ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"Desarrollo de una librería de componentes para la implementación de aplicaciones Web usando Java Server Faces (JSF) y tecnología AJAX"

TESIS DE GRADO

Previo a la obtención del Título de:

INGENIERO EN COMPUTACIÓN ESPECIALIZACIÓN EN SISTEMAS TECNOLÓGICOS

Presentada por:
Ricardo Andrade González.
Carlos Oñate Bravo.
Denisse Lissette Blum Luna.

GUAYAQUIL – ECUADOR
2008

AGRADECIMIENTO

Agradecemos a Dios por ayudarnos a concluir este trabajo a pesar de todas las adversidades, a nuestros padres y a todas las personas que de manera directa o indirecta contribuyeron a la realización de este tema de tesis; de manera especial queremos agradecer a Ing. Carmen Vaca, Directora de Tesis por su ayuda, paciencia y colaboración a lo largo del proyecto.

DEDICATORIA

A mi hijito por ser mi mayor motivación para seguir adelante; A mis hermanos, mi mamá y a mi papá que con paciencia supieron guiarme y estuvieron siempre para apoyarme incondicionalmente a lo largo de mi carrera y en mi vida personal.

Ricardo Andrade González.

DEDICATORIA

A Dios antes que todo, seguido de mi Madre Lupe, mis hermanos, mi mamá Meche, mi tía Patricia, a mi familia en general que estuvo involucrada en este proceso; por empujarme en los momentos que me quedaba y alentarme a seguir hasta el final.

Carlos Oñate Bravo.

DEDICATORIA

A mis padres, y muy especialmente a mis compañeros de tesis por su increíble paciencia y fortaleza para soportar todas las adversidades y realizar la mayoría de este trabajo para graduarme sin mucho esfuerzo.

Denisse Blum Luna.

TRIBUNAL DE GRADUACIÓN

Ing. Holger Cevallos

SUBDECANO DE LA FACULTAD

Ing. Carmen Vaca

DIRECTORA DE TESIS

Ing. Xavier Ochoa

MIEMBRO DEL TRIBUNAL

Ing. Verónica Macías

MIEMBRO DEL TRIBUNAL

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesis de Grado, me corresponde exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL".

(Reglamento de Graduación de la ESPOL).

Ricardo Martín Andrade González.

Carlos Francisco Oñate Bravo.

Denisse Lissete Blum Luna.

RESUMEN

En este trabajo se presenta el proceso de creación de una librería de componentes JSF desde el análisis previo a su desarrollo hasta su implementación en una aplicación de prueba; Además incluye las consideraciones previas para tomar una decisión correcta e informada sobre desarrollar o no una librería de componentes. Luego del desarrollo los mismos se utilizarán para la implementación de una pequeña aplicación Web y se llevarán a cabo encuestas para realizar comparaciones contra el desarrollo de la misma aplicación sin utilizar la librería mencionada.

ÍNDICE GENERAL

LISTA DE ABREVIATURAS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CSS	Cascade StyleSheet
DOM	Document Object Model
EL	Expression Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Developement Enviroment
J2EE	Java to Enterprise Edition
JSF	Java Server Faces
JSP	Java Server Page
MVC	Model View Controller
PDF	Portable Document Format
RAD	Rapid Application Development
RDF	Resource Description Framework
RSS	Really Simple Syndication (RSS 2.0)
TLD	Tag Library Descriptor
URI	Uniform resource identifier
W3C	World Wide Web Consortium
XHTML	eXtensible Hypertext Markup Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation

ÍNDICE FIGURAS

ÍNDICE TABLAS

ÍNDICE CÓDIGO FUENTE

INTRODUCCIÓN

La capacidad de llegar virtualmente a cualquier usuario conectado a internet junto con las mejoras constantes de los navegadores permite una evolución en el desarrollo de aplicaciones. Dando lugar al término: “Aplicación Web”.

Sin embargo, se enfrentan dos problemas principales al momento de desarrollar este tipo de soluciones: el aprendizaje por parte de los desarrolladores y el tiempo de desarrollo. Como consecuencia, se requiere un mecanismo orientado a componentes que puedan ser fácilmente configurados y que implementen funcionalidad utilizada comúnmente, para acelerar así el proceso de desarrollo.

La falta de componentes para aplicaciones Web hace que muchas veces los desarrolladores incurran en reescribir código para realizar las mismas funciones. Esto puede resultar en la aplicación poco eficiente este tipo de soluciones ya que estas funcionalidades pueden ser encapsuladas en una librería para su uso futuro y evitar este problema.

En este trabajo se presentará la creación de una librería de componentes JSF, estos se utilizarán para la implementación de una pequeña aplicación Web y se llevarán a cabo encuestas para realizar comparaciones contra el desarrollo de la misma aplicación sin utilizar la librería mencionada.

CAPÍTULO 1

1. Frameworks y librerías para aplicaciones Web

0.1. Antecedentes

El crecimiento en la implementación de aplicaciones Web ha dado lugar a un nuevo paradigma denominado WEB 2.0. Este tiene como fin crear clientes Web interactivos, con mejoras en la interfaz de usuario utilizando una combinación de tecnologías ya existentes como Javascript, CSS, DOM, entre otros, tratando de acercarse a la interacción obtenida en aplicaciones de escritorio.

Un ejemplo de esta nueva tendencia es el uso de AJAX Esta técnica permite actualizar únicamente elementos o secciones de la página Web en lugar de toda. Esto se logra a través de la unión de varias tecnologías, las cuales son:

- JavaScript que permite la interacción con el navegador por medio de respuesta a eventos.
- DOM para acceder y manipular la estructura HTML de la página.
- CSS y XHTML que hacen posible una buena presentación.
- XML y XSLT para el intercambio y manipulación de datos entre el cliente y el servidor.
- Objeto *XMLHttpRequest* provisto por JavaScript para la comunicación asincrónica.

El uso de este mecanismo se hace posible principalmente cuando están presentes dos elementos importantes, estos son:

- Un navegador con soporte para JavaScript y que implemente los objetos *XMLHTTP* o *XMLHttpRequest*.
- Un servidor que acepte requerimientos HTTP.

Entre las ventajas que AJAX ofrece destacan las siguientes:

- Comunicación asincrónica. No hay bloqueo del cliente hasta obtener la respuesta del servidor.
- Basado en herramientas, plataformas y lenguajes de código abierto.
- Mejoras en la interfaz de usuario al brindar un ambiente más parecido a las aplicaciones de escritorio.
- Optimización de los recursos del servidor al traspasar parte del procesamiento al cliente y solo actualizar las secciones necesarias.

El siguiente gráfico muestra la forma en que dichas tecnologías funcionan en conjunto para actualizar una sección de una página con los nuevos datos del servidor.

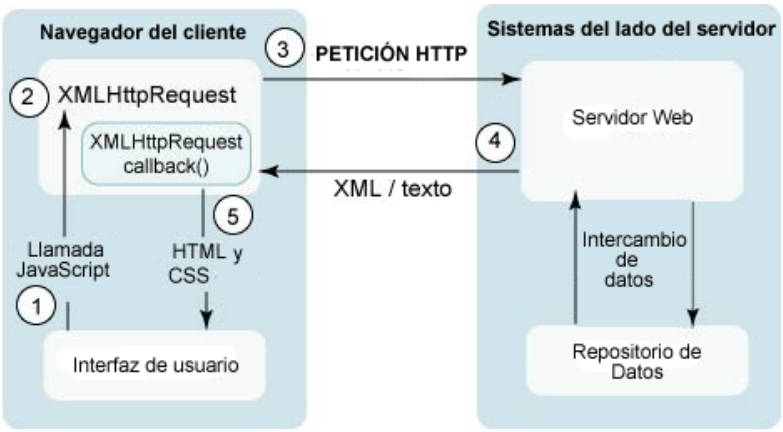


Figura 1.1: Secuencia general del Requerimiento AJAX (1).

Los pasos involucrados en un requerimiento AJAX ilustrados en la Figura 1.1 se describen a continuación:

1. El usuario genera un evento y se ejecuta una porción de código JavaScript encargado de manejar dicho evento.
2. Se crea un objeto *XMLHttpRequest*, se definen el recurso a solicitar al servidor con sus respectivos parámetros y la función de *callback*, la cual se invocará al recibir la respuesta por parte del servidor.
3. El objeto *XMLHttpRequest* hace una petición asincrónica al servidor Web. Un objeto (como un Servlet) recibe la petición, la procesa y obtiene o almacena información en el repositorio de datos.
4. El objeto que recibió la petición (el Servlet) envía un documento XML o texto plano que contenga toda la información actualizada que necesita ir al cliente.

5. El objeto *XMLHttpRequest* recibe los datos, los procesa a través de la función de *callback*, y actualiza el código HTML por medio de DOM con los nuevos datos.

Para demostrar la difusión de esta tecnología en el ámbito empresarial de E.E.U.U. presentaremos los resultados de un estudio publicado por SDTimes en Septiembre del 2006 realizado por BZ Research. El estudio fue realizado a 578 subscriptores de la revista y tiene una exactitud de +/- 3 %.; Los datos son los siguientes:

- 18,9 % ya tienen desplegada tecnología AJAX en producción
- 12,0% están desarrollando aplicaciones que entrarán en producción
- 14,2% están en la fase de piloto
- 37,7% están estudiando la tecnología
- 9,5% no tienen planes todavía
- 7,6% no sabían

En el siguiente grafico se presentan los lenguajes más utilizados para el desarrollo de aplicaciones que implementan AJAX.

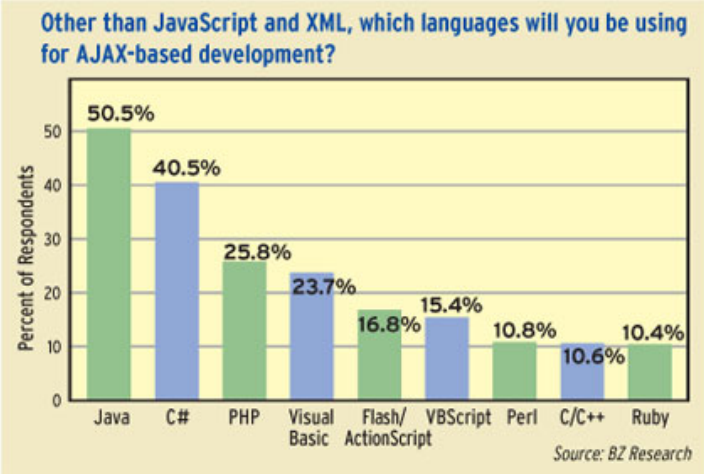


Figura 1.2: Lenguajes usados en aplicaciones AJAX (2).

El estudio muestra que el lenguaje más utilizado para el desarrollo basado en AJAX es Java, correspondiente al 50.5% de los encuestados. El desarrollo basado en Java, incluye tecnologías como JSP y JSF, esta última será la herramienta utilizada para el desarrollo de los componentes.

0.2. Frameworks

Un framework es una arquitectura reusable que soporta aplicaciones; Trata de recopilar tareas comunes e interactúa con la capa de lógica de negocios. Sobre esta se construyen los requerimientos de software.

Los Frameworks son diseñados con el intento de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional (3).

Por qué se necesitan Frameworks

El objetivo de un Framework es alejar al equipo de desarrollo de las tareas de programación comunes y enfocar sus esfuerzos en la lógica de negocio. Sin embargo es una decisión que debe ser analizada ya que en algunos casos podría ser contraproducente al añadir código innecesario o el aprendizaje ser muy costoso. Como en cualquier proyecto es necesaria una evaluación inicial de costo vs beneficio para tomar una buena decisión.

JSF, Struts y otros frameworks

Para acelerar el proceso de desarrollo de aplicaciones Web existen diferentes frameworks como JSF, Struts, Spring, Tapestry, WebWorks entre otros los cuales hacen más compleja la etapa de inicio del proyecto ya que esta decisión debe mantenerse durante todo el desarrollo.

El siguiente grafico muestra una pila de servicios y características de una aplicación web junto a su relación con JSF, Struts, Servlets, JSP y un servidor web tradicional.

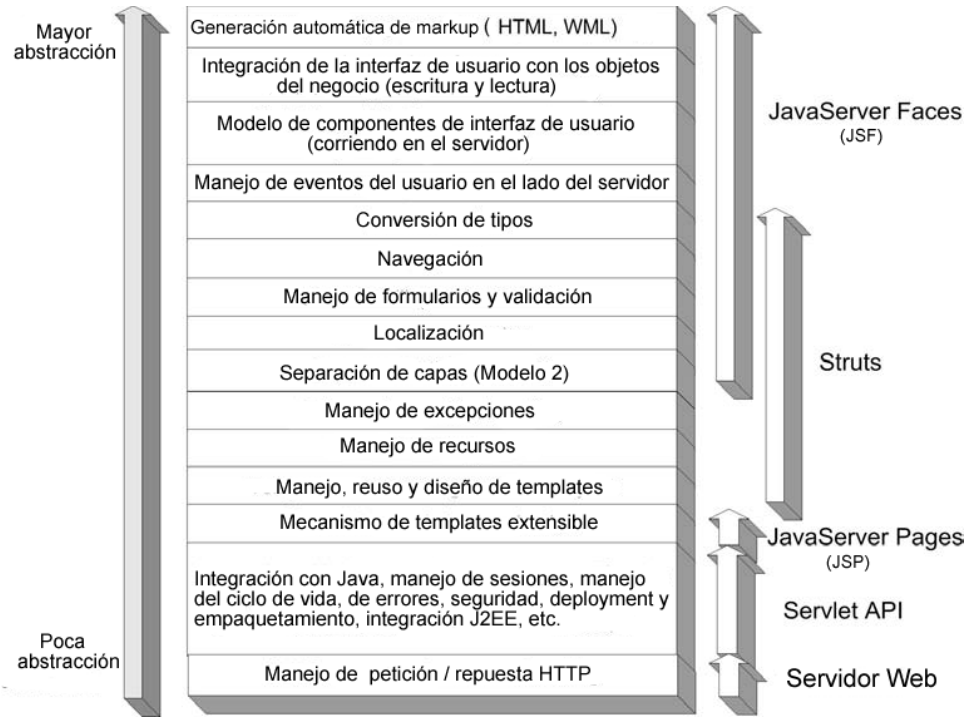


Figura 1.3: Infraestructura de una aplicación Web vista como una pila de servicios (4).

JSF puede ser visto como un framework de interfaz de usuario que puede ser complementado con el uso de otros frameworks como Struts o Spring para intentar abarcar la mayor cantidad de características descritas en la pila.

0.3. Estándares y tecnologías utilizadas

Para el desarrollo de los componentes JSF se usarán varias tecnologías y estándares cuya descripción se lista a continuación:

- **HTML (Hyper Text Markup Language):** Es un lenguaje de marcas diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas Web. Gracias a

Internet y a los navegadores como Internet Explorer, Opera, Firefox, Netscape o Safari, el HTML se ha convertido en uno de los formatos más populares y fáciles de aprender que existen para la elaboración de documentos para Web.

- **XML (eXtensible Markup Language):** Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). No es realmente un lenguaje de programación, sino una manera de definir vocabularios de datos para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.
- **CSS (Cascading Style Sheets):** Es un estándar desarrollado por el World Wide Web Consortium (W3C) con el fin de separar la estructura de las páginas Web de la presentación.
- **J2EE:** Es una plataforma que define estándares para el desarrollo de aplicaciones multicapa. Esta plataforma simplifica las aplicaciones basándolas en estandarizaciones, componentes modulares y proveyendo un conjunto completo de servicios para estos componentes y manejando automáticamente muchos detalles del comportamiento de la aplicación.
- **AJAX (Asynchronous JavaScript And XML):** Es una técnica de desarrollo Web para crear aplicaciones interactivas. Éstas se ejecutan en el lado del cliente, es decir, en el navegador del usuario, y

mantiene comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad en la misma.

- **RSS:** Desarrollado por Netscape, es un formato sencillo de datos que es utilizado para sindicar (redifundir) contenidos a suscriptores de un sitio Web. El cual se puede usar para tres estándares: Rich Site Summary (RSS 0.91), RDF Site Summary (RSS 0.9 y 1.0), Really Simple Syndication (RSS 2.0).
- **JSP (Java Server Pages):** Es una tecnología Java que permite generar contenido dinámico para Web, en forma de documentos HTML, XML o de otro tipo siendo un desarrollo de la compañía Sun Microsystems. Las páginas JSP permiten la utilización de código Java mediante scripts. Además es posible utilizar algunas acciones JSP predefinidas mediante etiquetas. Estas etiquetas pueden ser enriquecidas mediante la utilización de Librerías de Etiquetas (Tag Libs o Tag Libraries) externas e incluso personalizadas.
- **EL (Expresión Language):** Es un lenguaje que simplifica la escritura de expresiones en JSP 2.0 y J2EE 1.4.
- **URI (Uniform Resource Identifier):** Es una cadena corta de caracteres que identifica unívocamente un recurso (servicio, página,

documento, dirección de correo electrónico, enciclopedia, etc). Normalmente estos recursos son accesibles en una red o sistema.

- **RAD (Rapid Application Development):** Es una técnica cuyo objetivo principal es permitir el desarrollo de herramientas poderosas por medio de un conjunto de componentes reusables.

0.4. Java Server Faces

Java Server Faces (JSF) es un framework para el desarrollo de aplicaciones Web que se ejecutan sobre un servidor Java y que presentan una interfaz de usuario como respuesta al cliente.

JSF es una tecnología relativamente nueva, que simplifica el desarrollo de sistemas Web, centrándose en una arquitectura MVC, la cual claramente define una separación entre la lógica y la presentación de la aplicación haciendo fácil conectar la capa de presentación con el código. Este diseño le permite a cada miembro del grupo desarrollador de la aplicación Web enfocarse en su parte del proceso de desarrollo, y también provee un simple modelo de programación para unir todas las partes.

Por lo mencionado anteriormente esta arquitectura permite implementar aplicaciones que sean fáciles de mantener, característica deseable en un aplicación Web, ya que por lo general una modificación en este tipo de sistemas es mucho más costosa que en aplicaciones de escritorio.

¿Por qué usar JSF?

Actualmente existen dos técnicas populares de desarrollo Web:

- El estilo "desarrollo rápido", usando un ambiente de desarrollo visual tal como Microsoft ASP.NET.
- El estilo "codificación costosa", escribiendo grandes cantidades de código para lograr un alto rendimiento, usando como por ejemplo J2EE.

J2EE es una plataforma atractiva, altamente escalable y portable a múltiples plataformas. Por otro lado, ASP.NET hace más fácil la creación de interfaces atractivas para el usuario evitando la programación tediosa. Para los programadores lo ideal sería tener todas esas ventajas en una sola plataforma, JSF promete ser un framework que combina el desarrollo rápido con un buen rendimiento y una alta escalabilidad.

Para los programadores que están familiarizados con el desarrollo de Java del lado del cliente pueden ver a JSF como "Swing para aplicaciones del lado del servidor". Para los que tienen experiencia en JSP (Java Server Pages), JSF provee mucho más de lo que tienen que implementar "a mano" y para los que usan frameworks como Struts, JSF, la arquitectura propuesta resultará muy familiar.

1.5. Librería de componentes propuesta

Muchos desarrolladores realizan sus aplicaciones Web de forma ineficiente debido a que reescriben código que tienen la misma funcionalidad usada anteriormente. La solución propuesta para dicho problema, es el desarrollo de una librería de componentes para el desarrollo de aplicaciones Web usando Java Server Faces (JSF). A continuación se muestran los objetivos de la misma.

Objetivos de la propuesta

El objetivo principal es desarrollar una librería de componentes JSF usando AJAX cuando la funcionalidad implementada lo requiera. Los objetivos específicos son los que se detallan a continuación:

1. Proporcionar al desarrollador componentes adicionales a los componentes incluidos en la distribución de JSF y que puedan ser adaptados a los requerimientos particulares de una aplicación Web.
2. Establecer el procedimiento a seguir para el desarrollo de los componentes y documentarlo.
3. Promover la reutilización de código a través de la distribución de componentes que encapsulan funcionalidad comúnmente implementada en aplicaciones Web.

4. Crear componentes usando conceptos de Web 2.0 que impulsan clientes Web ricos y eficientes a través del modelo de programación propuesto por AJAX.
5. Comparar tiempo y costos de desarrollo de una aplicación Web pequeña en dos escenarios: con la librería de componentes desarrollada y sin ella.

A continuación se detallan los componentes que se desarrollarán para la librería

Componentes a desarrollar

JSF propone un conjunto de componentes básicos y además la ventaja de que el desarrollador pueda crear componentes propios que se ajusten a sus necesidades. En este trabajo se explotará esta capacidad de JSF para desarrollar componentes personalizados; Los componentes de la librería a desarrollar son:

1. **Reproductor de Video:** Reproduce listas de videos en formato MPEG almacenados en un servidor, especificando su respectiva ruta.
2. **Reproductor de Mp3:** Reproduce archivos de música en formato MP3 en la página Web.
3. **Login:** Formulario estándar (Usuario/Contraseña) para autenticar usuarios usando encriptación.

4. **Formulario de registro:** Crea un formulario para ingreso de datos a través de un JavaBean. Creando una nueva instancia para ser manejada por el desarrollador.
5. **Exportar datos en XML a PDF:** Transforma un archivo XML a otro en formato PDF a través de un archivo convertidor proveído por el desarrollador escrito en XSLT-FO. El acceso al documento transformado será por medio de un hiperenlace.
6. **Rol de noticias RSS:** Presenta noticias pertenecientes a una categoría especifica en la página Web que son alimentadas por un servidor de noticias y se procesa para darle formato CSS especificado por el desarrollador.
7. **Álbum de fotos:** Presenta las fotos en forma de tira. Teniendo propiedades editables como la orientación y longitud de la tira, además de la ubicación de los botones de navegación.
8. **Filtro XML:** Realiza una búsqueda a partir de una fuente XML dada.
9. **Gráficos estadísticos:** Genera gráficos de barras y pastel dada una colección de datos, donde se muestra la información adicional correspondiente a los mismos.

CAPÍTULO 2

2. Java Server Faces

2.1. Funcionamiento general de una aplicación JSF

En un escenario típico del funcionamiento de una aplicación JSF se tendría una secuencia de acciones como la mostrada en la Figura 2.1. El cliente, que puede ser un navegador Web o un dispositivo móvil, realiza un requerimiento HTTP.

1. EL requerimiento llega al servidor por medio de la página JSP, la cual en algún punto medio del procesamiento llama a los componentes JSF que estén dentro de la misma.
2. Los componentes dentro de la página son llamados para su procesamiento, y sus atributos pueden estar relacionados con atributos o métodos de clases (beans).
3. Después de una serie de pasos, la parte de la presentación asociada a cada componente JSF, se ejecuta para que finalmente sea devuelta al cliente como respuesta HTTP.

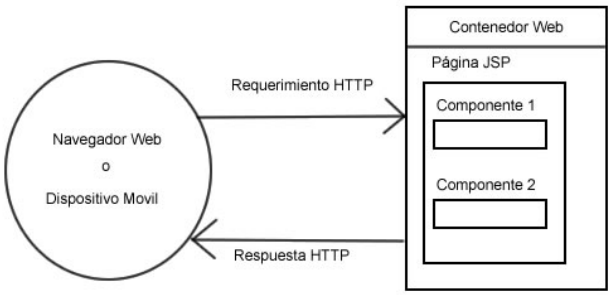


Figura 2.1: Interacción básica entre el cliente y la aplicación Web.

JSF utiliza una arquitectura en la cual el concepto central es el uso del modelo propuesto en el patrón MVC el cual se describe a continuación.

2.2. Arquitectura JSF

Patrón Modelo-Vista-Controlador

MVC es el patrón de diseño recomendado para aplicaciones interactivas en Java, permite centralizar el control y hace que la aplicación sea más extensible. MVC también ayuda a los desarrolladores con diferentes aptitudes a enfocarse en sus habilidades principales y a colaborar a través de interfaces claramente definidas.

MVC propone distribuir las partes de una aplicación en: modelo, vista y controlador. A continuación se detalla cada uno de ellos:

- **Modelo**.- Manipula los datos que la aplicación utiliza. Implementa las reglas del negocio y se encarga del manejo de la fuente de datos.

Los datos provistos por el modelo son independientes de la representación visual, por esto pueden ser usados por cualquier vista sin que sea necesario repetir código.

- **Vista**.- Presenta el modelo en una interfaz de usuario (presentación visual de los datos). La vista está aislada de las operaciones que se realizan sobre los datos.
- **Controlador**.- Procesa y responde a eventos, define el comportamiento de la interfaz del usuario (vista) y el modelo ante la entrada del usuario. El usuario dispara un evento que cambia el modelo, el cual en turnos, notifica a las vistas registradas para que actualicen o refresquen sus datos.

En la Figura 2.2 se muestra el funcionamiento del modelo MVC que se inicia en la petición hecha por el cliente y recibida por el controlador (Servlet), este a su vez crea y maneja el modelo (Beans), realiza las operaciones y re direcciona a la vista (JSP), el cual accede al modelo para presentar la respuesta al cliente.

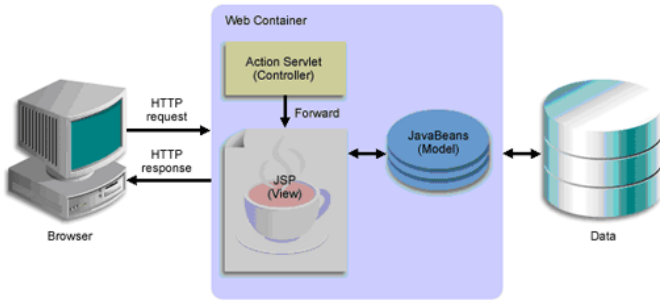


Figura 2.2: Modelo 2 MVC (5).

JSF usa MVC, el *modelo* es manejado por los Backing beans, la *vista* es implementada con páginas JSP y el *controlador* es un Servlet. Una vez que se ha descrito el patrón JSF, se presentarán los componentes que forman la arquitectura JSF.

2.3. Manejo de Beans en JSF

Un bean es un POJO (Plain Old Java Object), por sus siglas en inglés, es decir, que es una clase de Java que define métodos setters y getters, cuya única regla básica es tener un constructor por defecto sin parámetros.

JSF está basado en el manejo de beans, relacionando los atributos y los métodos definidos en estas clases con los atributos de los componentes que se usan dentro de una página JSP. Usando JSF lo beans pueden ser creados sin necesidad de programarlos. A continuación se describen el alcance de los beans.

Alcance de los Beans

En una aplicación JSF los beans pueden tener tres tipos de alcance (6):

- **Alcance de petición (Request Scope):** Es el más corto, comienza cuando un requerimiento HTTP es enviado y termina cuando la respuesta es enviada de regreso al cliente.

- **Alcance de sesión (Session Scope):** Es aquel que dura mientras dure la sesión que fue creada en la comunicación con el cliente. La duración puede estar dada por lógica de Negocio o porque su tiempo de vida ha terminado (timeout). Son los usados para aplicaciones tipo carrito de compras.
- **Alcance de aplicación (Application Scope):** Duran mientras dure la aplicación Web. Se puede compartir su información tanto con el alcance de petición como el de sesión.

A continuación se describe cómo configurar los beans dentro de la aplicación.

Configuración de los Beans

Aunque la configuración de beans se puede hacer desde archivos que contienen solamente sus declaraciones como por ejemplo bean.xml, el archivo de configuración faces-config.xml es el más usado. Estos archivos son colocados dentro del directorio WEB-INF de la aplicación Web, para definir un bean se usa la etiqueta <*managed-bean*> que va dentro de la etiqueta raíz <*faces-config*>.

A continuación se muestra el contenido de un archivo faces-config.xml que configura un bean:

```
<faces-config>
  <managed-bean>
    <managed-bean-name>usuario</managed-bean-name>
    <managed-bean-class>
      ec.edu.espol.fiec.BeanPersona
    </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
</faces-config>
```

Código Fuente 2.1: Contenido de un archivo faces-config.xml que configura un bean.

Se pueden observar 3 etiquetas dentro de <*managed-bean*>:

- <*managed-bean-name*>**:** Se le asigna el nombre al bean.
- <*managed-bean-class*>**:** Se define la clase del bean, declarándola desde el paquete que la contiene.
- <*managed-bean-scope*>**:** Se establece el alcance del bean.

Anteriormente se indicó que un bean puede ser configurado y declarado desde otros archivos. Para indicar a la aplicación el lugar en el que está definido el archivo de los beans es el archivo de configuración web.xml que también está dentro de WEB-INF. Se usa la etiqueta <*context-param*> para declarar parámetros, en la cual se utiliza <*param-name*> y <*param-value*> para asignar el nombre y el valor respectivamente.

A continuación se muestra un ejemplo de configuración de beans por medio del parámetro mencionado anteriormente.

```
<web-app>
  <context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>
      WEB-INF/navigation.xml,WEB-INF/beans.xml
    </param-value>
  </context-param>
  ...
</web-app>
```

Código Fuente 2.2: Ejemplo de configuración de beans (6).

De esta manera se puede separar la navegación, los beans, etc. Una vez que el bean ha sido definido puede ser accedido por componentes JSF. El atributo de un componente y el atributo de un bean se pueden relacionar de la siguiente manera:

```
<h:inputSecret value="#{user.password}"/>
```

Código Fuente 2.3: Línea de declaración de componente.

En el código mostrado se está relacionando el atributo *value* del componente *inputSecret* con el atributo *password* de la clase *user*.

En conclusión, en una aplicación JSF, los beans son comúnmente usados para los siguientes propósitos:

- Para componentes UI (beans tradicionales).
- Para agregar comportamiento de un formulario Web (backing beans).
- Para objetos de negocio cuyos atributos son mostrados en páginas Web.

- Para servicios tales como fuentes externas de datos que necesitan ser configuradas cuando la aplicación es ensamblada.

Una vez que se detalló la relación entre los Beans y JSF se describen elementos los Backing Beans.

2.4. Backing Beans

Los Backing beans tienen la misma funcionalidad que los beans, pero además de almacenar valores de atributos de los componentes también pueden contener objetos relacionados con el componente, tales como: captadores de eventos, mensajes de errores, excepciones, entre otros, o con la aplicación, como reglas de navegación por ejemplo.

La manera de enlazar algún campo del Backing bean, dentro de una página JSP, con el atributo de un componente es a través de EL, mediante la forma #{}. El término *campo* se refiere tanto a un atributo como a un método.

A continuación se muestra un ejemplo de relación entre el valor de un componente de ingreso de valores con el atributo id y de su validador con el método validar del bean *BeanUsuario*.

```
<h:inputText id=" idUsuario "    value="#{ BeanUsuario.id }"
  validator="#{ BeanUsuario.validar}" />
```

Código Fuente 2.4: Línea de declaración de componente caja de texto.

La navegación entre páginas es otro de los beneficios que en JSF puede ser configurada mediante archivos en forma estática o en forma dinámica. A continuación se describe el funcionamiento de la navegación entre páginas en JSF.

2.5. Navegación

La navegación entre páginas dentro de la aplicación JSF se puede manejar por medio de archivos de configuración donde se especifica hacia dónde dirigirse dependiendo desde qué página venga y qué evento ha ocurrido en la aplicación. A continuación se presenta la regla de navegación que parte desde la página inicial *login.jsp* hacia la página de bienvenida *welcome.jsp*, donde el evento *login* es el único caso posible.

```
<navigation-rule>
  <from-view-id>/login.jsp</from-view-id>
  <navigation-case>
    <from-outcome>login</from-outcome>
    <to-view-id>/welcome.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Código Fuente 2.5: Descripción de regla de navegación.

Las etiquetas para definir la navegación son:

- <*navigation-rule*> Etiqueta raíz para las reglas de navegación, puede haber más de una.
- <*from-view-id*> Nombre de la página donde proviene la navegación.

- <*navigation-case*> Permite definir el destino, contiene dos etiquetas que son:
  - <*from-Outcome*> Identificador del caso de navegación.
  - <*to-view-id*> Nombre de la página destino.

Navegación estática

La navegación estática se puede dar en el caso de un formulario, en el cual se sabe de antemano hacia dónde dirigir los datos para que puedan ser procesados. Mediante la regla previamente establecida.

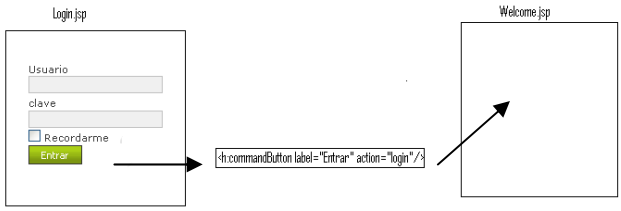


Figura 2.3: Navegación entre páginas

En la Figura 2.3: Navegación entre páginas se puede observar la regla de navegación que corresponde al Código Fuente 2.5: Descripción de regla de navegación. Mediante el atributo *action* de un componente *commandButton* se puede especificar hacia dónde dirigirse solo con indicar el nombre del caso de navegación.

Navegación dinámica

Una de las grandes ventajas que JSF nos ofrece es que el atributo *action* de un componente también acepta métodos, en donde se puede retornar el destino dinámicamente.

Por ejemplo el login en una página nos lleva a dos escenarios: éxito o falla, para implementar esto el botón debe tener un método en donde se hagan las validaciones correspondientes.

```
<h:commandButton label="Login"    action="#{login.verificarUsuario}" />
```

Código Fuente 2.6: Línea de declaración de componente botón.

En el Código Fuente 2.6 se puede apreciar que al atributo *action* se le ha pasado como parámetro el método *verficarUsuario()* del bean *login*, que usando EL se escribe *#{login.verificarUsuario}*.

Este método puede tener la siguiente forma:

```
String verificarUsuario() {
  if (...)
    return "exito";
  else
    return "falla";
}
```

Código Fuente 2.7: Método correspondiente a la acción de un componente.

En este escenario la regla de navegación tendrá dos casos de navegación: *éxito* y *falla*. En el caso de éxito la página destino será *welcome.jsp* y en el caso falla la página destino será *failure.jsp*.

```
<navigation-rule>

   <from-view-id>/login.jsp</from-view-id>
    <navigation-case>
    <from-outcome>exito</from-outcome>
    <to-view-id>/welcome.jsp</to-view-id>
  </navigation-case>

  <navigation-case>
    <from-outcome>falla</from-outcome>
    <to-view-id>/failure.jsp</to-view-id>
  </navigation-case>

</navigation-rule>
```

Código Fuente 2.8: Reglas de navegación entre páginas con caso de éxito y caso de falla.

2.6. Componentes JSF

En JSF, un componente se puede definir como un grupo de clases que interactúan juntas y proveen una pieza reusable. Basada en código de interfaz de usuario en Web. El componente está compuesto de tres clases que trabajan juntas: la clase que maneja la lógica, la clase que maneja la etiqueta y la clase que maneja la presentación del componente al cliente, estas clases serán explicada a profundidad en el Capítulo III (3.4)

La Figura 2.4 muestra como ejemplo un componente JSF denominado *HtmlInputText*. La clase *TextRenderer* presenta el componente visualmente en la página, la clase *Tag* obtiene los valores ingresados y realiza la

asociación. Estas clases juntas logran que los usuarios puedan ingresar cantidades de texto a través de dicho componente.

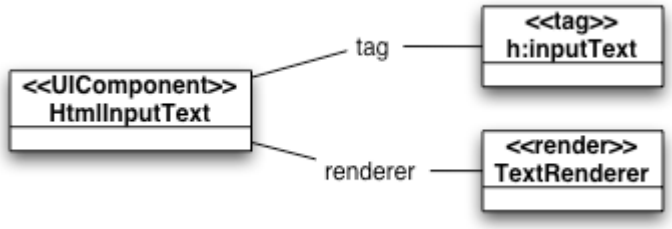


Figura 2.4: Relación entre las tres piezas principales de este componente (7).

Elementos del Componente

Cada vista JSF es implementada por medio de la clase *UIViewRoot* la cual contiene todas las instancias de los componentes UI en una página. La tecnología JSF incluye:

- Un API para la representación de los componentes de la interfaz de usuario, para el manejo de sus estados, eventos, validación de entradas, conversión de datos, definición de navegación entre páginas, y soporte para internacionalización y accesibilidad.
- Una librería de etiquetas JavaServer Pages (JSP) para expresar una interfaz JSF dentro de una página JSP.

A continuación, la Figura 2.5 muestra los elementos JSF mencionados anteriormente.

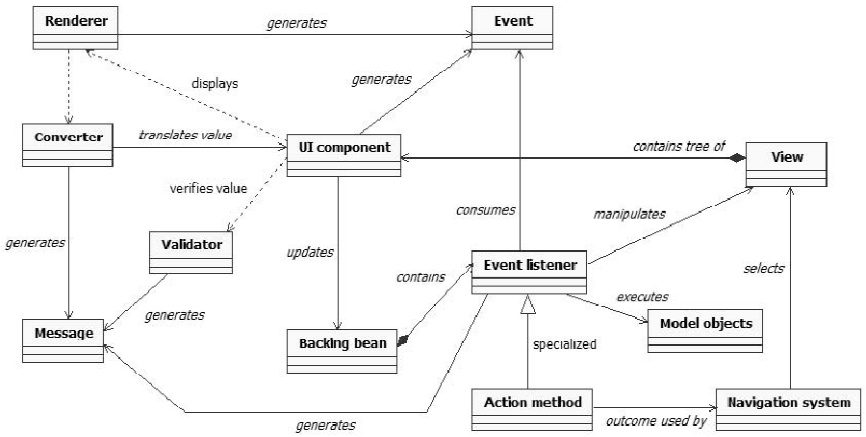


Figura 2.5: Modelo que muestra los conceptos principales de JSF que están relacionados unos con otros. Estos se comunican a través de eventos y mensajes (7).

Termino	Descripción
Componente UI	Es un objeto que maneja diferentes estados, se encuentra alojado en el servidor y provee funcionalidad específica para la interacción con el usuario final
Presentador (Renderer)	Responsable de mostrar un Componente UI y de traducir una entrada del usuario a un valor del componente. Estos pueden ser designados para trabajar con uno o más componentes UI, y un componente UI puede ser asociado con diferentes presentadores.
Validador (Validator)	Responsable de asegurarse que el valor ingresado por el usuario sea aceptable. Uno o más validadores pueden ser asociados a un componente UI.
Backing beans	JavaBeans especializados que contienen valores de los componentes UI e implementan los métodos de los manejadores de eventos. Estos también pueden mantener referencias a componentes UI.
Convertidores (Converter)	Convierte el valor de un componente a una cadena para ser mostrado. Un componente UI puede ser asociado a un solo convertidor.
Eventos y manejadores de eventos (Event y Event listeners)	JSF usa el modelo de eventos y manejadores de eventos de los JavaBeans (también usados por Swing). Los componentes UI (y otros objetos) generan eventos y sus "escuchadores" (listeners) pueden ser registrados para manejar dichos eventos.
Mensajes	Información que es mostrada al usuario. Cualquier parte de la aplicación (backing beans, validadores, convertidores, etc) puede generar información o mensajes de error.
Navegación	Es la capacidad de moverse desde una página a la siguiente. JSF tiene un poderoso sistema de navegación que integra manejadores de eventos especializados

Tabla I: Descripción de los conceptos principales de JSF (7).

En la Tabla I se explica de manera breve los elementos principales de JSF mostrados en la Figura 2.5.

El modelo de programación de JSF y la librería de Tags para expresar componentes UI en una página JSP, ayudan significativamente al desarrollo y mantenimiento de aplicaciones Web del lado del servidor ya que con menos esfuerzo permiten:

- Enlazar eventos generados en el cliente con el código que reside en el lado del servidor, mediante los atributos del componente.
- Mapear componentes UI en una página con datos del lado del servidor.
- Construir un UI con componentes reusables y que se pueden extender.
- Guardar y recuperar estados del UI más allá de la vida de las peticiones del servidor.

En la Figura 2.6 se muestra la arquitectura JSF manejando los conceptos anteriores. Se puede apreciar que existen diferentes tipos de clientes, esto es gracias a que JSF permite reconocer el tipo de cliente y crear una presentación específica para cada uno de ellos.

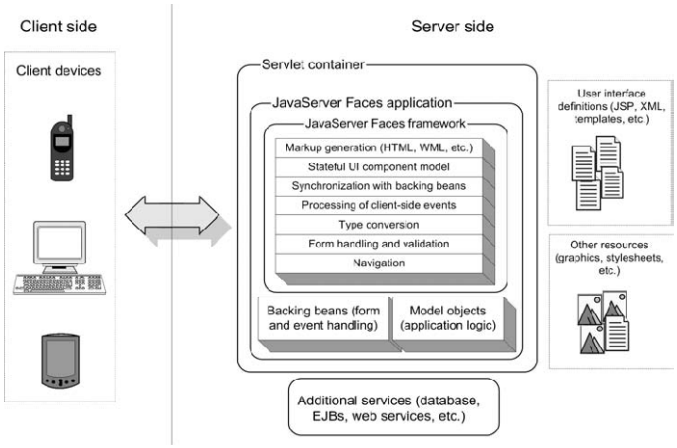


Figura 2.6: Arquitectura JSF (7).

Existen diferentes tipos de componentes tales como estándar, personalizados, código abierto (Open source) y propietarios (Third Party). Cada uno de ellos se describe en las siguientes secciones.

Componentes Estándar

El API estándar de JSF provee un conjunto de componentes básicos que pueda necesitar el desarrollar para la elaboración de una aplicación Web. En la Figura 2.7 se muestran algunos componentes JSF estándares tales como: *UISelectOne*, tipo radio button; *UISelectMany*, tipo combo y *UIInput* tipo texfield.

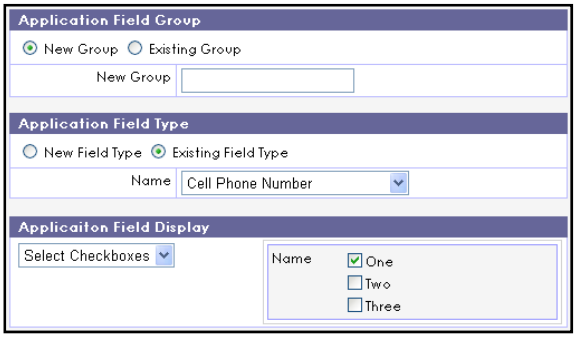


Figura 2.7: Componentes JSF estándar (8).

Componentes Personalizados

Otra de las fortalezas importantes en las que JSF se diferencia de otras tecnologías Web es que sus componentes puedan ser extendidos o personalizados para la creación de un comportamiento específico, es decir JSF permite escribir componentes reusables que no sólo son capaces de presentar HTML sino de proveer eventos, validaciones y conversiones.

En la siguiente figura se muestran dos ejemplos de componentes personalizados, que vale indicar que no son parte de los estándares provistos:

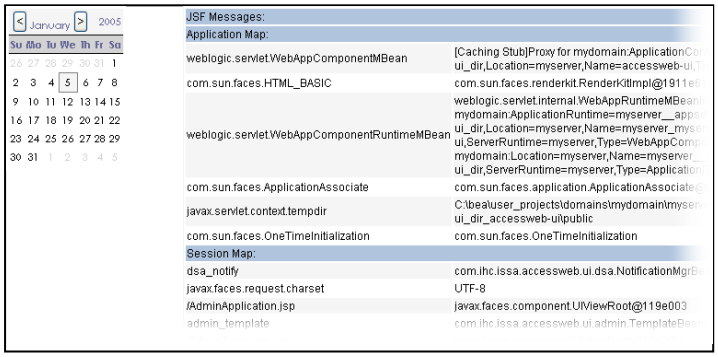


Figura 2.8: Componentes JSF personalizados (8).

Componentes de código abierto (Open Source)

Actualmente existen componentes JSF que pueden ser utilizados directamente en las páginas JSP y cuyo código puede ser modificado ya que es proporcionado por sus desarrolladores. Un ejemplo de esto son los componentes de Apache MyFaces, y los componentes AJAX de SweetDEV RIA.

En la siguiente ilustración se pueden apreciar diferentes componentes de código abierto:

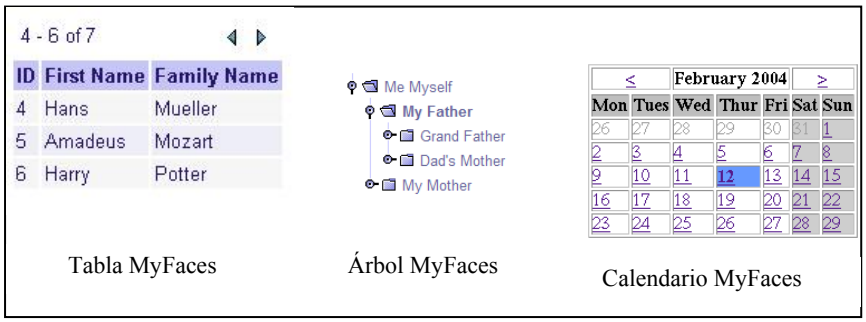


Figura 2.9: Componentes JSF de código abierto conocidos también como Open Source (8).

Componentes Propietarios

Existen librerías de componentes que son comercializadas para el desarrollo rápido y flexible de las aplicaciones Web, como por ejemplo *QuipuKit* que contiene diferentes componentes tales como calendarios, gráficas estadísticas, etc.

A continuación se presentan algunos componentes que se comercializan a través de páginas Web:

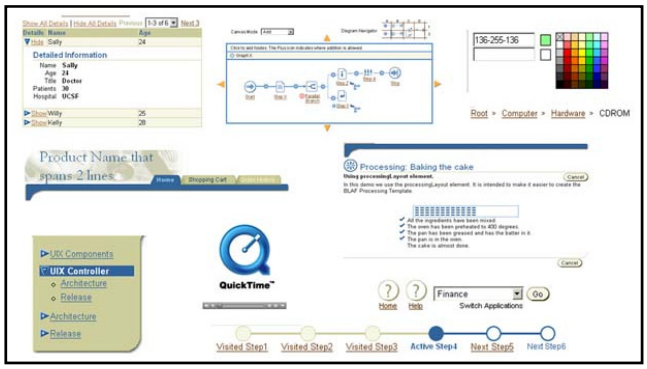


Figura 2.10: Componentes JSF propietarios conocidos también como Third Party (8).

Una vez que hemos concluido con los tipos de componentes, la siguiente sección presente conceptos sobre el manejo de eventos en JSF.

2.7. Manejo de eventos

Se ha explicado que JSF permite al desarrollador manejar eventos generados por acciones del usuario. Para esto se registran manejadores de eventos en los componentes. Por ejemplo se puede registrar un manejador para un menú de la siguiente forma:

```
<h:selectOneMenu    valueChangeListener="#{form1.ciudadModificada}" ... />
```

Código Fuente 2.9: Línea de declaración de un componente combo.

En el Código Fuente 2.9 el método *ciudadModificada()* del bean *form1* se enlaza con el listener del componente *SelectOneMenu*. Este método es invocado después que el usuario realiza una selección del menú.

Dependiendo del comportamiento que se quiera del componente, un en evento se puede registrar utilizando alguno de los tres tipos siguientes:

- **Eventos que cambian valores:** Son aquellos eventos generados al cambiar el valor de un componente de entrada tal como una caja de texto.
- **Eventos de acción:** Son aquellos eventos generados por botones.
- **Eventos de fase:** Son aquellos eventos que pueden ser generados por el ciclo de vida de JSF, los cuales tienen que ser registrados por la clase del evento generador; las fases en las cuales se puede registrar un evento son:
    - Restaurar la vista
    - Aplicar valores de petición
    - Validaciones
    - Actualizar valores del modelo
    - Invocar la aplicación
    - Presentar la respuesta

2.8. Componentes JSF en una página Web

A continuación se muestra un ejemplo del uso de componentes JSF en una página:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<f:loadBundle basename="jsfks.bundle.messages" var="msg"/>
<html>
 <head>
  <title>enter your name page</title>
 </head>
 <body>
  <f:view>
   <h1>
    <h:outputText value="#{msg.inputname_header}"/>
   </h1>
   <h:form id="helloForm">
    <h:outputText value="#{msg.prompt}"/>
    <h:inputText value="#{personBean.personName}" />
    <h:commandButton action="greeting" value="#{msg.button_text}"/>
   </h:form>
  </f:view>
 </body>
</html>
```

Código Fuente 2.10: Página jsp con etiquetas soportadas en JSF.

Para utilizar un componente JSF en cualquier página se requieren dos pasos:

- Incluir en la primera línea una directiva que nos indica en donde encontrar etiquetas JSF que definen elementos HTML, la segunda directiva nos dice donde encontrar etiquetas JSF que definen elementos JSF estándar. La tercera línea carga las propiedades del archivo que contiene mensajes que se desean mostrar en la página JSP. Todo esto se puede apreciar en las 4 primeras líneas del Código Fuente 2.10.
- Se agregan los componentes con sus correspondientes prefijos, nombres y atributos, los cuales son enlazados con sus valores a través de EL. En el resto de las líneas del ejemplo se muestran componentes

gregados a las páginas tales como formularios, cajas de textos, botones.

CAPÍTULO 3

3. Desarrollo de un componente JSF

3.1. Análisis previo a la creación de un componente

Desarrollar nuestros propios componentes para implementar tareas específicas no es una decisión que deba ser tomada a la ligera. Desarrollar un componente puede llegar a ser muy costoso en recursos y tiempo. Por lo tanto el análisis previo es imprescindible y debe considerar algunos aspectos y preguntas claves como:

- Identificar si realmente es necesario el desarrollo de un componente.
- Extender un componente vs. desarrollar uno nuevo

Identificar si es necesario el desarrollo de un componente

Por lo general cambiar atributos de los componentes básicos para que cumpla con las necesidades de nuestra aplicación es suficiente, cambiar la apariencia o el modo de manejar las peticiones del componente son opciones que demandan menor esfuerzo y deben ser tomadas en cuenta.

Debe considerarse crear un componente nuevo que se requiera manejar algún tipo de evento adicional. Ya que por razones de seguridad los tipos de eventos de un componente estándar no pueden extenderse ni usarse para otro componente. Además al análisis debe contemplar si se está haciendo una

“sobre-ingeniería”. Es decir se va a realizar un esfuerzo innecesario para un componente que en realidad no se utilizará sino una sola vez ya sea dentro de la misma aplicación o en otras. Por ejemplo, desarrollar un componente para consultas específicas no es una decisión acertada.

Extender un componente vs. Desarrollar uno nuevo

En el caso de que necesitemos extender propiedades o modificar la forma de codificar, validar y la forma de manejar estados deberíamos extender el nuevo componente de un componente estándar para heredar las propiedades básicas y sobrescribir y agregar algún nuevo comportamiento.

Esta pregunta se complementa con la anterior, ya que es necesaria una investigación para encontrar alguna opción que ya esté desarrollada y contemplar la opción (dependiendo de la licencia del componente escogido) de extender las propiedades y ajustarlo a nuestras necesidades. Existen muchos componentes de código abierto los cuales pueden ser modificados con la opción de contribuir con las modificaciones según sea el caso.

Para el presente proyecto de tesis se requería funcionalidad no implementada por los componentes JSF estándar y por lo tanto de decidió crear un librería de componentes nuevos. Para el desarrollo de una librería se requiere la

configuración de varios archivos que se describen en las secciones siguientes.

3.2. Archivo Descriptor de librerías de etiquetas (TLD)

Casi desde el inicio los autores de la especificación JSP notaron que ningún conjunto de componentes podría hacer todo lo que se necesita en una aplicación Web. Para resolver este problema estos autores crearon un mecanismo para que los programadores puedan crear nuevas etiquetas y que además sean fácilmente implementadas por una página JSP. Un punto principal de este mecanismo es el archivo TLD (*tag library descriptor*)

Un TLD es un archivo en formato XML que contiene la información sobre toda la librería y sobre cada componente definido en ella. Los TLDs son usados por un contenedor Web para validar las etiquetas contenidas en la librería y por herramientas de desarrollo para brindar información sobre los componentes al programador. Si queremos distribuir componentes se deben declarar las etiquetas de los mismos en un TLD.

Características de archivo TLD

Para que un TLD sea interpretado correctamente por un contenedor Web debe cumplir las siguientes reglas:

- Deben tener extensión ".tld".

- Encontrarse en cualquiera de las siguientes ubicaciones a partir del directorio raíz de la aplicación:
    - En el directorio /WEB-INF/.
    - En el directorio: /META-INF/
    - En un subdirectorio de la librería empacada en un JAR

Estructura de archivo TLD

El elemento raíz debe ser <*taglib*> en donde se especifica la versión de JSP requerida y los espacios de nombres que van a ser utilizados. Por ejemplo:

```
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
version="2.0">
```

Código Fuente 3.1: Línea de de la libreria de etiquetas.

La Tabla II muestra solo las etiquetas utilizadas dentro del elemento <*taglib*> para la creación de los componentes de la librería. En elANEXOS ANEXO A se detallan los elementos restantes.

Elemento	Descripción
Description	(Opcional) La descripción de la librería.
tlib-version	La versión de la librería.
short-name	(Opcional) Nombre corto de la librería
uri	El URI *(Uniform resource identifier)* que identifica únicamente a la librería.

Tabla II Etiquetas usadas en los archivos TLD.

A continuación se detalla un ejemplo de la ubicación de los elementos de la Tabla II dentro de un TLD. Se ha tomado como ejemplo el Componente RssViewer.

```
<taglib>
   …
   <description>...</description>
   <tlib-version>0.01</tlib-version>
   <jsp-version>1.2</jsp-version>
   <short-name>lb</short-name>
   <uri>http://fiec.espol.edu.ec/jsf/componentes</uri>
   <description>Libreria de Componentes JSF</description>
   …
   <tag>
      …
   </tag>
</taglib>
```

Código Fuente 3.2: Ubicación de los elementos de la Tabla II

Dentro de este archivo los elementos <*tag*> describen los componentes. A continuación se presentan los detalles de este elemento.

Descripción del Elemento *<tag>*

Cada componente en la librería debe ser especificado y declarado con un elemento <*tag*>. En este elemento se declaran el nombre, la clase que maneja el componente, los atributos e información de las variables creadas por este componente. Cada atributo contiene una indicación sobre si el atributo es requerido, si puede ser determinado por expresiones, el tipo de atributo entre otras.

La Tabla III muestra los subelementos mínimos que se pueden declarar en el elemento <*tag*>. En el

ANEXO B se menciona los otros subelementos que pueden usarse según se requieran.

Elemento	Descripción
Name	El nombre único del componente.
tag-class	El nombre completo de la clase manejadora del componente (HANDLER).
attribute	Declara un atributo del componente.

Tabla III: Subelementos que pertenecen al elemento *<tag>*.

Ubicación de los elementos de la Tabla III dentro de un TLD, se ha tomado como ejemplo el Componente

RssViewer

```
<tag>
   <name>RssViewer</name>
   <tag-class>ec.edu.espol.fiec.componentes.jsf.rss.RssViewerUITag</tag-class>
   …
   <attribute>
      …
   </attribute>
</tag>
```

Código Fuente 3.3: Declaración del elemento *<tag>* en el archivo TLD.

Una vez declarado el componente mediante la etiqueta <*tag*>, se debe establecer los atributos que este tiene, y se lo hace mediante la etiqueta <*attribute*>. El uso de esta se detalla a continuación.

Descripción del Elemento *<attribute>*

Por cada atributo de la etiqueta debe especificarse si es requerido, si el valor puede determinarse por una expresión, el tipo de atributo (Opcional) y si el

atributo es un fragmento. Si el atributo *rtexprvalue* toma el valor "true" o "yes" entonces el tipo de elemento define el tipo de retorno esperado de cualquier expresión especificada como el *value* del atributo. Para valores estáticos el tipo es siempre: *java.lang.String*.

Si un atributo no es requerido el manejador debería proporcionar un valor por defecto. La Tabla IV muestra los subelementos que pertenecen al elemento <*attribute*> que han sido usados en la creación de los componentes. En el ANEXO C se detallan los elementos restantes.

Elemento	Descripción
description	(Opcional) Una descripción del atributo.
name	El nombre único del atributo que está siendo declarado. Un error de traducción resulta si más de un atributo dentro de la misma etiqueta aparece con el mismo nombre.
required	(Opcional) Describe si el atributo es requerido. El valor por defecto es "false".

Tabla IV: Subelementos que pertenecen al elemento <*attribute*>.

Ubicación de los elementos de la Tabla IV dentro de un TLD, se ha tomado como ejemplo el Componente RssViewer.

```
<tag>
   ...
  <attribute>
     <name>url</name>
     <required>true</required>
     <description>Direccion del canal Rss</description>
   </attribute>
  ...
</tag>
```

Código Fuente 3.4: Declaración del elemento <*attribute*> en el archivo TLD

El siguiente código muestra un ejemplo de un TLD con los elementos típicos. Se muestra como ejemplo el Componente RssViewer.

```
<?xml version="1.0" encoding="ISO-8859-1" ?><!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
   <tlib-version>0.01</tlib-version>
   <jsp-version>1.2</jsp-version>
   <short-name>lb</short-name>
   <uri>http://fiec.espol.edu.ec/jsf/componentes</uri>
   <description>Libreria de Componentes JSF</description>
   <!--  COMPONENTE RSS  -->
   <tag>
      <name>RssViewer</name>
      <tag-class>ec.edu.espol.fiec.componentes.jsf.rss.RssViewerUITag</tag-class>
      <!-- Atributos de RssViewer-->
      <attribute>
         <name>url</name>
         <required>true</required>
         <description>Direccion del canal Rss</description>
      </attribute>
      <attribute>
         <name>numeroNoticias</name>
         <required>true</required>
         <description>Numero de noticias a ser mostrados.</description>
      </attribute>
      <attribute>
         <name>style</name>
         <required>false</required>
         <description>Estilo Css del componente tipo inline.</description>
      </attribute>
   </tag>
</taglib>
```

Código Fuente 3.5: Declaración de elementos típicos en un archivo TLD.

3.3. Archivo Faces-Config para la librería de componentes

Es un archivo XML de configuración exclusivo para JSF donde se da valor a diferentes parámetros, entre esos el elemento <*component*> donde se declaran los componentes que vamos a utilizar en la librería.

Descripción del Elemento <*component*>

Representa una implementación concreta de una clase que extienda de UIComponent y que debe ser registrada con el tipo de identificador especificado. Junto con sus propiedades y atributos. Los Tipos de componentes deben ser únicos dentro de la aplicación Web.

La Tabla V muestra los subelementos mínimos que se usan dentro de la etiqueta <*component*>. En elTabla C.I Elementos de la etiqueta <*atributte*>

ANEXO D se detallan los elementos restantes.

Elemento	Descripción
component-type	Representa el nombre bajo el cual la correspondiente clase del componente va a ser registrado.
component-class	Representa el nombre completo de la clase de un componente específico.

Tabla V: Subelementos que pertenecen al elemento <*component*>.

Ubicación de los elementos de la Tabla V dentro del archivo faces-config.xml, se muestra como ejemplo el Componente RssViewer.

```
<faces-config>
   ...
   <!-- COMPONENTE RSSVIEWER -->
   <component>
      <component-type>RssViewer</component-type>
      <component-class>
         ec.edu.espol.fiec.componentes.jsf.rss.RssViewerUI
      </component-class>
   </component>
   ...
</faces-config>
```

Código Fuente 3.6: Declaración de un elemento <*component*> en un archivo Faces-Config.

3.4. Definición de clases básicas

A continuación se detallarán los elementos necesarios para la elaboración de un componente básico.

Manejador de la lógica del componente

Esta clase puede extender ya sea de *UIComponentBase*, para crear un componente desde cero, o puede extender de algún componente ya existente, como por ejemplo una etiqueta de texto. Es en esta clase donde reside la

lógica del componente, y como tal puede incluso contener el comportamiento para mostrarse a sí mismo al cliente, aunque para eso existe la clase *Render*. A esta clase se la denominará *claseUI*.

Manejador de la etiqueta del componente

Como lo indica su nombre, es la clase que se invocará cuando la etiqueta asociada a ella se encuentre en la página JSP. Algo importante es que esta clase es requerida para el uso en un entorno de desarrollo JSP, tomando en cuenta que un componente puede ser usado en otra clase de entornos. A esta clase se la llamará *claseTag*.

Presentador del componente

Es la clase encargada de contener el código para la representación del componente. Esta puede proveer diferentes representaciones para el componente. Como por ejemplo un hiperenlace o un botón. También permite generar respuestas distintas usando representaciones para clientes distintos como navegadores, PDAs, entre otros, que hayan hecho la petición. A esta clase se la llamará *claseRenderer*.

3.5. Desarrollo del componente Persona.

Antes de empezar a implementar componentes nuevos para JSF se decidió implementar un componente básico cuya estructura se tomará como referencia para el desarrollo de los componentes propuestos para la librería.

El componente a desarrollar será sencillo, el funcionamiento consistirá en mostrar en la página Web el nombre y el apellido ingresado previamente como parámetros del componente.

Diagrama de clase del componente

Se muestra el diagrama básico de clases que tiene el componente de ejemplo Persona que se ha desarrollado como prueba.

Figura 3.1: Diagrama de clase para el componente de prueba Persona

Desarrollo de la clase *ClaseUI*

A continuación se muestra el código perteneciente a la clase *PersonaUI* que servirá de ejemplo en la creación de un componente.

```
package ec.edu.espol.fiec.componentes.jsf.formbuilder;

import java.io.IOException;
import javax.faces.component.UIComponentBase;
import javax.faces.context.FacesContext;
import javax.faces.context.ResponseWriter;

public class PersonaUI extends UIComponentBase {

public void encodeEnd(FacesContext context) throws IOException {
      ResponseWriter writer = context.getResponseWriter();
      String apellido = (String)getAttributes().get("apellido");
      String nombre = (String)getAttributes().get("nombre");
      writer.startElement("b", this);
         writer.writeText("Nombre: ",null);
      writer.endElement("b");
      if(apellido != null && nombre!=null)
         writer.writeText(nombre,null);
      else
         writer.writeText("-",null);

      writer.startElement("br", this);
      writer.endElement("br");

      writer.startElement("b", this);
         writer.writeText("Apellido: ",null);
      writer.endElement("b");
      if(apellido != null && nombre!=null)
         writer.writeText(apellido,null);
      else
         writer.writeText("-",null);
   }
}
```

Código Fuente 3.7: Código perteneciente a la clase PersonaUI

Como se puede apreciar, se han escrito dos métodos: *encodeBegin()* y *getFamily()***.** Esto es requerido para una clase que extiende de la clase abstracta *UIComponentBase*, puesto que puede venir de cualquier familia de componentes.

Como referencia podemos decir que las clases del comportamiento o clases Renderer usan métodos cuyo nombre empieza con el prefijo *encode* para indicar que son métodos en los cuales va el modo de presentación al usuario.

El método *encodeBegin()* recibe una referencia del contexto de la aplicación de tipo FacesContext creado por el contenedor, a través del cual se tiene acceso a una gran cantidad de objetos útiles tanto de JSF, JSP y Servlets. Aquí sólo utilizamos el escritor (writer), que es el que nos permitirá construir la respuesta al cliente Web.

```
public void encodeBegin(FacesContext context) throws IOException {
ResponseWriter writer = context.getResponseWriter();
```

Código Fuente 3.8:Método encodeBegin

A continuación se obtiene los valores de los atributos *nombre* y *apellido* del Componente Persona que fueron enviados desde la etiqueta.

```
String apellido = (String)getAttributes().get("apellido");
String nombre = (String)getAttributes().get("nombre");
```

Código Fuente 3.9: Obtiene los valores de los atributos nombre y apellido del Componente Persona.

Luego se genera la representación de la información para el cliente en leguaje HTML.

```
writer.startElement("h3", this);
 if(apellido != null && nombre!=null){
    writer.writeText(nombre+" "+apellido,null);
 } else{
    writer.writeText("Los valores son null :(", null);
 }
 writer.endElement("h3");
```

Código Fuente 3.10: Generación del código HTML para el cliente.

Por último tenemos la sobre escritura del método *getFamily()*, dado que no se ha creado ninguna familia de componentes se retornará un simple String.

```
public String getFamily()  {
 return "LibreriaComponentes";
}
```

Código Fuente 3.11: Método getFamily()

Registro del componente en el archivo FacesConfig

Para que el componente pueda ser utilizado, hay que registrarlo en el archivo faces-config.xml. En las líneas siguientes se presenta el código necesario para el registro del componente.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC
 "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
 "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
 <component>
     <component-type>ComponentePersona</component-type>
        <component-class>
            ec.edu.espol.fiec.componentes.jsf.formbuilder.PersonaUI
        </component-class>
  </component>
</faces-config>
```

Código Fuente 3.12: código necesario para el registro del componente

Con la etiqueta <*component*> se registra el componente, dentro se usan dos etiquetas que son: <*component-type*> para indicar el nombre. Tiene que ser el mismo que se encuentra registrado en el archivo TLD que veremos más adelante, y <*component-class*> en donde se especifica la clase que maneja el componente *PersonaUI*.

Después de haber registrado la clase del componente en el archivo de configuración faces-config.xml el enlace entre los dos archivos se vería como lo muestra la siguiente figura.

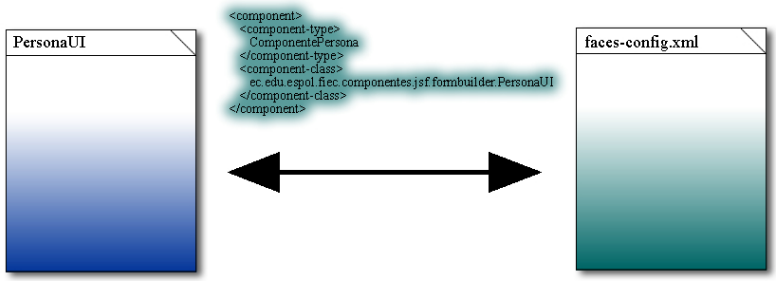


Figura 3.2: Relación entre la clase *PersonaUI* y el archivo de configuración faces-config.xml

Desarrollo de la *claseTag*

Para implementar la clase encargada de la etiqueta del componente se extiende de la clase *UIComponentTag*. A continuación se expone las líneas de código de esta clase.

```
...
public class PersonaUITag extends UIComponentTag {
   public String getComponentType() {
      return "ComponentePersona";
   }
   public String getRendererType() {
      return null;
   }
   protected void setProperties(UIComponent component) {
      super.setProperties(component);
      if (this.getApellido() != null) {
         if (isValueReference(getApellido())) {
            FacesContext context = FacesContext.getCurrentInstance();
            Application app = context.getApplication();
            ValueBinding vb = app.createValueBinding(this.getApellido());
            component.setValueBinding("apellido", vb);
         } else
            component.getAttributes().put("apellido", this.getApellido());
      }
      if(this.getNombre() !=null)
      {
         if(isValueReference(this.getNombre()))
         {
            FacesContext context = FacesContext.getCurrentInstance();
            Application app = context.getApplication();
            ValueBinding vb = app.createValueBinding(this.getNombre());
            component.setValueBinding("nombre", vb);
         }
         else
            component.getAttributes().put("nombre", this.getNombre());
      }
   }
   public void release() {
      super.release();
      setApellido(null);
      setNombre(null);
   }
```

Código Fuente 3.13: Código ClaseTag

```
    //Propiedades del Componente..
    private String apellido = null;
    private String nombre = null;
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public String getNombre() {
        return nombre;
    }
}
```

Código Fuente 3.14: Propiedades de la clase ClaseTag.

Los principales propósitos de esta clase son:

- Asociar una etiqueta de una página JSP a una *claseUI*.
- Asociar una clase renderer, si se necesita, a la *claseUI*.
- Establecer los valores de los atributos de la etiqueta enviada por el cliente.

Lo primero que podemos notar en una vista rápida, es el comportamiento de un bean; es decir métodos *setters* y *getters* dentro de la clase al final de la misma, cuyos atributos son apellido y nombre.

Se puede visualizar dos métodos que relacionan a la etiqueta con el componente *PersonaUI*, creado anteriormente, el método

getComponentType() hace referencia a la línea registrada bajo el elemento <*component-type*> en el archivo faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>
...
<faces-config>
  <component>
      <component-type>
ComponentePersona
      </component-type>
...
   </component>
</faces-config>
```

```
public String getComponentType() {
      return "ComponentePersona";
   }
```

Código Fuente 3.15: Declaración del componente en archivo faces-config.xml.

Y el método *getRendererType()***,** retorna la clase renderer del componente, pero en este caso se retorna *null* debido a que el comportamiento de la presentación para el cliente fue incluida en la misma clase del componente y por lo tanto no se utilizó una clase renderer.

En este fragmento de código tenemos el método *setProperties()*, donde primeramente se invoca al método del mismo nombre de la clase padre para la asignación de los valores a los atributos pertenecientes a la etiqueta del componente en la página JSP.

Nótese que lo primero que se hace con los atributos es conocer si está referenciado, en donde dichas referencias toman la forma de expresiones EL

de JSF ((Ej, #{Bean.Propiedad}). La forma para obtener el valor del atributo es diferente a la que se usaría si éste no estuviera referenciado.

Código Fuente 3.16: Método setProperties() del componente Persona.

Y por último tenemos al método *release()*, que es el encargado de liberar los recursos asignados a los atributos y regresarlos al estado "no usado".

Código Fuente 3.17: Método release del componente Persona.

Desarrollo del archivo descriptor de la librería de etiquetas

Como se describió anteriormente, el archivo TLD contiene las etiquetas en donde se relaciona al componente con su respectiva clase de etiqueta.

En ANEXO E se muestra el contenido del archivo TLD para el componente de prueba.

Luego de haber desarrollado la clase de manejo de la etiqueta del componente y haber creado la descripción de la etiqueta con sus respectivos atributos en el archivo TLD, la conexión entre archivos se vería de la siguiente manera.

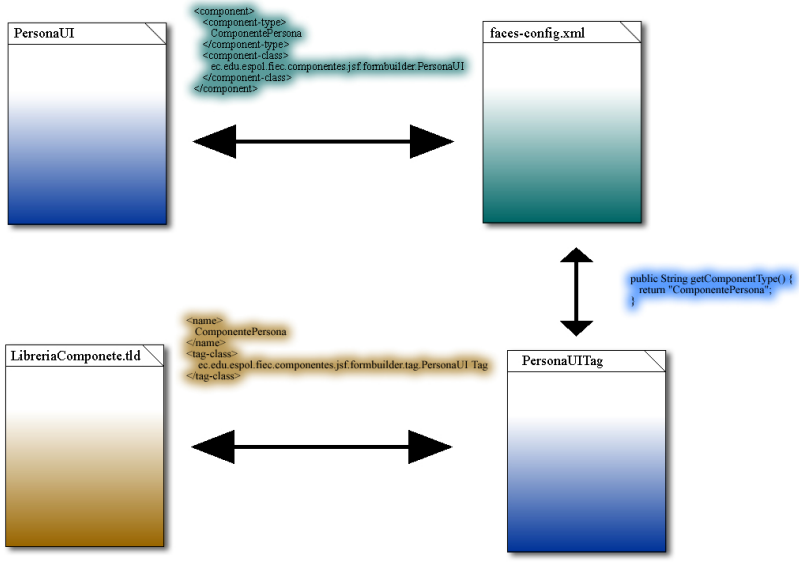


Figura 3.3: Relación entre la clase del comportamiento, faces-config-xml, clase de manejo de la etiqueta y el archivo TLD.

Declaración del componente en página JSP

Para declarar un componente en una página JSP es necesario primero especificar el espacio de nombre con el cual vamos a hacer referencia al componente dentro de la página. El espacio de nombre puede declararse utilizando la directiva <*taglib*> o como atributo de la etiqueta <*jsp:root*>.

```
<%@ taglib lb=" http://fiec.espol.edu.ec/jsf/componentes" %>
```

```
<jsp:root version="1.2"   xmlns:lb="http://fiec.espol.edu.ec/jsf/componentes" />
```

Código Fuente 3.18: Declaración de uso de la librería en una página Web.

Para luego declarar el componente con su respectiva etiqueta.

```
<lb:ComponentePersona nombre="Juan" apellido="Bravo"/>
```

Código Fuente 3.19: Declaración del componente en la página Web.

Dependencias entre clases y archivos de configuración

A continuación se presenta una representación grafica de cómo están relacionados todos los archivos que intervienen en el componente de ejemplo Persona

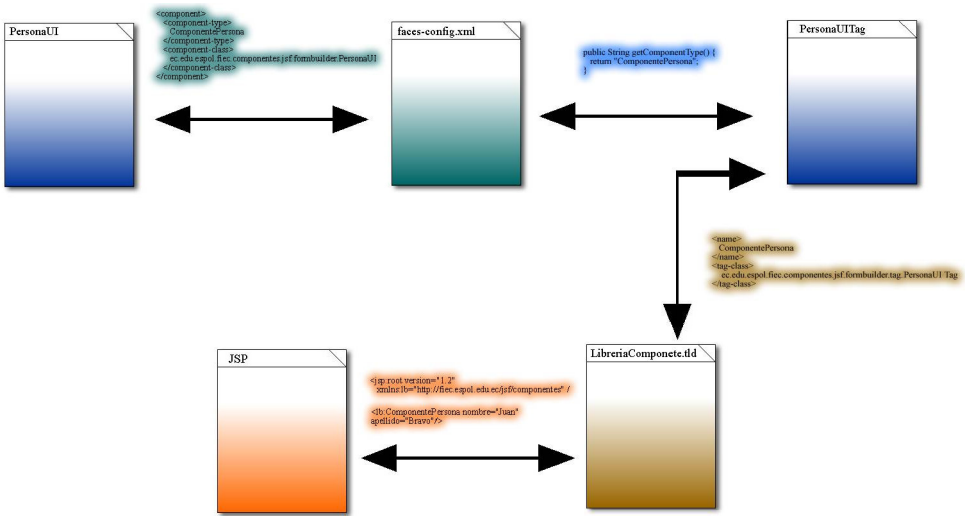


Figura 3.4: Diagrama de conexión entre los archivos que manejan un componente.

Vista del componente en navegador Web

En la figura que se muestra a continuación se puede apreciar como se ve el componente en un navegador.

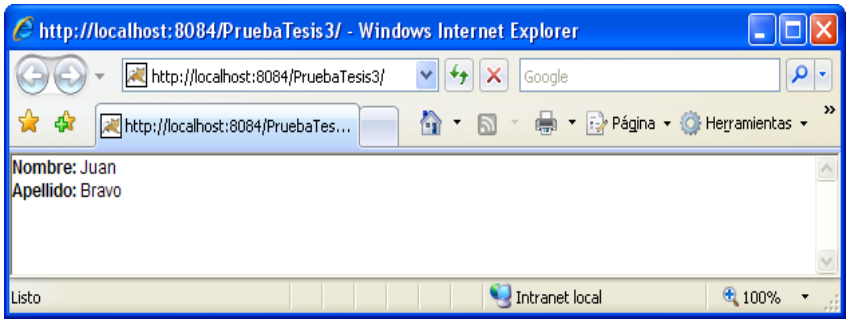


Figura 3.5: Vista del componente persona en navegador Web

CAPÍTULO 4

4. Componentes desarrollados

4.1. Componente RssViewer

En el año de 1999 Netscape introdujo al mercado el formato RSS 0.9, donde el objetivo era crear una plataforma y vocabulario basado en RDF para poder sindicar datos en su portal y su navegador, desafortunadamente por varias razones se decidió no continuar con el proyecto RSS; lo cual dio origen a la aparición de varios formatos.

RSS es parte de la familia de los vocabularios basados en XML desarrollado específicamente para sitios que se actualicen con frecuencia y por medio del cual se puede compartir la información y usarla en otros sitios web o en aplicaciones de escritorio. A esto se le conoce como redifusión o Sindicación web (5).

Tomando en cuenta el gran número de sitios en la Web que están ofreciendo el servicio de sindicación, surgió la idea de crear un componente dedicado específicamente a la lectura de documentos de noticias también llamados feeds.

El componente RssViewer presenta, en la página Web, noticias pertenecientes a un canal de noticias. Estas son provistas por un servidor que se procesan para darle formato utilizando un documento CSS especificado por el desarrollador, la fuente de la noticia puede ser cambiada por el desarrollador en tiempo de diseño o de ejecución.

Declaración de la etiqueta en la página Web.

A continuación se muestra la forma que tiene la etiqueta del componente (esta etiqueta fue autogenerada por el IDE).

El componente RssViewer tiene dos atributos que son básicos:

- *url*: este atributo permite especificar el canal de noticias que se utilizará como fuente.
- *numeroNoticias:* que indica el número de noticias que se presentaran en la página.

```
<fiec:RssViewer binding="#{Rss1.RssViewer1}" id="RssViewer1"
numeroNoticias="3"
style=" height: 214px; left: 120px; top: 72px; position: absolute; width: 214px "
url="http://www.eluniverso.com/rss/portada.xml"/>
```

Atributos del componente.

En la Tabla VI se describe en detalle cada una de las propiedades disponibles para el componente RssViewer y el tipo de datos esperado para cada una de ellas.

Nombre	Descripción	Tipo de dato	Requerido
Id	EL nombre único del componente que lo identifica.	String	Sí
altText	Texto descriptivo del componente.	String	No
numeroNoticias	EL número de noticias a ser presentado.	Integer	Sí
url	La dirección del canal de Noticias.	String	Sí
height	Alto que va a tener el componente en la presentación.	Integer	No
nombreArchivoCss	Ruta de archivo de estilo, en caso de no querer usar el predeterminado.	String	No
style	Estilo tipo inline que es usado por el IDE para la ubicación del componente.	String	No
width	Ancho que va a tener componente en la presentación.	Integer	No

Tabla VI: Atributos del componente RssViewer.

Vista del componente en la página Web

Las noticias, como se había explicado antes, se reciben en formato RSS, es decir, texto incluido dentro de tags. Utilizando la hoja de estilos por defecto o la provista por el desarrollador, es posible dar formato a ese texto y mostrarlo al usuario de forma que le sea sencillo visualizar las noticias provenientes del feed.

En la Figura 4.1 se puede observar como se ve el componente en la página Web, aplicando el estilo contenido en un archivo CSS provisto por el desarrollador. El archivo CSS define la ubicación del logo y el formato de cada noticia.

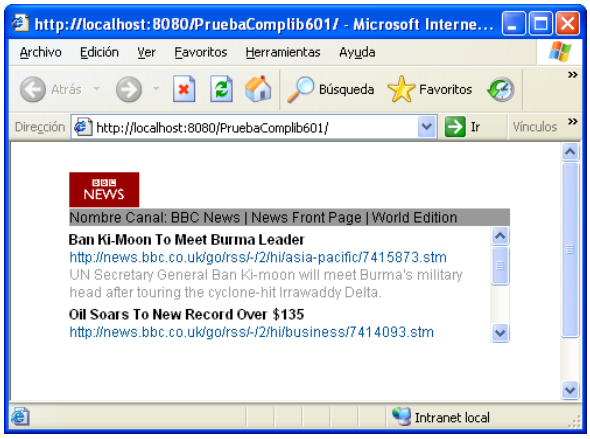


Figura 4.1 Vista final del componente RssViewer.

Código HTML y CSS del componente.

Para poder presentar al cliente (navegador) el componente, se estableció una estructura a la cual se le puede aplicar estilos (CSS). En cada etiqueta se establece un nombre único y estático para la respectiva aplicación del estilo. A continuación se presenta la estructura.

Para que el diseñador implemente una hoja de estilo que pueda ser aplicada al componente bastará con que utilice las clases mostradas en la figura Figura 4.2. Por ejemplo si se le quiere dar estilo l título de la noticia, tiene que sobrescribir la clase *tituloGeneralClass*.

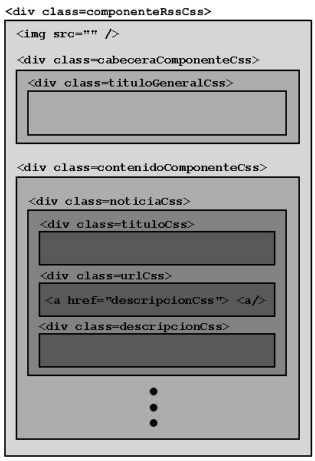


Figura 4.2: Estructura HTML y sus respectivos nombres para la clase de estilo del componente Rss Viewer.

Diagrama de clases del componente

El siguiente es el diagrama de clases interno, con las cuales se maneja la información que se presenta por medio del componente.

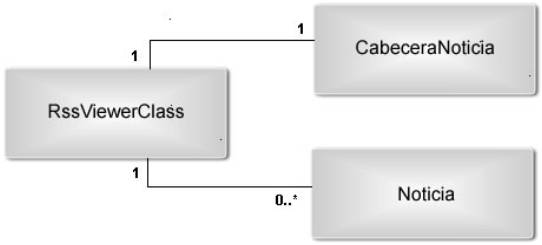


Figura 4.3: Diagrama de clase para el manejo de información de las noticias RSS.

La clase principal es la *RSSViewerClass* que es la encargada de proveer al componente la cabecera y la lista de noticias, las cuales están representadas por las clases *CabeceraNoticia* y Noticia respectivamente.

Clases para el manejo de las noticias RSS

Debido a que para la sindicación se han desarrollado tres estándares hasta el momento, se ha hecho uso de una librería de código abierto que es capaz de reconocer y leer estos estándares. Esta librería que es resultado de un proyecto, se la puede conseguir en http://java.sun.com/developer/technicalArticles/javaserverpages/rss_utilities/ . Dado que la implementación y manejo de RSS no era el objetivo de este trabajo, se consideró oportuno utilizar esta librería de código abierto.

En esta clase se manejan tres objetos básicos para la lectura de las noticias:

- RssParser: que se encarga de las conversiones del formato XML
- Rss: el que contiene toda la información.
- Channel: Es el canal donde están publicadas las noticias.

Estos tres objetos son inicializados en el constructor.

Uno de los métodos permite recuperar la información básica del canal, este devuelve la información en un objeto *CabeceraNoticia*, donde se encuentra el titulo, url, descripción, imagen, fecha de publicación, todo esto correspondiente al canal. También está el método que me permite obtener las noticias como objeto Noticia donde están el titulo, una pequeña descripción, el enlace y fecha de publicación de cada noticia.

Explicación y desarrollo de la clase de presentación

Dentro de esta clase se llevan a cabo algunas verificaciones importantes, en la primera se verifica el atributo *url* que se pasó como parámetro al componente no esté vacío. Usando este url, se forma la clase *RssViewerClass*, dentro de la cual se verifica si existe la cabecera con los datos mencionados anteriormente, luego se procesa la lista de noticias que contiene la clase, y finalmente se escribe al cliente (navegador) todos los datos obtenidos mediante el objeto *ResponseWriter*, dentro del método *escribirNoticias()*.

4.2. Componente TiraImagenes

En los últimos tiempos los sitios y aplicaciones web para presentación de imágenes y video se han venido multiplicando especialmente en los sitios conocidos como redes sociales, como: *www.hi5.com*, *www.facebook.com*, *www.myspace.com*, entre otros. Es por esto que surgió la idea de crear un componente que sea capaz de presentar las imágenes en forma horizontal, en donde se pueda apreciar un número de imágenes previas y una imagen central en tamaño original.

En este componente se hace uso de AJAX para las peticiones asincrónicas, manipulación de DOM para la modificación visual en el navegador y JSON para enviar y recibir información entre el Servlet controlador y el componente.

Declaración de la etiqueta en la página Web.

A continuación se muestra la forma que tiene la etiqueta del componente, que fue autogenerada por el IDE. Los atributos dos básicos del componente Tira de Imágenes son :

- *listaImagenes*: Donde se especifica la lista de direcciones de las imágenes
- *numeroImagenesEnTira*: El número de imágenes previas que van a ser presentados.

```
<fiec:TiraImagenes binding="#{Page6.tiraImagenes04}" id="tiraImagenes04"
listaImagenes="#{tiraImagenesBeanUno.lista}" numeroImagenesEnTira="3"/>
```

Atributos del componente.

En la **¡Error! No se encuentra el origen de la referencia. ¡Error! No se encuentra el origen de la referencia.**se describe en detalle cada uno de los atributos con su respectivo tipo de dato.

Nombre	Descripción	Tipo de dato	Requerido
id	EL nombre único del componente que lo identifica.	String	Si
listaImagenes	Lista de las rutas de las imágenes a presentarse.	List<String>	Si
numeroImagenesEnTira	El número de imágenes previas a presentarse en la parte superior.	Integer	Si
style	Estilo tipo inline que es usado por el IDE para la ubicación del componente.	String	No

Tabla VII: Atributos del componente Tira de Imágenes.

Vista del componente en la página Web.

En la Figura 4.4 se puede observar como se ve, para el usuario, el componente dentro de una página web. En este caso son 3 figuras pequeñas en la parte superior y la principal en la parte inferior, de tamaño natural.

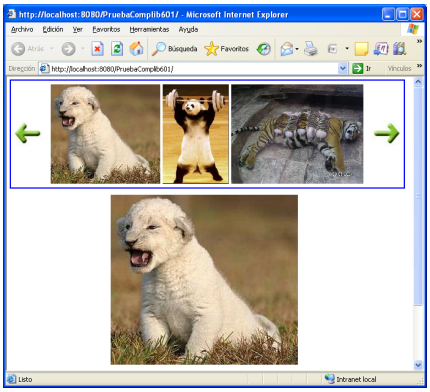


Figura 4.4: Vista final del componente Tira de Imágenes.

Estructura del componente en la página Web

Para poder presentar al cliente (navegador) el componente, se estableció una estructura a la cual se le puede aplicar estilos (CSS). En cada etiqueta se establece un nombre único y estático para la respectiva aplicación del estilo. A continuación se presenta la estructura.

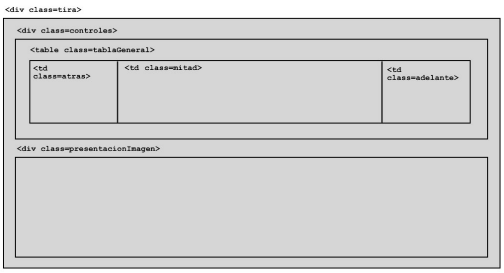


Figura 4.5: Estructura HTML y sus respectivos nombres para la clase de estilo del componente Tira de Imágenes.

Diagrama de clases para manejo interno del componente

Para el componente TiraImagenes se implementaron solamente dos clases. La clase *ParametrosTiraImagenes* se manejan los parámetros de las imágenes como la lista de ubicaciones de las imágenes, lista de id de las imágenes previas, número de imágenes por presentación, número de página y el id del componente.

Figura 4.6: para el manejo de parámetros del componente Tira de Imágenes.

La clase *ControladorTiraImagen* es el Servlet que procesará las peticiones AJAX por el componente desde el cliente (navegador Web).

Figura 4.7: para el control de imágenes del componente Tira de Imágenes.

Clases para el manejo de la tira de imágenes.

En el ANEXO F se muestra el código implementado para las dos acciones principales dentro del Servlet *ControladorTiraImagen* que son: presentar las imágenes siguientes y las imágenes anteriores.

Las acciones manejadas por el Servlet se ejecutan como respuesta a los eventos generados por el usuario al interactuar con la tira de imágenes desde el navegador.

Explicación y desarrollo de la clase de presentación

En la implementación de la clase de presentación se procesa el objeto instanciado de la clase *ParametrosTiraImagenes* en donde se tiene la información correspondiente a las imágenes.

Dentro de esta clase de presentación se obtiene como primer dato, la lista y el número de imágenes que se quiere presentar en tamaño reducido. En esta clase también se recupera el valor de variables claves como: numero de página y sublista por página, para el manejo de las acciones de presentación de imágenes siguientes y anteriores. La sublista son las imágenes correspondientes a una página determinada, por ejemplo, si existen 20 imágenes y se quiere presentar 5 simultáneamente, se tiene 4 páginas.

4.3. Componente Exportación de datos en XML a PDF

En la actualidad el uso de XML como lenguaje de intercambio de datos ha sido usado ampliamente en todos los campos de la computación, lo que lo ha hecho extremadamente popular y que tenga soporte en la mayoría de plataformas. Los archivos PDF también han ganado popularidad desde su aparición, muchos sistemas proveen la opción de exportar la información en este formato.

Con lo expuesto en el párrafo anterior, se vio la oportunidad de crear un componente que permita exportar información al formato PDF, lo cual en realidad es una aplicación de presentación de datos en un archivo XML por medio de un archivo XSL-FO.

Declaración de la etiqueta en la página Web.

De los atributos del componente ConvertidorPdf que se pueden apreciar, los 4 básicos son:

- *archivoXml*: es el nombre del archivo XML con su respectiva ruta dentro de la aplicación Web.
- *archivoXslfo:* es el nombre del archivo XSL-FO con su respectiva ruta dentro de la aplicación Web.
- *rutaDestino*: es la ruta completa en donde se va a colocar el archivo convertido.
- *nombreArchivoConvertido*: es el nombre del archivo que va a ser convertido.

```
<fiec:ConvertidorPdf  id="ConvertidorPDF1"  binding="#{pdf2.ConvertidorPDF1}"
archivoXml="nombreArchivoXML.xml"  archivoXslfo="nombreArchivoXslfo.xsl"
rutaDestino="archivosPdf" nombreArchivoConvertido="nombreArchivoConvertido.pdf"
style="height: 22px; left: 72px; top: 24px; position: absolute; width: 240px"/>
```

Atributos del componente.

En la Tabla VIII se describe en detalle cada uno de los atributos con su respectivo tipo de dato.

Nombre	Descripción	Tipo de dato	Requerido
id	El nombre único del componente que lo identifica.	String	Si
archivoXml	Ruta y nombre del arhivo xml.	String	Si
archivoXslfo	Ruta y nombre del archivo xslfo para conversión.	String	Si
rutaDestino	Ruta destino en la cual se creará en archivo convertido.	String	Si
nombreArchivoConvertido	Nombre que tendrá en archivo a convertir.	String	Si
style	Estilo tipo inline que es usado por el IDE para la ubicación del componente.	String	No

Tabla VIII: Atributos del componente Convertidor de XML a PDF.

Vista del componente en la página Web.

En la Figura 4.8 se muestra como se presenta el componente al usuario, es un simple enlace hacia el archivo convertido, pero la mayor parte del trabajo se lo hace internamente.

Se puede apreciar que cuando se pone el curso sobre el enlace, en la barra de estado aparece la dirección completa del enlace, que consta del nombre que el programador le da, mas el id de la sesión creada para el usuario navegador.

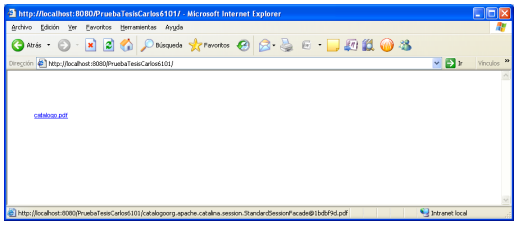


Figura 4.8: Vista final del componente Convertidor de XML a PDF.

Estructura del componente en la página Web

A continuación se presenta la estructura del componente ConvertidorPdf, en el cual la etiqueta principal es <a /> la cual representa un enlace, que está dentro de un <*div*> para que se le pueda aplicar un estilo personalizado.

```
<div class=enlaceArchivoPdf>
<a src='http://archivoConvertido.pdf'>
```

Figura 4.9: Estructura HTML y sus respectivos nombres para la clase de estilo del componente Convertidor de XML a PDF.

Diagrama de clases para manejo interno del componente

Para este componente se ha implementado una clase principal llamada ConvertidorPdf, que está encargada de verificar las rutas y existencia de los archivos, para después crear el archivo PDF mediante la aplicación del archivo XSL-FO al archivo XML.

Figura 4.10: Clase para el manejo de la conversión de archivos XML a PDF.

Clases para el manejo de la Exportación de datos en XML a PDF

Ya que el entendimiento y estructura de un archivo XML y un archivo XSL-FO no es objetivo de este trabajo, se ha hecho uso de una librería de código abierto que se la puede conseguir en http://xmlgraphics.apache.org/fop/download.html.

En esta clase se manejan dos métodos:

- procesarConversion: método público que toma como argumentos los datos pasados como atributos del componente y se hacen las verificaciones de datos ingresados.
- convertirArchivos: método privado, usado en el nombrado anterior a este, que aplica la conversión haciendo uso de las clases de la librería de código abierto.

Explicación y desarrollo de la clase de presentación

Debido a que este componente se presenta al usuario como un simple enlace al archivo PDF creado, las condiciones que se toman en cuenta es que todos los atributos necesarios estén con datos ingresados, para luego crear la etiqueta <*a*> en donde su atributo *href* contiene la dirección al archivo PDF.

4.4. Componente de video

Este componente se encarga de reproducir un video desde una página web. Posee varias implementaciones para cubrir los diferentes formatos propietarios actuales. Presenta algunos retos tecnológicos ya que involucra

interacción entre el servidor, el cliente y hardware del encargado de procesar el video y mostrarlo.

A continuación se describe el papel de cada elemento en el proceso de reproducción del video.

- **Servidor:** En éste se encuentra alojado el video a ser reproducido, No debe confundirse con el servidor donde está alojada la aplicación Web ya que puede ser un servidor dedicado exclusivamente para este fin o en su defecto el video puede estar incluido dentro de la misma aplicación web. Esto brinda la facilidad de separar totalmente la aplicación del servidor de archivos de video. Puede incluir seguridades especiales para proteger el contenido y que sólo sea accedido desde la aplicación.
- **Cliente:** Este se encarga de reproducir el video e interactúa con el hardware de su propia estación. Debe almacenar temporalmente los segmentos de video a medida que se van descargando y reproducirlos, ya que sería poco eficiente descargar primero todo el video para luego reproducirlo.

Para superar estas barreras tecnológicas se aprovechó de la capacidad de los navegadores web para descargar plug-ins especiales para procesar video. El componente interactúa directamente con el navegador y le ordena descargarse el plug-in si no lo detecta instalado.

El componente recibe una dirección de un servidor donde está publicado el video que va a ser procesado, esta puede ser una dirección web o de un servidor de archivos destinado para este propósito.

Declaración del componente dentro de una página web

A continuación se muestra la forma que tiene la etiqueta del componente (esta etiqueta fue autogenerada por el IDE).

El componente tiene dos atributos que son básicos:

- *implementación*: este atributo permite especificar el tipo de reproductor a utilizar.
- *url:* este atributo indica la dirección en el servidor del video a reproducir.

```
<fiec:VideoPlayer binding="#{TestVideoPlayer.videoPlayer1}" height="360"
id="videoPlayer1" implementacion="divx" style="left: 72px; top: 120px; position:
absolute"  url="http://host:8080/ServidorMultimedia /test.mpg" width="360"/>
```

Atributos del componente.

Atributos	Tipo	Descripción	Requerido
General			
autoStart	Boolean	Indica si el video debe ser tocado sin la acción específica del usuario.	SI
ShowControls	Boolean	Indica si debe presentar los controles al usuario.	SI
showPositionControls	Boolean	Indica si se debe presentar la barra de tiempo del video.	SI
showStatusBar	Boolean	Indica si se debe presentar la barra de estado.	SI
url	String	Es la dirección completa del video a ser presentado en la página.	SI
Apareance			

displaySize	Integer	---	SI
height	Integer	La altura del componente presentado.	SI
implementacion	String	Especifica el tipo de reproductor a utilizar. Puede tomar los valores indicados en la tabla de implementaciones.	SI
width	Integer	Ancho del componente.	SI
style	String	Estilo del componente.	SI
Advanced			
rendered	Boolean	Indica si el componente debe ser presentado.	SI

Tabla IX: Atributos del componente de Video

El componente puede reproducir los siguientes formatos bajo diferentes implementaciones:

Reproductor	Parámetro implementación	Formatos
Windows Media Player	WM	*.wmv, *.mpeg, *.mpg
Real Player	RP	*.ra, *.ram
Quick Time	QT	*.qt
Divx Player	DIVX	*.divx

Tabla X: Implementaciones disponibles

Vista del componente en la página Web

En la Figura 4.11 se puede observar cómo se presenta el componente en la página Web, con la implementación Windows Media Player.

Figura 4.11: Vista del componente de Video en una aplicación de prueba.

Aspectos de diseño:

En el paquete *ec.edu.espol.fiec.componentes.jsf.videoplayer.impl* se encuentran las clases que corresponden a las implementaciones de

reproductores donde cada una posee el código específico estándar para que el navegador efectúe el llamado al plug-in para realizar la reproducción.

Cada implementación contiene un método encargado de realizar la presentación del componente y este varía según el reproductor.

```
public void buildPlayer(ResponseWriter writer,UIComponent component) throws IOException
```

Para el caso del Windows Media Player se desea que el navegador interprete el siguiente código el cual crea un objeto donde tiene la referencia del plug-in y la dirección para ser descargado en caso de ser necesario.

```
<div style="left: 288px; top: 144px; position: absolute">
    <object id="_id1" class="mediaPlayer" height="304" width="304"
classid="CLSID:22D6f312-B0F6-11D0-94AB-0080C74C7E95" standby="Loading
Windows Media Player components..." type="application/x-oleobject"
codebase="http://activex.microsoft.com/activex/controls/mplayer/en/nsmp2inf.cab#Ve
rsion=6,4,7,1112" name="wmp">
        <param name="FileName"
value="http://localhost:8084/ServidorMultimedia/Videos/Angra.mpg" />
        <param name="AutoStart" value="true" />
        <param name="ShowControls" value="true" />
        <param name="ShowStatusBar" value="true" />
        <param name="ShowPositionControls" value="true" />
        <param name="DisplaySize" value="4" />
        <embed type="application/x-mplayer2"
pluginspage="http://www.microsoft.com/netshow/download/player.htm"
swliveconnect="true" name="wmp" id="_id1" height="304" width="304"
AutoStart="true" ShowControls="true" ShowPositionControls="true"
DisplaySize="4" src="http://localhost:8084/ServidorMultimedia/Videos/Angra.mpg">
    </embed>
  </object>
</div>
```

Código Fuente 4.1: Objeto HTML creado para el componente de Video.

La clase *VideoPlayerUIRenderer* es la encargada de elegir el tipo de reproductor en base al parámetro de implementación.

```
   @Override
   public void encodeEnd(FacesContext context, UIComponent component) throws
IOException {
      ResponseWriter writer = context.getResponseWriter();
      VideoPlayerUI player = (VideoPlayerUI) component;
      String impl = null;
      if (player != null) {
         impl = player.getImplementacion();
      }
      //Verifica el tipo de implementación configurado, según el mismo
      //se llama al método especifico de render.
      if (impl == null) {
         new WindowsMediaPlayer().buildPlayer(writer, component);
      } else if (impl.equalsIgnoreCase(VideoPlayerAtributes.WINDOWS_MEDIA_PLAYER)) {
         new WindowsMediaPlayer().buildPlayer(writer, component);
      } else if (impl.equalsIgnoreCase(VideoPlayerAtributes.REAL_PLAYER)) {
         new RealPlayer().buildPlayer(writer, component);
      } else if (impl.equalsIgnoreCase(VideoPlayerAtributes.QUIICK_TIME_PLAYER)) {
         new QuickTimePlayer().buildPlayer(writer, component);
      } else if (impl.equalsIgnoreCase(VideoPlayerAtributes.DIVX_PLAYER)) {
         new DivXPlayer().buildPlayer(writer, component);
      } else {
         new WindowsMediaPlayer().buildPlayer(writer, component);
      }
   }
```

Código Fuente 4.2: Codificación de la clase VideoPlayerUIRender.

Este componente no tiene asociado un diagrama de clases relevante puesto que su implementación lo realiza la clase de presentación.

4.5. Componente Reproductor MP3

Este componente se encarga de reproducir audio en formato mp3, Posee las mismas barreras tecnológicas que el Componente de video pero en este caso serán sorteadas con un enfoque diferente. Este componente utiliza un Applet firmado digitalmente para interactuar con el cliente. La firma digital es estrictamente necesaria para que el Applet tenga los permisos necesarios de ejecución en el equipo cliente.

En este caso se utilizó una librería de código abierto (LGPL) llamada jlGui que reproduce archivos de audio en formatos WAV, AU, AIFF, MP3 bajo plataforma Java y con ciertas modificaciones en su estructura, con especial énfasis en su portabilidad se lo encapsuló en el componente; Se crearon directorios especiales que deben estar bajo el mismo directorio padre donde se ubicaron los diferentes archivos necesarios para el funcionamiento del Applet que se describen a continuación.

- **Carpeta de configuración (/conf).** Contiene el archivo de configuración jlgui.ini que define algunos parámetros propios de la librería jlGui. Se detalla un ejemplo a continuación.

```
allowed_extensions=m3u,pls,wsz,snd,aifc,aif,wav,au,mp1,mp2,mp3,ogg
equalizer_auto=false
equalizer_enabled=true
equalizer_on=true
last_dir=
last_equalizer=50,32,32,50,50,50,50,50,50,32,32
last_playlist=
last_skin=
last_url=h
origine_x=0
origine_y=0
playlist_enabled=true
playlist_impl=javazoom.jlgui.player.amp.playlist.BasePlaylist
proxy_login=
proxy_password=
proxy_port=-1
proxy_server=
repeat_enabled=true
screen_limit=true
shuffle_enabled=false
taginfo_mpeg_impl=javazoom.jlgui.player.amp.tag.MpegInfoApplet
taginfo_oggvorbis_impl=javazoom.jlgui.player.amp.tag.OggVorbisInfoApplet
```

Código Fuente 4.3: Ejemplo de archivo de configuración jlgui.ini

- **Carpeta de librerías (/lib)** Contiene los archivos de extensión jar que necesita el Applet para funcionar correctamente y ser descargados por el cliente. Se encuentran también los archivos necesarios para firmar los archivos jar del Applet. Se proporciona una firma por defecto pero puede ser reemplazada por el programador.
- **Carpeta de listas de reproducción (/listas)** En esta carpeta se ponen las listas de reproducción para ser leídas por el componente, ya que deben ser publicadas.
- **Carpeta skins. (/skins)** En esta carpeta se colocan los skins (Máscaras de interfaz) para el Applet que son los mismos que utiliza Winamp v2.0

La siguiente es una vista de la estructura que debe tener la aplicación Web incluida la carpeta del componente.

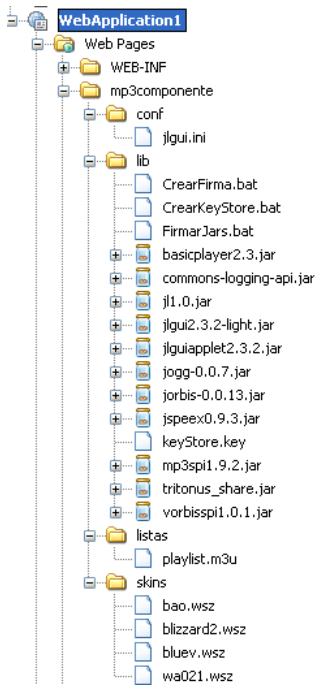


Figura 4.12: Estructura de Aplicación Web con el componente incluido.

Declaración del componente dentro de una página web

A continuación se muestra la forma que tiene la etiqueta del componente (esta etiqueta fue autogenerada por el IDE).

El componente tiene dos atributos que son básicos:

- *ubicacion*: este atributo permite especificar la ruta en la que están ubicadas las librerías.
- *playlist:* este atributo indica el nombre de la lista de reproducción que se encuentra en la carpeta de listas de reproducción (/listas).

```
<fiec:Mp3Player binding="#{TestMp3Player.mp3Player1}" id="mp3Player1" playlist="playlist.m3u" style="height: 430px; left: 120px; top: 24px; position: absolute; width: 286px" ubicacion="mp3componente"/>
```

Atributos del componente

	Tipo	Descripción	Requerido
General			
Skin	String	El nombre del archivo de skin a utilizar.	NO
Appearance			
playlist	String	El nombre de la lista publicada en formato m3u.	SI
Style	String	El estilo del componente	SI
ubicacion	String	La ubicación de la carpeta principal del componente que debe estar en la misma aplicación Web.	SI
Advanced			
Render	Boolean	Indica si en componente es presentado o no.	SI

Tabla XI: Atributos del componente reproductor de MP3.

Vista del componente en la página Web.

En la Figura 4.13 se puede observar cómo se presenta el componente en la página Web.

Figura 4.13: Vista del componente Reproductor de MP3 en una aplicación de prueba.

Aspectos de diseño

El componente construye un objeto HTML que es interpretado por el navegador para que éste ejecute un llamado a un Applet de Java.

```
<div style="left: 288px; top: 144px; position: absolute">
    <object id="_id1" class="mediaPlayer" height="304" width="304"
classid="CLSID:22D6f312-B0F6-11D0-94AB-0080C74C7E95" standby="Loading
Windows Media Player components..." type="application/x-oleobject"
codebase="http://activex.microsoft.com/activex/controls/mplayer/en/nsmp2inf.cab#Versi
on=6,4,7,1112" name="wmp">
        <param name="FileName"
value="http://localhost:8084/ServidorMultimedia/Videos/Angra.mpg" />
        <param name="AutoStart" value="true" />
        <param name="ShowControls" value="true" />
        <param name="ShowStatusBar" value="true" />
        <param name="ShowPositionControls" value="true" />
        <param name="DisplaySize" value="4" />
        <embed type="application/x-mplayer2"
pluginspage="http://www.microsoft.com/netshow/download/player.htm"
swliveconnect="true" name="wmp" id="_id1" height="304" width="304"
AutoStart="true" ShowControls="true" ShowPositionControls="true" DisplaySize="4"
src="http://localhost:8084/ServidorMultimedia/Videos/Angra.mpg">
        </embed>
    </object>
 </div>
```

Código Fuente 4.4: Objeto HTML generado para ejecutar el Applet del componente.

Al abrir la página con el componente aparecerá la siguiente ventana que indica que la firma no puede ser verificada ya que es una firma digital de prueba.

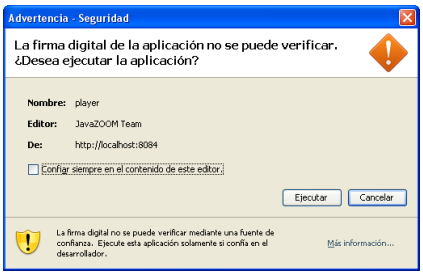


Figura 4.14: Advertencia de seguridad al ejecutar el Applet.

Este componente no tiene asociado un diagrama de clases relevante puesto que su implementación lo realiza la clase de presentación.

4.6. Componente de Formulario dinámico.

Este componente construye un formulario de ingreso de información a partir de un objeto proporcionado por el desarrollador. Los datos ingresados por el usuario en el formulario son asignados automáticamente al objeto y devueltos para ser procesados posteriormente por el desarrollador y aplicar la lógica respectiva.

Declaración del componente dentro de una página web

A continuación se muestra la forma que tiene la etiqueta del componente (esta etiqueta fue autogenerada por el IDE). Su atributo principal es:

- *objetoBean*: este atributo especifica la referencia al objeto en lenguaje EL.

```
<fiec:FormBuilder bgcolor="lightgray" binding="#{Page1.formBuilder1}" border="1"
columns="2" id="formBuilder1" objetoBean="#{SessionBean1.formulario}"
style="height: 120px; width: 336px" width="312"/>
```

Atributos del componente.

	Tipo	Descripción	Requerido
General			
objetoBean	String	La referencia al objeto para crear el formulario, Esta debe extender de FormBuilderWrapper	SI
Appearance			
Style	String	El estilo del componente	SI
bgcolor	String	Define el color de fondo del formulario. Ej green, blue, lightgray etc.	NO
border	Int	Valor que representa el ancho del borde que rodea el formulario	NO
columns	Int	El número de columnas del formulario	NO
Advanced			
Render	Boolean	Determina si el componente es presentado no.	SI

Tabla XII: Atributos del componente Reproductor de MP3.

Vista del componente en la página Web.

Es necesario configurar el objeto para que el componente obtenga los campos y las características que el desarrollador desea. Se creó una clase Usuario de prueba para la presentación. A continuación se muestra el código de dicha clase y su respectiva presentación en la Figura 4.15.

```
public class Usuario extends FormBuilderWrapper {
   private String nombre;
   private String apellido;
   private String tipoIdentificacion;
   private List tiposIdentificacion;
   private String identificacion;
   private Date fechaNacimiento;
   private Integer numeroTicket;
   private SimpleDateFormat formato = new SimpleDateFormat("dd-MM-yyyy");

   @Override
   public void config() {
      setDateFormat(formato);
      AddInputField("Nombre", "nombre", true, 20);
      AddInputField("Apellido", "apellido", true, 20);
      AddComboField("Tipo de ID.", "tipoIdentificacion", true, getTiposIdentificacion());
      AddInputField("Identificacion", "identificacion", true, 13);
      AddInputField("Fecha de Nacimiento", "fechaNacimiento", false, 15);
      AddInputField("No. Ticket", "numeroTicket", true, 4);
      AddButtonPrincipal("Ingresar", "#{FormBuilderTest.ingresar}");
      AddButtonSecundario("Cancelar", "#{FormBuilderTest.cancelar}");
   }
.....
```

Código Fuente 4.5: Código de clase Usuario de ejemplo.

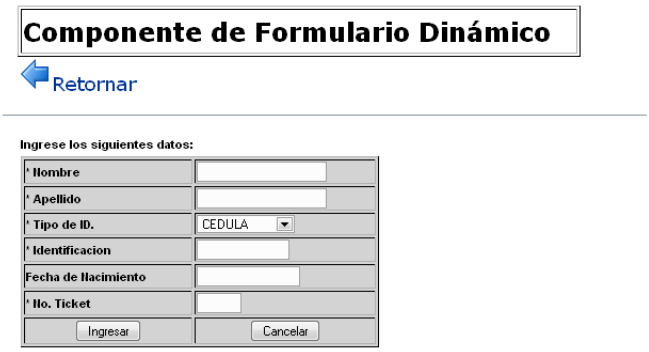


Figura 4.15: Vista del componente de Formulario Dinámico

Aspectos de diseño

El componente crea instancias estándar JSF en tiempo real para presentar los subcomponentes del formulario tales como los campos de texto, las etiquetas y los botones que son añadidos a una tabla. Cada uno de estos subcomponentes poseen su propio render estándar y se encargan por si solos de presentarse en la página Web.

Librerías utilizadas

Se utilizó *Apache BeanUtils* para describir el objeto, ya que proporciona sus atributos y propiedades sin conocerlos de antemano y en tiempo de ejecución. Es muy útil en este caso ya que posee métodos que permiten obtener el tipo de atributo, los datos ingresados y la asignación automática de la información ingresada en la instancia del objeto respectiva.

FormBuilderWrapper

Esta clase es la encargada de encapsular el objeto de lógica de negocio para construir el formulario. El objeto debe extender de esta clase y sobrescribir el método config(). Este método es el encargado de determinar los campos que van a presentarse en el formulario. Los métodos disponibles para configurar el formulario son los siguientes:

AddInputField

public void AddInputField(java.lang.String nombreDisplay,

java.lang.String nombreBean,

boolean requerido,

int longitudMaxima)

Agrega un campo de ingreso al formulario

Parametros:

nombreDisplay - El texto a presentar en el campo

nombreBean - Nombre que representa el atributo del objeto

requerido - indica si el campo es obligatorio

longitudMaxima - indica la máxima longitud que permite ingresar el campo

---

AddOutputField

public void AddOutputField(java.lang.String nombreDisplay,

java.lang.String nombreBean)

Agrega un campo que no puede ser editado al formulario

Parametros:

nombreDisplay - El texto a presentar en el campo

nombreBean - Nombre que representa el atributo del objeto

AddComboField

public void AddComboField(java.lang.String nombreDisplay,

java.lang.String nombreBean,

boolean requerido,

java.util.List opciones)

Agrega un campo de selección única al formulario

Parametros:

nombreDisplay - El texto a presentar en el campo

nombreBean - Nombre que representa el atributo del objeto

opciones – Lista de las opciones a presentar

AddButtonPrincipal

public void AddButtonPrincipal(java.lang.String nombreButton,

java.lang.String nombreMetodo)

Agrega el botón principal y su respectivo metodo a ejecutar al formulario

Parametros:

nombreButton - El texto que presenta el botón.

nombreMetodo - La sentencia EL que referencia al objeto del cual se obtiene el formulario.

---

setDateFormat

public void setDateFormat(java.text.SimpleDateFormat format)

Formato con el cual se van a manipular las fechas de este formulario

Parametros:

format – Formato de la fecha.

4.7. Componente de Gráficas Estadísticas

En estadística denominamos gráficos a aquellas imágenes que, combinando la utilización de sombreado, colores, puntos, líneas, símbolos, números, texto y un sistema de referencia (coordenadas), permiten presentar información cuantitativa.

Los gráficos son herramientas muy usadas y a menudo los más convenientes para presentar datos, se emplean para tener una representación visual de la totalidad de la información. Los gráficos estadísticos presentan los datos en forma de dibujo de tal modo que se pueda percibir fácilmente los hechos esenciales y compararlos con otros.

La utilidad que proporcionan es doble, ya que pueden servir no sólo como sustituto a las tablas, sino que también constituyen por sí mismos una poderosa herramienta para el análisis de los datos, siendo en ocasiones el medio más efectivo no sólo para describir y resumir la información, sino también para analizarla en diferentes campos.

Existe una gran variedad de gráficos para representar información, los seleccionados para el desarrollo de este componente son los que se numeran a continuación debido a que son los más comunes y útiles:

- Gráficas de Barras.

- Gráficas de Barras Agrupadas.
- Gráficas de Pastel.

El objetivo principal del componente “Gráficas Estadísticas” es representar los datos gráficamente con información adicional que ayude a la compresión y análisis de los mismos.

Para obtener una gráfica estadística, el desarrollador solamente requiere crear colecciones de datos con el formato descrito en secciones siguientes, las cuales deben permanecer en sesión con el fin de usarlos como parámetros en el componente.

A continuación se presentan el uso del componente en una página para gráficas de barras y pasteles.

Declaración de la etiqueta en la página Web

La siguiente ilustración, muestra la etiqueta del componente que es autogenerada por el IDE. Tal como se puede observar, la fuente de datos de la gráfica puede ser provista por un bean de sesión, el cual es denominado datos en el siguiente ejemplo, así como los colores, las listas de tooltips y funciones de javascript por cada categoría o sección, tipo de gráfica (pastel, barras y barras agrupadas). Adicionalmente, el componente contiene datos

de prueba, es decir que una gráfica de barras agrupadas es mostrada sin requerir de alguna configuración.

```
<lb:GraficaEstadistica alpha="60" alto="300" ancho="400" data="#{datos.datosBarras}" es3d="false" id="Comp3" labelX="Causas" labelY="% de homicidios" leyenda="true" listaColores="#{datos.listaColores}" listaFunOnClick="#{datos.listaFunciones}" listaTooltips="#{datos.listaTooltips}" orientacion="horizontal" overlib="true" tipo="barras" titulo=" Gráfica de barras "/>
```

Etiqueta Ejemplo de una gráfica de barras

```
<lb:GraficaEstadistica alpha="60" alto="300" ancho="400" data="#{datos.datosBarras}" es3d="true" id="Comp6" labelX="Causas" labelY="% de homicidios" listaColores="#{datos.listaColores}" listaFunOnClick="#{datos.listaFunciones}" listaTooltips="#{datos.listaTooltips}" tipo="barrasGrupo" titulo="Gráfica de barras agrupadas"/>
```

Etiqueta Ejemplo de una gráfica de barras agrupadas

```
<lb:GraficaEstadistica alpha="60" alto="300" ancho="400" data="#{datos.datosPie}" es3d="true" id="Comp2" listaColores="#{datos.listaColores}" listaFunOnClick="#{datos.listaFunciones}" listaTooltips="#{datos.listaTooltips}" tipo="pie" titulo=" Gráfica de Pastel "/>
```

Etiqueta Ejemplo de una gráfica tipo pastel

Atributos del componente

En la siguiente tabla se aprecian los atributos más importantes del componente, los cuales pueden ser configurados por el desarrollador, con sus respectivos tipos de datos.

Nombre	Descripción	Tipo de dato	Requerido
id	Identificación de la grafica, esta debe ser única	String	Sí
tipo	Tipo de la gráfica (Barras/BarrasGrupo/Pastel)	String	Sí

data	Dataset a utilizar como fuente de datos, alimentado previamente por el desarrollador	Expresión EL/Lista	Sí
alto	Alto de la gráfica	Integer	No
ancho	Ancho de la gráfica	Integer	No

Tabla XIII: Descripción de las propiedades del componente.

En el ANEXO G se adjunta un detalle de todos los atributos que se pueden personalizar en el componente.

Vista del componente en la página Web

Como se muestra en las siguientes figuras, las gráficas estadísticas poseen atributos comunes entre todos los tipos, los cuales son título, leyenda, tooltips, 2D ó 3D, etc.

En la siguiente figura se presenta una gráfica de barras agrupadas en 3D representadas en porcentajes con leyenda y título.

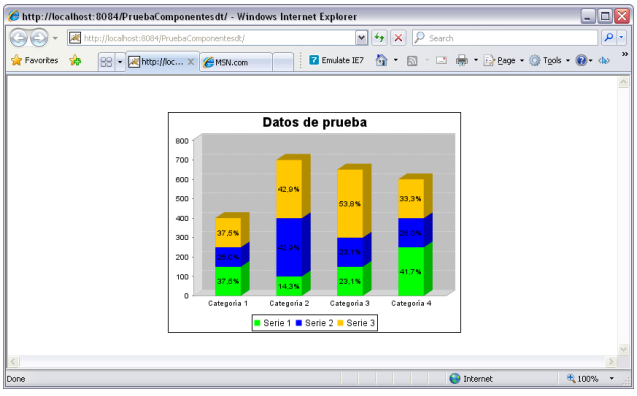


Figura 4.16: Gráfica de barras agrupadas mostrada en la página

La Figura 4.17 muestra una gráfica de pastel en 2D con leyenda, título, porcentajes e información adicional representada en un tooltip.

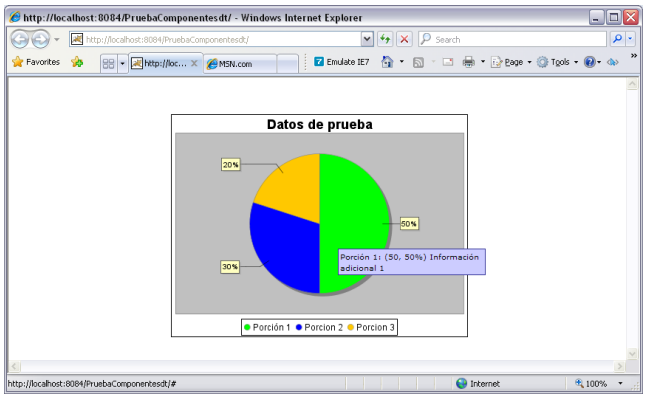


Figura 4.17: Gráfica de pastel mostrada en la página

Diagrama de clases del componente

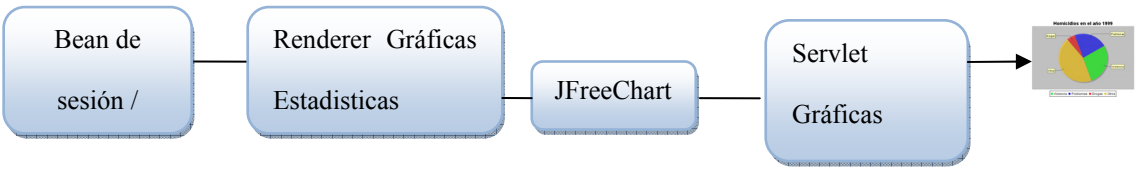


Figura 4.13: Clases que intervienen en el componente.

Bean de sesión (Clase DatosGráficas) / Listas

Para el uso del componente, el desarrollador tiene la opción de llenar un bean que resida en sesión para su posterior uso al momento de construir la gráfica, el mismo que deberá contener listas para los siguientes atributos del componente:

***List* datos:** lista que contiene los datos representados en la gráfica. Estos contienen arreglos de Strings los cuales deben ser ingresados de la siguiente forma:

Gráficas de barras: en este caso el arreglo contiene tres valores:

- Valor correspondiente a la serie y categoría.
- Nombre de la serie.
- Nombre de la categoría.

A continuación se muestra un ejemplo de una lista que será la fuente de datos para una gráfica de barras:

```
List data= new ArrayList();
data.add(new String[]{"5","Hombres","Violencia"});
data.add(new String[]{"5","Hombres","Problemas"});
data.add(new String[]{"15","Hombres","Drogas"});
data.add(new String[]{"25","Hombres","Otros"});
```

Código Fuente 4.6: Origen de datos de la gráfica de barras.

Gráficas de pasteles: en este caso el arreglo contiene dos valores:

- Etiqueta asociada a la porción de datos.
- Valor de la porción de datos.

A continuación se muestra un ejemplo de una lista que será la fuente de datos para una gráfica de pasteles:

```
List dataPie= new ArrayList();
dataPie.add(new String[]{"Violencia","50"});
dataPie.add(new String[]{"Problemas","40"});
dataPie.add(new String[]{"Drogas","10"});
dataPie.add(new String[]{"Otros","80"});
```

Código Fuente 4.7: Origen de datos de la gráfica tipo pastel.

- *List* **listaTooltips:** lista que almacena la información adicional de cada categoría en el caso de las gráficas de barras o en su defecto de cada porción en la de pastel, la cual será mostrada a través de tooltips (Paso del mouse).
- *List* **listaFunciones:** lista que contiene los nombres de las funciones Javascript a ser ejecutadas a través del evento clic sobre cada categoría en el caso de barras o de cada porción en el caso de pasteles.
- *List* **listaColores:** lista de colores a ser utilizados en las barras o en las porciones. Se utiliza la clase *java.util.Color* para la representación de colores.

El Bean contiene datos de prueba, los cuales se llenan en su constructor y se muestran en el ANEXO H; A continuación se presenta el llenado de las listas descritas anteriormente.

```
List tooltips = new ArrayList();
tooltips.add("Violencia familiar");
tooltips.add("Problemas personales");
tooltips.add("Adicción a las drogas ");
tooltips.add("Abusos sexuales,alcoholismo,ect.");
datos2.setListaTooltips(tooltips);

List colores = new ArrayList();
colores.add(Color.GREEN);
colores.add(Color.BLUE);
colores.add(Color.RED);
colores.add(Color.orange);
datos2.setListaColores(colores);

//Se agrega la misma función en las 4 categorías
List funciones = new ArrayList();
funciones.add("mensaje");
funciones.add("mensaje");
funciones.add("mensaje");
funciones.add("mensaje");
```

Código Fuente 4.8: Listas del componente explicadas anteriormente.

Cabe recalcar que las listas pueden ser proporcionadas por el desarrollador directamente al componente sin el uso del bean.

Detalles de implementación

Una vez que los datos son puestos en sesión la clase renderer procede a crear la gráfica usando las clases de JFreeChart.

JFreeChart es una librería libre 100% Java que le permite a los desarrolladores mostrar gráficas profesionales en sus aplicaciones, en este caso fue utilizada para el desarrollo del componente. Esta puede ser descargada de la siguiente dirección: *http://www.jfree.org/index.html*.

Luego de que la gráfica es creada, es puesta en sesión con el id del cliente del componente para que el Servlet la utilice y genere el png a ser mostrado en una etiqueta <img>. Cabe recalcar que la clase renderer del componente genera un map que va a ser asociado a dicha etiqueta <img>, el cual contiene la información extra de cada parte de las gráficas en forma de tooltips y funciones Javascript.

Adicionalmente, el mapping del servlet debe ser agregado al web.xml del proyecto que está creando el desarrollador, tal como se muestra a continuación.

```
<servlet>
        <servlet-name>
                grafica
        </servlet-name>
        <servlet-class>
                ec.edu.espol.fiec.componentes.jsf.graficasEstadisticas.servlets.Grafi
        casEstadisticas
        </servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>
                grafica
        </servlet-name>
        <url-pattern>
                /grafica
        </url-pattern>
</servlet-mapping>
```

Código Fuente 4.9: Configuración del servlet Gráficas Estadísticas.

4.8. Componente Ingreso de usuario

Autenticación es un modo de asegurar que el usuario que intenta realizar funciones en un sistema es de hecho el usuario a quien se le ha autorizado ejecutar dichas funciones.

Mecanismo general de autenticación

El primer elemento necesario (y suficiente) para la autenticación es la existencia de identidades con un identificador único. Los identificadores de usuarios pueden tener muchas formas siendo la más común una sucesión de caracteres.

El proceso general de autenticación consta de los siguientes pasos:

1. El usuario solicita acceso a un sistema.
2. El sistema solicita al usuario que se autentique.
3. El usuario aporta las credenciales que le identifican y permiten verificar la autenticidad de la identificación.
4. El sistema valida según sus reglas si las credenciales aportadas son suficientes para dar acceso al usuario o no.
5. Una de las tareas más comunes de las aplicaciones web es la implementación de autenticación de usuarios, el objetivo principal del componente “Ingreso de Usuario” es cubrir dicha necesidad en un solo paso.

La siguiente figura muestra el medio de autenticación de usuarios en las aplicaciones web.

Figura 4.18: Login/Ingreso de usuario.

Declaración de la etiqueta en la página Web

La siguiente ilustración, muestra la etiqueta del componente que es autogenerada por el IDE. Tal como se puede observar, se deben indicar los datos correspondientes a la conexión de la base de datos tales como, motor a utilizar, nombre de tabla, nombre del campo usuario, nombre del campo contraseña.

```
<lb:Login  campoPass="pass" campoUsuario="username" motorBD="1"
nombreTabla="usuarios"  paginaExito="prueba.jsp" />
```

Código Fuente 4.10: Etiqueta Ejemplo del componente Login/Ingreso de usuario

Atributos del componente

En la Tabla XIV se describe en detalle cada una de las propiedades del componente con sus respectivos tipos de datos.

Nombre	Descripción	Tipo de datos	Requerido
id	Identificación del componente, esta debe ser única	String	Sí
motorBD	Número que representa el motor de base de datos a utilizar	String	Sí
nombreTabla	Tabla sobre la cual se realizará la consulta	String	Sí

campoUsuario	Nombre del campo usuario de la base de datos	String	Sí
campoPass	Nombre del campo contraseña de la base de datos	String	Sí
paginaExito	Página que se muestra si se logra la autenticación del usuario	String	Sí
errorFallo	Mensaje de error que se muestra si no se logra la autenticación del usuario. "Usuario y/o contraseña incorrecta" por defecto.	String	No
errorBD	Mensaje de error que se muestra si hubo algún problema con la BD. "Error, comuníquese con el administrador", por defecto	String	No
archivoCSS	Ruta y nombre del archivo CSS a utilizar. Por defecto CssDefaultLogin.css	String	No

Tabla XIV: Descripción de las propiedades del componente.

Vista del componente en la página Web

La figura mostrada a continuación representa al componente agregado a un proyecto Web.

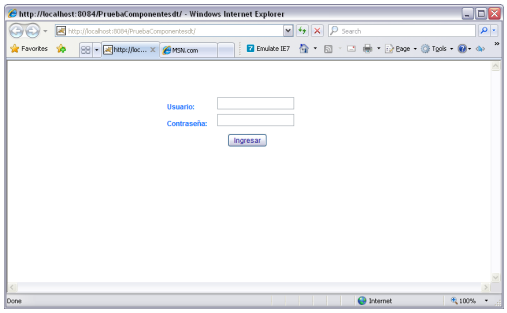


Figura 4.19: Login/Ingreso de usuario

Las siguientes figuras muestran casos de autenticación fallida, por error de usuario y/o contraseña o por error que se refiera a base de datos.

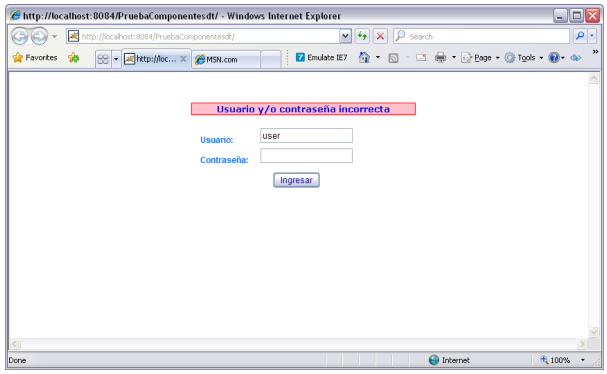


Figura 4.20: Login/Ingreso fallido.

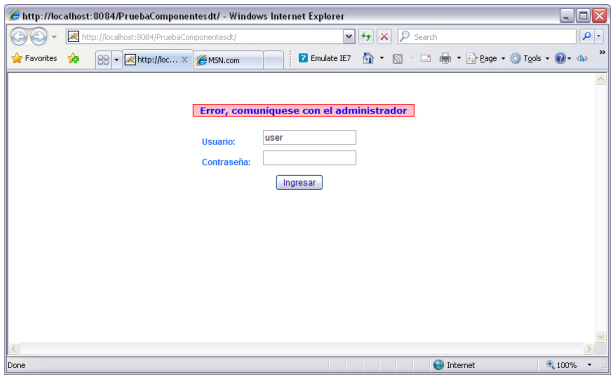


Figura 4.21 : Login/Ingreso de usuario fallido debido a error de conexión a la base de datos.

En el caso de autenticación exitosa se redirecciona a la página indicada por el usuario.

Estructura HTML del componente

El componente posee la siguiente estructura HTML dada por la clase renderer:

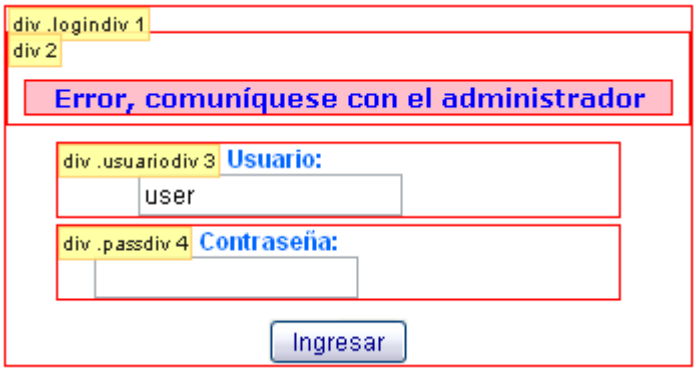


Figura 4.22: Estructura HTML del componente.

Como se puede observar en la figura anterior se presenta el nombre de las clases asociadas a cada div, excepto el div que contiene el mensaje de error (div 2), ya que este está compuesto de un iframe que a su vez contiene una etiqueta <p> con la respuesta enviada por el servlet, en este caso de error, se maneja una clase denominada 'errorP'.

Diagrama de clases del componente

Figura 4.23: Clases que intervienen en el componente Login.

Clase DaoHelper

La clase DaoHelperForJdbc es utilizada para la conexión con la base de datos por medio de JDBC, esto lo realiza por medio de un archivo properties donde se indican los datos necesarios para lograr dicha conexión. Los datos requeridos en este archivo son:

- El nombre del driver a utilizar (jdbc_drv_motor).
- El url de conexión respectivo (jdbc_url_motor).
- El usuario de la base de datos (jdbc_usr_motor) .
- La contraseña del usuario de base de datos (jdbc_pss_motor).

Este archivo debe ser ubicado en la siguiente ruta (docBase)/WEB-INF/clases. A continuación se muestra un ejemplo de una conexión con Mysql por el puerto por defecto, de la base de datos test con usuario y contraseña root.

```
jdbc_drv_motor1=com.mysql.jdbc.Driver
jdbc_url_motor1=jdbc:mysql://localhost/test?autoReconnect=true
jdbc_usr_motor1=root
jdbc_pss_motor1=root
```

Código Fuente 4.11: Archivo properties para la conexión de base de datos.

Como se puede apreciar en el ejemplo anterior se pueden ubicar varias conexiones con diferentes motores de bases de datos por medio de un número, con el cual relacionaremos el tipo de base de datos que utilizará el componente a través de un atributo. Los drivers de base de datos deben constar entre las librerías de la aplicación web.

Servlet Login

En este componente la validación principal la realiza un servlet, Este se conecta a través de la clase DaoHelperForJdbc a la base de datos y verifica la existencia del usuario con la contraseña respectiva. Si la operación fue

exitosa, almacena en sesión todos los datos del usuario, excepto la contraseña, encontrado a través de un Map. En dicho map se ubica como par key-valor, el nombre del campo y su respectivo dato. Finalmente se redirecciona a la página de éxito proporcionada por el desarrollador.

Si la autenticación falla el servlet envía un mensaje de error al usuario indicándole que el usuario y/o la contraseña son inválidas, o en su defecto si hubieron problemas con la conexión de la base de datos, se envía otro mensaje de error, ambos son configurables por el desarrollador.

Adicionalmente, el mapeo del Servlet debe ser agregado al web.xml del proyecto que está creando el desarrollador, tal como se muestra a continuación.

```
<servlet>
        <servlet-name>
        login
        </servlet-name>
        <servlet-class>
                ec.edu.espol.fiec.componentes.jsf.login.servlets.login
        </servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>
                login
        </servlet-name>
        <url-pattern>
        /ingreso
        </url-pattern>
</servlet-mapping>
```

Código Fuente 4.12: Configuración del Servlet Login.

Manejo de MD5

Debido a que las contraseñas no deben viajar por la red sin ser protegidas por algún algoritmo de encriptación, el componente utiliza md5 como una opción de seguridad para las claves. Utilizando un script *webtoolkit.md5.js* de http://www.webtoolkit.info/javascript-md5.html , se convierte la contraseña antes de ser enviada al servlet, este cambio se lo realiza a través de un hidden que almacena la contraseña encriptada a medida que se escribe en la caja de texto para finalmente ser reemplazada por la contraseña original. Como consecuencia a esto, las bases de datos utilizadas, deben manejar contraseñas MD5.

4.9. Componente Filtro XML

XML es una tecnología que tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más flexible y útil.

Adicionalmente, representa una manera distinta de compartir los datos con los que se trabaja a todos los niveles, por todas las aplicaciones y soportes. Juega un papel muy importante en este mundo actual, que tiende a la globalización y la compatibilidad entre los sistemas, ya que es la tecnología que permitirá compartir la información de una manera segura, fiable, fácil.

Debido a que el XML actualmente es una fuente universal de datos, se desarrolló un componente “Filtro XML” cuyo principal objetivo es permitir búsquedas avanzadas de información a partir de un archivo XML dado.

Declaración de la etiqueta en la página Web

La siguiente ilustración, muestra la etiqueta del componente que es autogenerada por el IDE. Tal como se puede observar, se deben indicar el nombre de la carpeta donde se guardará el XML y si se desea renombrar el archivo, el nuevo nombre del mismo. Adicionalmente se puede indicar el archivo CSS que utilizará el componente.

```
<lb:FiltroXML carpeta="dense" archivo="dense" />
```

Etiqueta Ejemplo del componente filtro XML

Atributos del componente

En la Tabla XV se describe en detalle cada uno de los atributos del componente con sus respectivos tipos de datos

Nombre	Descripción	Tipo de dato	Requerido
id	Identificación del componente, esta debe ser única	String	Sí
carpeta	Nombre de la carpeta que contenga el archivo XML. “Default” más la fecha, sería el valor por defecto	String	No
archivo	Nombre del archivo XML a subir. Nombre original de archivo subido más la hora, sería el valor por defecto	String	No
archivoCSS	Ruta y nombre del archivo CSS a utilizar. Por defecto CssDefaultFiltro.css	String	No

Tabla XV: Descripción de las propiedades del componente.

Vista del componente en la página Web

En la siguiente figura se puede apreciar cómo se presenta el componente en una página Web. Una vez que el archivo XML a procesar fue subido satisfactoriamente se genera un árbol por cada etiqueta externa, donde se muestra una caja de texto por cada elemento o atributo que este posee y en el caso de elementos que posean más elementos se presenta de la misma forma, éste último se detallará más adelante.

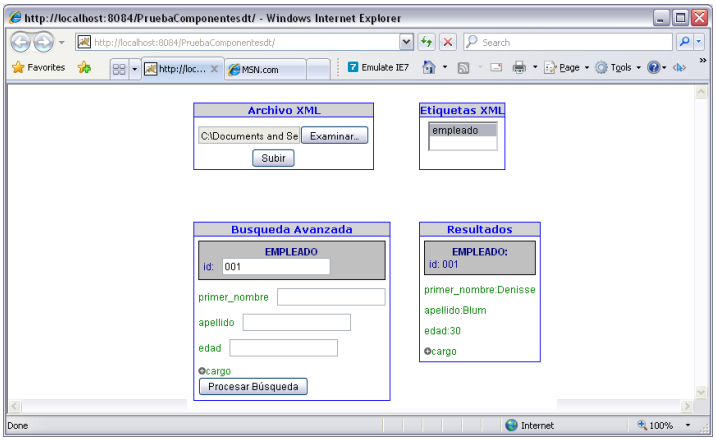


Figura 4.24: Componente "Filtro XML" en una página.

En el caso de que no se encuentre información relacionada a los criterios de búsqueda ingresados se muestra el siguiente mensaje.

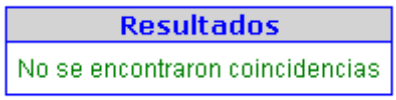


Figura 4.25: El componente no encontró resultados.

Estructura HTML del componente

El componente posee la siguiente estructura HTML dada por la clase renderer, éste consta de cuatro partes:

Subida del XML:

Solo se pueden subir obviamente archivos XML (con extensión .xml).

Selección de etiqueta a filtrar:

Lista que contiene las etiquetas externas del xml.

Ingreso de datos de búsqueda.

Cajas de texto en las cuales se ingresan los criterios de búsqueda.

Resultados.

Datos resultantes que coinciden con los criterios ingresados.

En la siguiente figura se muestra lo mencionado anteriormente junto con las clases de cada div que contiene el componente.

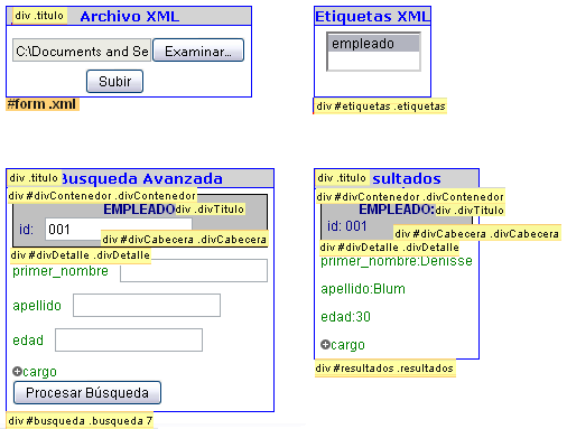


Figura 4.26: Estructura HTML

Como se puede observar en la gráfica anterior tanto la parte de búsquedas y resultados poseen una cabecera y un detalle, los títulos y cada uno de los

divs contenedores tienen una clase asociada para asignarle estilos al componente.

Diagrama de clases del componente

Figura 4.27: Clases que intervienen en el componente Filtro

Servlet uploadFichero

Este servlet sube el archivo solicitado al servidor para que luego pueda ser procesado. Los nombres de las carpetas y los archivos son proporcionados por el desarrollador, a los cuales se les agrega la fecha a las carpetas y la hora los archivos correspondientes a la fecha/hora de subida. En el caso de que no se proporcionen nombres para la carpeta y el archivo, el componente creará una carpeta con nombre “default” con la fecha de subida y el archivo se quedará con su nombre original más la hora de subida. Estos se almacenan en el directorio (docBase)/build/Web.

Un ejemplo de nombre de una carpeta puede ser (creada el 2008-04-03): dense2008_4_3 y un archivo contenido en la carpeta (subida a las 7:54:51): dense7_54_51.xml

Adicionalmente, el mapping del Servlet debe ser agregado al web.xml del proyecto que está creando el desarrollador, tal como se muestra a continuación.

```
<servlet>
        <servlet-name>
                fichero
        </servlet-name>
        <servlet-class>
                ec.edu.espol.fiec.componentes.jsf.filtroXML.servlets.uploadFicher
        o
        </servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>
                fichero
        </servlet-name>
        <url-pattern>
                /fichero
        </url-pattern>
</servlet-mapping>
```

Código Fuente 4.13: Configuración del Servlet uploadFichero.

Filtro JS

Este script se encarga de hacer la petición del archivo subido, por medio de AJAX, con el fin de poderlo recorrer y sacar sus etiquetas y campos a filtrar, sus atributos y elementos que a su vez poseen atributos y elementos.

Los elementos que poseen la imagen ( ), son aquellos que poseen atributos y/o elementos internos por los cuales se puede filtrar también, una vez expandidos se tiene la opción de recoger el elemento ( ) que no se desee incluir en la búsqueda, tal como se muestra en la siguiente figura.

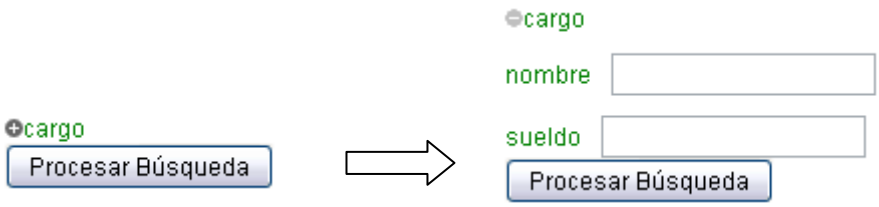


Figura 4.28: Elementos que poseen otros elementos.

Las imágenes deben ser colocadas en la carpeta *Resources* de la aplicación y a su vez en una carpeta llamada *images.*

CAPÍTULO 5

5. Integración de los componentes con el IDE Netbeans

La necesidad de reducir el tiempo de desarrollo y la tendencia a automatizar las tareas repetitivas ha tenido un efecto positivo en la evolución de las herramientas de programación. Ahora éstas además permiten que la experiencia de desarrollar software sea lo más intuitiva y fluida posible.

Una librería difícil de utilizar se convierte en improductiva y desdibuja el propósito inicial para la cual fue creada. Java provee estándares que son aplicados por los desarrolladores de herramientas de programación para mejorar la experiencia del desarrollador como se mencionó anteriormente. En este capítulo se explicarán los pasos para aplicar dichos estándares y la importación de la librería implementada a una de las herramientas de desarrollo más populares: Netbeans.

En la actualidad para el lenguaje Java existen bastantes herramientas de desarrollo como JBuilder de Borland Software, JCreator de Xinox Software, IntelliJIDEA de JetBrians, Eclipse de Eclise Foundation Software, Netbeans de Sun MicroSystems, entre otros.

De los nombrados en el párrafo anterior, los dos últimos se destacan por ser los más aceptados entre los desarrolladores además de ser de libre uso. A continuación

se presenta una comparación entre ambos en base a las características deseables en un IDE y desde el punto de vista de un desarrollador.

	Eclipse 3.3	NetBeans 6.0
Facilidad de Uso/Edición	2,80	3,60
Scripting/otros lenguajes	3,00	3,60
Soporte Enterprise	3,20	3,00
Sitema de Plugin	3,80	2,70
Total	3,20	3,21

Tabla XVI: Comparación en términos generales entre Netbeans y Eclipse (1).

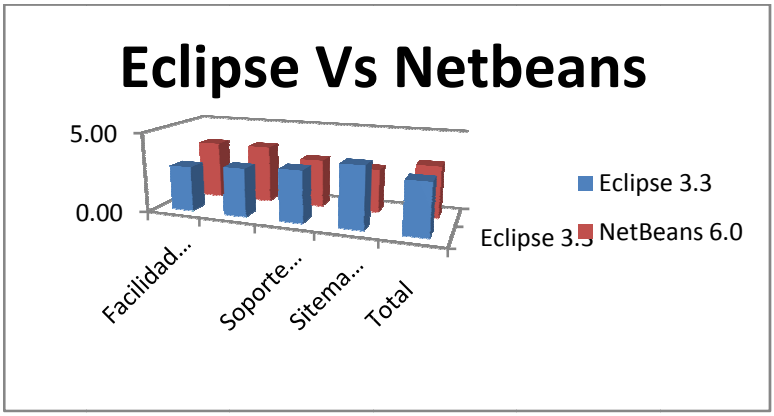


Figura 5.1: Gráfico comparativo entre las calificaciones obtenidas.

En el gráfico se puede apreciar que las dos herramientas obtienen calificaciones similares, y aunque Eclipse ha sido la preferencia de la mayoría de programadores debido a su temprana aparición, facilidad de uso, variedad de herramientas, entre otros, Netbeans ha venido mejorando con el pasar de los años y las nuevas características incluidas en versiones recientes, han hecho que cada vez más desarrolladores lo prefieran.

5.1. Componentes en la barra de herramientas (Pallete)

Comúnmente en java cuando se habla de librerías, se habla de archivos .jar, que son los que contienen las clases compiladas (archivos .class). Se ha trabajado de tal forma que se aproveche la ventaja que dan los IDEs para poder arrastrar componentes hacia la aplicación.

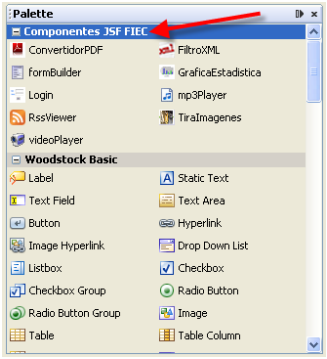


Figura 5.2 : Paleta de componentes disponibles

Además de poder arrastrar el componente, el IDE también se encarga de autogenerar el código con los primeros requerimientos para cada componente.

Una vez importada la librería los componentes estarán disponibles para arrastrarlos a la página de diseño desde la Paleta de Componentes.

5.2. Barra de atributos del componente

Los atributos y propiedades del componente pueden ser manipulados directamente desde la sección de propiedades y el IDE se encarga de escribir

el código para la asignación de las mismas. Aquí se encuentran las propiedades separadas por categorías según se hayan configurado en las clases de diseño.

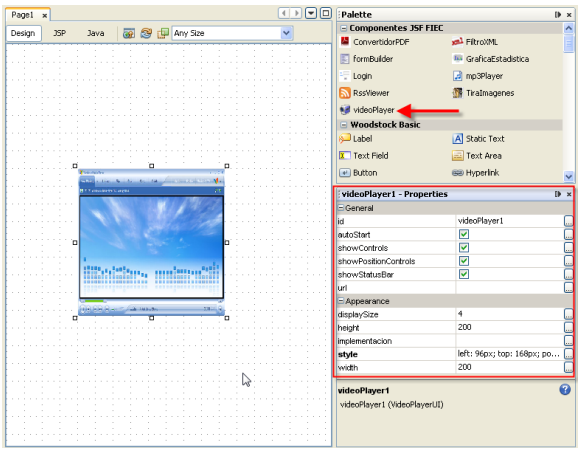


Figura 5.3: Sección de propiedades para el componente VideoPlayer.

5.3. Integración con el IDE

Para la definir el comportamiento de cada uno de los componentes en tiempo de diseño, se tiene que asignar valores a ciertos parámetros en una clase que hace posible leer la información perteneciente al componente y ubicarla ya sea en la barra de herramientas o de atributos.

Toda librería que requiera integración con Netbeans debe definir una clase que extienda de la clase *SimpleBeanInfo*, que es la que hace posible la comunicación entre el IDE y los miembros de las clases de los componentes. En este proyecto se ha definido la clase *BeanInfoHelper* que implementa los métodos básicos necesarios para que todos los componentes puedan integrarse con Netbeans.

A continuación se muestra el paquete en el cual se encuentra la clase *BeanInfoHelper* con sus respectivos archivos properties donde se establecen los nombres de las categorías de propiedades importadas por la clase *CategoryDescriptors*.

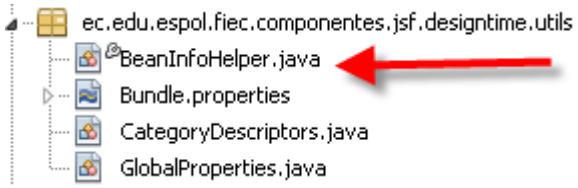


Figura 5.4: Vista de la clase BeanInfoHelper.

Manejo del componente en Tiempo de Diseño

Cada clase de los componentes para tiempo de diseño que usa Netbeans, extiende de *BeanInfoHelper* mencionada anteriormente. Dentro de esta clase se asigna los valores que le corresponde a cada atributo, los principales son:

- La clase del componente
- El icono para la representación del componente en la barra de herramientas.
- El nombre de la etiqueta del componente.
- El nombre con que aparece en la barra de herramientas.
- Una descripción del componente.

También se implementa el método en el cual se establecen todos los atributos que serán presentados en la sección de atributos del IDE y la categoría por la que se agrupan.

```
public class Mp3PlayerUIBeanInfo extends BeanInfoHelper {

   public Mp3PlayerUIBeanInfo() {
      beanClass = Mp3PlayerUI.class;
      iconFileName_C16 = Mp3PlayerAtributes.RECURSO_ICONO_DT;
      iconFileName_C32 = Mp3PlayerAtributes.RECURSO_ICONO_DT;
      instanceName = Mp3PlayerAtributes.INSTANCE_NAME;
      tagName = Mp3PlayerAtributes.COMPONENT_TYPE;
      displayName = Mp3PlayerAtributes.DISPLAY_NAME;
      shortDescription = Mp3PlayerAtributes.SHORT_DESCRIPTION;
      resources                                                    =
ResourceBundle.getBundle(Mp3PlayerUIBeanInfo.class.getPackage().getName() +
            ".Bundle-JSF",
            Locale.getDefault(),
            Mp3PlayerUIBeanInfo.class.getClassLoader());
   }
...
```

Código Fuente 5.1: Ejemplo de constructor para clase de manejo en tiempo de diseño.

```
  @Override
   protected void setComponentPropertyDescriptors(Vector vectorDescriptores)
throws IntrospectionException {
      AttributeDescriptor attrib = null;
      PropertyDescriptor prop_skin =
            new PropertyDescriptor(Mp3PlayerAtributes.ATRIBUTO_SKIN,
 beanClass, "getSkin", "setSkin");
      prop_skin.setDisplayName(resources.getString("SKIN_DISPLAYNAME"));
      prop_skin.setShortDescription(resources.getString("SKIN_DESCRIPCION"));
      prop_skin.setExpert(false);
      prop_skin.setHidden(false);
      prop_skin.setPreferred(false);

      attrib = new AttributeDescriptor(Mp3PlayerAtributes.ATRIBUTO_SKIN, false,
null, true);
      prop_skin.setValue(Constants.PropertyDescriptor.ATTRIBUTE_DESCRIPTOR,
            attrib);
      prop_skin.setValue(Constants.PropertyDescriptor.CATEGORY,

ec.edu.espol.fiec.componentes.jsf.designtime.utils.CategoryDescriptors.GENERAL);
...
...
      vectorDescriptores.add(prop_skin);
...
   }
```

Código Fuente 5.2: Ejemplo donde se expone la propiedad "skin" del componente reproductor de Mp3 para ser manipulada por herramientas de diseño.

La Figura 5.5 muestra los paquetes donde se encuentra la clase del manejo de tiempo de diseño para el Componente Reproductor MP3 y su respectivo paquete de recursos, que en este caso contienen los iconos para la barra de herramientas.

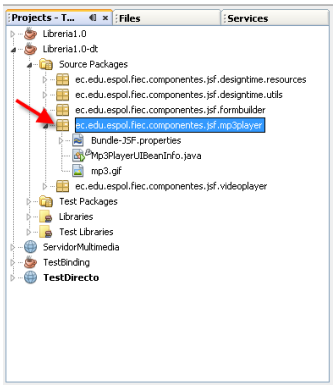


Figura 5.5: Vista del paquete de tiempo de diseño para el componente Reproductor Mp3

Creación de archivo complib para Netbeans

Una librería JSF es un conjunto de componentes que pueden ser agrupados y distribuidos en un solo archivo. Este archivo es un archivo comprimido al cual se le asigna la extensión ".complib" y contiene los componentes, jars de diseño, zip de código fuente, zip de java-doc además de otros jars de tiempo de ejecución o de diseño requeridos; Tiene una estructura como la siguiente:

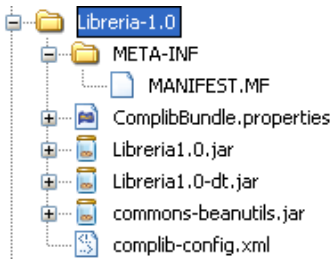


Figura 5.6: Estructura de complib.

A continuación se describen los archivos de configuración para un archivo complib.

- **Archivo Manifest** Es un archivo de metadatos llamado MANIFEST.MF y organizado con pares nombre-valor. Se puede crear con cualquier editor de texto, hay que introducir un fin de línea (enter) tras la última línea. En este archivo indicamos el archivo de configuración del complib.

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.6.2
Created-By: 1.4.2_08-b03 (Sun Microsystems Inc.)
X-Rave-API-Compatibility-Version: 2.0
X-Rave-Complib-Configuration: complib-config.xml
```

Código Fuente 5.3: Archivo MANIFEST.MF que indica el archivo de configuración para el complib.

- **Archivo ComplibBundle.properties** En este archivo se colocan los pares nombre-valor que van a ser importados por el archivo de configuración complib-config.xml.

```
# ResourceBundle para libreria de componentes.
#
#Titulo del grupo de componentes
MainCategory=Componentes JSF FIEC

#Nombre de Libreria
NombreLibreria=Libreria JSF FIEC
```

Código Fuente 5.4: Archivo de propiedades para el configuración del

- **Archivo de configuración de un complib** Este es un archivo XML que define los elementos de la librería cuyos elementos serán descritos a continuación:

Elemento	Descripción
complibConfiguration	Aquí se coloca el nombre del archivo bundle de donde se importan los keys.
identifier	
Uri	El URI
Versión	La versión de complib
titlekey	Su valor indica el key a importar del archivo bundle especificado.
runtimePath	Contiene uno o más <*pathelement*> que especifican los jars necesarios que serán utilizados en tiempo de ejecución.
designTimePath	Contiene uno o más <*pathelement*> que especifican los jars necesarios que serán utilizados en tiempo de diseño.
java-doc	Contiene uno o más <*pathelement*> que especifican el archivo ZIP donde se encuentra el java-doc.
initialPalette	Define la paleta donde van a ir ubicados los iconos que representan a los componentes.
folder	Contiene el atributo "key" que apunta al valor especificado en el bundle y le da el nombre a la carpeta que contiene los componentes especificados en los subelementos <*ítem*>
Item	Contiene el atributo "class-name" cuyo valor apunta a la claseUI del componente.

Tabla XVII: Elementos del archivo de configuracion para el complib.

A continuación de muestra un archivo de ejemplo que contiene las características antes mencionadas.

```
<?xml version="1.0" encoding="UTF-8"?>
<complibConfiguration
 version="1.0"
 resourceBundleBaseName="ComplibBundle">
 <identifier>
  <uri>http://fiec.espol.edu.ec/jsf/componentes</uri>
  <version>1.1</version>
 </identifier>
 <titleKey>SampleTitleKey</titleKey>
 <runtimePath>
  <pathElement>Libreria1.0.jar</pathElement>
  <pathElement>commons-beanutils.jar</pathElement>
 </runtimePath>
 <designTimePath prependRuntimePath="true">
  <pathElement>Libreria1.0-dt.jar</pathElement>
 </designTimePath>
 <initialPalette>
  <folder key="MainCategory">
   <item className="ec.edu.espol.fiec.componentes.jsf.videoplayer.VideoPlayerUI"/>
        <item className="ec.edu.espol.fiec.componentes.jsf.mp3player.Mp3PlayerUI"/>
        <item className="ec.edu.espol.fiec.componentes.jsf.formbuilder.FormBuilder"/>
        ...
  </folder>
 </initialPalette>
</complibConfiguration>
```

Código Fuente 5.5: Ejemplo de archivo de configuración para un complib.

5.4. Integración de la librería con Netbeans

Antes de ser utilizada en proyectos, la librería debe ser importada al entorno de desarrollo. A continuación se describen los pasos para realizar esta tarea.

- Ubicar en el Menú Tools. La opción *Component Library Manager*. Para la versión 5.5 está ubicada en *Tools -> Component Library Manager*, mientras que desde la versión 6 está en *Tools -> Component Librearies*. A continuación el IDE muestra la interfaz que permite escoger el archivo .complib previamente generado.

- Una vez que se ha importado la librería, se presentan los componentes contenidos en la misma y puede ser utilizada en cualquier proyecto Web JSF.

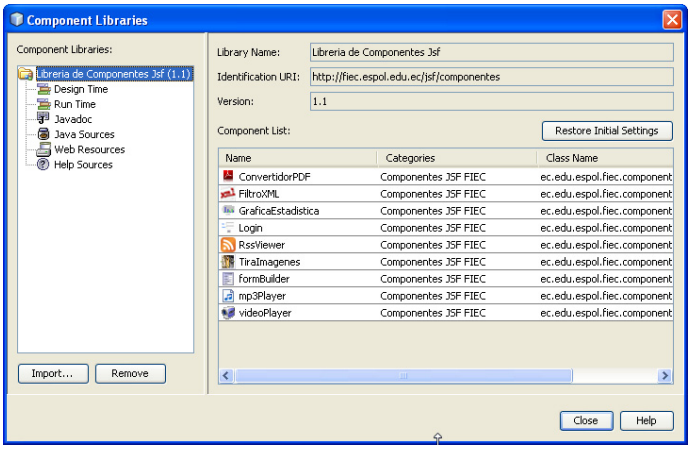


Figura 5.7: Vista de lista de componentes incluidos en el complib.

Importación de la librería a una Aplicación Web JSF.

Una vez creado el proyecto web con soporte para aplicaciones web visuales, al hacer clic derecho sobre el elemento *Component Libraries* de la vista de proyecto, se presenta un menú de contexto con la opción "Add Component Library..".

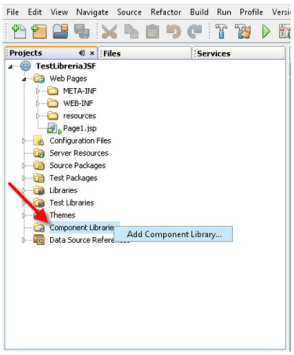


Figura 5.8: Vista de la ubicación desde donde se puede importar la librería en el proyecto.

Al ejecutar esta opción se despliega una lista con las librerías importadas anteriormente al entorno de desarrollo. Aquí se selecciona la librería importada en el paso anterior y se presiona el botón *Add Component Library*.

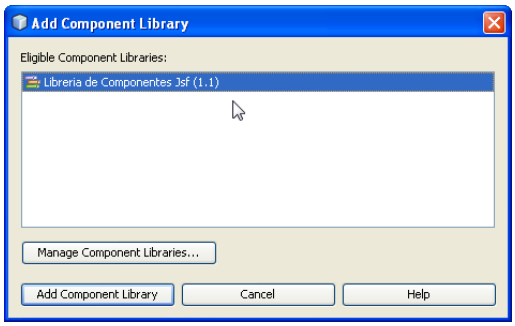


Figura 5.9: Interfaz de elección de la librería a importar.

Una vez que se importa la librería al proyecto se puede observar los componentes en la paleta de componentes. Estos se encuentran disponibles para su uso dentro del proyecto y es posible arrastrarlos al área de diseño de página.

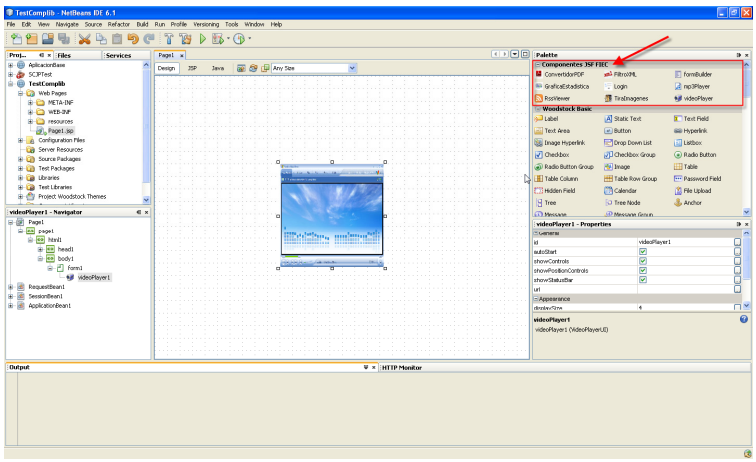


Figura 5.10: Vista de los componentes dentro de la paleta de componentes dentro de su propia pestaña.

Como ejemplo, al arrastrar el componente de video automáticamente se ubica una imagen que representa la manera como se va a presentar el componente en la página web que se está desarrollando.

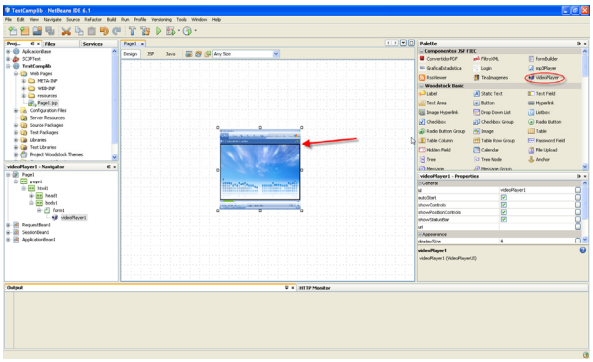


Figura 5.11: Ejemplo de un componente arrastrado dentro una página.

5.5. Pruebas de la librería realizadas por desarrolladores

Para verificar si la librería creada cumple con su propósito principal de facilitar el desarrollo de aplicaciones, se diseñó un grupo de pruebas sencillas divididas en 3 partes donde cada parte tiene 3 componentes. Luego de una charla y taller sobre JSF cuyo programa se detalla en el ANEXO I, cada parte fue entregada a una persona para que sea desarrollada usando la librería. Durante la prueba no hubo ninguna indicación adicional a las descritas en el

ANEXO J.

Luego de terminar la prueba se realizó una encuesta descrita en el ANEXO K

, la cual mide lo siguiente:

- La usabilidad, facilidad de uso y nivel de personalización de cada componente.
- La estimación del ahorro de tiempo conseguido al utilizar la librería para implementar una determinada funcionalidad.
- Verificar si existe una disminución en el nivel de experiencia requerido para implementar una aplicación JSF usando la librería.
- Probabilidad de uso de los componentes en una aplicación de la vida real.

Resultados de las pruebas

A continuación se presentarán los resultados de las preguntas realizadas a las personas que han colaborado para la realización de las pruebas de los componentes de la librería. Para cada una de las preguntas se presenta los gráficos correspondientes junto con su análisis.

Experiencia en JSF

Se solicita la experiencia de cada persona para saber el nivel de conocimiento de la tecnología, esto nos da una pauta para obtener una relación entre el nivel de experiencia y uso de los componentes.

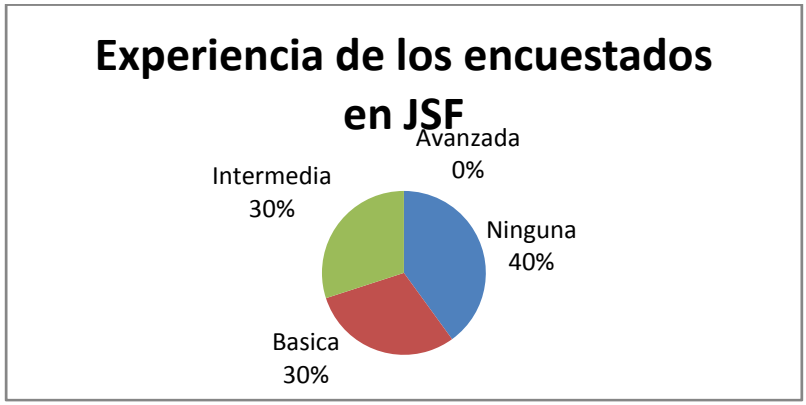


Gráfico 5.1: Experiencia de los encuestados en JSF.

Tiempo estimado

Para sacar conclusiones sobre el tiempo que se necesita para la realización de las tareas con la librería y sin la librería se realizaron dos preguntas, las cuales se muestran a continuación.

- Pregunta #1: **¿Cree Ud. Que podría realizar la prueba sin la librería?**

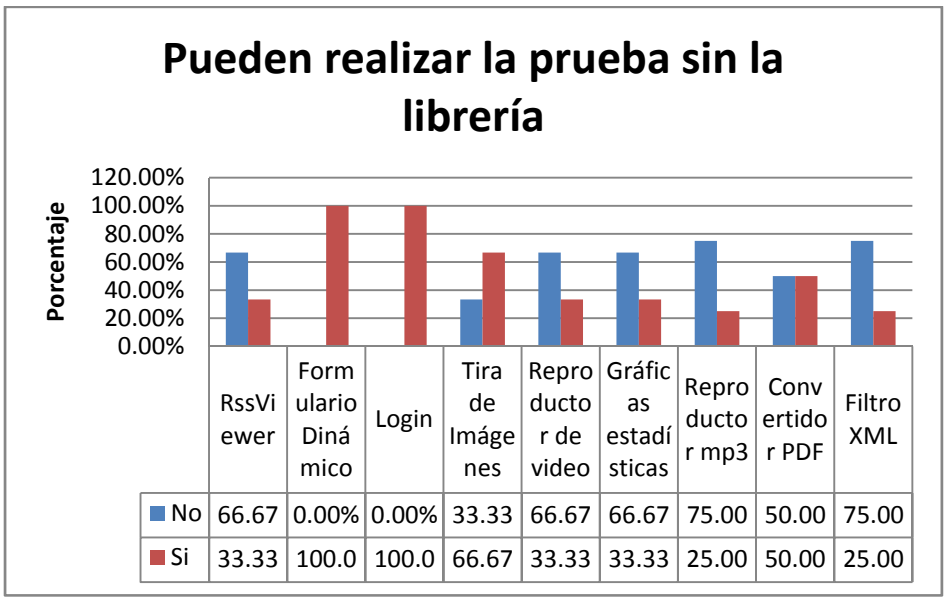


Gráfico 5.2: Resultados de la pregunta #1.

De los 9 componentes utilizados para las pruebas, se puede apreciar que para 6 de ellos más del 50% de los desarrolladores encuestados piensan que no pueden ser realizados sin la librería.

- Pregunta #2: ¿**Cuánto tiempo estima que le hubiera tomado realizar esta tarea <u>SIN</u> la librería**?

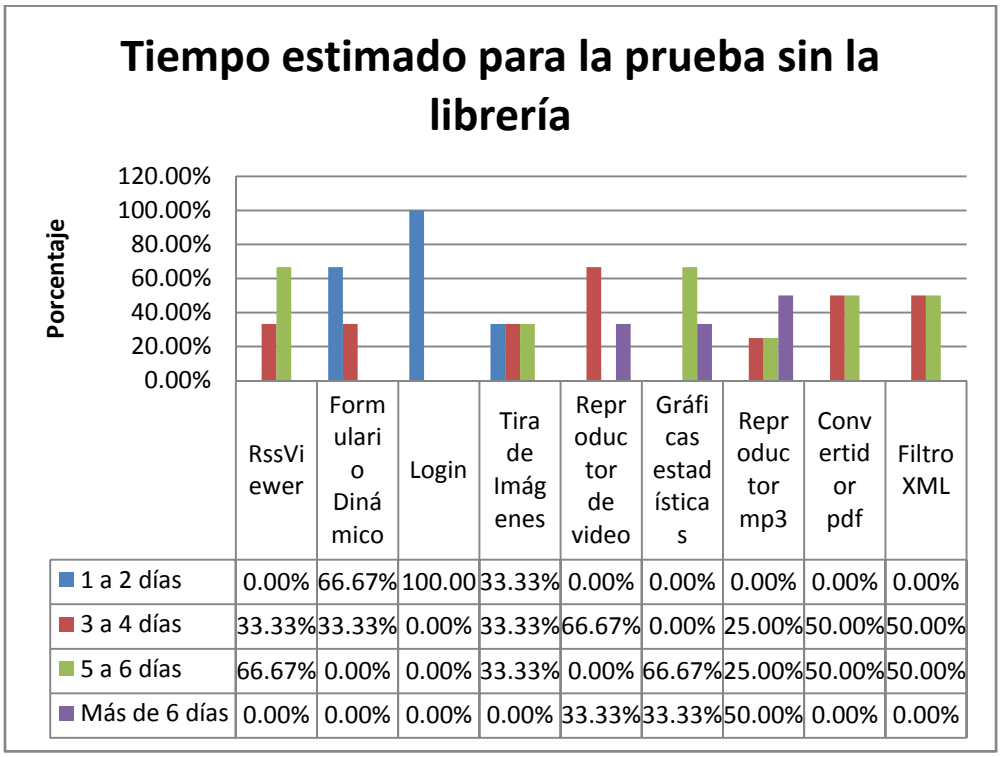


Gráfico 5.3: Resultados de la pregunta #2.

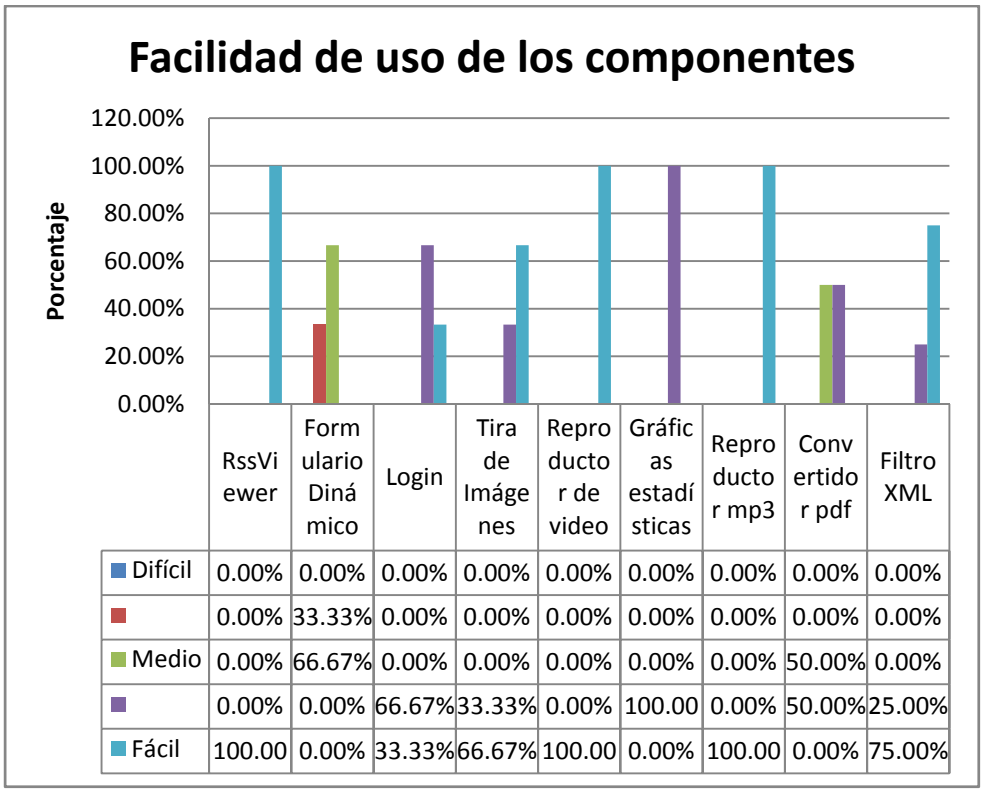
Analizando el Gráfico 5.3, las personas que piensan que si podrían realizar la misma tarea, pronostican entre 4 a 5 días para la implementación de la misma funcionalidad.

Sin duda con el uso de la librería el tiempo de desarrollo se reduce a arrastrar los componentes a la página de la aplicación y configurar su comportamiento según se requiera.

Usabilidad

Asimismo como para la estimación de tiempos, para saber parte de la usabilidad de cada uno de los componentes se realizaron dos preguntas que evalúan la facilidad de uso y la personalización mediante las propiedades de los componentes.

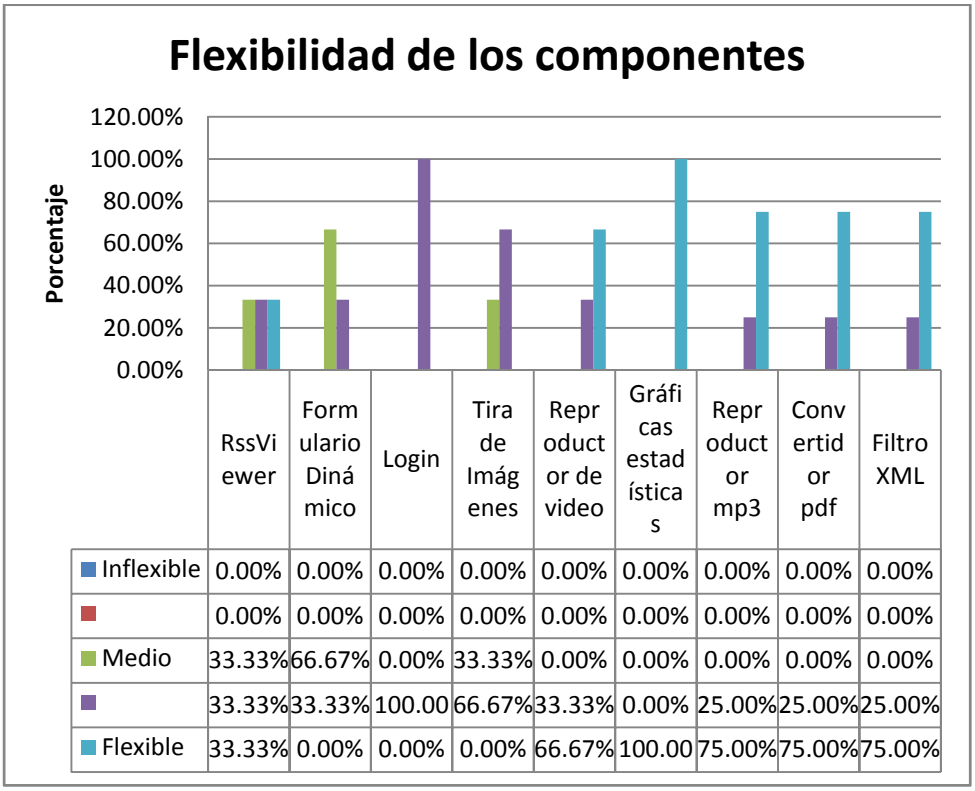
- Pregunta #3: ¿**Qué tan fácil se le hizo utilizar los componentes**?



	RssViewer	Formulario Dinámico	Login	Tira de Imágenes	Reproductor de video	Gráficas estadísticas	Reproductor mp3	Convertidor pdf	Filtro XML
Difícil	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	0.00%	33.33%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Medio	0.00%	66.67%	0.00%	0.00%	0.00%	0.00%	0.00%	50.00%	0.00%
	0.00%	0.00%	66.67%	33.33%	0.00%	100.00	0.00%	50.00%	25.00%
Fácil	100.00	0.00%	33.33%	66.67%	100.00	0.00%	100.00	0.00%	75.00%

Gráfico 5.4: Resultados de la pregunta #3

- Pregunta #4: ¿**Qué tan flexible (personalizable) encuentra el componente**?



	RssViewer	Formulario Dinámico	Login	Tira de Imágenes	Reproductor de video	Gráficas estadísticas	Reproductor mp3	Convertidor pdf	Filtro XML
Inflexible	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Medio	33.33%	66.67%	0.00%	33.33%	0.00%	0.00%	0.00%	0.00%	0.00%
	33.33%	33.33%	100.00	66.67%	33.33%	0.00%	25.00%	25.00%	25.00%
Flexible	33.33%	0.00%	0.00%	0.00%	66.67%	100.00	75.00%	75.00%	75.00%

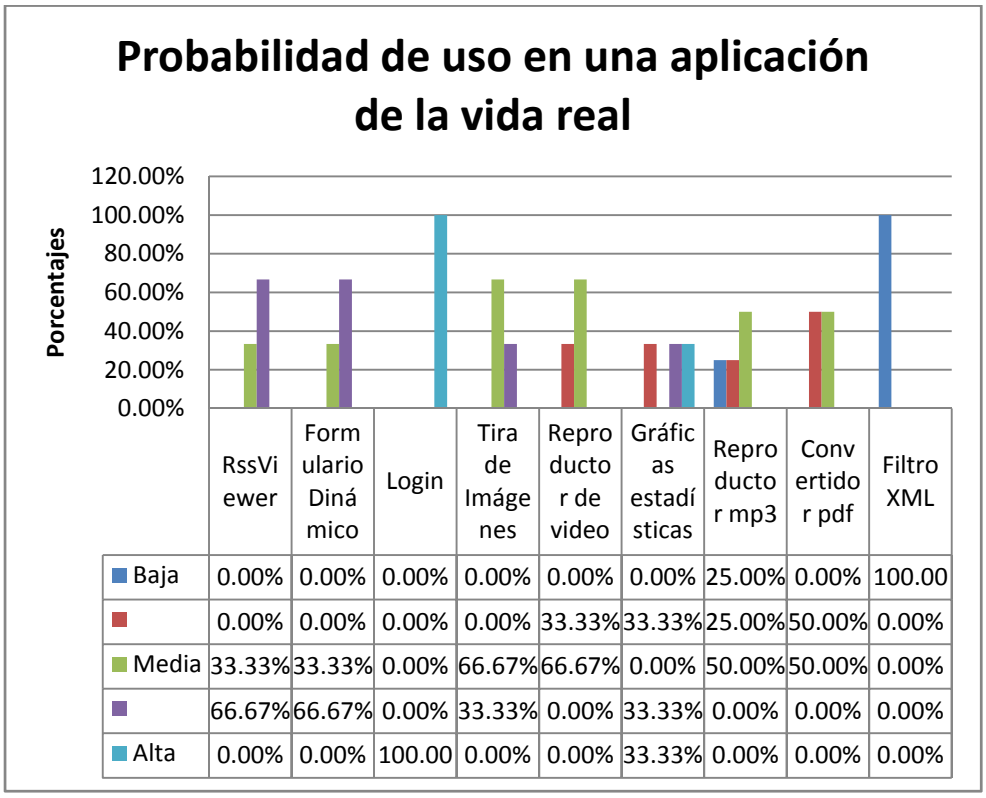
Gráfico 5.5: Resultados de la pregunta #4

Más del 60% de los desarrolladores piensan que los componentes son flexibles y fáciles de usar. Solo se detectó cierta dificultad para las personas que no tienen experiencia en JSF al momento de declarar los Servlets en el archivo de configuración web.xml.

Uso de la librería en la vida real

Para darnos una idea de las probabilidades de uso de los componentes ya en la vida real, se preparó la siguiente pregunta.

- Pregunta #5: ¿**Qué tan probable es que en algún momento Ud. necesite utilizar uno de estos componentes en una aplicación en su área de trabajo**?



	RssViewer	Formulario Dinámico	Login	Tira de Imágenes	Reproductor de video	Gráficas estadísticas	Reproductor mp3	Convertidor pdf	Filtro XML
Baja	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	25.00%	0.00%	100.00
	0.00%	0.00%	0.00%	0.00%	33.33%	33.33%	25.00%	50.00%	0.00%
Media	33.33%	33.33%	0.00%	66.67%	66.67%	0.00%	50.00%	50.00%	0.00%
	66.67%	66.67%	0.00%	33.33%	0.00%	33.33%	0.00%	0.00%	0.00%
Alta	0.00%	0.00%	100.00	0.00%	0.00%	33.33%	0.00%	0.00%	0.00%

Gráfico 5.6: Resultados de la pregunta #5

En las preguntas anteriores se evaluó facilidad de uso y flexibilidad, aun quedaba saber cuál es la probabilidad de uso de los componentes, y se puede apreciar que para la gran mayoría de los componentes la probabilidad es al menos de un nivel medio.

CONCLUSIONES Y RECOMENDACIONES

1. JSF es un framework que permite reducir tiempos de desarrollo mediante el uso de componentes, los cuales incrementan la productividad.

2. La configuración de componentes por medio de archivos permite llevar un control más flexible de su comportamiento.

3. Una librería encapsula conocimiento avanzado y/o tareas repetitivas complejas, permitiendo que este sea implementado por desarrolladores menos avanzados.

4. Mientras más flexible sea el componente, además de reducir el tiempo de desarrollo, reduce el nivel de conocimiento requerido para realizar una tarea.

5. Aún cuando el uso de componentes reduce el tiempo de desarrollo, es importante realizar primero un análisis y evaluar su factibilidad, puesto que el desarrollo de un componente puede llevar más tiempo que la propia tarea.

6. El desarrollo de componentes para JSF es una tarea compleja y consumidora de recursos, por lo que debe tomarse en cuenta este tiempo de aprendizaje en la etapa de análisis previo al desarrollo de una librería.

ANEXOS

ANEXO A ELEMENTOS DE LA ETIQUETA TAGLIB

Elemento	Descripción
display-name	(Opcional) Nombre que va a ser presentado por herramientas
icon	(Opcional) Icono que puede ser usado por herramientas.
validator	(Opcional) Se especifica este elemento si se va a usar para validar las etiquetas.
listener	(Opcional) Especifica listeners para el componente.
tag-file | tag	Declara los archivos de etiquetas que en este caso no serán usados.
function	(Opcional) Se pueden definir funciones EL.
tag-extension	(Opcional) Extensiones que proveen información extra para herramientas de desarrollo.

Tabla A.I Elementos de la etiqueta *<taglib>*

Descripción del Elemento *<validator>*.

Elemento que puede ser utilizado para validar la conformación de cualquier página JSP que importa esta librería.

La siguiente tabla muestra los subelementos que pertenecen al elemento <*validador*>

Elemento	Descripción
validator-class	La clase que implementa javax.servlet.jsp.tagext.TagLibraryValidator
init-param	(Opcional) Parámetros de inicialización.
description	(Opcional) Descripción del validador implementado.

Tabla A.I Elementos de la etiqueta *<taglib>*

Ubicación de estos elementos dentro de un TLD.

```
<validator>
   <validator-class>...</validator-class>
   <init-param>...</init-param>
   <description>...</description>
</validator>
```

Código Fuente A.1:Declaración de los elementos de *<validator>* en el archivo TLD.

Descripción del Elemento *<listener>*.

Una librería de tags puede especificar clases que escuchan diferentes eventos, el contenedor instancia los listeners y los registra en una forma análoga. El único subelemento es *<listener-class>* que contiene el nombre completo de la clase del listener.

```
<listener>
    <listener-class>...</listener-class>
</listener>
```

Código Fuente A.2: Declaración del elemento *<listener>* en el archivo TLD.

ANEXO B - ELEMENTOS DE LA ETIQUETA TAG

Elemento	Descripción
description	(Opcional) Descripción del componente.
display-name	(Opcional) Nombre que puede ser usado por herramientas.
tag-class	El nombre completo de la clase manejadora del componente (HANDLER).
tei-class	(Opcional) Subclase de javax.servlet.jsp.tagext.TagExtraInfo.
body-content	El tipo del contenido cuerpo.
variable	(Opcional) Declara una variable EL expuesta por la etiqueta a la página que lo llama.
dynamic-attributes	Dice si la etiqueta soporta atributos adicionales con nombres dinámicos. El valor por defecto es falso. Si es verdadero, la clase manejadora debe implementar la interface javax.servlet.jsp.tagext.DynamicAttributes.
example	(Opcional) Descripción informal de un ejemplo del uso del componente.
tag-extension	(Opcional) Extensiones que proveen información extra sobre el componente para uso de herramientas.

Tabla B.I Elementos de la etiqueta *<tag>*

ANEXO C - ELEMENTOS DE LA ETIQUETA ATRIBUTTE

Elemento	Descripción
rtexprvalue	(Opcional) Determina si el valor de este atributo puede ser calculado por una expresión EL. El valor por defecto es “false”
type	(Opcional) El tipo de dato que determina el valor. El valor por defecto es: “java.lang.String” si no es especificado.
fragment	(Opcional) Indica si este atributo va a ser evaluado por el Manejador del componente (true) o es un atributo normal que será evaluado por el contenedor y luego pasado al manejador. Si este elemento es verdadero no se especifica el valor del elemento rtextprvalue ya que el contenedor fija este valor a verdadero y además no se especifica tampoco el valor de el elemento “type” ya que el contenedor especifica el tipo javax.servlet.jsp.tagext.JspFragment. El valor por defecto es “false”.

Tabla C.I Elementos de la etiqueta <*atributte*>

ANEXO D - ELEMENTOS DE ETIQUETA *<component>*

Elemento	Descripción
description	Contiene una descripción textual del componente.
display-name	Es un nombre corto y describe el componente, es para uso exclusivo de herramientas de desarrollo.
icon	Contiene 2 sub elementos: <small-icon> y <large-icon> que especifican la ruta como recurso para imágenes GIF o JPG para representar el componente en alguna herramienta gráfica de desarrollo.
attribute	Representa los atributos del componente
property	Representa las propiedades del componente
facet	Representa una sección con nombre dentro de un componente en la página JSP
component-extension	Mediante este elemento se pueden modificar o extender el comportamiento de un componente Base o agregar funcionalidad extra.

Tabla D.I Elementos de la etiqueta *<component>*

Descripción del Elemento *<attribute>*

Identifican atributos genéricos que son reconocidos por la lógica de implementación de este componente.

Elemento	Descripción
description	Contiene una descripción textual del atributo.
display-name	Es un nombre corto y describe el componente, es para uso exclusivo de herramientas de desarrollo.
icon	Contiene 2 sub elementos: <small-icon> y <large-icon> que especifican la ruta como recurso para imágenes GIF o JPG para representar el componente en alguna herramienta gráfica de desarrollo.
Attribute-name	Representa los atributos del componente
attribute-class	Clase que representa el atributo, el valor por defecto es “String”
default-value	Valor por defecto que toma el atributo
suggested-value	Valor sugerido para el programador
attribute-extension	

Tabla D.II Elementos de la etiqueta *<attribute>*

Ubicación de estos elementos dentro del archivo faces-config.xml

```
<attribute>
   <description>...</description>
   <display-name>...</display-name>
   <icon>...</icon>
   <attribute-name>...</attribute-name>
   <attribute-class>...</attribute-class>
   <default-value>...</default-value>
   <suggested-value>...</suggested-value>
   <attribute-extension>...</attribute-extension>
</attribute>
```

Código Fuente D.1: Declaración de un elemento <attribute> en un archivo Faces-Config.

Descripción del Elemento *<property>*

Elementos <*property*> también pueden ser anidados e identifican propiedades del JavaBean de la clase que implementa el componente para ser manipuladas y expuestas por herramientas de desarrollo.

La siguiente tabla muestra los subelementos que pertenecen al elemento <*component*>

Elemento	Descripción
description	Contiene una descripción textual del componente.
display-name	Es un nombre corto y describe el componente, es para uso exclusivo de herramientas de desarrollo.
Property-name	Contiene 2 sub elementos: <*small-icon*> y <*large-icon*> que especifican la ruta como recurso para imágenes GIF o JPG para representar el componente en alguna herramienta gráfica de desarrollo.
Property-class	Nombre del facet
default-value	Valor por defecto que toma el atributo
suggested-value	Valor sugerido para el programador
Property-extension	

Tabla D.III: Subelementos que pertenecen al elemento <*component*>

Ubicación de estos elementos dentro del archivo faces-config.xml

```
<property>
   <description>...</description>
   <display-name>...</display-name>
   <icon>...</icon>
   <property-name>...</property-name>
   <property-class>...</property-class>
   <default-value>...</default-value>
   <suggested-value>...</suggested-value>
   <property-extension>...</property-extension>
</property>
```

Código Fuente D.2: Declaración de un elemento *<property>* en un archivo Faces-Config.

Descripción del Elemento *<facet>*

Elemento que define una sección con nombre dentro de un componente.

La siguiente tabla muestra los subelementos que pertenecen al elemento *<facet>*

Elemento	Descripción
description	Contiene una descripción textual del componente.
display-name	Es un nombre corto y describe el componente, es para uso exclusivo de herramientas de desarrollo.
icon	Contiene 2 sub elementos: *<small-icon>* y *<large-icon>* que especifican la ruta como recurso para imágenes GIF o JPG para representar el componente en alguna herramienta gráfica de desarrollo.
Facet-name	Nombre del facet
Facet-extension	

Tabla D.IV: Subelementos que pertenecen al elemento *<facet>*

Ubicación de los elementos de la Tabla D.IV dentro del archivo faces-config.xml

```
<facet>
   <description>...</description>
   <display-name>...</display-name>
   <icon>...</icon>
   <facet-name>...</facet-name>
   <facet-extension>...</facet-extension>
</facet>
```

Código Fuente D.3: Ubicación de los elementos de la Tabla D.IV dentro del archivo faces-config.xml.

ANEXO E – ARCHIVO TLD PARA COMPONENTE PERSONA

```
<?xml version="1.0" encoding="ISO-8859-1" ?><!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
   <tlib-version>0.01</tlib-version>
   <jsp-version>1.2</jsp-version>
   <short-name>simple</short-name>
   <uri>http://libreria.com</uri>
   <description>Libreria de Componentes JSF</description>
   <tag>
      <name>ComponentePersona</name>
      <tag-class>
         ec.edu.espol.fiec.componentes.jsf.formbuilder.tag.PersonaUITag
      </tag-class>
      <attribute>
         <name>binding</name>
         <description>A value binding that points to a bean property</description>
      </attribute>
      <attribute>
         <name>id</name>
         <description>The client id of this component</description>
      </attribute>
      <attribute>
         <name>rendered</name>
         <description>Is this component rendered?</description>
      </attribute>
      <attribute>
         <name>apellido</name>
         <description>El apellido de la persona</description>
      </attribute>
      <attribute>
         <name>nombre</name>
         <description>El nombre de la persona</description>
      </attribute>
   </tag>
</taglib>
```

Código Fuente E.1: Archivo TLD para el componente persona.

ANEXO F – CODIGO ADICIONAL DEL COMPONENTE RSS

A continuación se presenta la implementación de la clase RssViewerClass, que está encargada de armar las noticias con las clases creadas para el almacenamiento de la información requerida para el componente.

```
public class RssViewerClass {

    RssParser parser = null;
    Rss rss = null;
    Channel channel = null;

    /** Creates a new instance of RssViewer */
    public RssViewerClass(String urlRss) {
        try {
            parser = RssParserFactory.createDefault();
            rss = parser.parse(new URL(urlRss));
            channel = rss.getChannel();
        } catch (RssParserException ex) {
            ex.printStackTrace();
        } catch (MalformedURLException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
```

Código Fuente F.1: Clase RssViewerClass.

```
   public CabeceraNoticia getCabeceraNoticia(String urlRss)
throws RssParserException, MalformedURLException, IOException{
      if (channel == null)
         return null;
      Rss rss = null;
      String tituloRss = "";
      CabeceraNoticia cabeceraNoticia = new CabeceraNoticia();
      Title titulo = channel.getTitle();
      if (titulo!=null)
         cabeceraNoticia.setTitulo(titulo.toString());
         //tituloRss = titulo.toString();
      Image imagen = channel.getImage();
      if (imagen!=null)
         cabeceraNoticia.setUrlImagen(imagen.getUrl().toString());
      return cabeceraNoticia;
   }
  public Collection<Noticia> getNoticias(String urlRss)
throws RssParserException, MalformedURLException, IOException{
      if (channel == null)
         return null;
      Collection items = channel.getItems();
      Collection<Noticia> listaNoticias = new LinkedList<Noticia>();
      Iterator itRss = items.iterator();
      //int contador = 0;
      while( itRss.hasNext() ) {//&& (++contador<5) ){
         Item item = (Item) itRss.next();
         Noticia noticia = new Noticia();
         Title titulo = item.getTitle();
         if ( titulo!=null )
            noticia.setTitulo(titulo.toString());

         Link url = item.getLink();
         if ( url!=null )
            noticia.setUrl(url.toString());

         Description descripcion = item.getDescription();
         if ( descripcion!=null )
            noticia.setDescripcion(descripcion.toString());

         PubDate fechaPub = item.getPubDate();
         if ( fechaPub!=null )
            noticia.setFechaPublicacion(fechaPub.toString());

         Author autor = item.getAuthor();
         if ( autor!=null )
            noticia.setAutor(autor.toString());
```

Código Fuente F.1: Métodos de la clase RssViewerClass.

ANEXO G – DETALLE DE ATRIBUTOS DEL COMPONENTE GRÁFICAS ESTADÍSTICAS

Nombre	Descripción	Tipo de dato	Requerido	Valor por defecto
id	Identificación de la grafica, esta debe ser única	String	Sí	Proporcionado por el IDE
tipo	Tipo de la gráfica (Barras/BarrasGrupo/Pastel)	String	No	Barras Agrupadas
data	Dataset a utilizar como fuente de datos, alimentado previamente por el desarrollador	Expresión EL / List	No	Lista de datos de muestra
titulo	Título de la grafica estadística	String	No	"Gráfica Estadística"
labelX	Etiqueta del eje X, usado en gráficas de barras	String	No	
labelY	Etiqueta del eje Y, usado en gráficas de barras	String	No	
alto	Alto de la gráfica	Integer	No	300
ancho	Ancho de la gráfica	Integer	No	400
orientacion	Orientación de la gráfica (Vertical/Horizontal), aplica sólo en gráficas de barras.	String	No	Vertical
leyenda	Indica sí se muestra leyenda (True/False).	Boolean	No	True
es3d	Indica si la gráfica es 3d (True/False).	Boolean	No	True
listaTooltips	Lista de tooltips que se presentan por categoría o porción de datos	Expresión EL/ List	No	Lista de muestra
listaFunOnClick	Lista de funciones javascript que se ejecutan al dar click en la categoría o porción de datos	Expresión EL/ List	No	Lista de muestra
overlib	Indica si la gráfica si el tooltip usa overlib o el tooltip tradicional (True/False).	Boolean	No	True
listaColores	Lista de colores a ser usadas en la gráfica	Expresión EL/List	No	Lista de muestra
colorFondo	Color de fondo de la gráfica	Color	No	Blanco
colorPlot	Color de fondo del área de la gráfica	Color	No	Plomo
colorLineas	Color de las líneas de la	Color	No	Blanco

	gráfica. Sólo aplica en barras			
alpha	Porcentaje de alpha aplicado a las porciones de las gráficas.	Double	No	0%
tamanoMaxBarra	Porcentaje que corresponde al tamaño máximo de las barras	Double	No	10
sentidoReloj	Indica si el pastel se dibuja a favor de las manecillas del reloj o en contra. (a favor por defecto)	Boolean	No	True
anguloInicio	Angulo de inicio del pastel.	Double	No	90º
style	Estilo del componente	String	No	No

Tabla G.I: Atributos del componente Gráficas Estadísticas.

ANEXO H – DATOS DE PRUEBA DEL COMPONENTE GRAFICAS ESTADISTICAS.

A continuación se presenta la implementación de los datos de prueba para el componente de Gráficas Estadísticas.

```
// DATOS DE GRAFICA DE BARRAS
this.datosBarras= new DefaultCategoryDataset();
this.datosBarras.addValue(150,"Serie 1","Categoría 1");
this.datosBarras.addValue(100,"Serie 1","Categoría 2");
this.datosBarras.addValue(150,"Serie 1","Categoría 3");
this.datosBarras.addValue(250,"Serie 1","Categoría 4");
this.datosBarras.addValue(100,"Serie 2","Categoría 1");
this.datosBarras.addValue(300,"Serie 2","Categoría 2");
this.datosBarras.addValue(150,"Serie 2","Categoría 3");
this.datosBarras.addValue(150,"Serie 2","Categoría 4");
this.datosBarras.addValue(150,"Serie 3","Categoría 1");
this.datosBarras.addValue(300,"Serie 3","Categoría 2");
this.datosBarras.addValue(350,"Serie 3","Categoría 3");
this.datosBarras.addValue(200,"Serie 3","Categoría 4");

// DATOS DE GRAFICA DE PASTEL
this.datosPie=new DefaultPieDataset();
datosPie.setValue("Sección 1",40);
datosPie.setValue("Sección 2",80);
datosPie.setValue("Sección 3",120);
datosPie.setValue("Sección 4",150);

//LISTA DE TOOLTIPS A MOSTRAR
List tooltips = new ArrayList();
tooltips.add("Información adicional 1");
tooltips.add("Informacion adicional 2");
tooltips.add("Informacion adicional 3");
tooltips.add("Informacion adicional 4");
this.setListaTooltips(tooltips);

//LISTA DE COLORES
List colores = new ArrayList();
colores.add(Color.GREEN);
colores.add(Color.BLUE);
colores.add(Color.ORANGE);
colores.add(Color.PINK);
this.setListaColores(colores);
```

Código Fuente H.1: Datos de prueba del componente de Gráficas Estadísticas.

ANEXO I – PROGRAMA DE LA CHARLA DE JSF

- Introducción y conceptos básicos de JSF.
- Tipos de beans involucrados y su alcance.
- Archivos de configuración.
- Navegación estática y dinámica.
- Diseño de interfaz orientada a componentes.
- Importación de librerías al IDE.
- Aplicación de prueba.

ANEXO J – PRUEBAS DE LA LIBRERÍA

TALLER DE USO DE COMPONENTES (1)

Luego de importar la librería implementar cada uno de los requerimientos descritos a continuación:

RssViewer

Crear una página Web y Conectarse al rol de noticias del universo (o cualquier otro), que presente máximo 3 noticias:
http://www.eluniverso.com/rss/gran_guayaquil.xml

Formulario Dinámico.
Crear un formulario de ingreso para el bean *Persona*. Este debe tener mínimo 2 atributos como *nombre* y *apellido*.

Instrucciones del componente:

- Crear un paquete test junto con la clase Persona con los atributos nombre, apellido. Usar convenciones para JavaBeans escribiendo los sets y gets para cada atributo del bean.
- Extender la clase *Persona* de la clase *FormBuilderWrapper* y sobreescribir el método *config()*. Agregar dentro de este las llamadas a los métodos AddXXX según corresponda. Ej.

AddInputField
public void AddInputField(java.lang.String nombreDisplay,
java.lang.String nombreBean,
boolean requerido,
int longitudMaxima)
Agrega un campo de ingreso al formulario
Parameters:
`nombreDisplay` - El texto a presentar en el campo
`nombreBean` - Nombre que representa el atributo del objeto
`requerido` - indica si el campo es obligatorio
`longitudMaxima` - indica la máxima longitud que permite ingresar el campo

AddButtonPrincipal
public void AddButtonPrincipal(java.lang.String nombreButton,
java.lang.String nombreMetodo)
Agrega el botón principal y su respectivo método a ejecutar al formulario
Parameters:
`nombreButton` - El texto que presenta el botón.
`nombreMetodo` - La sentencia EL (#{}) que referencia al objeto del cual se obtiene el formulario.

AddButtonSecundario
public void AddButtonSecundario (java.lang.String nombreButton,
java.lang.String nombreMetodo)
Agrega el botón secundario y su respectivo método a ejecutar al formulario

Parameters:
`nombreButton` - El texto que presenta el botón.
`nombreMetodo` - La sentencia EL (#{}) que referencia al objeto del cual se obtiene el formulario.

- Crear los métodos para los botones en el *BackingBean* de la página. Deben tener la siguiente forma:

```
public String ok(){
 //Cualquier código para probar que pase por aquí.
}
```

- Crear el objeto *Persona* a referenciar en el *SessionBean* y configurarlo en el componente.
- Configurar columnas, color de fondo o incluir casos de navegación según su criterio.

Login

Crear una página Web que contenga un formulario de autenticación. Para la conexión utilice los siguientes parámetros:

Base de datos: Mysql.
Servidor: portatilcarlos
Esquema: BaseDatos
Usuario: root
Clave: caos14

Instrucciones del componente:

- Realizar la configuración de los parámetros indicados anteriormente en el archivo *DaoHelperForJdbc.properties* previamente proporcionado.
- Utilizar el recurso entregado *Exito.jsp* como página de autenticación exitosa.
- Configurar el *Servlet* en el archivo de configuración *web.xml.*

```
<servlet>
        <servlet-name>
        login
        </servlet-name>
        <servlet-class>
                ec.edu.espol.fiec.componentes.jsf.login.servlets.login
        </servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>
                login
        </servlet-name>
        <url-pattern>
        /ingreso
        </url-pattern>
</servlet-mapping>
```

Luego de esto llene la encuesta de usabilidad para los componentes utilizados.

TALLER DE USO DE COMPONENTES (2)

Luego de importar la librería implementar cada uno de los requerimientos descritos a continuación:

Reproductor de Video
Elaborar una página que pueda reproducir uno de los siguientes videos:

- http://portatilcarlos:8080/ServidorMultimedia/Videos/Angra.mpg
- http://portatilcarlos:8080/ServidorMultimedia/Videos/Bunny.divx
- http://portatilcarlos:8080/ServidorMultimedia/Videos/sample.mov
- http://portatilcarlos:8080/ServidorMultimedia/Videos/video.ram

Instrucciones del componente:

- Tener en cuenta el formato de cada video para configurar el tipo de reproductor en el componente.
- Si el *plug-in* no se descargara automáticamente solicite los instaladores manuales.

Tira de Imágenes

Crear una tira de imágenes que tome las siguientes imágenes del servidor y solo muestre 3 figuras simultáneas.

- http://portatilcarlos:8080/ServidorMultimedia/Imagenes/1.jpg
- http://portatilcarlos:8080/ServidorMultimedia/Imagenes/2.jpg
- http://portatilcarlos:8080/ServidorMultimedia/Imagenes/3.jpg
- http://portatilcarlos:8080/ServidorMultimedia/Imagenes/4.jpg
- http://portatilcarlos:8080/ServidorMultimedia/Imagenes/5.jpg

Instrucciones del componente:

- Poner las flechas proporcionadas con el mismo nombre en directorio bajo la carpeta /Web llamado *images*.
- Crear una lista (*ArrayList* o cualquier subclase de *List*) de tipo *String* en el *AplicationBean* con los *url* antes mencionados y proporcionarla al componente.
- Configurar el *Servlet* en el archivo de configuración *web.xml.*

```
<servlet>
    <servlet-name>
        ControladorTiraImagen
    </servlet-name>
    <servlet-class>
        ec.edu.espol.fiec.componentes.jsf.tiraImagen.servlets.ControladorTiraImagen
    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>
        ControladorTiraImagen
    </servlet-name>
    <url-pattern>
        /ControladorTiraImagen
    </url-pattern>
</servlet-mapping>
```

Graficas estadísticas

Agregar una gráfica estadística a la página *Resultados.jsp*, el cual muestra los resultados de una votación previamente programada en la página *Encuesta.jsp*.

Instrucciones del componente:

- Tomar los datos de la gráfica mediante un método estático *getDatos* de la clase *DatosGraficas* proporcionada, el cual accede a base de datos.
- Configurar el *Servlet* en el archivo de configuración *web.xml.*

```
<servlet>
        <servlet-name>
                grafica
        </servlet-name>
        <servlet-class>
                ec.edu.espol.fiec.componentes.jsf.graficasEstadisticas.servlets.Grafic
        asEstadisticas
        </servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>
                grafica
        </servlet-name>
        <url-pattern>
                /grafica
        </url-pattern>
</servlet-mapping>
```

Luego de esto llene la encuesta de usabilidad para los componentes utilizados.

TALLER DE USO DE COMPONENTES (3)

Luego de importar la librería implementar cada uno de los requerimientos descritos a continuación:

Reproductor de MP3
Elaborar una página que reproduzca un mp3 de la siguiente dirección:
http://portatilcarlos:8080/ServidorMultimedia/Musica/sow.mp3

Instrucciones del componente:

- Copie el directorio *mp3component* a su aplicación web bajo la carpeta /Web.
- Configurar el atributo *ubicación* del componente con el nombre de la carpeta del componente.
- En este directorio se encuentran subdirectorios que contienen las listas de reproducción y skins. Edite el *playlist* para indicar las *url* de los archivos mp3.
- No olvide colocar las extensiones para *playlist* y *skins*.

Convertidor a PDF
Permitir descargar un archivo en formato PDF producto de la conversión de un archivo XML con una transformación XLS.

Instrucciones del componente:

- Crear 3 directorios (*xml*, *xsl* y *pdf*) bajo el directorio /Web. Ubicar los archivos proporcionados en sus respectivos directorios.
- Configurar las rutas relativas de los archivos y directorios con sus extensiones en el componente.

Filtro XML
Crear una página web que lea un archivo XML, permita realizar búsquedas sobre el mismo y además presente los resultados.

Instrucciones del componente:

- Crear el archivo XML o solicitar uno de prueba.
- Configurar el *Servlet fichero* en el archivo de configuración *web.xml.*

```
<servlet>
        <servlet-name>fichero</servlet-name>
        <servlet-class>
                ec.edu.espol.fiec.componentes.jsf.filtroXML.servlets.uploadFichero
        </servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>fichero</servlet-name>
        <url-pattern>/filtroXML</url-pattern>
</servlet-mapping>
```

ANEXO K - ENCUESTA DE USO DE LIBRERÍA JSF

Fecha: _____

Datos del encuestado:

- Especialización: _____
- Nivel: _____
- Experiencia en JSF:

Ninguna ☐ Básica ☐ Intermedia ☐ Avanzada ☐

Preguntas:

1. **¿Cree Ud. Que podría realizar la prueba sin la librería? Marque con una cruz su respuesta.**

Componente	SI	NO
Reproductor de Video		
Reproductor de Mp3		
Login		
Formulario dinámico		
Exportar datos en XML a PDF		
Noticias RSS		
Álbum de fotos		
Filtro XML		
Gráficos estadísticos		

2. **¿Cuánto tiempo estima que le hubiera tomado realizar esta tarea <u>SIN</u> la librería? Marque con una cruz su respuesta.**

	Días			
Componente	1–2	3-4	5-6	> 7
Reproductor de Video				
Reproductor de Mp3				
Login				
Formulario dinámico				
Exportar datos en XML a PDF				
Noticias RSS				
Álbum de fotos				
Filtro XML				
Gráficos estadísticos				

3. **¿Qué tan fácil se le hizo utilizar los componentes? Marque con una cruz su respuesta para cada componente.**

	Difícil <- -> Fácil				
Componente	1	2	3	4	5
Reproductor de Video					
Reproductor de Mp3					
Login					
Formulario dinámico					
Exportar datos en XML a PDF					
Noti cias RSS					
Álbum de fotos					
Filtro XML					
Gráficos estadísticos					

4. **¿Qué tan flexible (personalizable) encuentra el componente? Marque con una cruz su respuesta para cada componente.**

	Inflexible <- -> Flexible				
Componente	1	2	3	4	5
Reproductor de Video					
Reproductor de Mp3					
Login					
Formulario dinámico					
Exportar datos en XML a PDF					
Noticias RSS					
Álbum de fotos					
Filtro XML					
Gráficos estadísticos					

5. **¿Qué tan probable es que en algún momento Ud. necesite utilizar uno de estos componentes en una aplicación en su área de trabajo? Marque con una cruz su respuesta para cada componente.**

	Poco Probable <- -> Muy probable				
Componente	1	2	3	4	5
Reproductor de Video					
Reproductor de Mp3					
Login					
Formulario dinámico					
Exportar datos en XML a PDF					
Noticias RSS					
Álbum de fotos					
Filtro XML					
Gráficos estadísticos					

BIBLIOGRAFÍA

1. Ball, Gregory Murray and Jennifer. Including Ajax Functionality in a Custom JavaServer Faces Component. Sun Developer Network. [En línea] http://java.sun.com/javaee/javaserverfaces/ajax/tutorial.jsp.

2. **SDTimes.** *SDTimes.* [En línea] Septiembre de 2006. http://www.sdtimes.com/article/story-20060901-12.html.

3. **Wikipedia®.** Wikipedia®. *Wikipedia®.* [En línea] http://es.wikipedia.org/wiki/Framework.

4. Java Sever Faces in Action.

5. **Wikipedia.** Wikipedia. *Wikipedia.* [En línea] http://es.wikipedia.org/wiki/RSS.

6. **Cavaness, Chuck.** [En línea] http://www.onjava.com/pub/a/onjava/2005/11/02/what-is-struts.html.

7. *Core Java Server Faces.*

8. *Java Server Faces in Action.* s.l. : M A N N I N G.

9. **Intermountain Health Care.** [En línea] www.ujug.org/stuff/JavaServerFaces.ppt.

10. **JavaBeans (TM), tutorial.** [En línea] http://java.sun.com/docs/books/tutorial/javabeans/whatis/index.html.