Escuela Politécnica de Ingeniería
Grado de Ingeniería Informática en Tecnologías de la Información

Tecnologías Web

Tema 3

Tecnologías web de servidor: JSF 2.2 1/2

Índice

- *Introducción*
- *Beans gestionados*
- *Configuración de JSF*
- *Lenguaje de expresiones*
- *Manejo de eventos*
- *Tablas a partir de colecciones*
- Archivos de Propiedades e Internacionalización
- Integración con AJAX
- Validación
- Plantillas con facelets

Introducción

- JSF = Java Server Faces
- Definición: Especificación de un Framework MVC basado en JEE para el desarrollo Web
- JSF incluye:
  - APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entrada, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad.
  - Un conjunto por defecto de componentes para la interfaz de usuario.
  - Dos bibliotecas de etiquetas personalizadas para JavaServer Pages que permiten expresar una interfaz JavaServer Faces dentro de una página JSP/ XHTML.
  - Un modelo de eventos en el lado del servidor.
  - Administración de estados.
  - Beans gestionados (Managed Beans).

Introducción

- La especificación de JSF fue desarrollada por la Java Community Process como JSR (Java Specification Request) 127, que definía JSF 1.0 y 1.1, JSR 252 que define JSF 1.2, JSR 314 para JSF 2.0 y JSR 344 para JSF 2.2
- Implementaciones de JSF:
  - **Oracle Mojarra**
    - Dirección: http://javaserverfaces.java.net/
    - Requiere servlets 2.5 o superior, también integrado en Glassfish 3
  - **Apache MyFaces**
    - **Dirección: http://myfaces.apache.org/core20/**
    - **Requiere servlets 2.5 o superior, también integrado** en Apache Geronimo 3
  - **Cualquier servidor Java EE 6**
    - **JSF 2.2 es parte de la especificación Java EE 6**
    - **Jboss 7/WildFly, Glassfish**, WebLogic xx, WebSphere x, etc.
- Extensiones de JSF
  - **RichFaces** Agrega componentes visuales y soporte para AJAX.
  - **ICEfaces** Contiene diversos componentes para interfaces de usuarios más enriquecidas, tales como editores de texto enriquecidos, reproductores de multimedia, entre otros.
  - **jQuery4jsf** Contiene diversos componentes sobre la base de uno de los más populares framework javascript jQuery.
  - **PrimeFaces** Es una librería muy liviana, todas las decisiones hechas son basadas en mantener a PrimeFaces lo más liviano posible. PrimeFaces es una librería muy simple que no necesita dependencias y configuraciones.
  - **OpenFaces** Librería open source que contiene diferentes componentes JSF, un Framework Ajax y un Framework de validación por parte del cliente.

Introducción

- Alternativas:
  - Struts2
  - Spring MVC
  - Apache Wicket
  - Apache Tapestry
  - jMaki

Configuración de JSF

- **Librerías**
  - Mandatorio el jar JSF 2.2; recomendado JSTL 1.2
  - Se pueden omitir para Glassfish 3, JBoss 7, y otros servidores Java EE 6
- **faces-config.xml (configuración de JSF)**
  - En el se definen: Beans gestionados, reglas de navegación, …
  - Permite el empleo de anotaciones Java para los Beans Gestionados y mapeos por defecto de acciones de controladores a páginas resultado.
- **web.xml (descriptor de despliegue)**
  - se definen el controlador de Faces (FacesServlet)
  - Y las reglas de mapeo a ese controlador. Patrones *.jsf/faces/* o *.faces (o el que se desee)
- Normalmente se establece **PROJECT_STAGE** a Development para obtener más detalle en caso de errores
- **El acceso a páginas pagina.xhtml**
  Usar URL pagina.jsf (que coincida con la regla de mapeo de web.xml)

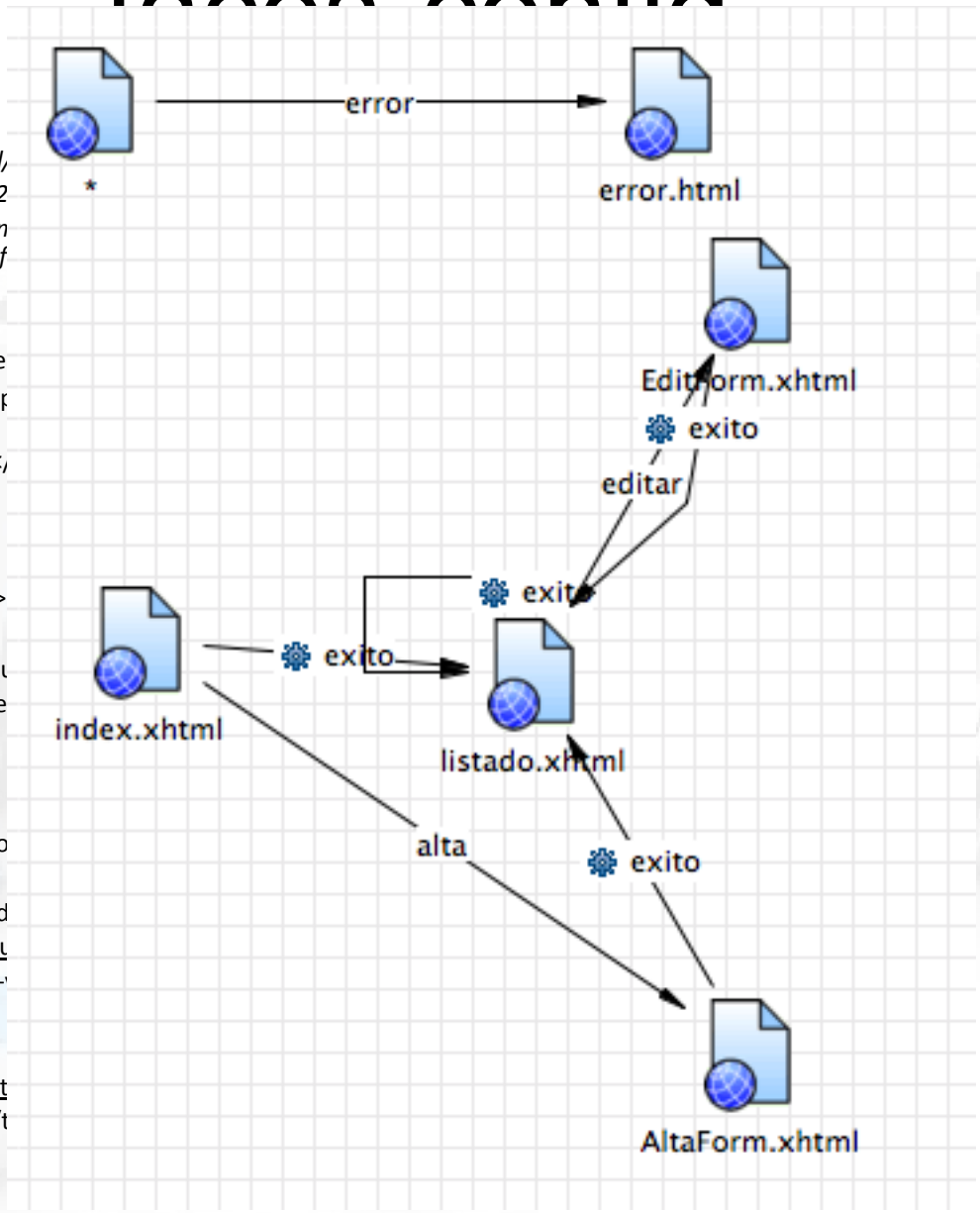
web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
  <display-name>tew-gestioneitorv6_0</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
  </servlet-mapping>
<param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

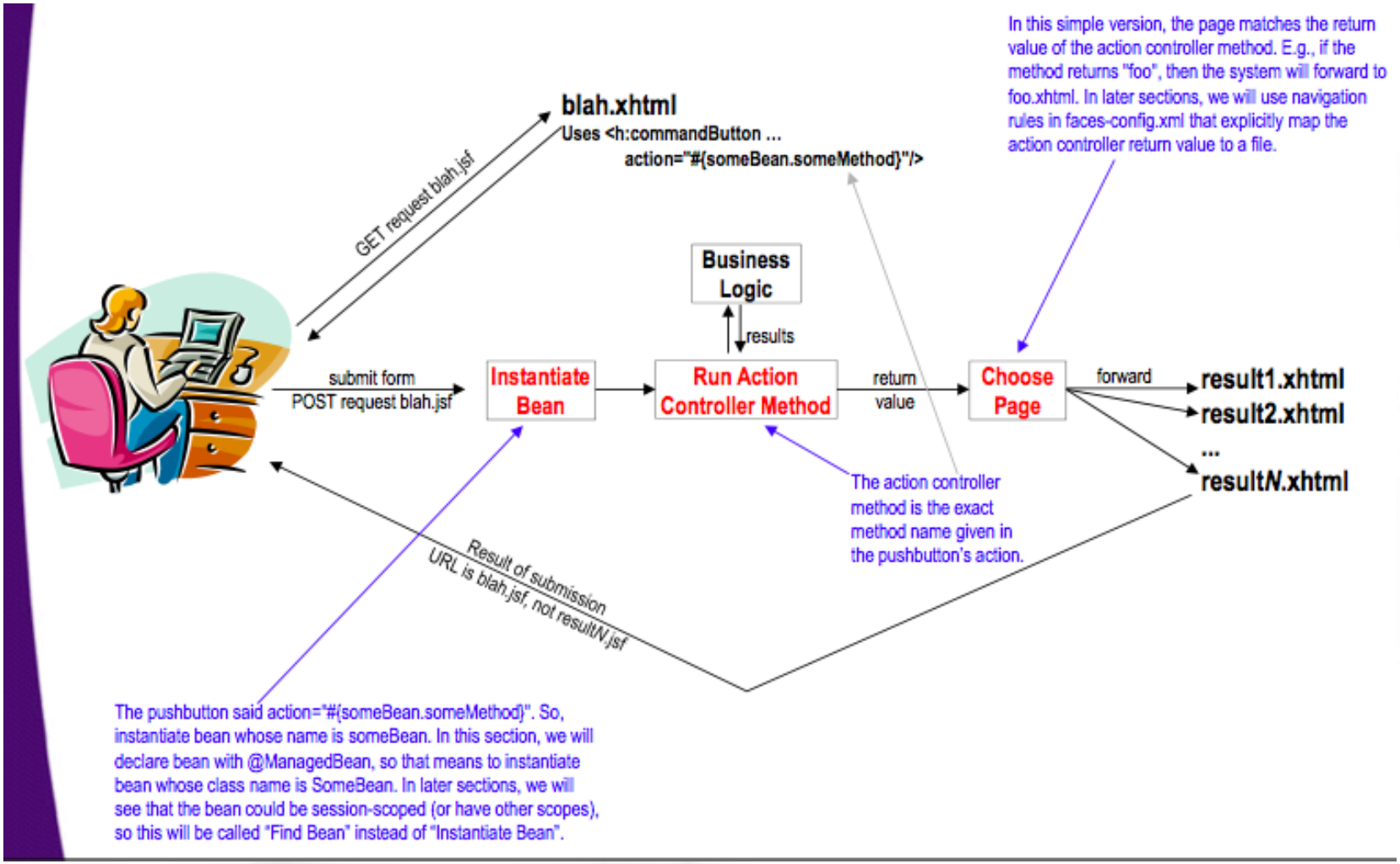
faces-config

```
<?xml version="1.0"?>
<faces-config
   xmlns="http://xmlns.jcp.org/xml/
   xmlns:xsi="http://www.w3.org/2
   xsi:schemaLocation="http://xmln
xmlns.jcp.org/xml/ns/javaee/web-f
   version="2.2">
 <managed-bean>
   <managed-bean-name>controlle
   <managed-bean-class>com.tew.p
managed-bean-class>
   <managed-bean-scope>session</
 </managed-bean>

 <navigation-rule>
   <from-view-id>*</from-view-id>
    <navigation-case>
     <from-outcome>error</from-ou
     <to-view-id>/error.html</to-vie
   </navigation-case>
 </navigation-rule>
 <navigation-rule>
   <from-view-id>/index.xhtml</fro
   <navigation-case>
     <from-action>#{controller.listad
     <from-outcome>exito</from-ou
     <to-view-id>/listado.xhtml</to-
   </navigation-case>
   <navigation-case>
     <from-outcome>alta</from-out
     <to-view-id>/AltaForm.xhtml</t
   </navigation-case>
 </navigation-rule>
 <navigation-rule>
```



JSF flujo de control



Formato de las páginas faces

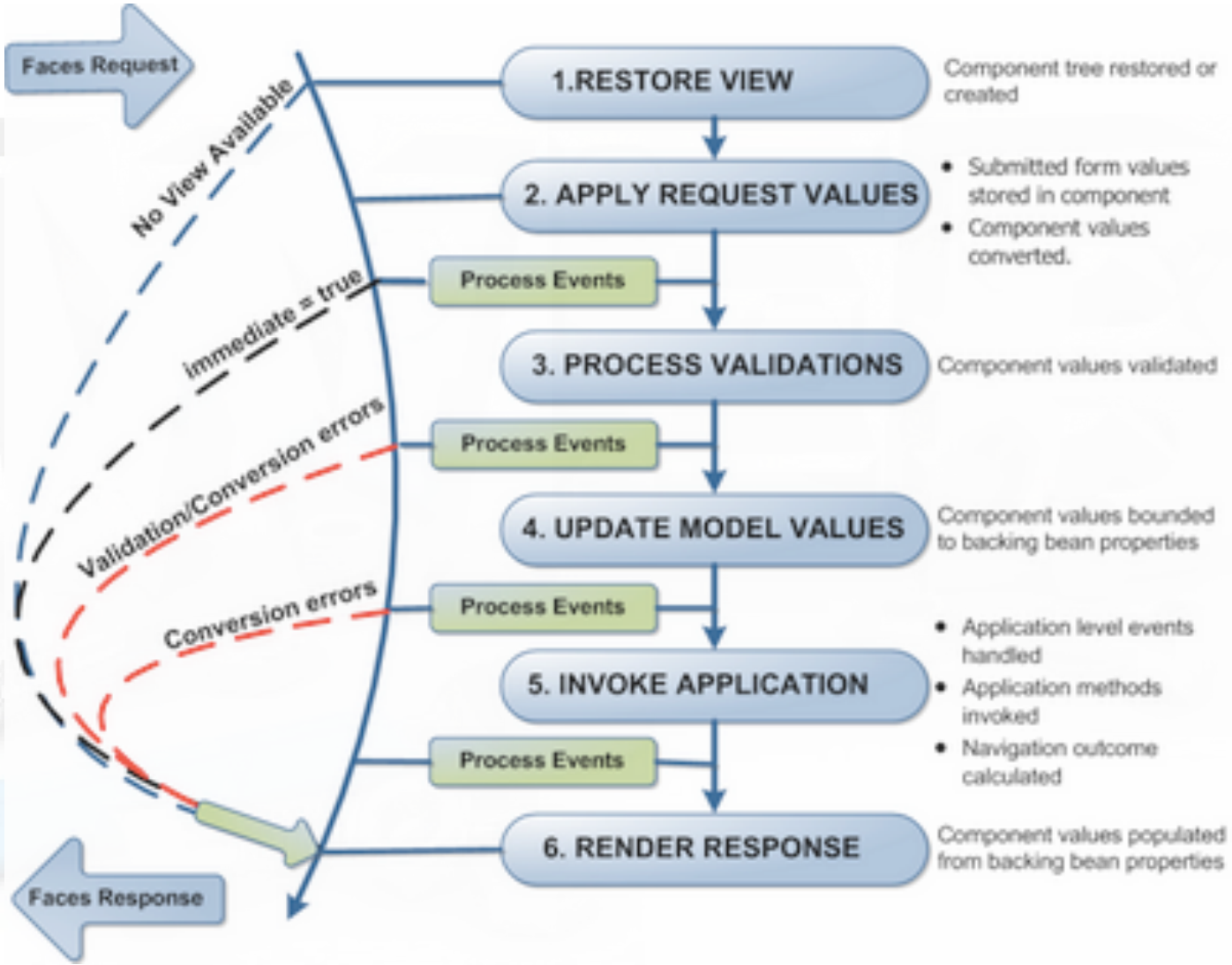
- En JSF 1.x se usaban JSPs con etiquetas JSF
- En JSF 2.x se usa el formato xhtml con etiquetas JSF.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h=http://java.sun.com/jsf/html
    xmlns:f="http://java.sun.com/jsf/core">
<h:head>
<title>Piloto de TEW</title>
</h:head>
<h:body>
<center>
<h1>Gestioneitor!</h1>
<br/>
<h2>Aplicación de gestión de alumnos</h2>
</center>
<br/>
<br/>
<br/>
<h:form>
 <h:commandLink value="Listado de alumnos" action="#{controller.listado}"/>
 <br></br>
 <h:commandLink value="Alta de alumno" action="alta"/>
</h:form>
</h:body>
</html>
```

Librerías de etiquetas

Tag Library	URI	Prefix	Example	Contents
JavaServer Faces Facelets Tag Library	http://java.sun.com/jsf/facelets	ui:	ui:component ui:insert	Tags for templating
JavaServer Faces HTML Tag Library	http://java.sun.com/jsf/html	h:	h:head h:body h:outputText h:inputText	JavaServer Faces component tags for all UIComponent objects
JavaServer Faces Core Tag Library	http://java.sun.com/jsf/core	f:	f:actionListener f:attribute	Tags for JavaServer Faces custom actions that are independent of any particular render kit
JSTL Core Tag Library	http://java.sun.com/jsp/jstl/core	c:	c:forEach c:catch	JSTL 1.2 Core Tags
JSTL Functions Tag Library	http://java.sun.com/jsp/jstl/functions	fn:	fn:toUpperCase fn:toLowerCase	JSTL 1.2 Functions Tags

JSF 2.x ciclo de vida



JSF 2.x ciclo de vida II

- Comienzo/fin JSF: Solicitud/respuesta.
- HTTP stateless protocol: Solución basada en vistas mantenidas en el servidor
- Una vista es un árbol de componentes que representa la UI del usuario. Así, mientras que nosotros nos centramos en desarrollar los componentes, el ciclo de vida de JSF se preocupa de sincronizar estas vistas del lado del servidor y lo que se le muestra al usuario.
  Cuando un usuario pincha en un botón o un link comienza el ciclo de vida de JSF.

  1) Restore View

  Se crea o restaura el árbol de componentes (la vista) en memoria. Cuando la vista se crea por primera vez, se almacena en un contenedor padre conocido como FacesContext, y se pasa directamente a la última fase (Render Response), ya que la petición no tendrá valores que estudiar. Todas las operaciones realizadas en el FacesContext utilizan un hilo por petición de usuario, así que no hay que preocuparse porque múltiples peticiones de usuarios puedan colisionar.

  2) Apply Request Values

  Se itera sobre los componentes del árbol, comprobando qué valor de la petición pertenece a qué componente, y los van guardando. Estos valores almacenados se llaman 'valores locales'.

  3) Process Validations

  Se realizan las validaciones y conversiones necesarias de los valores locales. Si ocurre algún error en esta fase, se pasa a la fase Render Response, mostrándole al usuario otra vez la página actual y dándole así una nueva oportunidad para que pueda introducir los datos correctos.

JSF 2.x ciclo de vida III

4) Update Model Values

Se modifican los valores de los beans asociados a los componentes de la vista con los valores locales. Ej. #{userBean.name}

5) Invoke Application

Se invoca el método asociado al action del botón o link que pinchó el usuario (Ej. #{userBean.addUser}), y que hizo que se activara el ciclo de vida de la petición. Estos métodos devuelven un String que le indica al gestor de navegación qué página tiene que devolverle al usuario.

6) Render Response

El servidor devuelve la página de respuesta al navegador del usuario y guarda el estado actual de la vista para poder restaurarla en una petición posterior.

Si añadimos a un botón o a un link el atributo immediate a true, se saltará la fase de validación. Por ejemplo, si tenemos un botón "Volver", normalmente no querremos que valide los valores que hay introducidos sino que nos devuelva diréctamente a la página anterior.

Beans gestionados I

```
@ManagedBean
@SessionScoped
public class BeanAlumnos implements Serializable {
 private static final long serialVersionUID = 55555L;
// Se añade este atributo de entidad para recibir el alumno concreto selecionado de la tabla o de un formulario
// Es necesario inicializarlo para que al entrar desde el formulario de AltaForm.xml se puede
 // dejar los avalores en un objeto existente.
private Alumno alumno = new Alumno();

        private Alumno[] alumnos = null;

        public BeanAlumnos()   {
           iniciaAlumno(null);
        }

       public Alumno[] getAlumnos () {
         return(alumnos);
       }
}
```

Beans gestionados II

- **Clases POJO con constructores sin parámetros**
  - Por defecto o explícito
  - Sin propiedades públicas
    Las propiedades Blah persistentes se acceden mediante getBlah and setBlah
  - Para acceder a la propiedad title se puede hacer así:
    - En JSF #{book.title} llama al método getTitle del objeto book.
  - Las propiedades puede usar isBlah en vez de getBlah
  - Importa el método no la propiedad de instancia
- **Regla para acceder los métodos getter y setter desde el lenguaje de expresiones**
  - Eliminar el sufijo get/set y cambiar la siguiente letra a minúscula. La variable de instancia es irrelevante
    - Método: getFirstName
    - Propiedad firstName
    - Ejemplo: #{customer.firstName}
  - Excepción 1: Propiedades booleanas
    - Si el getter retorna boolean o Boolean
    - Método: getPrime o isPrime
    - Propiedad: prime
    - Ejemplo: #{myNumber.prime}
  - Excepción 2: Mayúsculas consecutivas
    - Si van dos mayúsculas después de get/set
    - Método: : getURL
    - Propiedadme: URL (no uRL) •
    - Ejemplo: #{webSite.URL}

Beans gestionados: ejemplos

Method Names	Property Name	Example JSF Usage
getFirstName setFirstName	firstName	#{customer.firstName} <h:inputText value="#{customer.firstName}"/>
isExecutive setExecutive (boolean property)	executive	#{customer.executive} <h:selectBooleanCheckbox value="#{customer.executive}"/>
getExecutive setExecutive (boolean property)	executive	#{customer.executive} <h:selectBooleanCheckbox value="#{customer.executive}"/>
getZIP setZIP	ZIP	#{address.ZIP} <h:inputText value="#{address.ZIP}"/>

Beans gestionados

- **JSF gestiona automáticamente los "Beans gestionados"**
  - Los instancia (por eso necesita zero-arguments constructor)
  - Controla su ciclo de vida: - Scope (request, session, application)
  - Invoca a los métodos setter: p.e., para <h:inputText value="#{customer.firstName"/>, el valor se pasa a setFirstName
  - Invoca a los método getter: p.e. , el uso de #{customer.firstName} invoca a getFirstName
- **Declaración de M. Beans**
  - @ManagedBean antes de la clase (por defecto scope request)
  - Más potente: declararlo en una sección: <managed-bean> en faces-config.xml

Beans gestionados: mejora del rendimiento de los getter

- **Problema**

  Los método getter de beans gestionados se invocan varias veces.

  - Como mínimo una vez cuando se muestra el formulario (<h:inputText value="#{user.customerId}"/>) y otra al mostrar el resultado (#{user.customerId}).
  - Aunque frecuentemente más veces.
    De modo que cuando el método get accede a una BD o un realiza un cálculo pesado, el rendimiento cae espectacularmente en nuestro servicio

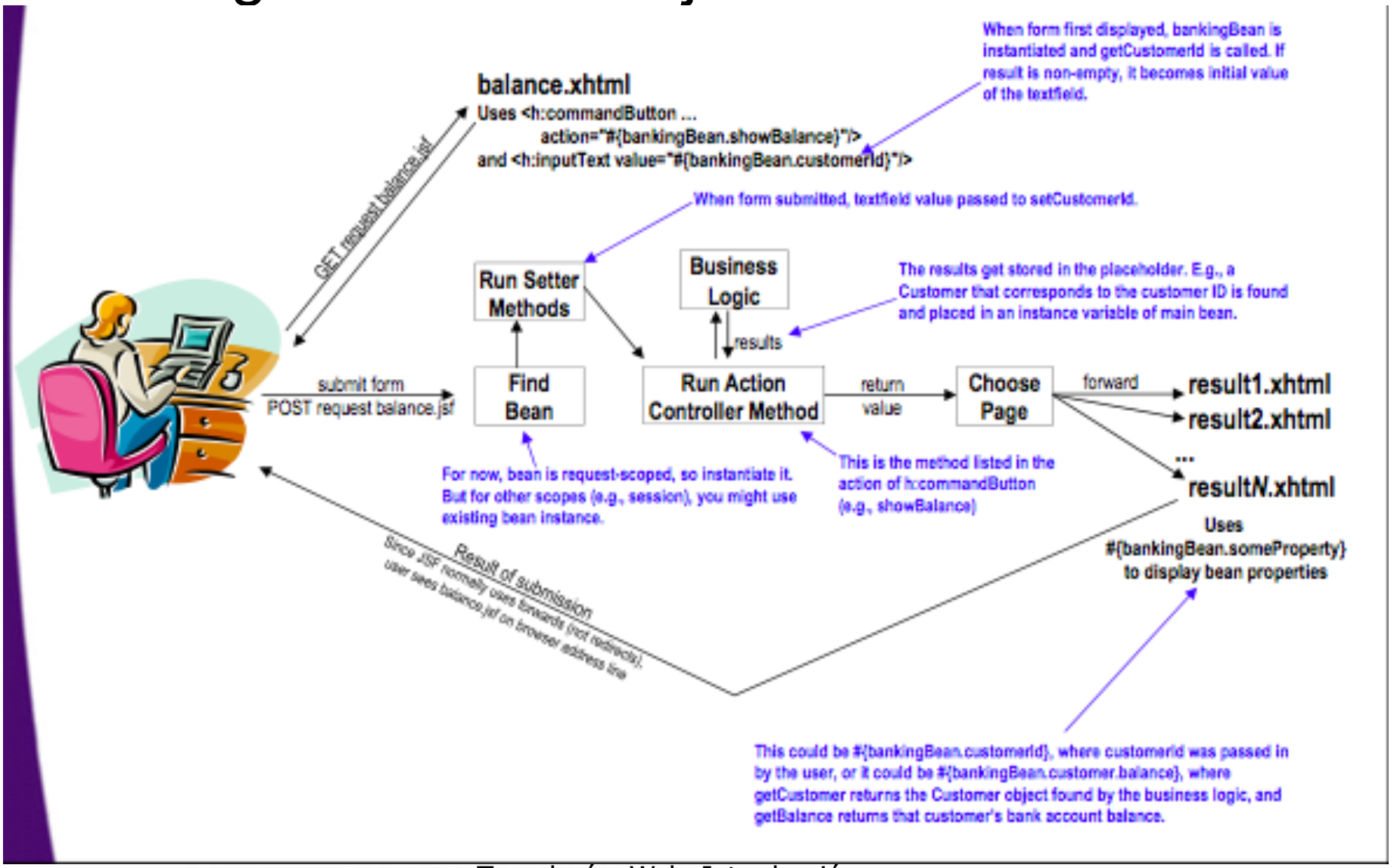
- **Solución**

  - Almacenar los datos en variables de instancia con getter que retorna dichos datos.

Beans gestionados: estructura

- **Los beans gestionados tienen tres secciones:**
  - Propiedades para representar las entradas (p.e., pares de getters y setters)
  - Métodos Action controller
    - Puede ir más de uno por M. Bean
  - Propiedades para los resultados
    - Normalmente derivadas de las entradas

Beans gestionados: flujo de control actualizado



El atributo name

- **Atributo de un M. Bean @ManagedBean**

  @ManagedBean(name="anyName")
  public class BeanName { ... }
- Para referirse al bean se puede usar #{anyName.blah}, siendo anyName el valor del atributo name del Bean.
- Se mantiene el scope request y sin necesidad de entradas en faces-config.xml

Navigator2.java

package coreservlets; import javax.faces.bean.*; @ManagedBean(name="customName") public class Navigator2 extends Navigator {}

Start-page.xhtml

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html">

<h:head>...</h:head>
<h:body>

...
<fieldset>
<legend>Random Results Page</legend> <h:form>

Press button to get one of three possible results pages.
<h:commandButton action="navigator2.gotoPage" value="Go to Random Page"

</h:form> </fieldset> ... </h:body></html>

Scope Beans

- **Idea**
  - Pensar en el tiempo de vida de los beans y qué usuarios y solicitudes recibirá
- **JSF 1.x scopes**
  - Request, session, application – en faces-config.xml – Request es el valor por defecto
- **JSF 2.0 scopes**
  - request, session, application, view, none, custom
  - Especificados en faces-config.xml or haciendo uso de las nuevas anotaciones (p.e., @SessionScoped)
  - Request continua siendo el valor por defecto
- **Anotaciones en JSF 2.0**
  - @RequestScoped
  - @SessionScoped
  - @ApplicationScoped
  - @ViewScoped
  - @CustomScoped
  - @NoneScoped
- Habitualmente se incluye después de @ManagedBean

```
@ManagedBean
@SessionScoped
public class SomePojo { ... }
```

Scope Beans 2

- **@RequestScoped**
  - Valor por defecto. Crea una nueva instancia de bean por cada solicitud HTTP request. Ya que los beans son también empleados para representar los valores iniciales en un formulario de entrada, esto implica que generalmente estos bean se instancian dos veces (one cuando se visualiza el formulario y otra cuando se envia al servidor)
- **@SessionScoped**
  - Si el mismo usuario con la misma cookie (JSESSIONID) retorna antes del timout de sesion se emplea la misma instancia de bean. Este debería ser Serializable.
- **@ApplicationScoped**
  - Compartido por todos los usuarios. Este Bean debería tener un estado no-mutable o bien debería tener un acceso sincronizado cuidadoso.
- **@ViewScoped**
  - Se mantiene la misma instancia del bean mientras el usuario está en la misma página (p.e. con manejadores de eventos o consultas Ajax)
  - Nuevo scope en JSF 2.0
  - Bean debe implementar Serializable
- **• @CustomScoped(value="#{someMap}")**
  - Bean se almacena en un Map, de forma que puede controlarse el ciclo de vida
  - Nuevo scope en JSF 2.0
- **@NoneScoped**
  - El Bean no se almacena en un scope. Útil para bean referenciados por otros beans que sí están en un scope
  - Nuevo scope en JSF 2.0

Cuando usar Application scope

- **Empleado con beans que no modifican su estado**
  - Sólo representa reglas de navegación (sin getters ni setters)
  - Proveedores de listas de opciones para drop downs

```
<h:selectOneMenu value="#{requestScopedBean.choice}">
  <f:selectItems value="#{appScopedBean.options}"/>
</h:selectOneMenu>
```

  - Propiedades de tipo estructuras de datos que van en los beans principales
  - Uso de inyección de dependencia @ManagedBeanProperty
- Beans que son compartidos
  - Entre usuarios y entre páginas
  - Poner atención a la sincronización para evitar condiciones de carrera
  - Aunque es poco habitual este tipo de problemas

Cuando usar Application scope 2

- **Normalmente**

  @ManagedBean

  @ApplicationScoped

  public class SomeClassNoUserState { ... }

  – La clase se instancia la primera vez que se usa. Después de esto todos los usuarios y solicitudes comparten esta instancia.

- **Ocasional (Evaluación ansiosa)**

  @ManagedBean(eager=true)

  @ApplicationScoped

  public class SomeClassBigDataNoUserState { ... }

  – La instancia se crea una vez se carga el servidor. Útil cuando hay que instanciar datos de gran volumen en esa clase
  – Listas de opciones de drop downs
  – Mapas para lógica de negocio
  – Se solían emplear variables estáticas para abordar este problema. El scope Application es una alternativa razonable

Session scope

- **Se recicla la instancia del bean si:**
  - Se trata del mismo usuario
  - Se trata de la misma sesión del navegador
    Normalmente basado en cookies, pero también puede estar basado en reescritura del URL
- **Útil para**
  - Recordar preferencias de usuario
  - Prerellenar formularios
  - Carritos de la compra
- **Las clases de usuario debe ser Serializable**
  - Algunos servidores guardan los datos de sesión en disco para poner reciclarla
  - Aplicaciones web distribuidas necesitas replicar sesiones

Acceso a request y response

- **No hay acceso automático a request & response**
- **Métodos estáticos**
  - ExternalContext context = FacesContext.getCurrentInstance();
  - HttpServletRequest request = (HttpServletRequest)context.getRequest();
  - HttpServletResponse response = (HttpServletResponse)context.getResponse()
- **Cuando usar request**
  - Manipulación explícita de sesiones (P.e. cambiar el inactive interval o invalidar la sesión)
  - Manipulación explicitade cookies (P.e. en cookies persistentes)
  - Lectura de cabeceras de request (P.e. User-Agent)
- **Usos de response**
  - Establecer códigos de estado HTTP
  - Fijar cabeceras de respuesta
  - Fijar cookies persistentes

Inyección de dependencia

- Básicamente consiste en asignar un valor a una propiedad de un bean sin necesidad de escribir el código en Java, empleando anotaciones o faces-config.xml

```
@ManagedProperty(value="#{someBean}")
  private SomeType someField;
```

  - <managed-property> Permite hacerlo en faces-config.xml
  - Como JSF 1.x, pero en faces-config.xml se permite modificar Maps, con anotaciones no.
  - Se requiere un método setter para la propiedad a modificar (setSomeField())
- Bean principal

```
@ManagedBean
public class MainBean {
@ManagedProperty(value="#{lookupServiceBeanName}")
private CustomerLookupService service;
public void setService(...) { ... }
}
```

- Bean de búsqueda que se inyecta con instanciación impaciente

```
@ManagedBean(eager=true)
@ApplicationScoped
public class LookupServiceBeanName
```

Lenguaje de expresiones JSF

- Los scriptlets JSP no están soportados en Facelets pero se puede invocar a Java indirectamente:
  - #{employee.**f**irstName}, invoca a get**F**irstName siendo empleoyee un Bean gestionado.
  - <h:inputText value="#{employee.firstName}"/>, valor de input/output.
  - #{employee.addresses[0].zip}. Aquí estamos manejando un getter para una lista, array o mapa.
  - #{var.prop1.prop2.prop3}. Propiedades anidadas permitidas.
- Evaluación diferida vs inmediata:
  - Inmediata: Uso de ${}. Se evalua el resultado lo más pronto posible.
  - Diferida: Uso de #{}. Se posterga la evaluación al momento que detrmina el ciclo de vida de JSF.
- Se permite el uso de operadores aritméticos, relacionales y lógicos**.**
  - Aritméticos: +, - (binary), *, / y div, % y mod, - (unary)
  - Lógicos: and, &&, or, ||, not
  - Relacionales: ==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le
  - Empty: empty, retorna true si un valor es null
  - Condicional: A ? B : C

Lenguaje de expresiones JSF 2

- Variables predefinidas: request, session, application, etc.
- Paso de argumentos a métodos de Beans: a partir de la versión E.L. 2.2
  - <h:inputText value="#{userNumberBean.userNumber('5')}">
- **Uso de []**
  - Array = theArray[index] (get & set)
  - List = theList.get(index) or theList.set(index, *submittedVal)*
  - Map = theMap.get(key) or theMap.put(key, *submittedVal)*

Lenguaje de expresiones JSF vs. JSP

Feature	JSF 2.0 EL	JSF 1.*x* EL (with JSP)	JSP 2.0 EL
Format	#{blah} (immediate output values could be accessed with ${blah})	#{blah}	${blah}
Where used	Anywhere in facelets page Eg: #{customer.firstName}	Only in attributes of JSF tags. Eg: <h:outputText value= "#{customer.firstName}"/>	Anywhere in page Eg: ${customer.firstName}
Represents	Output data, later location for submitted data. Eg: <h:inputText value= "#{customer.firstName}"/>	Output data, later location for submitted data. Eg: <h:inputText value= "#{customer.firstName}"/>	Output data. Eg ${customer.firstName}
Where it looks for beans	Request, session, application (etc.) scopes and managed bean defs.	Request, session, application (etc.) scopes and managed bean defs.	Request, session, application scopes.
Declaration type	None needed for simplest usage. xlmns declaration for h:, ui:, f: tags.	@taglib	None needed
Environments	Java EE 6 servers or servlet 2.5 servers with JSF 2.0 JARs.	Java EE 5 servers or servlet 2.4 servers with JSF 1.*x* JARs.	Servlet 2.4+ servers

Lenguaje de expresiones: Variables predefinidas

- **Variables predefinidas**
  - facesContext. The FacesContext object.
    - p.e. #{facesContext.externalContext.remoteUser}
  - param. Request params.
    - E.g. #{param.custID}
  - header. Request headers.
    p.e. #{header.Accept} or #{header["Accept"]}
    - #{header["Accept-Encoding"]}
  - cookie. Cookie object (not cookie value).
    - P.e. #{cookie.userCookie.value} or #{cookie["userCookie"].value}
  - request, session
    #{request.contextPath}, #{request.queryString}, #{session.id}
    #{request.contextPath} useful for making relative URLs. See templating section. – initParam. Context initialization param.
- **Problema**
  - Desacopla la capa de vista de la de lógica de negocio por lo que se recomienda pasarlo a la capa de negocio.

Len

- **Arimét**
  - + -
- **Relacio**
  - == ó
  - En n vez
- **Lógicos**
  - &&
- **Empty**
  - emp
  - True
- **Condici**
  - A ? I
- **Recom**
  - Emple

EL Expression	Result
${1 > (4/2)}	false
${4.0 >= 3}	true
${100.0 == 100}	true
${(10*10) ne 100}	false
${'a' < 'b'}	true
${'hip' gt 'hit'}	false
${4 > 3}	true
${1.2E4 + 1.4}	12001.4
${3 div 4}	0.75
${10 mod 4}	2
${!empty param.Add}	False if the request parameter named Add is null or an empty string.
${pageContext.request.contextPath}	The context path.
${sessionScope.cart.numberOfItems}	The value of the numberOfItems property of the session-scoped attribute named cart.
${param['mycom.productId']}	The value of the request parameter named mycom.productId.
${header["host"]}	The host.
${departments[deptName]}	The value of the entry named deptName in the departments map.
${requestScope['javax.servlet.forward.servlet_path']}	The value of the request-scoped attribute named javax.servlet.forward.servlet_path.
#{customer.lName}	Gets the value of the property lName from the customer bean during an initial request. Sets the value of lName during a postback.
#{customer.calcTotal}	The return value of the method calcTotal of the customer bean.

Texto condicional con rendered

- El atributo rendered recibe un boolean que indica si el control se debe visualizar o no.

```
<h:inputText value="#{programmer.level}" size="12"
    rendered="#{programmer.levelEditable}"/>
<h:commandButton value="Update"
    rendered="#{programmer.levelEditable}">
    <f:ajax render="@form" execute="@form"/>
</h:commandButton>
<h:outputText value="#{programmer.level}"
    rendered="#{!programmer.levelEditable}"/>
```

Paso de argumentos a métodos

- Sólo soportado a partir de E.L. 2.2 (JSP 2.2)
- Sintaxis directa: #{someBean.someMethod(arg1, arg2)}
- No es parte de JSF 2.0
- El servidor debe soportar servlets 3.0
- Servidores con soporte para JEE 6
  - Glassfish 3, Jboss 6 y Tomcat 7
  - Falla en servidores jee 2.5

Reglas de navegación

- Fase del Ciclo de vida de JSF: 5. Invoke application
- Tipos de navegación
  - Dinámica (To-Ids dinámicos)
    - <to-view-id>#{exam.nextQuestionPage}</to-view-id>
  - Estática
    - <h:commandButton ... action=”pagina.xhtml"/>
- Tipos de navegación dinámica (Reglas implícitas vs explícitas)
  - Explícita. Haciendo uso de faces-config.xml
  - Implícita. Los “outcomes” de las acciones deben llamarse igual que la página destino sin la extensión.

Reglas de navegación II

- Reglas explícitas (faces-config.xml)
  - Un elemento <navegation-rule> por página origen
  - Subelemento <from-view-id>
    - Indica la página de partida del flujo
    - Permitido wildcards (*). Usados en páginas de servicios general como errores ejemplo.
  - Subelemento <navigation-case>
    - Indica un flujo de navegación que parte del <from-view-id>
  - SubSubelemento <from-action>
    - Indica el método que genera la etiqueta de navegación.
    - Es necesario cuando una misma vista tiene varias transiciones que parten de diferentes acciones que retornan la misma etiqueta (por ejemplo “exito”).
  - SubSubelemento <from-outcome>
    - Indica la etiqueta que debe generar la página de partida (ya sea una etiqueta constante o el resultado de un método de un MBean) para que la navegación vaya al siguiente <to-view-id>
  - SubSubelemento <to-view-id>
    - Indica la página de destino

```
<navigation-rule>
  <from-view-id>/index.xhtml</from-view-id>
  <navigation-case>
    <from-action>#{controller.listado}
    </from-action>
    <from-outcome>exito</from-outcome>
    <to-view-id>/listado.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>alta</from-outcome>
    <to-view-id>/AltaForm.xhtml
    </to-view-id>
  </navigation-case>
</navigation-rule>
```

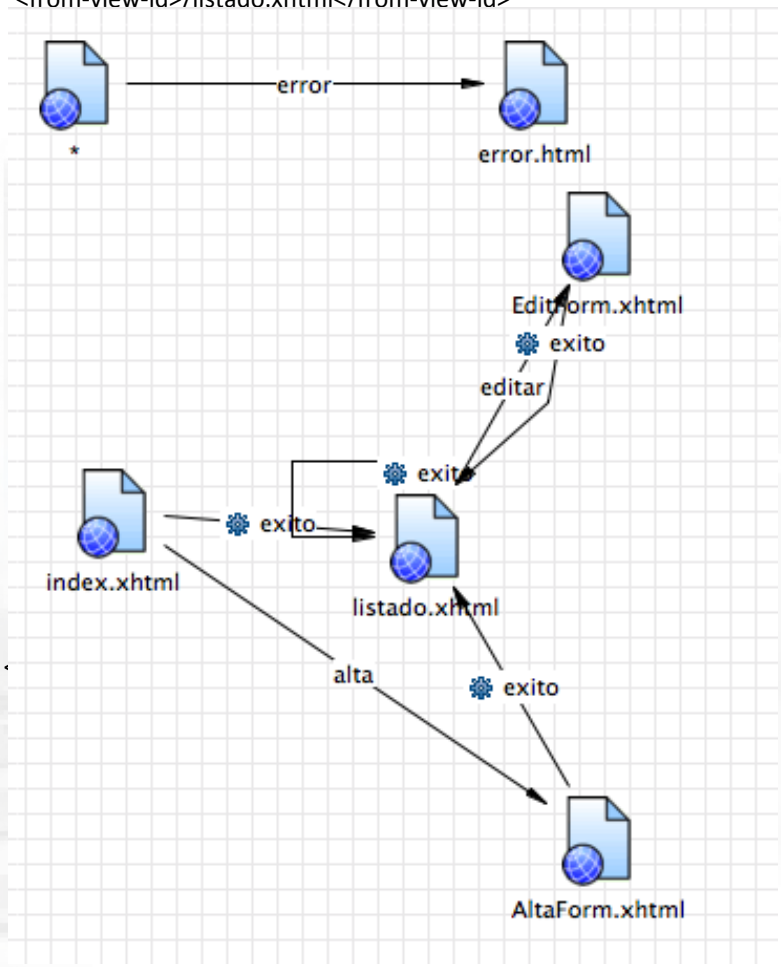
Ejemplo - Reglas de navegación

```
<?xml version="1.0"?>
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_2_2.xsd"
  version="2.2">
 <managed-bean>
  <managed-bean-name>controller</managed-bean-name>
  <managed-bean-class>com.sdi.presentation.Beanalumnos</managed-
bean-class>
  <managed-bean-scope>application</managed-bean-scope>
 </managed-bean>
 <navigation-rule>
  <from-view-id>*</from-view-id>
   <navigation-case>
    <from-outcome>error</from-outcome>
    <to-view-id>/error.html</to-view-id>
   </navigation-case>
 </navigation-rule>
 <navigation-rule>
  <from-view-id>/index.xhtml</from-view-id>
  <navigation-case>
   <from-action>#{controller.listado}</from-action>
   <from-outcome>exito</from-outcome>
   <to-view-id>/listado.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
   <from-outcome>alta</from-outcome>
   <to-view-id>/AltaForm.xhtml</to-view-id>
  </navigation-case>
 </navigation-rule>

 <navigation-rule>
```

Compartición de error.html

```
<from-view-id>/listado.xhtml</from-view-id>
```




```
</faces-config>
```

Manejo de Eventos en JSF: tipos

- **Existen dos tipos de eventos de usuario en JSF**
  - Eventos que ejecutan procesos del backend (Beans ….)
  - Eventos que sólo afectan al formato del interface de usuario
- **JSF denomina a los manejadores de ambos eventos: *action controllers (action)* and *event listeners* respectivamente (Actionlistener y ValueChangeListener)**
- Los Action controllers manejan los datos procedentes del envío de un formulario.
  - Se disparan después de haber rellenado el bean
  - Se disparan después de la validación lógica
  - Retorna las cadenas que produce la navegación entre páginas
- Los Event listeners manejan eventos del UI
  - Se disparan antes de rellenar un bean
  - Evita la validación lógica
  - Nunca afecta directamente a la navegación entre páginas

Action controller

- **Características**
  - Ejecutar una regla de negocio y si es necesaria navegación
  - La action puede retornar un string que genera un flujo de navegación (outcome-view-id)
  - EL retorno de null o void lleva a la misma página.
  - El retorno de la cadena vacía o el mismo View-id conlleva que los Beans con scope ViewScope se destruyan.
- Ejemplo:

```
<h:form>
<h:panelGrid columns="2" styleClass="formTable">
  NOMBRE:
  <h:inputText value="#{controller.alumno.nombre}"/>

  APELLIDOS:
  <h:inputText value="#{controller.alumno.apellidos}"/>

  USERID:
  <h:inputText value="#{controller.alumno.iduser}"/>

  EMAIL:
  <h:inputText value="#{controller.alumno.email}"/>
</h:panelGrid>
  <h:commandButton value="Salvar"
                   action="#{controller.salva}"/>
</h:form>
```

```
public String salva() {
  AlumnosService service;
   try {
     service = Factories.services.createAlumnosService();
      if (alumno.getId() == null) {
           service.saveAlumno(alumno);
       }
     else {    service.updateAlumno(alumno);  }
//Actualizamos el javabean de alumnos inyectado en la tabla

alumnos = (Alumno [])service.getAlumnos().toArray(new
Alumno[0]);
     return "exito";

      } catch (Exception e) {
    e.printStackTrace();
    return "error";
}
}
```

Tipos de manejadores de eventos: ActionListener

- Invocados por botones, imágenes o enlaces:
  - <h:commandButton value="..." .../>
  - <h:commandButton image="..." .../>
  - <h:commandLink .../>
- Estos elementos envían automáticamente el formulario
- Sintaxis
  - public void blah(ActionEvent e) { ...}
  - Con commandButton, el ActionEvent es ignorado pero es necesario declararlo.

```
<h:commandButton ... actionListener="#{bean.blah}"/>
```

Tipos de manejadores de eventos: ValueChangeListener

- Invocados por combo boxes, checkboxes, radio buttons, etc :
  - <h:selectOneMenu …/>
  - <h:selectBooleanCheckbox.../>
  - <h:selectOneRadio .../>
  - <h:inputText .../>
- Lo cuales no envián automáticamente el formulario
- Sintaxis
  - public void blah(ValueChangeEvent e) { ...}
    El evento contiene la opción seleccionada
  - <h:selectOneRadio ...onclick="submit()" valueChangeListener="#{bean.blah}"/>

Implementación de un ActionListener

- No necesariamente se creará un Bean
- Algunas veces se dispone de un Bean para datos del GUI. Scope Session.
- El ActionListener toma ActionEvent como argumento:
  - Método void (no String como los action controllers)
  - ActionEvent está en javax.faces.event
  - ActionEvent tiene la factoría getComponent para obtener una referencia a UIComponent
    - A partir de UIComponent, se obtiene el ID del componente, renderer, y más información
  - Ejemplo:

```
public void someMethod(ActionEvent event) {
doSomeSideEffects();
}
```

Ejemplo de ActionListener y Action controller

```
<!DOCTYPE ...>
....
<h:commandButton value="#{msgs.buttonLabel}" action="#{person.doRegistration}"/>
...
<h:form>
....
<div align="center">
<h:commandButton value="#{msgs.normalFont}"
actionListener="#{formSettings.setNormalSize}"
immediate="true"/>
<h:commandButton value="#{msgs.largeFont}"
actionListener="#{formSettings.setLargeSize}" immediate="true"/>
</div>
</h:form> </h:body></html>
```

Bean para el ActionListener

```
@ManagedBean
@SessionScoped
public class FormSettings implements Serializable {
 private boolean isNormalSize = true;
public String getBodyStyleClass()
 { if (isNormalSize) {
    return("normalSize");
} else { return("largeSize"); }
}
public void setNormalSize(ActionEvent event) {
isNormalSize = true;
}
public void setLargeSize(ActionEvent event) {
isNormalSize = false;
}
```

Bean para el ActionController

```
@ManagedBean
public class Person
{ private String firstName, lastName, emailAddress;
// Accessor methods: getFirstName, setFirstName, etc.
public String doRegistration() {
if (isAnyEmpty(firstName, lastName, emailAddress)) {
return("missing-input"); }
else {
return("confirm-registration"); }
}
... }
```

Manejo de Eventos: Orden de ejecución de los manejadores

- Reglas general: Los actionListeners se invocan siempre antes que los action y en el mismo orden que son declarados en la vista y asociados al componente.
- <h:commandLink value="submit”
  actionListener="#{bean.listener1}" action="#{bean.submit}">
  <f:actionListener type="com.example.SomeActionListener" />
  <f:setPropertyActionListener target="#{bean.property}" value="some" />
  </h:commandLink>
- En este caso el orden de ejecución sería:
  - 1.- bean#listener1() 2.- SomeActionListener#processAction(), 3.- Bean#setProperty() y 4.-Bean#submit.

Tablas a partir de colecciones: como manejar datos de longitud variable

- **Opciones**
  - Construyendo HTML a partir de una propiedad HTML de un bean
  - Empleando el componente h:dataTable
  - Mediante un componente propio
  - Iterando mediante ui:repeat
- Uso de **h:dataTable**
  - Básico: h:dataTable y h:Column
  - Cabeceras
  - Hojas de estilo
  - Tablas con ajax activado
  - Tablas con valores condicionales

Sintaxis

- **Atributos**
  - var, value, border
- Elementos anidados
  - h:column
- Ejemplo

```
<h:dataTable var="someVar" value="#{someCollection}” border="...">
<h:column>#{someVar.property1}</h:column>
<h:column>#{someVar.property2} </h:column> ...
</h:dataTable>
```

- Valores legales para el atributo **“value”**
  - Array, List, ResultSet, Result, DataModel
    Result es de JSTL ; DataModel es parte de JSF

Ejemplo

```
<h2>Programmers at
#{company1.companyName}</h2> <h:dataTable
var="programmer"
value="#{company1.programmers}” border="1">
<h:column>#{programmer.firstName}</h:column>
<h:column>#{programmer.lastName}</h:column>
<h:column>#{programmer.level}</h:column>
<h:column>#{programmer.languageList}</
h:column>
</h:dataTable>
```

Cabeceras y títulos

- Problema

  Habitualmente el contenido de un datatable es un fila de datos de modo que h:column se repite para cada fila

- Solución
  - Marcar las cabeceras con f:facet, que sólo se renderizan para la primera fila

```
<h:dataTable var=”someVar" value="#{someCollection}”>
<h:column>
<f.facet name=“header”>First Heading</f:facet>
#{someVar.someProperty1}
</h:column>
</h:dataTable>
```

- Más opciones de facet: name=“footer”, name=“caption”

Más atributos

- **Sintaxis**
  <h:dataTable var="..." value="..." *otherAttributes*>...</*h:dataTable*>
- Atributos más importantes
  - border, bgcolor, cellpadding, cellspacing, width onclick, ondoubleclick, onmouseover, etc.
  - Los mismos que para la etiqueta <table>.
  - first, rows
    Primera instancia de la colección value, número total de filas a mostrar
  - id, rendered

  Los mismo que para el resto de elementos h:elements. Usar id para Ajax. Usar rendered si se desea omitir table en ciertas situaciones (p.e. si no tiene filas).

Ejemplo

```
<h:dataTable var="programmer" value="#{company1.programmers}" border="1">
<f:facet name="caption">Programmers at #{company1.companyName}</f:facet>
<h:column>
<f:facet name="header">First Name</f:facet> #{programmer.firstName}
</h:column>
<h:column>
<f:facet name="header">Last Name</f:facet> #{programmer.lastName}
</h:column>
<h:column> <f:facet name="header">Experience Level</f:facet>
#{programmer.level}
</h:column>
<h:column>
<f:facet name="header">Languages</f:facet> #{programmer.languageList}
 </h:column>
</h:dataTable>
```

CRUD sobre las filas de una tabla

```
<h:form>
<h:dataTable var="alumno"  value="#{controller.alumnos}"  border="1">
  <h:column><f:facet name="header">Nombre</f:facet>#{alumno.nombre}</h:column>
  <h:column><f:facet name="header">Apellidos</f:facet>#{alumno.apellidos}</h:column>
  <h:column><f:facet name="header">IdUser</f:facet>#{alumno.iduser}</h:column>
  <h:column><f:facet name="header">Email</f:facet>#{alumno.email}</h:column>
  <h:column><f:facet name="header">Baja</f:facet>
  <h:commandLink action="#{controller.baja}" type="submit" value="BAJA" immediate="true">
    <f:setPropertyActionListener target="#{controller.alumno}" value="#{alumno}"/>
  </h:commandLink>
  </h:column>
  <h:column><f:facet name="header">Edición</f:facet>
  <h:commandLink action="/EditForm.xhtml" type="submit" value="EDITAR" immediate="true">
    <f:setPropertyActionListener target="#{controller.alumno}" value="#{alumno}"/>
  </h:commandLink>
  </h:column>
</h:dataTable>
</h:form>
</h:body>
</html>
```

Bean de control CRUD

```
@ManagedBean
@SessionScoped
public class Beanalumnos implements Serializable{
private static final long serialVersionUID = 1L;
private Alumno alumno = new Alumno();
private Alumno[] alumnos = null;

….. Getters/setters

 public String edit() {
 AlumnosService service;
try {
// Acceso a la implementacion de la capa de negocio  a través de la factoría
service = Factories.services.createAlumnosService();
//Recargamos el alumno seleccionado en la tabla de la base de datos por si hubiera cambios.
alumno = service.findById(alumno.getId());
return "exito";

} catch (Exception e) {
e.printStackTrace();
return "error";
} }

…. Otros métodos
}
```

Bibliografía

- **www.coreservlets.com**
- **JEE 6**
  - http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html
- **JSF 2 Docs home page**
  - http://javaserverfaces.java.net/nonav/docs/2.2/
- **JSF 2 Java API**
  - http://javaserverfaces.java.net/nonav/docs/2.2/javadocs/
- **JSF2TagsAPI**
  - http://javaserverfaces.java.net/nonav/docs/2.2/vdldocs/facelets/
- **Java 7 API**
  - http://docs.oracle.com/javase/7/docs/api/