - Contenedor de EJB's – gestiona la ejecución de EJB's.
  - Contenedor web – gestiona la ejecución de JSP's y Serlvets.

Contenedores J2EE



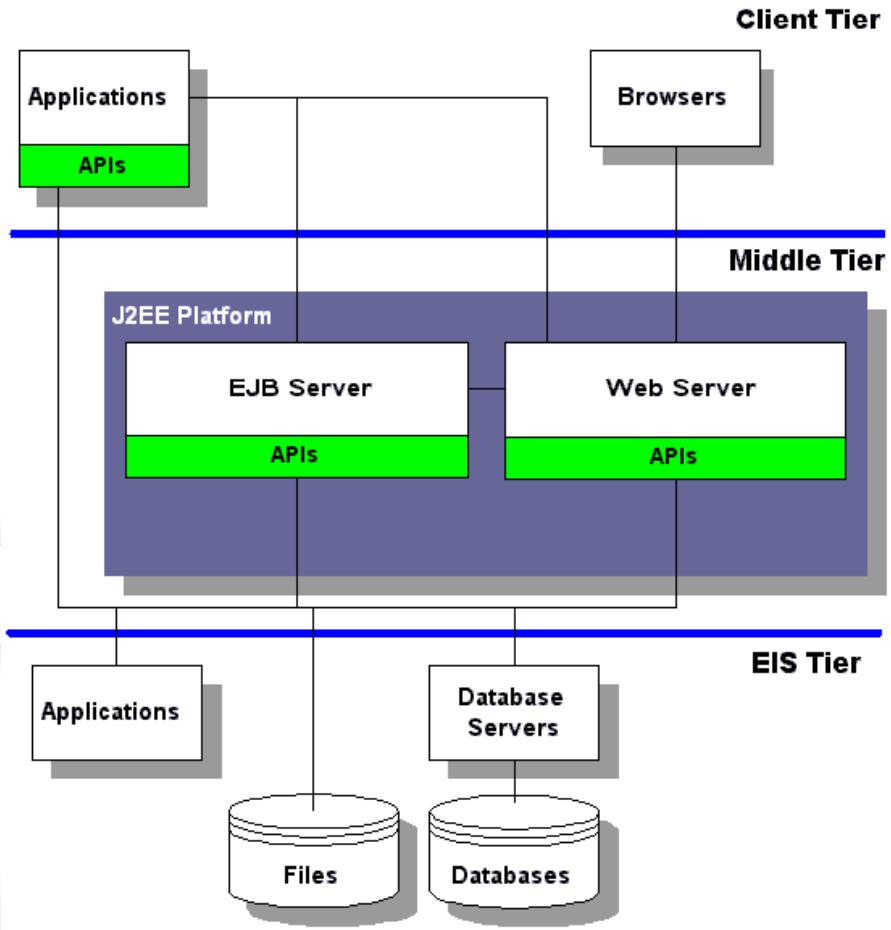
Empaquetado

- Módulo: uno o más componentes J2EE para el mismo tipo de contenedor + descriptor.
- Descriptor de despliegue (DD): documento XML que describe las propiedades de despliegue de un módulo.
- Tipos de módulos:
  - EJB's: .jar
  - Web: .war
  - Resource Adapter: .rar
  - Cliente de la aplicación: .jar
- Un fichero EAR contiene una aplicación J2EE, que puede estar compuesta por uno o varios módulos

Servidores de aplicaciones

- Software que ayuda al desarrollo, despliegue y control de aplicaciones empresariales (distribuidas).
- Sirven como contenedores de los componentes de una aplicación J2EE
- Provee middleware para acceso a servicios subyacentes
  - Seguridad
  - Persistencia
  - Acceso a datos
  - ...

Arquitectura de un servidor de aplicaciones



Contenedores de aplicaciones

Salenda

- Existen muchos servidores de aplicaciones
  - Comerciales
    - WebSphere (IBM)
    - WebLogic (BEA Systems)
    - Oracle AS (Oracle Corp)
  - Libres
    - JBboss (JBoss Inc)
    - Tomcat (Apache Foundation)
    - Jonas (ObjectWeb)
- Comparativa de servidores de aplicaciones
  - http://en.wikipedia.org/wiki/Matrix_of_Application_Servers#Java

Servidores de aplicaciones

Salenda

- Todos los comerciales ofrecen
  - Contenedores de jsp's y servlets
  - Contenedores de EJB's
  - Funcionalidades añadidas (dependiendo del vendedor)
- Existen otros servidores de aplicaciones que solo ofrecen contenedor de jsp's y servlets (tomcat)

Servlets

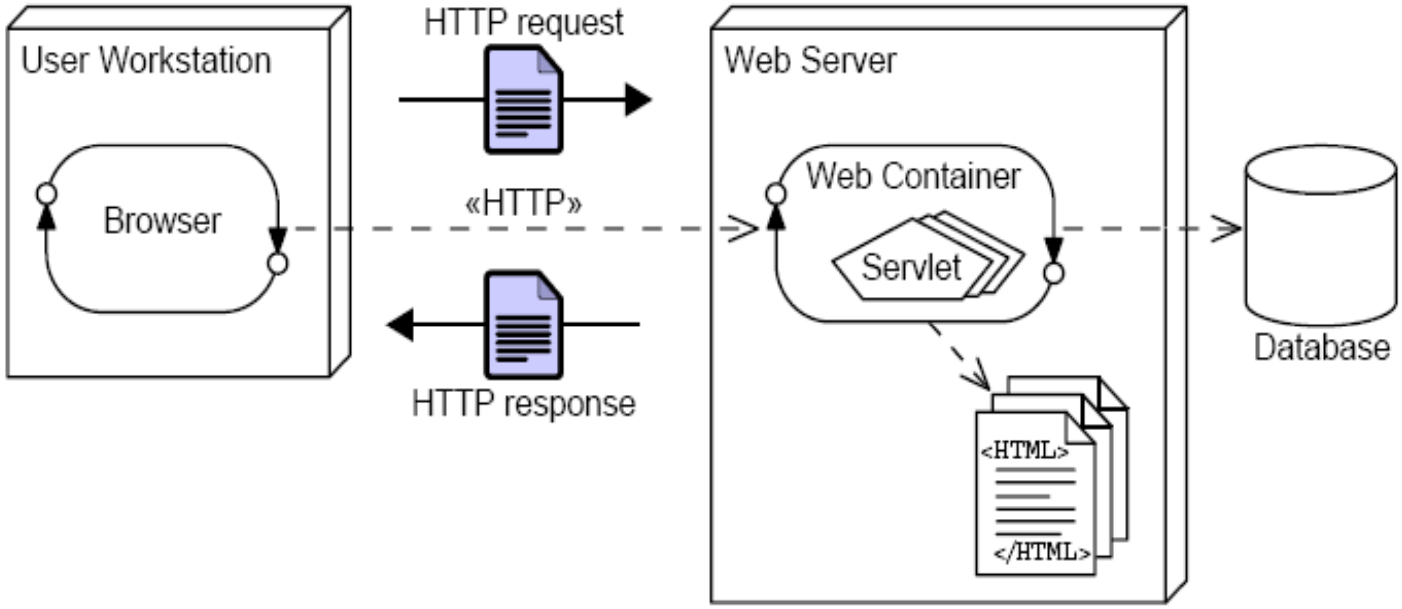
- Extienden la funcionalidad de un servidor web
- Componentes ejecutables en un contenedor web
- Encargados de escuchar a las peticiones de un cliente
- Encargados de responder a las peticiones de un cliente
- Mejor rendimiento que CGI : uso de threads

JSP

- Separacion entre logica de negocio y presentacion
- Facil mantenimiento
- Basado en la tecnologia de servlets
- Orientado a la logica de presentacion

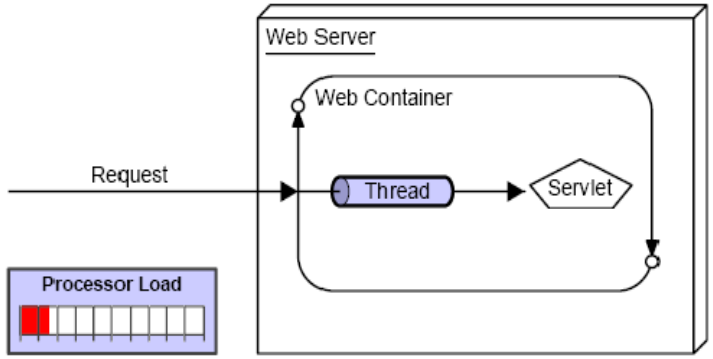
Servlets en el servidor web

- Diagrama de despliegue de un servidor web con un contenedor web:

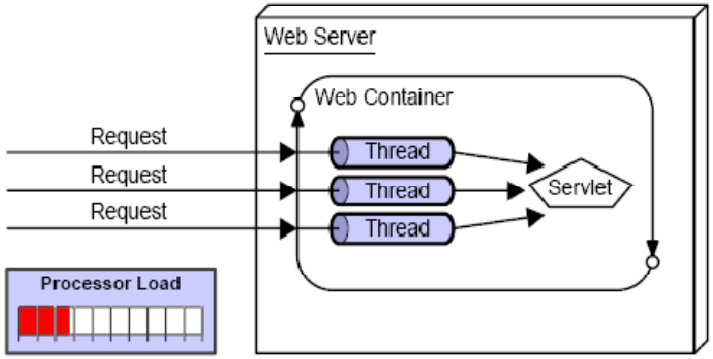


Ejecución de los servlets

- Para una petición:



- Para 'n' peticiones



Ventajas y desventajas de los Servlets

Salenda

- Ventajas:
  - Prestaciones (los threads son más rápidos y ligeros que los procesos)
  - Escalable
  - El lenguaje Java es orientado a objetos y robusto.
  - El lenguaje Java es multiplataforma
- Desventajas:
  - ¿Cómo separar la lógica de negocio y la presentación?
  - Cuestiones de concurrencia

Tecnología JavaServer Pages (JSP)

- Las plantillas de código son como páginas HTML estáticas, pero con código Java embebido para permitir la generación dinámica de datos y HTML.
- Ejemplo:

```
<table border="1" cellspacing="0" cellpadding='5'>
        <tr><th>number</th><th>squared</th></tr>
        <% for ( int i=0; i<10; i++ ) { %>
                <tr><td><%= i %></td>
                    <td><%= (i * i) %></td></tr>
        <% } %>
</table>
```

Table of numbers squared:

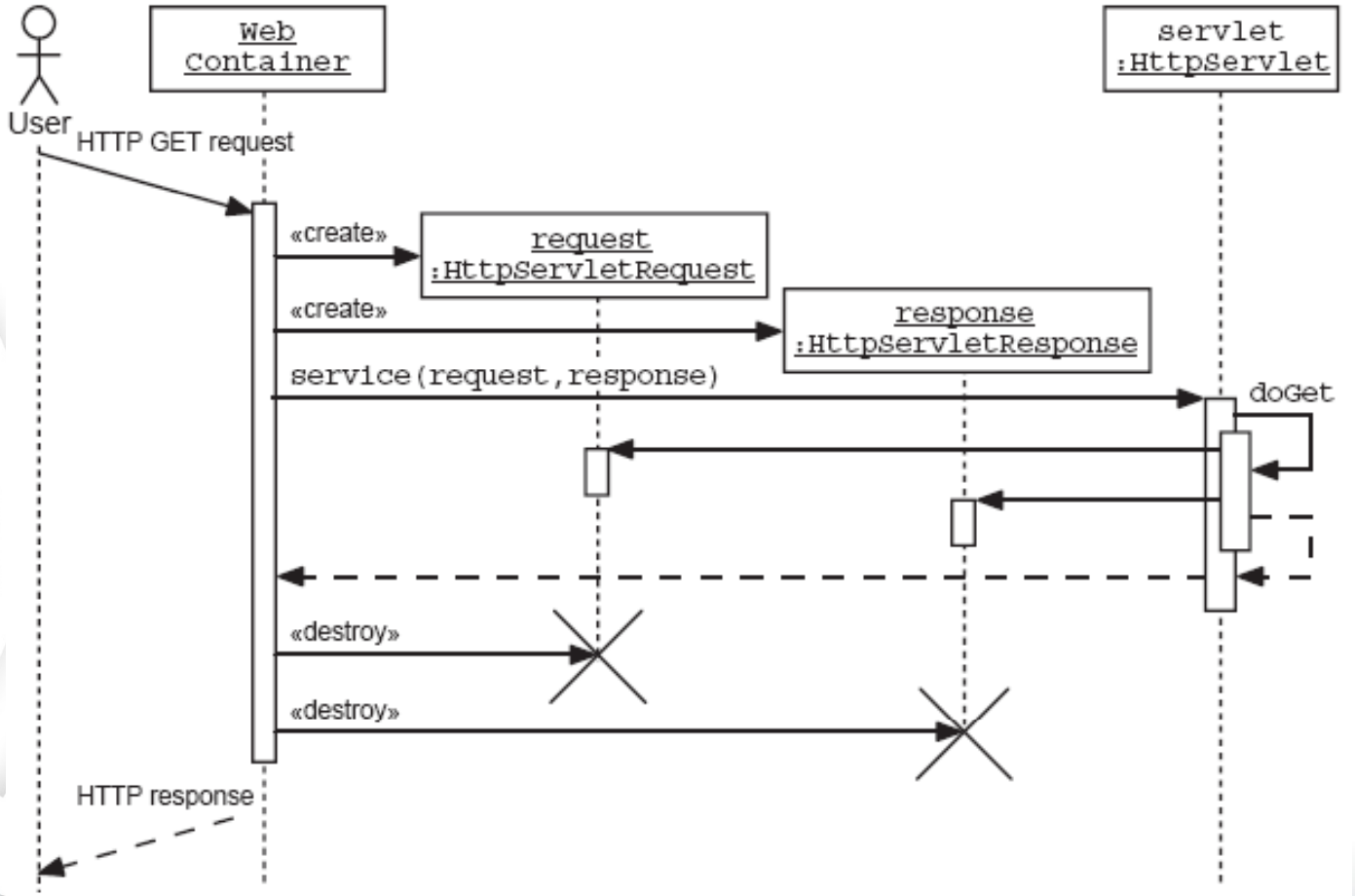
number	squared
0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81

Ventajas y desventajas de las JSP

Salenda

- Ventajas:
  - Todas las ventajas que tienen los servlets: Altas prestaciones, escalabilidad, independencia de plataforma, etc.
- Desventajas:
  - Si sólo se emplean páginas JSP, el código de script que realiza la lógica de negocio puede llegar a ser demasiado grande y confuso, haciendo difícil su depuración

Diagrama de secuencia de una petición GET



Resumen

- Puedes usar un componente de la vista para presentar datos, mostrar un formulario, mensajes de información y demás.
- El protocolo HTTP proporciona un mecanismo para solicitar vistas tanto estáticas como dinámicas
- El contenedor web intercepta las peticiones HTTP y activa el servlet que sea necesario
- Se puede desarrollas una clase servlet que implemente el método doGet para procesar una petición
- Se puede acceder a la información de la petición del inputstream a través del objeto request provisto por el contenedor.
- Se puede generar una vista escribiendo directamente en el outputstream del objeto request provisto por el contenedor

Integración de Servlets y JSP
El Patrón MVC

El patrón MVC

¿Porqué combinar Servlets y JSP?

Salenda

- Escenario típico: usar JSP para facilitar el desarrollo y mantenimiento del contenido HTML
  - Para código dinámico simple, invocar código servlet desde elementos de script.
  - Para aplicaciones un poco más complejas, usar clases propias invocadas desde elementos de script.

¿Porqué combinar Servlets y JSP?

Salenda

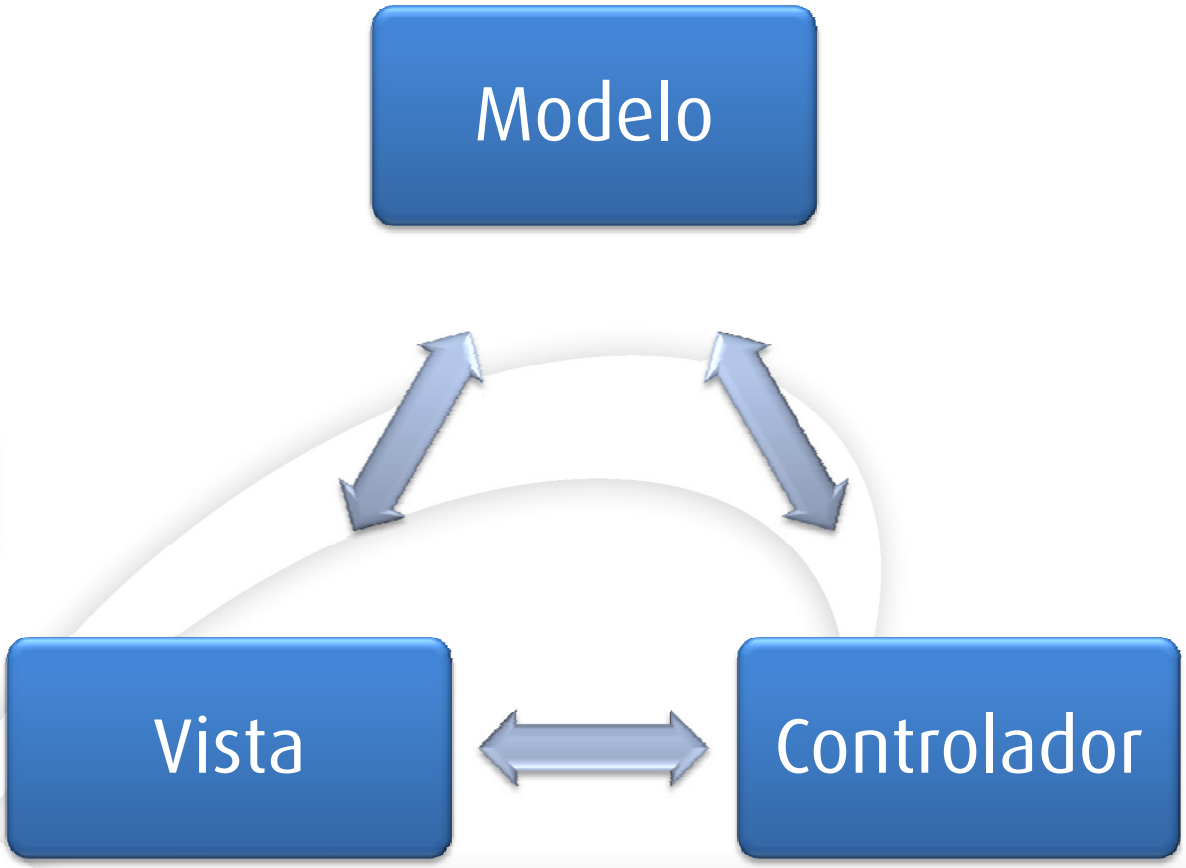
- Pero eso no es suficiente
  - Para procesamiento complejo, empezar con JSP es una mala aproximación
  - A pesar de la facilidad de separar el código real en clases separadas, beans y custom tags, asumir que una JSP es una *sola* página

Posibilidades para manejar una sola petición

Salenda

- Arquitectura MVC. Necesaria cuando:
  - Una sola petición puede derivar en múltiples resultados diferentes.
  - Existe un gran equipo de desarrollo con funciones bien separadas.
  - Se realiza un procesamiento complicado de los datos.

Patrón MVC



¿Qué es Apache Struts?

- ¿Un framework MVC?
  - Struts provee un framework unificado para el despliegue de aplicaciones web que usan la arquitectura MVC.
- ¿Una colección de utilidades?
  - Stuts provee una serie de clases de utilidades para manejar las tareas comunes del desarrollo de aplicaciones web

¿Qué es Apache Struts?

- ¿Un conjunto de librerías de etiquetas personalizadas ?
  - Struts provee librerías de etiquetas personalizadas para mostrar propiedades de beans, generar formularios HTML ...

Ventajas de Struts

- Configuración basada en archivos centralizados:
  - Archivos XML.
- Form beans.
- Bean tags:
  - Tags de acceso a las propiedades de los beans.
- HTML tags:
  - Asocia formularios HTML con beans.

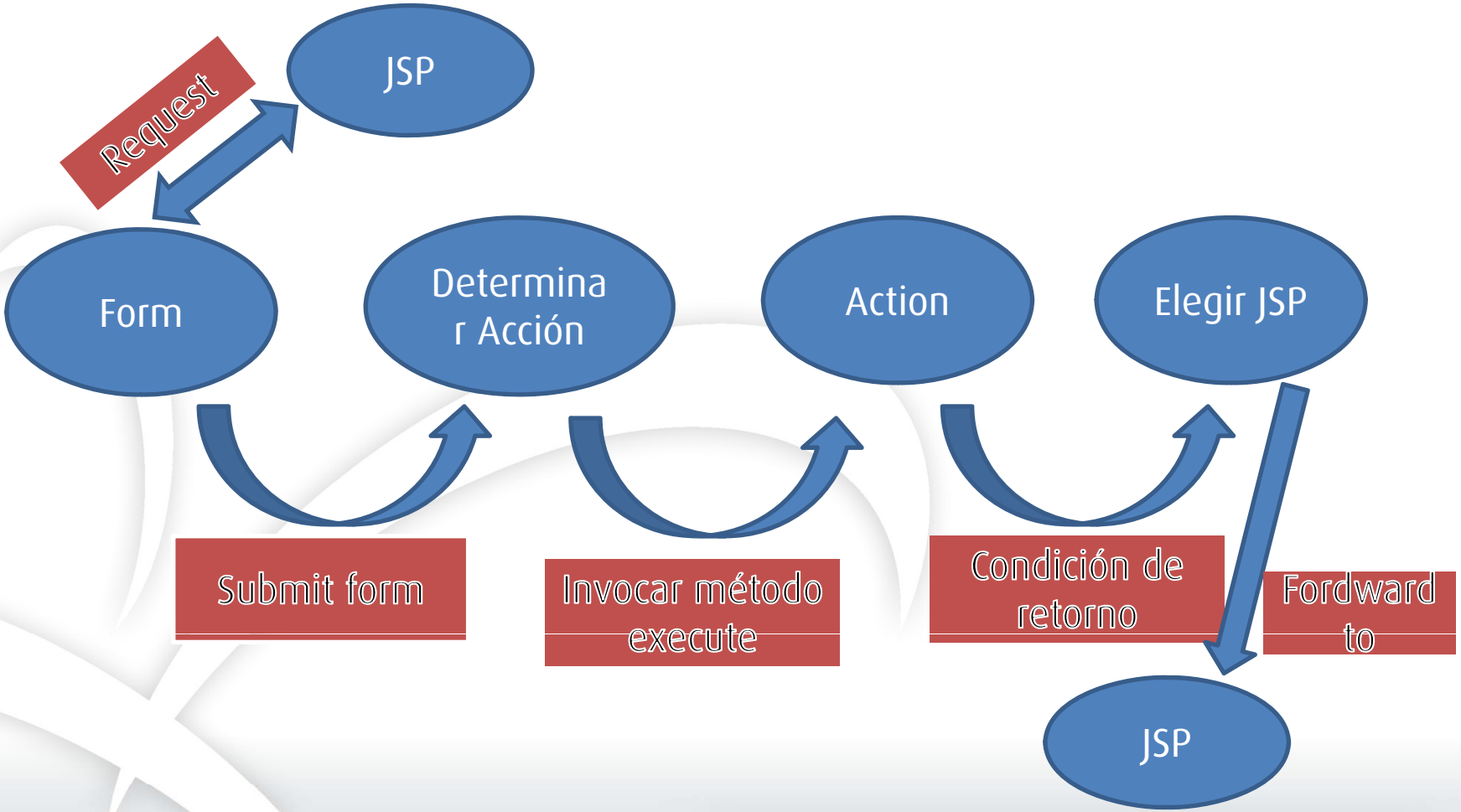
Ventajas de Struts

- Validación de formularios.
- Aproximación consistente con el patrón MVC.

Desventajas de Struts

- Mayor curva de aprendizaje
- Peor documentación:
  - Comparada con la documentación standard de servlets y JSP's.
- Menor transparencia:
  - El framework añade mayor complejidad
- Aproximación rígida:
  - Con Struts es muy difícil usar algo distinto a MVC.

Control de flujo de Struts



Ejemplo: Action.java

```
package es.salenda.action;

import com.opensymphony.xwork2.ActionSupport;

public class Action extends ActionSupport {

    private String mensaje;

    public String execute() throws Exception {
        //if(condicionDeSalida) return "failure";
        setMensaje(mensaje);
        return SUCCESS;
    }


    public void setMensaje(String mensaje) {
        this.mensaje = mensaje;
    }

    public String getMensaje() {
        return mensaje;
    }
}
```

Ejemplo: struts.xml


```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts><!-- Configuration for the default package. -->
<package name="es.salenda.action" extends="struts-default">
        <action name="accion" class="es.salendaaction.Action">
        <result>/index.jsp</result>
        </action>
</package>
</struts>
```

Ejemplo web.xml

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>My Application</display-name>
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <welcome-file-list>
  <welcome-file>inicio.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```

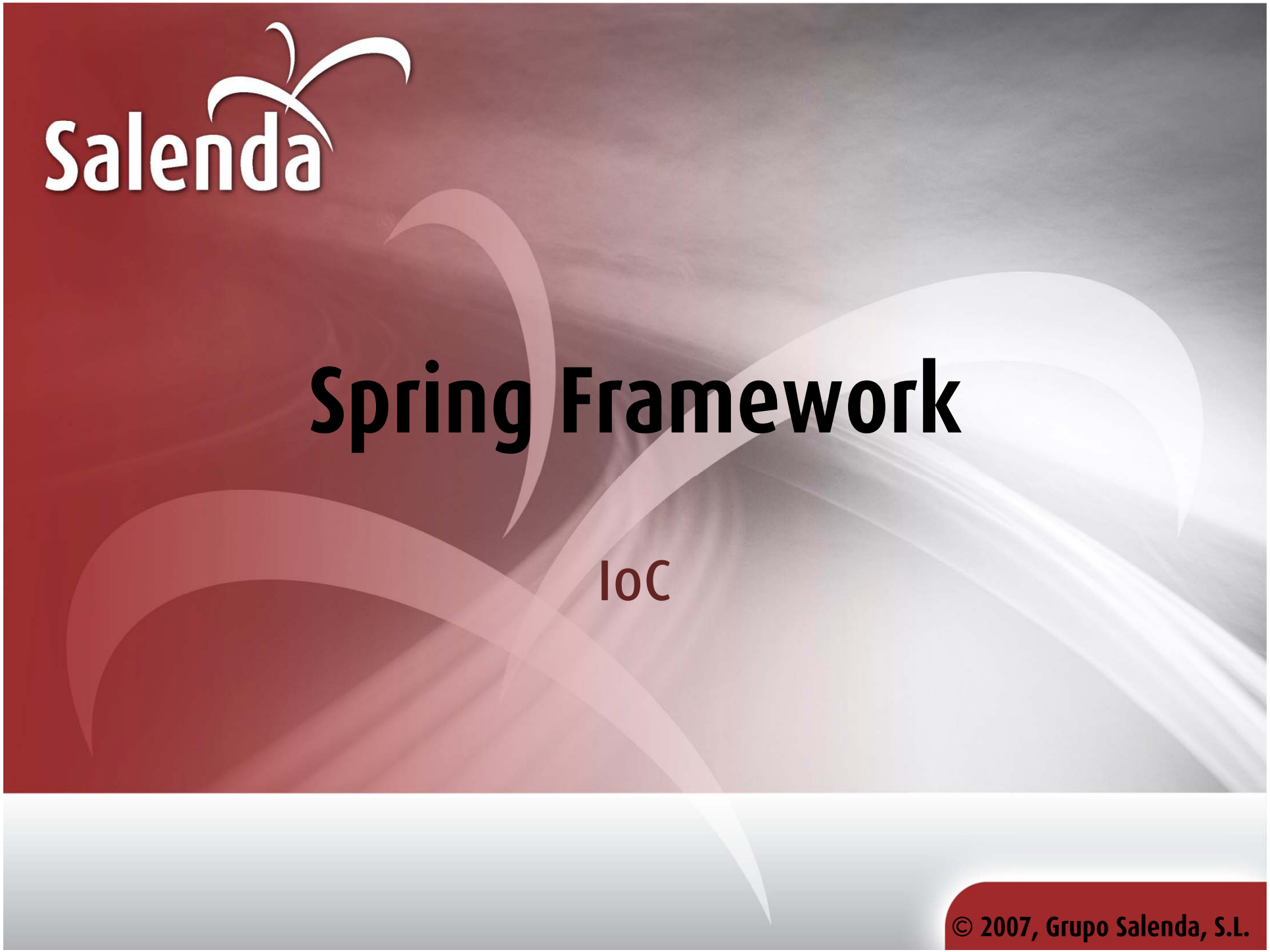
Ejemplo: inicio.jsp

- <%@ taglib prefix="s" uri="/struts-tags" %>
- <html>
- <head>
- <title>Inicio</title>
- </head>
- <body>
- <s:form action="accion">
- <s:textfield label="Echo server" name="mensaje"/>
- <s:submit/>
- </s:form>
- </body>
- </html>

Ejemplo: index.jsp

- <%@ taglib prefix="s" uri="/struts-tags" %>

- <html>
- <head>
- <title>Echo Server</title>
- </head>
- <body>
- Echo server reply:
- <h2><s:property value=“mensaje" /></h2>
- </body>
- </html>


Salenda
Spring Framework
IoC
© 2007, Grupo Salenda, S.L.

¿Por qué un contenedor ligero?

Salenda

- J2EE funciona bien pero ...
  - Es una arquitectura pesada con muchas restricciones
  - A menudo es “demasiado” para desarrollar aplicaciones simples
  - Es complicado el “test driven development”

Inversión de Control

Salenda

- Inversión de control a.k.a. Injección de dependencias
- El patrón en el núcleo de Spring
  - Hace que el código sea más fácil de probar.
  - Organiza los objetos de la capa intermedia (con o sin EJB's).
  - Con Spring te centras sólo en las propiedades de los JavaBeans.
  - Diseñado para que existan las menores dependencias posibles (casi todos los objetos de negocio NO dependen del framework).

Inversión de Control

- Provee de un framework consistente para el acceso a datos (JDBC o O/R mapping).
- Posibilita la construcción de la aplicación usando POJO's.

Diseño de sistemas sostenibles

Salenda

- El punto clave es la gestión de *dependencias.*
- Diseño por "contrato":
  - Define el comportamiento, no la implementación.
  - Se escribe una interfaz/clase para solucionar cada problema.
  - Fácil construir test's para comprobar la funcionalidad de cada implementación dada.

Diseño de sistemas sostenibles

Salenda

- Dependencia: “atar” un componente a otro mediante:
  - Herencia.
  - Composición.
  - Instanciación.
  - Signatura de métodos.
  - Uso de métodos estáticos o atributos

Diseño de sistemas sostenibles

Salenda

- La dependencia implica un cambio en el componente dependiente cuando el componente del que depende sufre algún cambio.
- No es necesariamente mala (es inevitable); el objetivo es:
  - Minimizar el número de dependencias en el diseño.
  - Depender únicamente de interfaces.

Cómo funciona

- Defines:
  - Interfaces e implementaciones.
  - Dependencias entre las clases/interfaces.
- El contenedor de IoC:
  - Construye al dependiente y al proveedor; *inyecta* al proveedor dentro del dependiente.
  - Te da la posibilidad de elegir el tipo de injección (por configuración, código o automáticamente –autowiring-)

Cómo funciona

Salenda

- Uso de POJOS para las implementaciones.
- No hay necesidad de desplegar en un contenedor pesado.
- Mejora la “testabilidad”.
- No es intrusivo:
  - No dependes de ninguna API específica del contenedor.
  - No hay interfaces que implementar, ni clases de las que heredar, salvo las propias.

"El principio de Hollywood"

Salenda

- No me llames, que ya te llamo yo.

- Sin IoC:
  - El componente tiene el control sobre sus dependencias, con lo que tiene que *conseguirlas.*
- Con IoC:
  - Los componentes de negocio no tienen control sobre sus dependencias.
  - El contenedor será el encargado de inyectárselas.

Ejemplo

```
class MyMain {
        public static void main(String[] args) {
        //initialize the IoC container (here it's Spring):
                XmlBeanFactory xmlBeanFactory =
                        new XmlBeanFactory(new ClassPathResource("beans.xml"));
                //retrieve MyLogic:
                MyLogic myLogic = (MyLogic) xmlBeanFactory.getBean("myLogic");
                //call the method:
                myLogic.doYourThing();
        }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans ...>
<!-- this is beans.xml -->
<beans>
        <bean id="theBusinessSvc" class="foo.BusinessServiceMock" />
        <bean id="myLogic" class="foo.MyLogic">
                <property name="businessService">
                        <bean ref="theBusinessSvc" />
                </property>
        </bean>
</beans>
```

Integración con Struts

```
Package packt;

import com.opensymphony.xwork2.ActionSupport;

public class Action extends ActionSupport {

    private servicios.MiBean mB;

    public Action(servicios.MiBean bean) {
                mB = bean;
    }

    public String execute() throws Exception {
          //Lógica ....
       return SUCCESS;
    }
```

Integración con Struts

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans ...>
<!-- this is beans.xml -->
<beans>
        <bean id=”claseAction” scope=“prototype”
                class=”packt.ActionClass”>
                <constructor-arg ref=“miBean”/>
        </bean>
        <bean id=”miBean” class=”servicios.MiBean”>
                <property name=”codigo” value=“765123”/>
        </bean>
</beans>
```


Salenda
EJB 3.0
© 2007, Grupo Salenda, S.L.

¿Qué son los EJB's?

- Es una tecnología para desarrollar aplicaciones multicapa.
- Un estándar implementador por muchos proveedores de contenedores.
- Componentes distribuidos de negocio y acceso a datos.
- Existen varios tipos

Tipos de EJB

Salenda

- Entity Beans:
  - Usados para mapear tablas de la BBDD a clases (O/R mapping).
  - En lugar de trabajar con “resultados de querys”, trabajas con objetos Java.
  - El servidor de aplicaciones provee de la funcionalidad para cardar, actualizar o borrar los valores de una clase de la BBDD.

Tipos de EJB

Salenda

- Beans de sesión:
  - Usados para implementar funcionalidad de la aplicación.
  - Hay dos tipos:
    - Con estado.
    - Sin estado.

Tipos de EJB

- Beans de sesión con estado:
  - El contenedor se encargar de mantener el estado del objeto a lo largo de la sesión del usuario. (p.e. un carro de la compra).
- Beans de sesión sin estado:
  - Es un componente con un ciclo de vida corto, no alcanza más allá de cada conexión del cliente. (p.e. implementar lógica para mandar un mail).

Tipos de EJB

- Message Driven Beans:
  - Representan un servicio sin estado de invocación asíncrona.

EJB's

- Proporcionan la arquitectura para el desarrollo de aplicaciones distribuidas basadas en componentes.
- Proporcionan portabilidad entre diferentes plataformas y protocolos de comunicación.
- El contenedor se encarga de detalles tales como seguridad, transaccionabilidad, gestión de su ciclo de vida ...

Beneficios de los EJB's

Salenda

- Simplifican el desarrollo de aplicaciones distribuidas de gran envergadura.
  - El contenedor le provee de servicios de bajo nivel.
  - La lógica de negocio se encuentra en los beans, y no en el cliente => clientes más ligeros.
  - Son portables y reusables.

¿Cuándo usar EJB's?

Salenda

- Cuando ...
  - La aplicación debe de ser escalable
    - Pueden ser distribuidos en varias máquinas y su localización seguirá siendo transparente para los clientes.
  - Se debe primar la integridad de los datos
    - Los EJB's soportan transacciones con este fin. Pueden acceder de forma concurrente a objetos compartidos.
  - La aplicación tendrá varios clientes
    - Clientes remotos Java, en otros lenguajes, navegadores web ...

EJB's de entidad

Salenda

- Nos centraremos en los EJB's de entidad.
  - Representan un objeto de negocio en un sistema de almacenamiento *persistente*.
- Persistencia:
  - El EJB's existe *más allá* de la vida de la aplicación.
  - Puede ser gestionada por el bean o por el contenedor.

EJB's de entidad

Salenda

- Acceso compartido:
  - Como los Beans de entidad pueden ser accedidos por múltiples clientes al mismo tiempo es imprescindible que realicen su trabajo dentro de transacciones.
- Clave primaria:
  - Cada bean de entidad *debe* de tener un único (y unívoco) identificador.

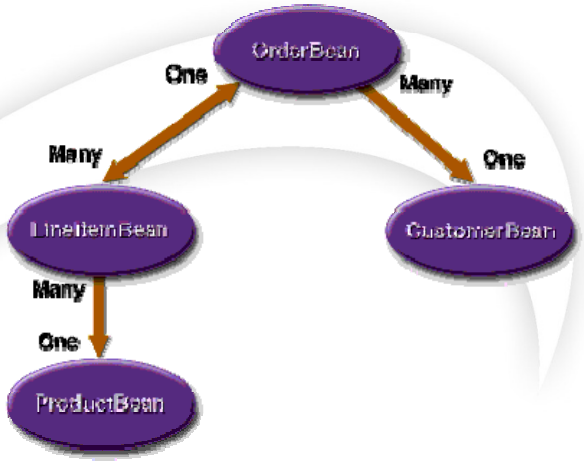
EJB's de entidad

Salenda

- Relaciones:
  - Al igual que una tabla en el modelo relacional, un EJB de entidad puede tener relaciones con otros (1:1, 1:n, n:m).
- Persistencia manejada por el contenedor:
  - El contendor maneja *todo* el acceso a la BBDD.
  - El EJB no contiene SQL, luego no está atado a ningún mecanismo de almacenamiento en concreto.

EJB's de entidad

- Debido a esto, los EJB's son portables entre distintas plataformas/contenedores/bbdd's.
- Se debe de proveer del "Abstract Schema"



EJB's de entidad

Salenda

- Campos persistentes:
  - Todos los campos persistentes son almacenados en la BBDD subyacente.
  - El contenedor de EJB se encarga de que SIEMPRE estén sincronizados.
  - Normalmente cada Entity se aloja en una tabla, y cada campo en una columna.

EJB's de entidad

Salenda

- Campos relacionales
  - Un campo relacional es una clave ajena en la base de datos.

Ejemplo de EJB

```
@Entity
@Table(name = "PURCHASE_ORDER")
public class Order implements java.io.Serializable
{
private int id;
private double total;
private Collection<LineItem> lineItems;

@Id @GeneratedValue(strategy=GenerationType.AUTO)
public int getId()
{
return id;
}

public void setId(int id)
{
this.id = id;
}

public double getTotal()
{
return total;
}

public void setTotal(double total)
{
this.total = total;
}
```

Ejemplo de EJB

```
public void addPurchase(String product, int quantity, double price)
{
if (lineItems == null) lineItems = new ArrayList<LineItem>();
LineItem item = new LineItem();
item.setOrder(this);
item.setProduct(product);
item.setQuantity(quantity);
item.setSubtotal(quantity * price);
lineItems.add(item);
total += quantity * price;
}

@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy="order")
public Collection<LineItem> getLineItems()
{
return lineItems;
}

public void setLineItems(Collection<LineItem> lineItems)
{
this.lineItems = lineItems;
}
}
```

Ejemplo de EJB

```
@Entity
public class LineItem implements java.io.Serializable
{
  private int id;
  private double subtotal;
  private int quantity;
  private String product;
  private Order order;


  @Id @GeneratedValue(strategy=GenerationType.AUTO)
  public int getId()  {
    return id;
  }

  public void setId(int id)  {
    this.id = id;
  }

  public double getSubtotal()  {
    return subtotal;
  }
  public void setSubtotal(double subtotal)  {
    this.subtotal = subtotal;
  }
  public int getQuantity()
  {
    return quantity;
  }
```

Ejemplo de EJB

```
    public void setQuantity(int quantity)  {
      this.quantity = quantity;
    }

    public String getProduct()  {
      return product;
    }

    public void setProduct(String product)  {
      this.product = product;
    }

    @ManyToOne
    @JoinColumn(name = "order_id")
    public Order getOrder()  {
      return order;
    }

    public void setOrder(Order order)  {
      this.order = order;
    }
}
```

Recursos

- http://java.sun.com
- http://springframework.org
- http://struts.apache.org