5.3.2 Java Server Faces (JSF)

Framework que implementa el patrón MVC (Modelo-Vista-Controlador)

- Basado en componentes y eventos del lado del servidor
- Mantiene del lado del servidor una representación del interfaz de usuario presentado en el cliente

(a) Elementos del framework

Vista: Conjunto de { ficheros JSP con las *tag libraries* de JSF ; Facelets (ficheros xhtml) ; otros PDLs (*Page Declaration Languages*) [p.ej. XUL] }

- Describen la jeraquía de componentes JSF que conforman cada una de las páginas (pantallas) del interfaz de usuario de la aplicación.
- Vinculan los componentes JSF con los *Managed Beans* (objetos de respaldo)
  - Se hace uso de la sintaxis del *Unified Expresion Language* para referenciar los *Managed Beans* y sus atributos (`#{objeto.atributo}`)

Modelo: *Managed Beans* (objetos de respaldo gestionados por el framework JSF)

- Objetos Java (*Java Beans*) responsables de la lógica de la aplicación
  - implementada directamente en los propios *Managed Beans*
  - delegando en componentes de negocio (EJBs, Beans SPRING, Servicios Web, etc,...)
- Responden a los eventos generados por los componentes JSF
- Controlan la navegación entre páginas (métodos de acción)

Controlador: *Faces Servlet* (configurado en `faces-config.xml` [opc. en JSF 2.0]) + métodos de acción de los *Managed Beans*.

- Todas las peticiones HTTP del usuario pasan por el *Faces Servlet*
- *Faces Servlet* examina las peticiones recibidas, actualiza la representación del interfaz del cliente y los datos de los *Managed Beans* e invoca los manejadores de eventos y las acciones sobre el modelo a través de los métodos de los *Managed Beans*

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" 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/jav
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <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>
    <session-config>
        <session-timeout>  30 </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Otros elementos JSF

Renderizadores *(renderer)*: objeto responsable de generar la representación de los componentes JSF a mostrar en los clientes y de recuperar las entradas de usuario recibidas (*Strings* en los parámetros HTTP) para actualizar los valores vinculados a los componentes

- por defecto, JSF incluye un *render kit* para HTML 4.0

Convertidores y validadores *(converter, validator)*

Convertidores:

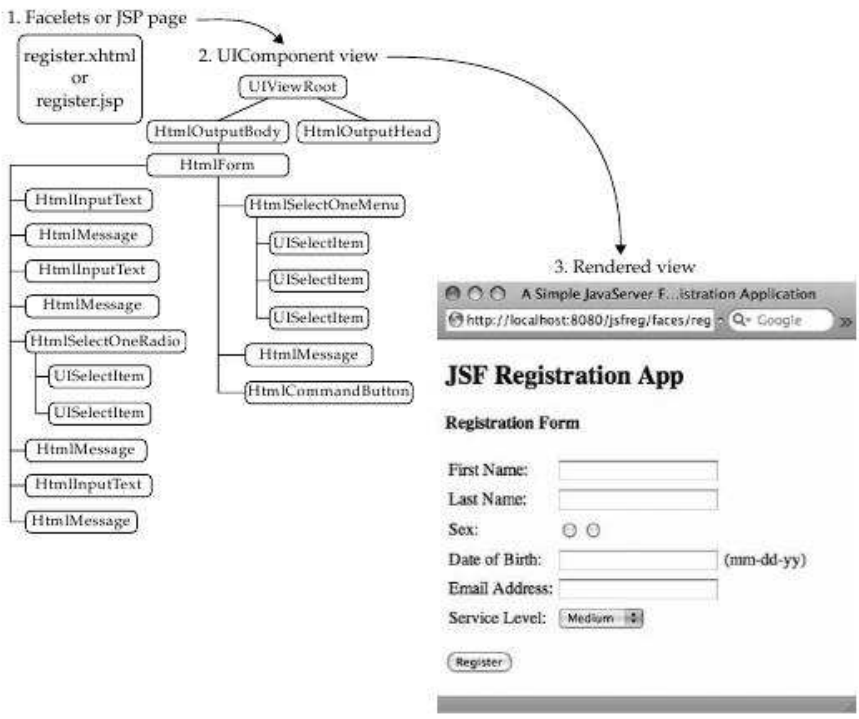
- *[Al generar la presentación]* transforman de forma conveniente los valores de los objetos Java vinculados a los componentes JSF en *Strings* (texto HTML)
- *[Al recuperar las entradas de usuario]* transforman los *Strings* enviados por el cliente (parámetros HTTP) en los objetos Java vinculados a los componentes JSF según el tipo que corresponda (*Integer*, *Double*, *Date*, etc)
- JSF incluye conversores para los tipos básicos
  - Paquete `javax.faces.convert`
  - Etiquetas `<f:convertDateTime>`, `<f:convertNumber>`, ...

Validadores:

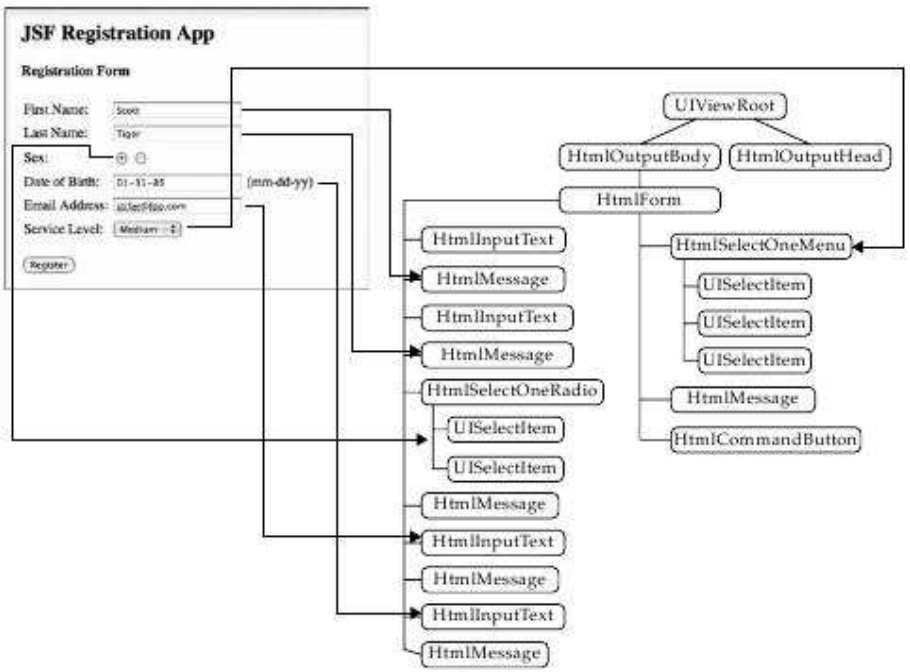
- Ojetos responsables de comprobar (antes de actualizar los atributos de los *Managed Beans*) que los valores recibidos y almacenados en los componentes JSF cumplen las restricciones especificadas. (validaciones del lado del servidor)
  - cada componente JSF que reciba datos de entrada puede tener asociado 1 o más validadores
  - JSF incluye una serie de validadores estándar: `<f:validateLength>`, `<f:validateDoubleRange>`, `<f:validateRegExp>`, ...

(b) Ciclo de vida de JSF

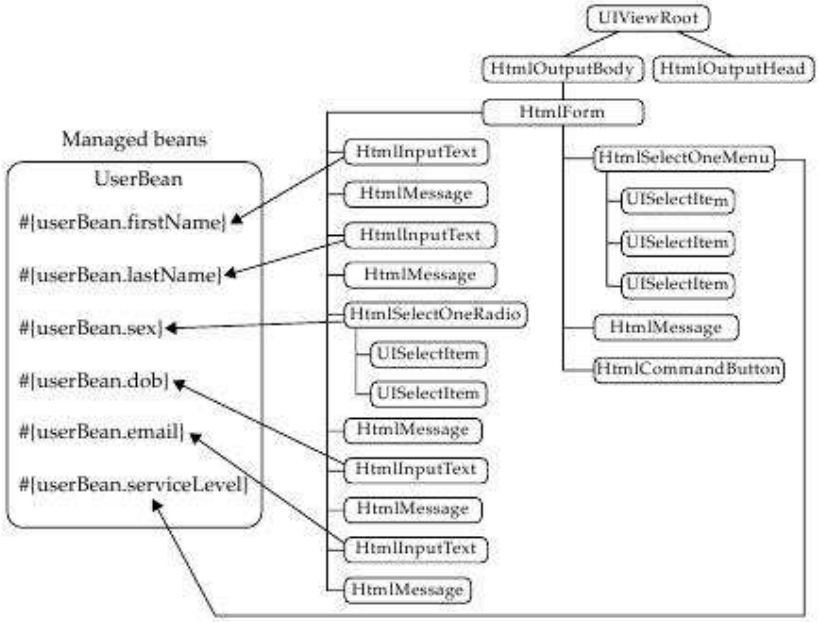
1. **Restaurar vista:** *Faces Servlet* localiza la vista correspondiente a la URL de la petición recibida
   - Si es la primera visita: genera el árbol de componentes de esa vista (`UIViewRoot`) a partir de su fichero de definición (JSP, Facelets, ...)
   - Si la vista ya fue generada: recupera el correspondiente (`UIViewRoot`) previo



2. **Actualizar valores petición:** A partir de los valores de entrada que llegan en la petición HTTP (campos de entrada, parámetros HTTP, cookies, campos cabecera HTTP, etc) se actualizan los valores vinculados a los nodos del árbol de componentes (`UIViewRoot`) que representa el interfaz de usuario.
   - Sólo se actualiza el estado de los componentes, no los valores de los *Managed Beans*
   - Se realizan las conversiones *String*-Objeto necesarias (actúan los *Converters*)
   - Si hay errores de conversión se salta a la fase de Renderizado para informar del error.
     - Se renderiza la misma página.
     - Si incluye componentes para mensajes de error (`<h:message>`, `<h:messages>`) se hacen visibles con los errores de conversión de los componentes implicados.



3. **Procesar validaciones:** JSF recorre recursivamente el árbol de componentes (`UIViewRoot`) que los valores recibidos por cada componente son aceptables.
   - Actúan los *Validators* asociados a cada componentes JSF.
   - Si hay errores de conversión se salta a la fase de Renderizado para informar del error.
     - Se renderiza la misma página.
     - Si incluye componentes para mensajes de error (`<h:message>`, `<h:messages>`) se hacen visibles con los errores de validación de los componentes implicados.
4. **Actualizar valores del modelo:** Se actualizan los valores de los atributos de los *Managed Beans* vinculados a los componentes JSF mediante EL (*expresión language*).
   - Esos valores ya han sido convertidos y validades $\Rightarrow$ es "seguro" actualizar los objetos de respaldo que conforman el Modelo.



5. **Invocar lógica de aplicación:** Se invocan los métodos de los *Managed Beans* (que ya tendrán sus atributos sincronizados con las entradas del usuario)

   a) Se invocan los manejadores de eventos registrados en los componentes JSF.

   - Eventos de cambio de valor (atributo `valueChangeListener` en componentes de entrada [`<h:inputText>`, `<h:inputTextArea>`, `<h:selectOneListbox>`, ...])

     ```
     <h:selectOneMenu valueChangeListener="#{model.menuValueChanged}"
                      value="#{model.serviceLevel}">
     ....


     public void menuValueChanged(ValueChangeEvent evt) {...}
     ```

   - Eventos de acción (atributo `actionListener` en componentes de acción [`<h:commandLink>`, `<h:commandButton>`])

     ```
     <h:commandButton value="Confirm"
                       actionListener="#{UserBean.confirmActionListene
       ....
     public void confirmActionListener(ActionEvent evt) {...}
     ```

   b) Se invocan los *métodos de acción* y se toman las decisiones de navegación.

   - Determina cúal será la página JSF a partir de la que se generará la respuesta a renderizar.

6. **Renderizar respuesta:** Una vez decidida cuál es la vista a enviar al cliente (página actual con componentes de mensaje "activados" si hubo error ó la siguiente página a mostrar si todo fue correcto) se generada la representación de sus componentes (HTML, XML, ...)

   En esta fase se almacena la representación en memoria de la vista actual (`UIViewRoot`) para posteriores visitas.

Adicionalmente se pueden vincular manejadores de eventos específicos que se incocarán en distintas etapas del ciclo de vida JSF (*lifecicle events*).

(c) *Managed Beans*

Objetos de respaldo gestionados por el framework JSF

- El contenedor los crea en tiempo de ejecución cuando son necesarios para la aplicación.
- Cualquier *Java Bean* (constructor sin argumentos + acceso con *getXXX()* y *setXXX()*) definido como *public* puede ser configurado como *Managed Beans*
- También se permiten listas (`java.util.List`) y tablas Hash (`java.util.Map`)

Son responsables del Modelo.

- Almacenan los datos a mostar en las vistas o los recuperadas de la entrada de usuario
- Implementan la lógica de la aplicación (directamente o delegada)
- Manejan la nevagación entre las vistas
- Pueden ser compartidos por varias vistas (dependerá de su *alcance*).
- Se vinculan a las vistas empleando el EL (*expression language*)

Declaración de *Managed Beans* .

En JSF 2.0 se declaran y configuran con la anotación `@ManagedBean`.

- Se les puede asignar un nombre (parámetro `name=''....''`)
- Puede especificarse su alcance (*scope* ó visibilidad)

Alcance	
`@ApplicationScoped`	Objetos disponibles para todas las peticiones de cualquier usuario en todas las vistas de la aplicación
`@SessionScoped`	Objetos disponibles para todas las peticiones que formen parte de la misma sesión de un cliente (valores permanecen entre peticiones de la misma sesión)
`@ViewScoped`	Objetos disponibles para todas las peticiones que se realicen sobre la misma vista (página JSF) (valores permanecen hasta que se navegue a otra página)
`@RequestScoped`	Objetos disponibles desde que se recibe una petición hasta que se la respuesta se envía al cliente (alcance por defecto)
`@NoneScoped`	Objetos no son visibles a las vistas JSF, sólo a otros *Managed Beans*

Ejemplo

```
@ManagedBean
@SessionScoped
public UsuarioController {....}
```

También es posible declararlos en el fichero `faces-config` (única opcion en JSF 1.x)

```
<managed-bean>
  <managed-bean-name> usuarioController </managed-bean-name>
  <managed-bean-class> controladores.UsuarioController </managed-bean-c
  <managed-bean-scope> session </managed-bean-scope>
</managed-bean>
```

Se pueden especificar los valores de las propiedades de los *Managed Beans*

- Con la anotación `@ManagedProperty` o en `faces-config` con `<managed-property>`
- Util para inicializar esos atributos o para inyectarles referencias a otros *Managed Beans* o a sus propiedades.

```
@ManagedBean
public class BookController {
   @ManagedProperty(value = "#{initController.defaultBook}")
   private Book book;
   @ManagedProperty(value = "this is a title")
   private String aTitle;
```

```
<managed-bean>
  <managed-bean-name> bookController </managed-bean-name>
  <managed-bean-class> controller.BookController </managed-bean-class>
  <managed-bean-scope> session </managed-bean-scope>
  <managed-property>
     <property-name> aTitle </property-name>
     <value> this is a title </value>
  </managed-property>
  <managed-property>
     <property-name> book </property-name>
     <value> #{initController.defaultBook} </value>
  </managed-property>
...
</managed-bean>
```

Ciclo de vida de los *Managed Beans*

- Los *Managed Beans* son creados por el Framework JSF cuando se necesitan (nunca se llama directamente a su *new()*)
- Se puede ejecutar código propio justo después de su creación anotando métodos con `@PostConstruct` y justo antes de su liberación, con `@PreDestroy`.

Referencias a *Managed Beans* .

Todo *Managed Beans* tiene asignado un identificador con el que es referenciado en las vistas JSF para las que sea visible empleando la sintaxis EL (
emphExpression Language).

Para especificar al acceso desde las vistas JSF a propiedades o los métodos a ejecutar de un *Managed Beans* se emplea la notacion `#nombreManagedBean.nombreAtributo`.

- En el caso de propiedades se especifican mediante el atributo `value=''...''` de los componentes JSF.
- Esas referencias se traducirán en llamadas a los respecticos métodos *getAtributo()* o *setAtributo(...)* según corresponda al componente JSF.
- En JSF 2.0 se permite pasar parámetros a los métodos especificados con EL.

(d) Navegación y acciones

Los componentes de acción (`<h:commandLink>`, `<h:commandButton>`) fuerzan el envio de una petición HTTP POST desde el cliente.

- Mediante su atributo `action=''...''` se puede especificar la siguiente vista JSF a la que navegar cuando se activen.
  - Puede especificarse directamente el nombre de la vista destino (la extensión es opcional) [navegación implícita]
  - Puede incluir una referencia a un método de acción de un *Managed Beans* accesible desde esa vista.

Los métodos de acción deben ser públicos y devolver un *String*

- El *String* de retorno especifica cúal será la siguiente vistas
  - Si el nombre de una vista (la extensión es opcional) se renderizará esa página. [navegación implícita]
  - Si el *String* está recogido en las reglas de navegación del fichero `<faces-config>` se renderizará esa página.
  - En otro caso se recarga la página actual (informando del error en los componentes de mensajes)
- En EL de JSF 2.0 se permite pasar parámetros a los métodos.
- Dependiendo de las necesidades de cada aplicación concreta, típicamente los *métodos de acción*:
  - Invocan a la lógica de aplicación con los datos que se acaban de recibir del cliente (ya sincronizados con las propiedades del *Managed Bean*)
  - Deciden a qué vista navegar en base a las propiedades del *Managed Bean*
  - Preparan los datos precisos para esa vista, invocando la lógica de la aplicación
  - Devuelven el *String* que indentifica la vista de destino

Ejemplo reglas de navegación

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config xmlns="..." version="2.0">
  <navigation-rule>
     <from-view-id>newBook.xhtml</from-view-id>
     <navigation-case>
         <from-outcome>sucess</from-outcome>
         <to-view-id>listBooks.xhtml</to-view-id>
     </navigation-case>
  </navigation-rule>
  <navigation-rule>
     <from-view-id>listBooks.xhtml</from-view-id>
     <navigation-case>
         <from-outcome>sucess</from-outcome>
         <to-view-id>newBook.xhtml</to-view-id>
     </navigation-case>
  </navigation-rule>
  <navigation-rule>
     <from-view-id>*</from-view-id>
     <navigation-case>
         <from-outcome>error-fatal</from-outcome>
         <to-view-id>error.xhtml</to-view-id>
     </navigation-case>
  </navigation-rule>
</faces-config>
```