# Universidad de las Ciencias Informáticas

## Facultad 1


# Título: “AJAK: JSF + AJAX para Kainos”

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

**Autores:**

César Lage Codorníu

Pedro Enrique Novales Hernández

**Tutor:**

Ing. Iósev Pérez Rivero

Ciudad de la Habana, junio del 2008

“Beethoven era un buen compositor porque utilizaba ideas nuevas en combinación con ideas antiguas. Nadie, ni siquiera Beethoven podría inventar la música desde cero. Es igual con la informática.“

Richard Stallman

## Declaración de autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 1 y a la Universidad de las Ciencias Informáticas para que haga el uso que estime pertinente con este trabajo, con carácter exclusivo.

Para que así conste firmamos la presente a los ___ días del mes de _____ del 2008

| _____ | _____ | _____ |
|---|---|---|
| Autor | Autor | Tutor |
| César Lage Codorníu | Pedro E. Novales Hernández | Iósev Pérez Rivero |

# DEDICATORIA

A mi mamá, por su confianza, por ser mi guía y por su inmenso amor...

A mi papá, por su optimismo y por darme fuerzas para seguir adelante...

A mi hermanita, por ser mi niña linda, por su cariño y por sus llamadas...

A mis abuelas, por su sabiduría y por sus mimos...

A toda mi familia, por su apoyo y por enseñarme que no hay nada más importante que ellos...

A todos mis amigos, por estar en el momento preciso...

Pedro Enrique

A mi madre, mi mejor ejemplo de que ser bueno es el único modo de ser dichoso.

A la FEU y sus actores.

A mis Amigos de la universidad.

A mis familias (Lage y Betancourt)

César

A la comunidad Java…

A todos aquellos que sientan que en la informática todo es posible…

# AGRADECIMIENTOS

A todos los que aportaron con lo mejor de su tiempo, conocimiento y paciencia…

# RESUMEN

En la Universidad de las Ciencias Informáticas (UCI) existen proyectos que incorporan AJAX[1] al desarrollo de la capa de presentación de aplicaciones web implementadas con JSF[2]. Sin embargo, aún resultan escasos los conocimientos sobre el tema.

En este trabajo se describe el proceso de implementación para la mejora de la capa de presentación del proyecto Kainos[3], mediante la integración de AJAX a JSF. Para ello se presenta el estado del arte de las tecnologías AJAX y JSF, así como el conjunto de herramientas a utilizar para el desarrollo de la aplicación. Se realiza un análisis crítico de los distintos frameworks[4] que permiten integrar ambas tecnologías y se define una propuesta que se ajusta a las necesidades del proyecto. Finalmente, se describen los pasos necesarios para implementar la propuesta definida.

[1] AJAX(JavaScript y XML Asíncronos): Técnica de desarrollo web que permite modificar la información de una página web sin tener que recargarla completamente, agilizando de esta forma la interacción con el usuario.

[2] JSF(JavaServer Faces): Framework para aplicaciones web basadas en Java que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

[3] Kainos: Proyecto cuyo objetivo es desarrollar una aplicación para la gestión de la información de la Federación Estudiantil Universitaria (FEU) a todos sus niveles.

[4] framework: Estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado.

# ÍNDICE

# ÍNDICE DE FIGURAS

# INTRODUCCIÓN

El mundo se mueve cada vez más hacia una sociedad que funciona sobre las nuevas tecnologías, donde se extiende la informatización de los procesos que regulan la vida cotidiana. Nuestro país lleva a cabo un programa de informatización de la sociedad que es sustento y motor del desarrollo. La Universidad de las Ciencias Informáticas (UCI), como centro de producción de software, juega un papel primordial; ya sea por la idea de construir un modelo de ciudad digital, como por el número creciente de proyectos que desde la universidad contribuyen a la informatización de los distintos sectores del país. Se ha apostado por el uso del software libre y ello requiere construir un software de calidad que cumpla con todos los requerimientos de los usuarios, además de estar en correspondencia con las tendencias de desarrollo de las aplicaciones actuales a nivel internacional.

El proyecto Kainos tiene como objetivo desarrollar una aplicación para la gestión de la información de la Federación Estudiantil Universitaria (FEU) a todos sus niveles. También se propone controlar la información actualizada de cada uno de sus miembros y cuenta con un sistema de reportes que facilita el cierre del funcionamiento mensual de la organización.

Para lograr estos propósitos se trabajó en una aplicación web que pudiera ser accesible desde todos los municipios del país. En la definición de la arquitectura inicial del proyecto se decidió utilizar la tecnología Java[5] y se implementó la capa de presentación de la aplicación usando JSF.

En la actualidad, el uso de JSF en la capa de presentación es muy común para sistemas multicapas desarrollados en la plataforma JEE[6]. Así mismo se ha popularizado AJAX, como una de las tendencias de la Web 2.0[7], por lograr un dinamismo e interactividad con el usuario similar al alcanzado en las aplicaciones de escritorio.

JSF proporciona una rica arquitectura para manejar el estado de los componentes, procesar los datos, validar la entrada del usuario y manipular eventos, funcionalidades que se encuentran del lado del servidor. Sin embargo, este último establece una comunicación síncrona con el cliente, pues ambos

[5] Java: Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90.

[6] JEE (Java Enterprise Edition): Versión empresarial de Java, que después de J2EE 1.4 es llamada JEE 5.0.

[7] Web 2.0: Segunda generación de Web basada en comunidades de usuarios y servicios, que fomenta la colaboración y el intercambio ágil de información entre los usuarios.

operan dependiendo del procesamiento de una petición y necesitan de un resultado para proseguir. Esto trae como consecuencia que las páginas de una aplicación web se recarguen completamente cuando sólo se necesitan actualizaciones parciales, lo que trae consigo demoras en la navegación. Al mismo tiempo, puede provocar pérdidas de datos, de la posición de la barra de desplazamiento o del foco de la página en los formularios, y demora en validaciones sencillas. Esta situación dificulta la interacción de los usuarios con la aplicación.

Lo explicado anteriormente constituye la situación problémica acontecida desde que se liberó la primera versión de la aplicación del proyecto Kainos. De esta situación se define el siguiente **problema científico**:

*¿Cómo integrar AJAX a la capa de presentación JSF del proyecto Kainos para mejorar la interacción del usuario con la aplicación?*

Teniendo en cuenta este problema, se define como **objeto de estudio**: *La capa de presentación de las aplicaciones web que usan JSF con AJAX.*

Se especifica como **campo de acción**: *La capa de presentación JSF del proyecto Kainos.*

Se plantea como **objetivo general**: *Integrar AJAX a la capa de presentación JSF del proyecto Kainos.*

Se plantean además como **objetivos específicos** los siguientes:

- Estudiar las experiencias de la integración de AJAX a JSF en el mundo.
- Definir una propuesta de integración de AJAX a JSF.
- Implementar la propuesta definida en la capa de presentación JSF del proyecto Kainos.

Para guiar la investigación se plantea como **idea a defender** que: *La integración de AJAX al ciclo de vida de JSF permitirá mejorar la capa de presentación del proyecto Kainos.*

Para dar cumplimiento a los objetivos específicos, se ha definido el siguiente conjunto de **tareas**:

- Revisión y selección bibliográfica sobre temas relacionados con AJAX, JSF y su integración.
- Análisis crítico de las tecnologías para integrar AJAX a JSF.
- Integración de AJAX a JSF en la capa de presentación del proyecto Kainos.

Los **métodos de investigación teóricos** que se han aplicado en la realización de este trabajo son:

- Analítico – Sintético: Se hace una división del fenómeno a estudiar en los componentes que lo integran, JSF y AJAX, para entender el funcionamiento de cada uno por separado. Como resultado de ello, se toma de ambos las características principales para lograr una solución eficaz de integración.
- Inductivo – Deductivo: A partir de JSF y AJAX se implementa la capa de presentación del proyecto Kainos y se define una propuesta para su reutilización en otros proyectos de la UCI.
- Histórico – Lógico – Tendencial: La propuesta de este trabajo se realiza sobre la base de la evolución histórica, las condiciones en que surgieron y las principales tendencias de las tecnologías JSF y AJAX.

En la presente tesis el contenido está estructurado en tres capítulos:

- Capítulo 1. Fundamentación teórica: Se presentan el estado del arte de las tecnologías que constituyen el objeto de estudio y el conjunto de herramientas a utilizar para el desarrollo de la aplicación.
- Capítulo *2*. Propuesta de integración: Se realiza un análisis técnico de los distintos frameworks que permiten integrar AJAX a JSF y se define una propuesta que se ajusta a las necesidades del proyecto.
- Capítulo *3*. Mejora de la capa de presentación: A partir de la propuesta definida, se describen los pasos necesarios a aplicar en la capa de presentación del proyecto Kainos, sobre la primera versión de la aplicación.

# Capítulo 1. Fundamentación teórica

En este capítulo se muestra el estado del arte de las aplicaciones web y se exponen las tendencias actuales de las diferentes tecnologías que se utilizan en el desarrollo de las mismas. Además, se describe JSF y AJAX por separado y se ofrece una panorámica acerca de su integración. Más adelante, se expone el ambiente de desarrollo para la implementación de la aplicación.

## *1.1. Aplicaciones Web*

Las aplicaciones web son sistemas informáticos que los usuarios utilizan accediendo a un servidor web, a través de Internet o de una intranet. Son populares debido a la practicidad del navegador web como cliente ligero y a la facilidad para actualizar y mantener estas aplicaciones sin distribuir e instalar software en miles de potenciales clientes. Aplicaciones como los webmails[8], wikis[9], weblogs[10] y tiendas en línea, son ejemplos bien conocidos de aplicaciones web. [12]

En sus inicios, la web era sencillamente una colección de páginas estáticas, documentos, imágenes, entre otros, para su consulta o descarga. El paso inmediatamente posterior en su evolución fue la inclusión de métodos para elaborar páginas dinámicas, permitiendo que las respuestas se generen a partir de los datos de las peticiones.

Uno de los métodos más conocidos es el Common Gateway Interface (CGI), que define un mecanismo mediante el cual se puede pasar información entre el servidor y ciertos programas externos. Los CGIs se utilizan ampliamente, la mayoría de los servidores web permiten su uso gracias a su sencillez.

El funcionamiento de los CGIs tiene, sin embargo, un punto débil: cada vez que se recibe una petición, el servidor debe lanzar un proceso para ejecutar el programa CGI. Como la mayoría de CGIs están escritos en lenguajes interpretados, como Perl o Python, o en lenguajes que requieren entorno de ejecución, como Java o Visual Basic, el servidor se ve sometido a una gran carga. La concurrencia de múltiples accesos al CGI puede crear graves problemas.

[8] webmail: Cliente de correo electrónico, que provee una interfaz web por la que acceder al correo electrónico.
[9] wikis: Contenidos creados por los usuarios que cualquiera puede modificar, corregir y ampliar.
[10] weblogs: Páginas web que contienen anotaciones ordenadas cronológicamente.

Se empiezan a desarrollar alternativas a los CGIs para solucionar el problema del rendimiento. Las soluciones llegan básicamente a través de dos sistemas. En el primero se diseñan sistemas de ejecución de módulos mejor integrados con el servidor, que evitan la instanciación y ejecución de varios programas. En el segundo se dota a los servidores de un intérprete de algún lenguaje de programación, que permita incluir el código en las páginas de forma que lo ejecute el servidor, reduciendo el intervalo de respuesta.

Se experimenta un aumento del número de lenguajes y arquitecturas que permiten desarrollar aplicaciones web. Todos siguen alguno de los sistemas antes mencionados y los más utilizados son los que permiten integrarlos. Esta integración consiste en un lenguaje que permite al servidor interpretar comandos incrustados en las páginas HTML[11] y un sistema de ejecución de programas mejor enlazado con el servidor, que no implica los problemas de rendimiento propios de los CGIs.

En este sentido el lenguaje Java, creado por la empresa de desarrollo de software Sun Microsystems, es uno de los más potentes. Dos de las potencialidades más importantes de Java para las aplicaciones web es la integración de un lenguaje que permite la incrustación de código en las páginas HTML que el servidor convierte en programas ejecutables, denominado JavaServer Pages (JSP), y un método de programación muy ligado al servidor, con un rendimiento superior a los CGIs, denominado servlet.

Otro lenguaje de programación interpretado por el servidor, muy utilizado en Internet y de los que más éxito tiene, es PHP. Se trata de un lenguaje que permite incrustar HTML en los programas, con una sintaxis que proviene de C[12] y Perl. Teniendo en cuenta su facilidad de aprendizaje, su sencillez y potencia, se ha convertido en una herramienta muy utilizada para el desarrollo de sistemas.

Uno de los patrones arquitectónicos más usados y recomendados en la actualidad para el desarrollo de aplicaciones web, es la arquitectura en capas. Inicialmente, casi todas las arquitecturas eran de dos niveles (cliente y servidor), donde uno solicitaba un servicio a través de una petición y otro brindaba la respuesta correspondiente. Actualmente, es muy común el uso de tres o más niveles (n capas).

La arquitectura en tres capas, como su nombre lo indica, se divide en tres capas lógicas distintas, cada una de ellas con un grupo de interfaces perfectamente definidas.

---

[11] HTML (Lenguaje de Marcado de Hipertexto): Lenguaje de marcado predominante para la construcción de páginas web.

[12] C: Lenguaje de programación creado en 1972 por Ken Thompson y Dennis M. Ritchie, orientado a la implementación de Sistemas Operativos.

- Primera capa: se denomina capa de presentación y normalmente consiste en una Interfaz Gráfica de Usuario (GUI[13]). Aquí es donde la aplicación presenta información a los usuarios y acepta de ellos entradas o respuestas para ser usadas por el programa. Idealmente, la GUI no debe desarrollar ningún procesamiento de negocios o reglas de validación de negocios. Por el contrario, la GUI debe delegar sobre la capa de negocio para manipular estos asuntos.
- Segunda capa: se denomina capa intermedia, capa de empresa, o también es conocida como capa de negocio, consiste en la lógica del negocio. Es básicamente el código al que recurre la capa de presentación para recuperar los datos deseados. La capa de presentación recibe entonces los datos y los formatea para su presentación, comunica la información y la captura. Esta separación entre la lógica del negocio y la interfaz de usuario, añade una importante flexibilidad al diseño de la aplicación. Pueden construirse y desplegarse múltiples interfaces de usuario sin cambiar en absoluto la lógica del negocio, siempre que esta presente una interfaz claramente definida para la capa de presentación.
- Tercera capa: se denomina capa de acceso a datos. La capa de negocio accede a esta capa cuando necesita hacer operaciones CRUD[14] (crear, leer, actualizar, borrar) sobre objetos persistentes, delegando estas responsabilidades en esta capa.

Con el objetivo de mantener las buenas prácticas en el diseño de arquitecturas, se han creado varios patrones arquitectónicos, los cuales definen una familia de sistemas informáticos en términos de su organización estructural. Los patrones arquitectónicos describen componentes y las relaciones entre ellos con las restricciones de la aplicación, así como el diseño para la construcción de dicha aplicación. Uno de los más utilizados en la construcción de sistemas empresariales distribuidos, es el Modelo-Vista-Controlador (MVC).

El MVC separa en tres clases diferentes el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario:

- Modelo: Administra el comportamiento y los datos del dominio de la aplicación, responde a requerimientos de información sobre el estado de la misma (usualmente formulados desde la

[13] GUI: Tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

[14] CRUD: Acrónimo de Create-Read-Update-Delete. Conocido como el padre de todos los patrones de capa de acceso a datos.

vista) y a instrucciones de cambiar dicho estado (habitualmente formulados desde el controlador).

- Vista: Maneja la visualización de la información.
- Controlador: Controla el flujo entre la vista y el modelo.

Tanto la vista como el controlador dependen del modelo, sin embargo, este no depende de las otras clases. Esta separación permite construir y probar el modelo, independientemente de la representación visual.

Las principales ventajas del MVC son las siguientes:

- Soporte de múltiples vistas: Dado que la vista se encuentra separada del modelo, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos. Por ejemplo, diferentes páginas de una aplicación Web pueden utilizar el mismo modelo de objetos mostrado de maneras diferentes.
- Adaptación al cambio: Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares o PDAs[15.] Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

Gracias a las facilidades que brinda MVC, en la actualidad muchas aplicaciones se basan en este estilo arquitectónico, tal es el caso del framework JSF.

## *1.2. JavaServer Faces (JSF)*

En el presente epígrafe se exponen las principales características de la tecnología JSF, término que puede definirse como especificación o framework; así como los principales elementos que la conforman: Servlet[16] y JSP. Además, se presentan ventajas y algunas acotaciones importantes en relación con la tecnología JSF.

[15] PDA (Asistente Digital Personal): Computador de mano originalmente diseñado como agenda electrónica con un sistema de reconocimiento de escritura.

[16] Ver epígrafe 1.2.3

### 1.2.1. De la especificación al framework

Java Community Process (JCP) es una comunidad donde intervienen compañías, organizaciones y empresas, todas involucradas en la producción y desarrollo de software. Su objetivo fundamental es definir los estándares de uso para la tecnología Java. Cada uno de los nuevos estándares se publica a través de especificaciones y se conocen como Java Specification Request (JSR).

JSF es una especificación de JCP que define una tecnología. Fue introducida a través de la JSR 127 por Sun MicroSystem en mayo del año 2001, dando a conocer la versión 1.1 de JSF; luego se publicó la JSR 252 en mayo del año 2006, con la versión 1.2. Actualmente JCP trabaja en la versión 2.0 de JSF.

La especificación JSF de JCP puede tener distintas implementaciones. Una implementación es la elaboración del código fuente necesario, a partir de los estándares definidos, de manera que se pueda utilizar como framework.

Una de las implementaciones de JSF más conocidas es MyFaces, la cual fue creada por Apache durante el proceso de especificación para que sirviera como referencia de implementación. Existe además, una implementación estándar desarrollada por Sun MicroSystem que tiene el mismo nombre que la tecnología, lo que puede ocasionar dudas, pues también se llama JSF. Se conocen otras implementaciones como Smile, Shale, ICEFaces (de ICESoft), ADF Faces (de Oracle), etcétera.

Actualmente, el framework JSF comparado con otras tecnologías basadas en MVC (Spring MVC, Struts, WebWork, Tapestry, entre otros) está más orientado a componentes y eventos, y tiene mayor nivel de abstracción. JSF es el más reciente, el más avanzado y con mejores perspectivas de desarrollo.

El framework JSF permite hacer el desarrollo de software más rápido y más fácil. Enmascara los elementos de complejidad del desarrollo web, para que el trabajo se concentre en construir una mejor aplicación a partir de los componentes.

### 1.2.2. La tecnología

JSF es un framework de interfaces de usuario del lado de servidor para aplicaciones Web sobre tecnología Java, basada en el patrón MVC. [8]

Los principales componentes de la tecnología JSF son:

- API[17] y una implementación de referencia: para representar componentes de interfaz de usuario y manejar sus estados, manejar eventos, validar y convertir datos del lado del servidor, definir la navegación entre páginas, soportar internacionalización y accesibilidad, y proporcionar extensibilidad para todas estas características.
- Librería de etiquetas personalizadas: para dibujar componentes de interfaz de usuario dentro de una página y para representar manejadores de eventos, validadores, navegación y otras acciones.

Una característica importante de la tecnología JSF es que los componentes de interfaz de usuario de la página están representados en el servidor como objetos con estado. Esto permite a la aplicación manipular el estado del componente y conectar los eventos generados por el cliente en el lado del servidor. Además, permite convertir y validar entradas sobre componentes individuales y reportar cualquier error antes de que se actualicen los datos en el lado del servidor.

La librería de etiquetas de componentes elimina la necesidad de codificar componentes de interfaz de usuario en HTML u otro lenguaje de marcas, resultando completamente reutilizables. Por otro lado, la librería principal facilita registrar eventos, validadores y otras acciones de los componentes.

Como la mayoría de las tecnologías, JSF tiene un conjunto de términos específicos que definen su base conceptual. El siguiente modelo de dominio muestra los conceptos fundamentales de la tecnología y cómo se comunican las aplicaciones implementadas con JSF. La mayoría de estos conceptos generan un evento o mensaje, elementos a través de los cuales se comunican estas aplicaciones. Los eventos representan las entradas de los usuarios u operaciones de la aplicación y los mensajes indican errores y notificaciones de la aplicación.

[17] API (Interfaz de Programación de Aplicaciones): Conjunto de especificaciones para comunicarse con una aplicación, normalmente para obtener información y utilizarla en otros servicios.

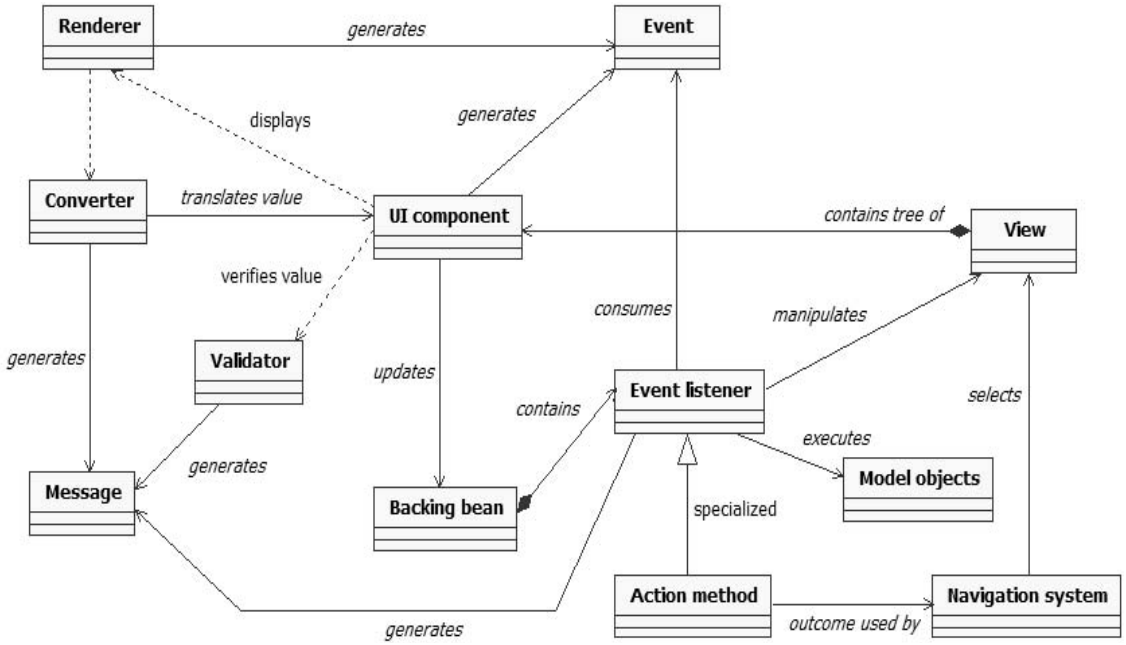


**Figura 1.2-1 Modelo de dominio de JSF [15]**

La arquitectura principal de la tecnología JSF está diseñada para ser independiente de un protocolo específico. Está encaminada a solucionar muchos de los problemas más comunes encontrados en el desarrollo de aplicaciones web para clientes HTML. Se comunica a través del protocolo HTTP[18] con servidores de aplicaciones Java y soporta aplicaciones basadas en servlets y JSP.

### 1.2.3. Servlet y JSP en JSF

Un servlet es un programa escrito en Java que se ejecuta del lado del servidor. Los servlets corren dentro del contexto de un contenedor de servlets (ej: Apache Tomcat, IBM WebSphere, Sun Java System Application Server) y extienden su funcionalidad. También pueden correr dentro de un servidor de aplicaciones (ej: Apache Tomcat) que es además un contenedor para servlets. [8]

El servlet recibe las peticiones que envía el navegador web y genera a partir de ellas una página web. El uso más común de los servlets es generar páginas de forma dinámica desde un servidor web. Otras opciones que permiten generar contenido dinámico son los lenguajes ASP (.Net), PHP, Python y JSP (Java).

[18] HTTP (Protocolo de Transferencia de Hipertexto): Protocolo usado en las transacciones de la Web.

JSP es una tecnología Java, desarrollada por la compañía Sun Microsystems, que permite generar contenido dinámico para la web en forma de documentos HTML, XML[19] o de otro tipo. JSP puede considerarse una manera alternativa y simplificada de los servlets, a los efectos de los desarrolladores.

JSP es una página web con etiquetas especiales y código Java incrustado, cuyo funcionamiento consiste en que el servidor de aplicaciones interpreta el código contenido en la página para construir el código Java del servlet a generar. Este servlet es el que genera el documento web que se presenta en la pantalla del navegador del usuario. Es por esto que se plantea que una página JSP puede hacer todo lo que un servlet puede hacer.

JSF está implementado como un servlet y puede usar páginas JSP como alternativa para la construcción de sus vistas. JSP construye un árbol de componentes, convirtiendo el equivalente de los métodos de un servlet (doGet() y doPost()) a un método jspService(). JSF lo recibe para generar otro árbol de componentes. Cada uno de ellos los crea con sus tags correspondientes. El ciclo de vida para el modelo de componentes JSF es totalmente independiente al producido por el servlet de JSP, lo que provoca una doble interpretación de las páginas en el servidor.

Por otra parte, JSF va más allá de un servlet, ya que este último cubre la infraestructura básica necesaria para construir una aplicación web, manejando las propiedades del protocolo HTTP. Las aplicaciones JSF tienen componentes de interfaz de usuario, que están asociados con clases de Java y pueden generar eventos que son consumidos por la lógica de las aplicaciones.

JSF utiliza el API Servlet para todo su funcionamiento interno. El API Servlet provee una serie de funcionalidades sobre la seguridad, la administración de usuarios, el ciclo de vida de los eventos, los filtros y otros elementos que constituyen la base fundamental de JSF. Todo esto posibilita que los desarrolladores se beneficien y puedan construir aplicaciones web sin tener que preocuparse por las especificidades del protocolo HTTP.

### 1.2.4. Ventajas

Una de las grandes facilidades de la tecnología JSF es que ofrece una clara separación entre el comportamiento y la presentación. Las aplicaciones web construidas con tecnología JSP consiguen parcialmente esta separación. La separación de la lógica de presentación le permite a cada miembro

[19] XML (Lenguaje de Marcas Extensible): Metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una manera de definir lenguajes para diferentes necesidades.

del equipo de trabajo de una aplicación web, centrarse en su parte del proceso de desarrollo y proporciona un sencillo modelo de programación para enlazar todas las piezas.

Otra ventaja de JSF se relaciona con los componentes de interfaz de usuario y la capa web, la cual no se limita a una tecnología de script en particular o un lenguaje de marcas. Aunque la tecnología JSF incluye una librería de etiquetas personalizadas para representar componentes en una página JSP, el API de la tecnología JSF se ha creado directamente sobre el API Servlet. Esto permite usar otra tecnología de presentación con o sin JSP, crear componentes personalizados directamente desde las clases de componentes y generar salida para diferentes dispositivos cliente.

Lo más importante de la tecnología JSF es que proporciona una rica arquitectura para manejar el estado de los componentes, procesar los datos, validar la entrada del usuario y manejar eventos. Una aplicación JSP no puede mapear peticiones HTTP al manejo de eventos específicos de componentes, ni manejar elementos de interfaz de usuario como objetos con estado en el servidor.

### 1.2.5. Acotaciones

Por otra parte, JSF no es recomendable para todas las aplicaciones web. Un simple sitio web dinámico es fácil de implementar y mantener usando sólo servlets y JSP, o sólo JSP y JSP Standard Tag Library (JSTL[20]). El mejor uso de JSF es para una aplicación web con amplia interacción de los usuarios.

JSF no necesariamente reemplaza las tecnologías actuales. Si bien es una tecnología que brinda la estructura y el soporte para aplicaciones de interfaz de usuario, necesita el complemento de AJAX para un mejor rendimiento.

## *1.3. Asynchronous JavaScript and XML (AJAX)*

La Web 2.0 no es una nueva versión de la web, ni un protocolo de comunicaciones, ni un nuevo lenguaje de programación; ni siquiera se trata de algo vinculado únicamente a Internet. La Web 2.0 es una nueva filosofía de hacer las mismas cosas. Por ello no es de extrañar que, en el ámbito tecnológico, los estándares sobre los que se apoyan las aplicaciones y los servicios Web 2.0 existieran desde mucho antes de acuñarse el concepto. [1]

[20] JSTL: Tecnología proporcionada por Sun Microsystems que extiende de JSP proporcionando cuatro librerías de etiquetas (Tag Libraries), con utilidades ampliamente utilizadas en el desarrollo de páginas web dinámicas.

Se plantean tres principios que pueden definir la Web 2.0 y que contribuyen a que los usuarios y muchas empresas exploren nuevas posibilidades en la web. Estos son:

- Comunidad: el usuario aporta contenidos, interactúa con otros usuarios, crea redes de conocimiento, etc.
- Tecnología: un mayor ancho de banda permite transferir información a una velocidad antes inimaginable; en lugar de paquetes de software, se pueden tener servicios web y la terminal puede ser cliente y servidor al mismo tiempo y en cualquier lugar del mundo.
- Arquitectura modular: favorece la creación de aplicaciones complejas de forma más rápida y a un menor costo

Estos principios contribuyen a que los usuarios y muchas empresas exploren nuevas posibilidades en la web.

Entre los conceptos asociados con la Web 2.0 se encuentran las RIA[21], también conocidas como interfaces ricas. Estas se ejecutan en el cliente, es decir, en el navegador de los usuarios y mantienen 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. Todas las RIA comparten una característica: presentan una capa intermedia de código, usualmente llamada motor del cliente, entre el usuario y el servidor. El motor del cliente actúa como una extensión del navegador, el cual asume la responsabilidad de representar la interfaz de usuario de la aplicación para la comunicación con el servidor. Este motor generalmente se implementa utilizando AJAX.

### 1.3.1. Definición y funcionamiento

El término AJAX apareció por primera vez en el artículo “Ajax: A New Approach to Web Applications” publicado por Jesse James Garrett, el 18 de Febrero de 2005. Hasta ese momento, no existía un término normalizado que hiciera referencia al nuevo tipo de aplicación web que estaba apareciendo. En realidad, el término AJAX es un acrónimo de *Asynchronous JavaScript and XML*, que se puede traducir como JavaScript Asíncrono y XML.

[21] RIA (Aplicaciones Ricas en Internet): Formas avanzadas de que un usuario interactúe con una aplicación o página web, ofreciéndole funciones y nuevas posibilidades útiles intentando al mismo tiempo mantener la simplicidad aparente.

El artículo define AJAX de la siguiente forma: “Ajax no es una tecnología en sí mismo. En realidad, se trata de la unión de varias tecnologías que se desarrollan de forma autónoma y que se unen de formas nuevas y sorprendentes.” [4]

AJAX combina un grupo de tecnologías ya conocidas, permitiéndole realizar funciones antes inalcanzables en la web tradicional, las tecnologías que forman AJAX son:

- XHTML[22] y CSS[23], para crear una presentación basada en estándares.
- DOM[24], para la interacción y manipulación dinámica de la presentación.
- XML, XSLT[25] y JSON[26], para el intercambio y la manipulación de información.
- XMLHttpRequest[27], para el intercambio asíncrono de información.
- JavaScript, para unir todas las anteriores tecnologías [19].

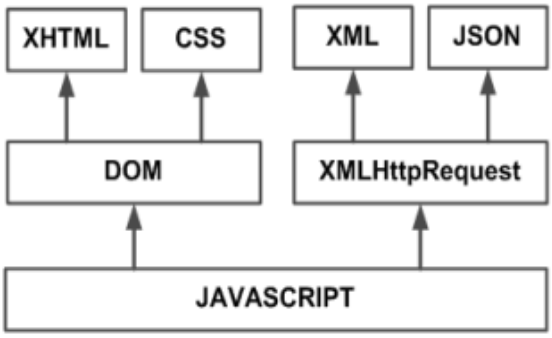


**Figura 1.3-1: Tecnologías asociadas con AJAX [19]**

[22] XHTML (Lenguaje Extensible de Marcas de Hipertexto): Versión XML más avanzada del lenguaje HTML que se utiliza para la creación y visualización de páginas web.

[23] CSS (Hojas de Estilo en Cascada): Lenguaje para definir la presentación de las páginas web, de modo que su aspecto quede separado del contenido en sí.

[24] DOM (Modelo en Objetos para la representación de Documentos): Modelo computacional a través del cual los programas y scripts pueden acceder y modificar dinámicamente el contenido, estructura y estilo de los documentos HTML.

[25] XSLT: Estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML.

[26] JSON (Notación de Objetos para JavaScript): Formato ligero para el intercambio de datos, es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

[27] XMLHttpRequest (Lenguaje de Marcado Extendido/Protocolo de Transferencia de Hipertexto): También referida como XMLHTTP, es una interfaz empleada para realizar peticiones HTTP y HTTPS a servidores WEB.

En las aplicaciones web tradicionales, las acciones del usuario en la página (dar click en un botón, seleccionar un valor de una lista, entre otros) desencadenan llamadas al servidor. Una vez procesada la petición del usuario, el servidor devuelve una nueva página HTML al navegador del usuario.

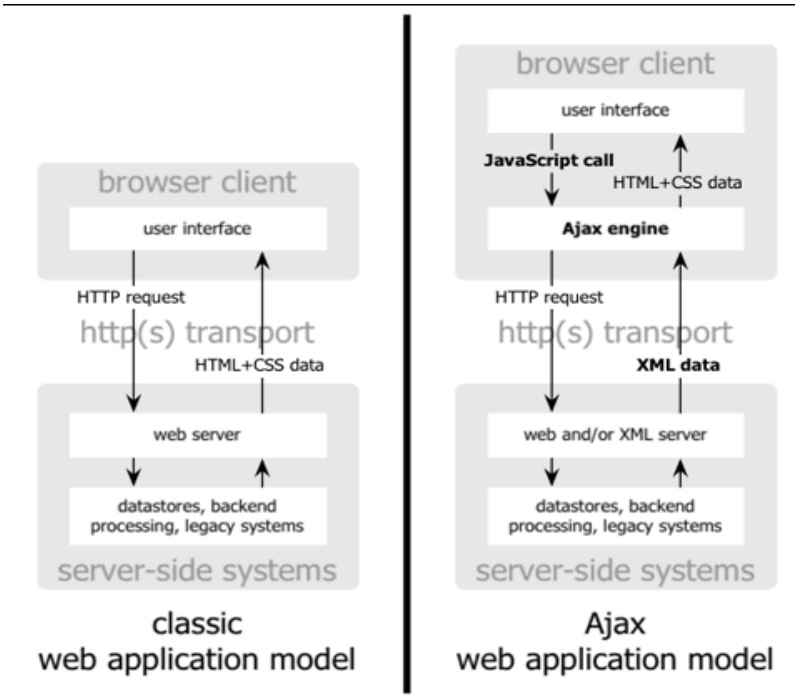


**Figura 1.3-2: Modelo de aplicación web [19]**

Una aplicación AJAX elimina la naturaleza *arrancar-frenar-arrancar-frenar* de la interacción en la web tradicional, ya que introduce entre el usuario y el servidor un motor AJAX. En vez de cargar una página web, al inicio de la sesión, el navegador carga al motor AJAX (escrito en JavaScript). Este motor es el responsable de mostrar la interfaz que el usuario ve y de comunicarse con el servidor en nombre del usuario. El motor AJAX permite que la interacción del usuario con la aplicación suceda asincrónicamente. Así el usuario nunca mira una ventana en blanco en el navegador o un ícono de reloj de arena, esperando a que el servidor emita una respuesta.

Cada acción de un usuario que normalmente generaría un requerimiento HTTP toma la forma de un llamado JavaScript al motor AJAX, en vez de ese requerimiento. Cualquier respuesta a una acción del usuario que no requiera un viaje de vuelta al servidor (como una simple validación de datos, edición de datos en memoria, incluso algo de navegación) es manejada por el motor.

Si el motor necesita algo del servidor para responder (enviar datos para procesar, cargar código adicional y recuperar nuevos datos) hace esos pedidos asincrónicamente, usualmente a través de XML, sin frenar la interacción del usuario con la aplicación.

Cuando el motor AJAX recibe la respuesta, se ejecuta el cambio en la interfaz del usuario basado en la información que recibe. Dado que este proceso disminuye la cantidad de información transmitida, la actualización de la interfaz de usuario es más rápida y el motor puede realizar el trabajo de manera más eficiente.

Cuando se interactúa con una aplicación AJAX lo primero que ocurre es que el usuario provoca un evento. Luego se crea y configura un objeto XMLHttpRequest, el cual realiza una llamada al servidor. Después, el servidor retorna un documento XML (XHTML, JSON) que contiene el resultado y el objeto XMLHttpRequest lo procesa. Finalmente se actualiza el DOM de la página asociado con el resultado devuelto.

### 1.3.2. Características

AJAX tiene numerosas características que hacen que cada día sea más potente, por ejemplo:

- Basada en estándares abiertos: todas las tecnologías que lo conforman son estándares abiertos, excepto el objeto XMLHttpRequest. Es soportada por los navegadores más utilizados en Internet, por ejemplo Internet Explorer, Mozilla y Opera.
- Gran usabilidad: permite a las páginas hacer una petición muy pequeña al servidor y recibirla sin necesidad de recargar la página completa.
- Válida en cualquier plataforma y navegador: resulta muy fácil programar aplicaciones AJAX en los navegadores que tienen el número uno en el mercado de Internet, como son Internet Explorer y Mozilla Firefox; y también se pueden construir aplicaciones web basadas en AJAX que funcionen en otros navegadores.
- Independiente del tipo de tecnología de servidor que se utilice: como mismo AJAX funciona con cualquier navegador, es compatible con cualquier tipo de servidor estándar y lenguaje de programación web, ejemplo: PHP, ASP, ASP.Net, Perl, etc.
- Mejora la estética de la web: se puede combinar toda la imaginación del desarrollador con la usabilidad de una aplicación web de forma tal que si una aplicación no estuviera dentro de un navegador, pudiera pasar por una aplicación de escritorio.

### 1.3.3. Ventajas

A la hora de analizar AJAX como técnica para desarrollar aplicaciones web, es importante destacar las ventajas que determinan su creciente utilización a nivel mundial:

- Tráfico mínimo: las aplicaciones AJAX deben enviar y recibir una pequeña cantidad de información desde y para el servidor, por lo que se minimiza el tráfico entre el cliente y el servidor, asegurándose que no reciba o envíe información innecesaria.
- Interactividad: recuperación asíncrona de datos, es decir el usuario no tiene que esperar después de una petición, la página no se recarga completamente.
- Portabilidad: no requiere plugins[28], es soportada por la mayoría de los navegadores modernos.
- Convenciones establecidas: no gasta tiempo inventando nuevos modelos de interacción con el usuario, utiliza tecnologías ya existentes.
- El usuario primero: diseña las aplicaciones con el usuario en mente.

### 1.3.4. Acotaciones

Se deben mencionar algunas dificultades que trae consigo la utilización de esta técnica.

Las aplicaciones AJAX combinadas con una red de pobre conexión o poco ancho de banda pueden hacer realmente difícil manejar una interfaz de usuario.

AJAX provoca alteraciones en la navegación, es decir, los botones *Atrás*, *Adelante*, *Actualizar* o la opción de *Añadir a Favoritos* no funcionan como se espera de ellos. Sin embargo, con un arduo trabajo de programación se pueden mitigar estas alteraciones.

Un problema de la gran mayoría de las aplicaciones AJAX es la baja compatibilidad entre navegadores, puesto que la capa JavaScript es de una gran complejidad y a menudo, por falta de experiencia en el lenguaje, se opta por programar sólo para Internet Explorer.

No debe usarse AJAX en formularios simples, búsquedas, navegación básica o para sustitución de textos, ya que para todas estas funcionalidades tan pequeñas se emplearía un código JavaScript

[28] plugins: Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.

bastante grande, el cual se ejecuta directamente en el cliente y en vez de ganar en eficiencia, se puede caer en un proceso bastante lento sin necesidad de ello.

Con AJAX es conveniente replantear el modelo de respuesta en la pantalla para que el usuario perciba que el sistema le está atendiendo, de lo contrario podrían aumentar las llamadas al servidor innecesariamente.

AJAX no permite modificar el navegador, ejecutar ficheros musicales, ni acceder a ficheros locales y al hardware. Requiere programadores que conozcan todas las tecnologías que intervienen en AJAX.

Es importante saber que los principales desafíos a la hora de crear aplicaciones AJAX no son técnicos ya que las principales tecnologías que la conforman son estables y bien conocidas. Sin embargo, el verdadero desafío es para los desarrolladores que deben olvidar las limitaciones de la web y comenzar a imaginarse todo con un nivel más amplio de posibilidades.

### 1.3.5. ¿Por qué AJAX y no Flash?

Existen numerosas interrogantes en cuanto a la conveniencia de distintas tecnologías a utilizar para desarrollar una aplicación web interactiva, entre las cuales se destacan AJAX y Flash.

Flash es una herramienta muy buena para diseñadores y animadores. Presenta calidad visual, sobre todo con el avance en el manejo de los textos, animación, multimedia, contenido interactivo, video, audio, así como un lenguaje de programación de alto nivel orientado a objetos. Sin embargo, los usuarios necesitan tener instalada en sus computadoras personales la última versión de Flash para poder verlo. Además, depende de un plugin propietario, por lo que resulta poco accesible.

AJAX está basada en XHTML, es decir, más estandarizable y accesible. Por su integración con CSS es sencillo separar los datos de la presentación. Una página de AJAX con HTML es más ligera que una con Flash. También presenta lenguajes interpretados en el navegador, sencillos y orientados a objetos.

### 1.3.6. Usos de AJAX

Se deben mencionar los principales usos de AJAX:

- Validación de datos de formularios en tiempo real
- Identificación de usuario, números de serie, códigos postales u otro código especial que necesite validación en el lado del servidor antes de ser enviado el formulario

- Autocompletado
- GUI avanzadas
- Control en árbol, menús, barras de progreso
- Refrescador de datos
- Notificaciones del servidor
- Actualización o eliminación de registros
- Expansión formularios web
- Devolución de peticiones simples de búsqueda

AJAX es una de las tecnologías más novedosas, atractivas y prometedoras de la denominada Web 2.0. AJAX reestructura todo lo que se conocía acerca de la manera de relacionarse con Internet, redefiniendo el término interactividad. Actualmente existe un grupo de aplicaciones que utilizan AJAX y validan estas afirmaciones. A continuación se listan algunas de las más conocidas:

- Gestores de correo electrónico: Gmail, Yahoo Mail, Windows Live Mail
- Sistemas de cartografía: Google Maps, Yahoo Maps, Windows Live Local
- Aplicaciones web y metapáginas: Netvibes, Google Docs, Google Personalized Home
- Otras: Digg (noticias), Meebo (mensajería), 30 Boxes (calendario), Flickr (fotografía)

Es conveniente destacar que la técnica AJAX se debe utilizar apoyada por un framework que le permita abstraerse de la tecnología subyacente y apropiarse de las funcionalidades que este ofrece.

## *1.4. JSF + AJAX*

Después de haber analizado ambas tecnologías por separado, a continuación se describe una panorámica sobre la integración de AJAX a JSF. AJAX suple la comunicación con el servidor, por lo que esta pasa a ser asincrónica. Las facilidades de validaciones de JSF se enriquecen pues con AJAX la mayoría de ellas se ejecutan en el cliente y no el servidor. AJAX, además, aporta una interfaz más rica para los componentes de JSF. Esta integración puede resultar complicada en el momento de su implementación si no se usa algún framework que permita abstraerse del código JavaScript necesario para la misma.

En la actualidad se han desarrollado varios frameworks y librerías de componentes que implementan esta integración. De ellos se han identificado veintisiete, entre los más conocidos están:

- Tobago
- Trinidad
- RCFaces
- Woodstock
- WebGalileoFaces
- ZK
- QuipuKit
- ICEFaces
- RichFaces

Teniendo en cuenta la difusión, la accesibilidad, su compatibilidad y la comunidad de desarrolladores que poseen, se seleccionan cinco para profundizar en sus principales funcionalidades:

- **WebGalileoFaces**: desarrollado por JSCAPE y SoftAspects. Inicialmente todos sus componentes estaban bajos licencia privativa, pero en la actualidad están liberados como open-source bajo la Licencia Apache y sus desarrolladores sólo comercian el soporte de los mismos. Tiene componentes web de interfaz de usuario altamente reusables y personalizables, para el desarrollo de aplicaciones web basadas en Java. Provee las herramientas necesarias para facilitar la captura y respuesta a eventos de interfaz de usuario en el navegador, con completo soporte para la especificación JSF de Sun MicroSystem y para AJAX.
- **ZK**: framework de aplicaciones web con AJAX, basado en componentes y eventos, incluyendo un motor de AJAX para manejar los eventos. Está desarrollado completamente en Java sobre código abierto y permite una rica interfaz de usuario para aplicaciones web sin usar JavaScript y con poca programación. Tiene un rico conjunto de componentes XUL[29] y XHTML y un

[29] XUL (Lenguaje basado en XML para la Interfaz de Usuario): Aplicación de XML a la descripción de la interfaz de usuario en el navegador Mozilla.

lenguaje llamado ZUML[30] (ZK User Interface Markup Language), por lo que no se necesita tener conocimientos de Java para usar ZK, solo los básicos para completar la aplicación.

- **QuipuKit**: incluye un conjunto extendido de componentes de JSF para la construcción de interfaces ricas de usuario web y un framework para validaciones que cambia la lógica tradicional de validación de JSF del lado del cliente.
- **ICEFaces**: desarrollado por la compañía IceSoft, es una librería de componentes JSF con AJAX incorporado. Tiene muchos componentes con funcionalidades avanzadas con soporte para estilos de interfaces. Tiene incluso 17 efectos que pueden ser configurados en otros componentes visuales. Contiene además 9 tipos de diagramas y soporte para componentes *drag and drop*[31].
- **RichFaces**: desarrollado por Jboss con una enorme librería de componentes ricos y con soporte para *skinnability*[32]. Es un framework de código abierto que adiciona capacidades AJAX a las aplicaciones JSF existentes, sin necesidad de escribir código JavaScript. Se integra completamente al ciclo de vida de JSF y permite escribir sus propios componentes con AJAX incluido.

Cada uno de estos se analiza con mayor detenimiento en el Capítulo 2, con el objetivo de definir cuál es el más eficiente para lograr la integración de AJAX a JSF.

## *1.5. Ambiente de desarrollo*

En la producción de software es importante establecer el ambiente de desarrollo (Development Environment) porque es donde se definen el conjunto de herramientas, tecnologías, versiones a utilizar y su integración, que más tarde intervienen en el proceso de desarrollo de software.

A continuación se presenta el conjunto de herramientas utilizadas en el desarrollo de este trabajo. Ellas son: un entorno de desarrollo integrado y sus plugins, un contenedor web, un control de versiones y los frameworks. Se exponen algunas de sus características y ventajas.

---

[30] ZUML: Lenguaje de Marcado para la definición de una potente Interfaz de Usuario soportado por ZK.

[31] drag and drop: Término muy usado en informática que se refiere a la acción de arrastrar y soltar con el ratón objetos de una ventana a otra o entre partes de una misma ventana o programa.

[32] skinnability: Propiedad que permite personalizar una serie de elementos gráficos que, al aplicarse sobre un determinado software, modifican su apariencia externa.

### 1.5.1. Entorno de desarrollo integrado

Un ambiente o entorno de desarrollo integrado (Integrated Development Environment, IDE) es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse exclusivamente a un sólo lenguaje de programación o puede utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solos o pueden ser parte de aplicaciones existentes. Estos proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Java, C#, Visual Basic, Object Pascal, etc.

#### Eclipse

Metafóricamente, Eclipse es como una tienda para herreros, donde no solamente se hacen productos, sino que además se hacen las herramientas para hacer los productos. Cuando se descarga el Eclipse SDK, se obtiene un equipo de instrumentos para desarrollo en Java o Java Development Toolkit (JDT) para escribir y depurar programas en Java. Además, se obtiene un ambiente de desarrollo de plugin o en inglés Plug-in Development Enviroment para heredar de Eclipse. Si todo lo que se quiere es un IDE para Java, no se necesita nada más que el JDT. Esto es para lo que la mayoría las personas usan Eclipse. [2].

Aunque Eclipse está implementado en Java y su principal uso es como IDE para Java, este es un IDE neutral. El soporte para desarrollo en Java es proveído por un componente enchufado o plugin, pero además están disponibles plugins para otros lenguajes, como C/C++, Cobol, C#.

En principio, Eclipse permite ejecutar un programa sobre cualquier plataforma. Es una plataforma extensible de código abierto para desarrollar herramientas. Es administrado y dirigido por un consorcio de compañías de desarrollo de software con un interés comercial en promover Eclipse como plataforma compartida para herramientas de desarrollo de software.

##### 1.5.1.1. Plugins

Un plugin es una aplicación que interactúa con otra para agregarle una funcionalidad específica y es ejecutada por la aplicación principal. En el caso particular de Eclipse es un conjunto de clases que permite hacer el IDE más extensible. [10]

### Web Tool Platform (WTP)

La plataforma de herramientas web de Eclipse o Eclipse Web Tool Platform (WTP) es un plugin de código abierto, que provee varias APIs para desarrollo de aplicaciones sobre la web y JEE. Estas incluyen editores gráficos, de código fuente para una variedad de lenguajes, asistentes y aplicaciones incorporadas para simplificar el desarrollo de servicios web, además de herramientas y APIs para soportar el despliegue, ejecución y prueba de aplicaciones.

Soporta integración con servidores Web dentro de Eclipse como ambiente de ejecución de primera clase para aplicaciones web. También incluye la configuración de servidores y su asociación con los proyectos web, permitiendo la depuración sobre el servidor de los recursos y las clases.

### Spring IDE

Spring IDE es un plugin que sirve como interfaz de usuario gráfica para la configuración de los archivos usados por Spring Framework. Permite el completamiento de etiquetas, valores de atributos y elementos en estos archivos de configuración. Analiza gramaticalmente expresiones de punto de cortes de AspectJ (AspectJ Pointcut Expressions) y @AspectJ-style pointcut expressions, mostrando errores si están incorrectas sintácticamente.

Dicho plugin muestra un árbol con todos los proyectos de Spring y sus archivos de configuración. Permite hacer búsquedas de beans en dichos archivos. Muestra gráficamente los beans y sus dependencias. Presenta un editor gráfico con completamiento y validaciones para los archivos de definiciones del flujo web usado por el Spring Web Flow.

### Subclipse

Subclipse es un plugin para Eclipse que adiciona integración para el control de versiones (específicamente Subversion), permitiendo operaciones de sincronización, actualización, entre otras. Permite bloqueos a recursos para que otros usuarios no puedan modificarlo. Dispone de una vista de comparación entre el recurso local y remoto, en caso de que exista conflicto entre la versión del recurso local con el remoto. Muestra una vista del historial de versiones de los recursos con un conjunto de atributos de las acciones realizadas sobre el recurso. Soporta conectarse a varios repositorios de control de versiones al mismo tiempo, permitiendo hacer operaciones sobre el repositorio directamente.

Es un plugin muy útil para el desarrollo colaborativo, en el que intervienen un conjunto de desarrolladores trabajando sobre el mismo proyecto, poniendo a disposición del equipo de desarrollo facilidades para el trabajo en equipo.

## 1.5.2. Contenedor Web

Los servidores de aplicaciones se crean con el fin de gestionar las peticiones del usuario y devolverle recursos a través de un protocolo de comunicación (HTTP, sirve para incursionar en los sitios de www en Internet).

Sin embargo en el mundo de JEE surge un término muy común, contenedor web, el cual además de interceptar solicitudes enviadas a los protocolos: HTTP, HTTPS[33], FTP[34], y otros, es esencialmente un período de ejecución Java que proporciona una implementación del API Servlet y facilidades para las páginas JSP. Además, es responsable de inicializar, invocar, y gestionar el ciclo de vida de los servlets Java y las páginas JSP.

### Apache Tomcat

Apache Tomcat es un contenedor de servlet usado en la implementación de referencia oficial para las tecnologías Java Servlet y JSP. Es desarrollado en un ambiente participativo y abierto. Apache Tomcat es usado en numerosas aplicaciones web de gran escala y en diversas industrias y organizaciones que se referencian en su sitio oficial. [11]

## 1.5.3. Control de versiones

El control de versiones es la gestión de versiones o revisiones de todos los elementos de configuración que forman la línea base de un producto o una configuración del mismo. Los sistemas para el control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, junto a las posibles especializaciones realizadas para algún cliente específico. Sin embargo, los mismos

[33] HTTPS: Protocolo de red basado en el protocolo HTTP, destinado a la transferencia segura de datos de hipertexto.

[34] FTP: Protocolo de transferencia de archivos entre sistemas conectados a una red TCP basado en la arquitectura cliente-servidor.

conceptos son aplicables a otros ámbitos y no sólo para código fuente sino para documentos, imágenes, archivos XML, XSD[35], DTD[36], JSP, HTML, entre otros.

Un sistema de control de versiones proporciona un mecanismo de almacenamiento de cada uno de los recursos que deba gestionarse (archivos de texto, imágenes, documentación, etc.), la posibilidad de modificar, mover, borrar cada uno de los elementos, un historial de las acciones realizadas con cada elemento pudiendo volver a un estado anterior dentro del mismo.

Aunque un sistema de control de versiones puede realizarse de forma manual, es aconsejable disponer de herramientas que faciliten esta gestión, por ejemplo, Subversion.

### Subversion

Subversion es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS[37], el cual posee varias deficiencias. Subversion es un software libre distribuido bajo una licencia de tipo Apache/BSD y se le conoce también como *svn* por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.

### TortoiseSVN

Es un cliente Subversion, implementado como una extensión al *shell*[38] de Windows. Es un software libre, liberado bajo la licencia GNU/GPL. Puede ser usado sin un entorno de desarrollo. Tiene pequeñas imágenes que decoran los íconos de los archivos mostrando los archivos o directorios que necesitan ser enviados al repositorio. Maneja la diferencia de interfaz de los documentos de Office tales como los creados con Microsoft Word.

---

[35] XSD (XML Schema Definition): Concebida como alternativa a las DTD.
[36] DTD (Definición de Tipo de Documento): Descripción de estructura y sintaxis de un documento XML.
[37] CVS (Concurrent Version System o Concurrent Versioning System): Sistema de control de versiones.
[38] shell: Intérprete de línea de comandos.

### 1.5.4. Frameworks

Recientemente, el interés en reutilizar software ha cambiado de la reutilización de componentes simples a diseño de sistemas enteros o estructuras de aplicaciones. Un software que pueda ser reutilizado en este nivel para la creación de aplicaciones completas es llamado framework. Los frameworks permiten la producción fácil de un conjunto de sistemas específicos pero similares, dentro de un cierto dominio, comenzando desde una estructura genérica. Los frameworks son arquitecturas genéricas integradas por un extensible conjunto de componentes. Además, los frameworks pueden contener subframeworks los cuales representan subconjuntos de componentes de un sistema más grande. [7]

Los frameworks juegan el rol de control entre la aplicación y la infraestructura. En sistemas convencionales, los propios desarrolladores de programas proveen toda la estructura y control del flujo de ejecución y realizan llamadas a librerías de funciones como sea necesario; pero usando un framework, los desarrolladores tienen solamente que preparar estos componentes que no están en el framework acorde al comportamiento deseado de la aplicación.

Un framework contiene clases o estructuras que implementan los componentes de una aplicación genérica también como componentes concretos que satisfacen tareas especializadas. Para desarrollar programas completos, los desarrolladores buscan e instancian los componentes apropiados.

No hay una definición única para los frameworks, pero la mayoría tiene un tema común: la reutilización. Una definición ampliamente aceptada es la que plantean R. E. Jonson y B. Foote en su publicación, en 1988. "Un framework es un conjunto de clases que personifican un diseño abstracto para soluciones de una familia de problemas relacionados...". [6]

#### JSF Framework

A partir de la JSR 252 de JCP que define la versión 1.2 de JSF se han liberado un grupo de implementaciones. En este caso se utiliza el framework JSF 1.2 implementado por la Sun MicroSystem. (Ver epígrafe 1.2)

# Capítulo 2: Propuesta de integración

En este capítulo se valoran las variantes de integración de AJAX a JSF a partir del funcionamiento de ambas. También se profundiza en cinco frameworks que permiten integrar AJAX a JSF. Para el análisis de cada uno de los frameworks se tienen en cuenta cinco aspectos: librería de componentes, compatibilidad, configuración, aplicación en la construcción de las vistas e integración a los componentes estándares de JSF. Luego se describe el framework Facelets debido a su importancia y pertinencia para la presente tesis. Finalmente, se ofrece una propuesta de integración para implementar en la capa de presentación del proyecto Kainos.

## *2.1. Integrando AJAX a JSF*

Agregar funcionalidades de AJAX a componentes de JSF permite proporcionarle a una aplicación todas las riquezas de ambos elementos. Para lograr esto de la mejor forma se analizan algunas características del funcionamiento y ciclo de vida de ambas.

AJAX es un simple proceso de tres pasos [3]:

- Se invoca una URL desde el código JavaScript en el cliente.
- Se maneja la URL en el servidor y se escribe la respuesta.
- Cuando la respuesta está completa se integra al DOM de la página en el cliente.

Estos pasos, como proceso general, tienen un comportamiento similar al que ejecuta JSF al recibir una petición del cliente. JSF maneja una petición en el servidor para finalmente sobrescribirla con la respuesta. Lo único que no cumple AJAX es el último paso de sobrescribir la respuesta, lo que quiere decir que en una petición AJAX no se refresca la página, sólo una parte de ella. Esta sutil diferencia permite realizar toda clase de interacciones interesantes en las aplicaciones web.

El ciclo de vida de JSF muestra con más detalles todo el proceso que transita una petición en este framework, una vez llegada al servidor. Para trabajar dentro del ciclo de vida de JSF, un cliente envía un objeto de tipo XMLHttpRequest a una URL que tiene el escuchador del servlet de JSF. Al llegar al servlet entra en las siguientes fases [9]:

- ***Restore view***: se construye un árbol con los componentes de la vista a partir de los datos de la petición o de los datos salvados en el servidor.

- ***Apply request values***: se le asignan los valores que vienen en la petición a los componentes en el árbol creado.
- ***Process validation***: se validan los valores que se asignaron a los componentes desde la petición.
- ***Update model values***: se actualizan las propiedades del modelo de la aplicación mediante el enlace con los componentes del árbol de la vista, usando la propiedad *values binding*.
- ***Invoke application***: los escuchadores de eventos invocan a métodos externos para procesar los datos actualizados.
- ***Render response***: la respuesta es enviada a la petición original.

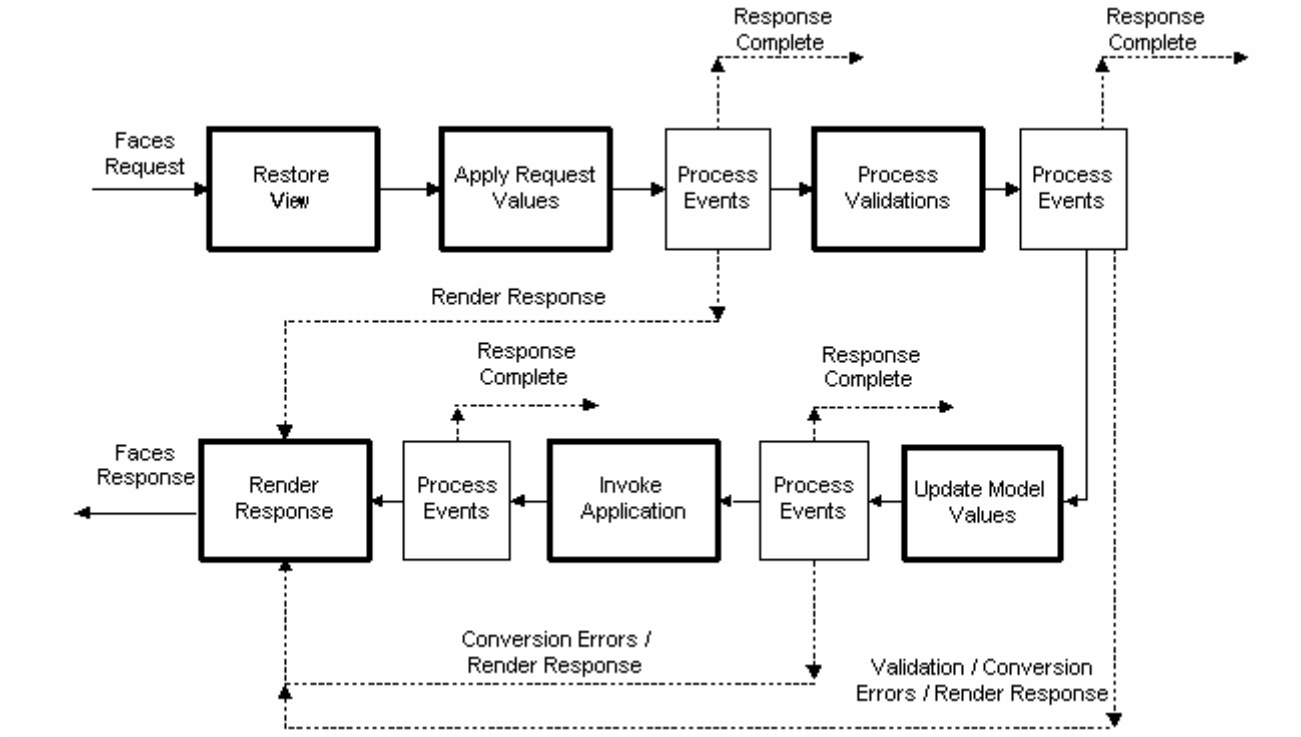


**Figura 2.1-1: Ciclo de vida de JSF [18]**

Después de cada fase los escuchadores de eventos son llamados. Estos pueden continuar normalmente el ciclo de vida, reportar errores y saltar a la última fase o generar ellos mismos la respuesta necesaria.

El ciclo de vida de JSF antes descrito ofrece ventajas para tratar dinámicamente el estado de los componentes y el procesamiento de los datos, así como para manejar los eventos que se generan

durante el funcionamiento de una aplicación. Por tanto, para obtener una propuesta de integración óptima JSF debe tomar de AJAX las ventajas de actualizaciones parciales de las páginas, que incrementan el rendimiento y la usabilidad de una aplicación, sin renunciar a ninguna de las bondades de JSF.

Existen varias formas de integrar AJAX a JSF [9]:

- Usar un servicio externo que manipule el objeto XMLHttpRequest. Esta estrategia evita el ciclo de vida de JSF.
- Usar un ciclo de vida externo para manejar las peticiones de AJAX.
- Usar un componente de JSF personalizado para procesar el código JavaScript y los objetos XMLHttpRequest de AJAX. En este caso la petición es procesada a través del ciclo de vida de JSF, como una petición más.

Utilizando las potencialidades de JSF y retomando la idea de una verdadera integración, se debe tener en cuenta como lógica a seguir la tercera variante de las antes mencionadas. La ventaja fundamental de esta propuesta está en la posibilidad de procesar las peticiones que entrega AJAX a través del mismo ciclo de JSF, logrando así que todo el proceso de peticiones al servidor funcione dentro de un único ciclo.

## *2.2. Análisis de frameworks*

Para analizar los frameworks seleccionados en el capítulo anterior se tuvieron en cuenta los siguientes aspectos:

- **Librería de componentes**: cantidad de componentes que la forman, así como la funcionalidad y las particularidades de los mismos.
- **Compatibilidad**: posibilidad de integrarse con otros frameworks, navegadores o IDEs.
- **Configuración**: pasos necesarios para poder integrarse al framework JSF.
- **Aplicación en la construcción de las vistas**: elementos necesarios a tener en cuenta para la programación de las páginas de una aplicación.
- **Integración a los componentes estándares de JSF**: posibilidad de agregarle propiedades AJAX a los componentes propios de JSF.

### 2.2.1. WebGalileoFaces

El framework WebGalileoFaces contiene tres subsistemas principales:

- ***core***: incluye las clases esenciales del núcleo del framework.
- ***tags***: da soporte a todas las implementaciones de los tags personalizados.
- ***faces***. hereda de las clases de core e implementa interfaces específicas de JSF.

Todos los componentes están construidos usando el patrón MVC. La separación de las capas permite la rápida modificación de todo el comportamiento de un componente, con mínimos cambios en su estructura.

Cada componente tiene un conjunto de eventos y cada tipo de evento puede ser manejado por el escuchador de eventos correspondientes. Todos los eventos heredan de *FacesEvents* y todos los escuchadores heredan de *FacesListener;* ambas clases de JSF, lo que permite una mejor integración. Igualmente todos los componentes que soportan AJAX tienen un *PhaseListener* predefinido que les posibilita adaptarse al ciclo de vida de JSF.

#### 2.2.1.1. Librería de componentes

La integración de componentes WebGalileoFaces con AJAX provee el soporte para las fases del estándar JSF como decodificación, validación, actualización de valores e invocación de componentes y escuchadores.

Los componentes están diseñados con un modelo de interfaces predefinido, el cual tiene un grupo de funcionalidades ya implementadas. Una de las ventajas principales de este framework es que los desarrolladores pueden heredar fácilmente de su implementación o re-implementar cualquiera de sus componentes para obtener nuevas funcionalidades. También pueden agregar un manejador de eventos del lado del servidor, implementando escuchadores de eventos específicos y agregándolos a los componentes. Además, los componentes pueden utilizar las reglas de navegación de JSF y estos permiten ser agregados al árbol de componentes en tiempo de ejecución.

Para utilizar los *beans*[39] *manejados* de JSF, se requiere de la creación de una clase modelo, la definición del *bean* correspondiente en los archivos de configuración de JSF y la especificación de

[39] beans: Son simples clases de Java con sus métodos de acceso.

directivas en las páginas JSP. Esto adiciona trabajo a los desarrolladores para el uso del framework, lo cual constituye una desventaja del mismo.

WebGalileoFaces cuenta con un amplio grupo de componentes, en su mayoría comunes con otros frameworks. Se distinguen especialmente los componentes *Charts* y *FlowChart*, que son utilizados para gráficos con muestra de datos y gráficos de flujo que se emplean en el seguimiento de procesos. La posibilidad de personalizar la propiedad *skinability* brinda grandes facilidades para los desarrolladores.

#### 2.2.1.2. Compatibilidad

WebGalileoFaces permite hacer más fácil la creación de RIAs con integración en tiempo de diseño en IDEs como Sun Java Studio Creator, IBM Rational Application Developer, Oracle JDeveloper y Eclipse. La posibilidad de personalizar la propiedad *skinability* brinda grandes facilidades para los desarrolladores.

WebGalileoFaces está creado preferentemente para la implementación JSF de Sun, específicamente para la versión 1.1, pero igualmente se podrían usar sus componentes con alguna implementación JSF que siga estrictamente esta especificación, como la implementación MyFaces de Apache. Se pueden usar las librerías en proyectos basados en Struts, así como también se pueden integrar sus componentes con Tiles y con Cocoon[40]. No solo se limita a usar tecnología JSP, también soporta el Framework Facelets. De igual manera brinda soporte para el uso de la tecnología Portlets[41].

#### 2.2.1.3. Configuración

Para usar WebGalileoFaces se debe ubicar en el directorio *lib* de la aplicación web los siguientes ficheros: *wgf-faces-X-X.jar*, *wgf-core-X-X.jar*, *wgf-tags-X-X.jar*, donde X-X es la versión del producto.

WebGalileoFaces contiene recursos web (componentes JavaScript, imágenes, recursos HTML, *Webgalieo.key*) que deben incluirse en cualquier aplicación web que use los componentes. Para ello se debe descompactar el archivo *resources.zip* en el directorio raíz de la aplicación. Dentro de este archivo se encuentran dos archivos más: uno llamado *WEB-INF,* donde se localizan el recurso *webgalieo.key* junto con otros ficheros de configuración; y otro llamado *repositorio* donde se localizan

[40] Cocoon (Apache Cocoon): Framework de desarrollo web que se enfoca en la publicación de XML y XSLT y está construido usando el lenguaje de programación Java.

[41] Portlets: Componentes modulares de interfaz de usuario gestionados y visualizados en un portal web.

los recursos web. Los recursos antes mencionados siempre deben corresponderse con la configuración del fichero *web.xml.*

Después de ubicar estos recursos web (usados por los componentes y necesarios para su implementación) dentro del directorio de la aplicación, se debe especificar la posición de los mismos editando las siguientes entradas en el fichero *web.xml* (todos parámetros obligatorios):

- *Init param – applicationRootContext*: muestra el contexto de la raíz de la aplicación web.
- *Init param – pathToImages:* señala el directorio de imagen dentro de la aplicación incluyendo el camino de la raíz.
- *Init param – pathToJavaScriptsExplorer:* señala los componentes JavaScript de Internet Explorer dentro de la aplicación incluyendo el camino de la raíz.
- *Init param – pathToJavaScriptsNetscape:* señala los componentes JavaScript de Netscape dentro de la aplicación incluyendo el camino de la raíz.
- *FileOrImageUploadServlet*: servlet usado en el componente *HtmlEditor*
- *CacheImageFilter* filtro de imágenes a caché para todos los componentes.

Otros ficheros a configurar además de *faces-config.xml* y que merecen atención son (todos de configuración obligatoria):

- *face-config.xm*l: configuración principal para los componentes
- *components-faces-config.xml:* configuraciones para componentes específicos
- *panelbar-face-config.xml*: configuraciones para el componente *PanelBar*
- *popupwindow-face-config.xml*: configuraciones para componentes *Popup*
- *flex-menu-faces-config.xml*: configuraciones para componentes *FlexMenu*
- *flowchart-faces-config.xml*: configuraciones para el componente *Flochart*
- *gmaps-faces-config.xml*: configuraciones para el componente *GMaps*
- *progressbar-faces-config.xml:* configuraciones para el componente *ProgressBar*
- *chart-faces-config.xml*: configuraciones para el componente *Charts*
- *dragarea-faces-config.xml:* configuraciones para soporte *drag-and-drop*

Como se muestra, la configuración necesaria para poder utilizar los componentes de WebGalileoFaces se hace realmente engorrosa para los desarrolladores, ya que hay que declarar en los archivos de configuración de JSF cada uno de los componentes que se usen. El fichero *web.xml* quedaría como se muestra:

```
<context-param>
        <param-name>javax.faces.CONFIG_FILES</param-name>
        <param-value>
            /WEB-INF/components-faces-config.xml,
            /WEB-INF/panelbar-faces-config.xml,
            /WEB-INF/popupwindow-faces-config.xml,
            /WEB-INF/progressbar-faces-config.xml,
            /WEB-INF/flex-menu-faces-config.xml,
            /WEB-INF/gmaps-faces-config.xml,
            /WEB-INF/flowchart-faces-config.xml,
            /WEB-INF/dragarea-faces-config.xml,
            /WEB-INF/chart-faces-config.xml,
            /WEB-INF/test-app-validators-config.xml,
            /WEB-INF/test-app-managed-beans-config.xml
        </param-value>
</context-param>

<context-param>
        <param-name>tree.control.images</param-name>
        <param-value>/images</param-value>
</context-param>

<filter>
        <filter-name>ExtraSpaceFilter</filter-name>
        <filter-class>
            com.jscape.framework.galileo.support.filters.ExtraSpaceRemover
        </filter-class>
</filter>

<filter>
        <filter-name>zipFilter</filter-name>
        <filter-class>
            com.jscape.framework.galileo.support.filters.GZIPFilter
        </filter-class>
</filter>

<filter>
        <filter-name>CacheImageFilter</filter-name>
        <filter-class>
            com.jscape.framework.galileo.support.filters.IEFlickerFixFilter
        </filter-class>
</filter>

<filter>
        <filter-name>uploadFilter</filter-name>
        <filter-class>
            com.jscape.framework.galileo.support.upload.UploadFilter
        </filter-class>
```

```
        <init-param>
            <param-name>maxFileSize</param-name>
            <param-value>1024</param-value>
            <description>max file size (KB)</description>
        </init-param>
 </filter>

<filter-mapping>
        <filter-name>CacheImageFilter</filter-name>
        <url-pattern>*.gif</url-pattern>
</filter-mapping>

<filter-mapping>
        <filter-name>CacheImageFilter</filter-name>
        <url-pattern>*.jpg</url-pattern>
</filter-mapping>

<filter-mapping>
        <filter-name>CacheImageFilter</filter-name>
        <url-pattern>*.png</url-pattern>
</filter-mapping>

<filter-mapping>
        <filter-name>uploadFilter</filter-name>
        <url-pattern>*.faces</url-pattern>
</filter-mapping>

<servlet>
        <servlet-name>ComponentsInitializerServlet</servlet-name>
        <servlet-class>
             com.jscape.framework.galileo.support.base.ComponentsInitializerServlet
        </servlet-class>
        <init-param>
            <param-name>applicationRootContext</param-name>
            <param-value>/webgalileofaces</param-value>
        </init-param>
        <init-param>
            <param-name>pathToImages</param-name>
            <param-value>/webgalileofaces/repository/images/</param-value>
        </init-param>
        <init-param>
            <param-name>pathToJavaScriptsExplorer</param-name>
            <param-value>/webgalileofaces/repository/scripts/</param-value>
        </init-param>
        <init-param>
            <param-name>pathToJavaScriptsNetscape</param-name>
            <param-value>/webgalileofaces/repository/scripts/ns/</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
</servlet>

<servlet>
        <servlet-name>FileOrImageUploadServlet</servlet-name>
        <servlet-class>
              com.jscape.framework.galileo.support.base.Upload
```

```
        </servlet-class>
        <init-param>
            <param-name>uploadDir</param-name>
            <param-value>/upload</param-value>
        </init-param>
 </servlet>

<servlet>
      <servlet-name>DisplayChart</servlet-name>
      <servlet-class>org.jfree.chart.servlet.DisplayChart</servlet-class>
</servlet>

<servlet-mapping>
        <servlet-name>FileOrImageUploadServlet</servlet-name>
        <url-pattern>/UploadServlet</url-pattern>
</servlet-mapping>

<servlet-mapping>
      <servlet-name>DisplayChart</servlet-name>
      <url-pattern>/chart/*</url-pattern>
</servlet-mapping>
```

#### 2.2.1.4. Aplicación en la construcción de las vistas

Para la construcción de las vistas utilizando los componentes de WebGalileoFaces es importante tener en cuenta que se debe declarar la directiva JSP correspondiente en el inicio de la página JSP para cada uno de los componentes que se usarán. Se puede usar cualquier prefijo para el nombre de la etiqueta. Por ejemplo, si se fueran a usar los componentes *tabbedPanel*, *table*, *toolbar* y *tree* las directivas a declarar serían*:*

```
<%@ taglib uri="/WEB-INF/tabbedPanel.tld" prefix="tabs" %>
<%@ taglib uri="/WEB-INF/table.tld" prefix="table" %>
<%@ taglib uri="/WEB-INF/toolbar.tld" prefix="toolbar" %>
<%@ taglib uri="/WEB-INF/tree.tld" prefix="tree" %>
```

Usar los componentes de este framework implica declarar en las páginas JSP una directiva por cada uno de los componentes a utilizar, lo cual hace más trabajoso la construcción de las páginas.

A continuación se muestra un ejemplo de código e imagen del componente *chart* del framework:

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib prefix="chart" uri="http://www.jscape.com/creator/chart"%>

<f:subview id="charBasicFeatures">
    <h:form id="form1">
        <h:panelGrid id="grid1" columns="2">
            <chart:chart id="chart1"
```

```
                        datasource="#{chartBean.pieChartSource}"
                        type="PIE"
                        alt="Sampe pie chart"
                        chartTitle="Pie chart"
                        colors="red,green,blue,cian,magenta,yellow"
                        is3D="false"
                        startAngle="290"/>
            <chart:chart id="chart2"
                        datasource="#{chartBean.pieChartSource}"
                        type="PIE"
                        alt="Sampe pie 3D chart"
                        chartTitle="Pie 3D chart"
                        colors="red,green,blue,cian,magenta,yellow"
                        is3D="true"
                        startAngle="290"
                        depth="30"
                        alpha="50"
                       output="JPEG"/>
      </h:panelGrid>
    </h:form>
</f:subview>
```

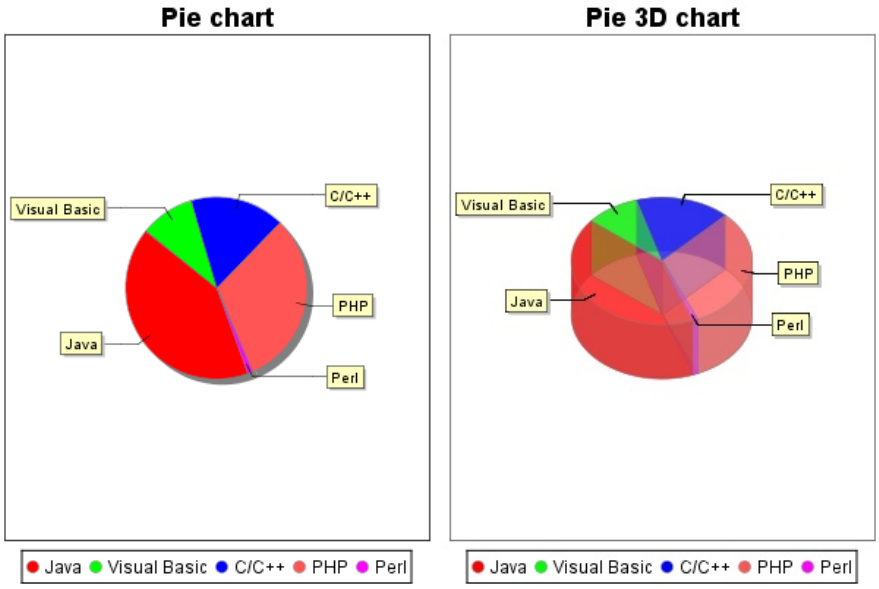


**Figura 2.2-1: Componente chart [27]**

### 2.2.1.5. Integración a los componentes estándares de JSF

Los componentes permiten manejar eventos adicionales como son:

- *ajaxClientHandle*: función de usuario que debe ser llamada antes de enviar la petición AJAX

- *ajaxListener*: escuchador del lado del servidor que debe ser llamado en el propio servidor

Estas propiedades sólo son válidas para el grupo de componentes que propone WebGalileoFaces, no permiten integrarle AJAX a los componentes propios de JSF, sólo sus componentes son construidos como componentes JSF integrados con AJAX. Para permitir operaciones AJAX se está obligado a establecer la propiedad *ajaxEnabled* = *true*.

### 2.2.2. ZK

ZK representa su aplicación en componentes XUL y XHTML y manipula los eventos provocados por la actividad del usuario, de modo similar a lo que se hace en las aplicaciones de escritorio. A diferencia de la mayor parte de otros frameworks, en este caso AJAX es una tecnología en la parte trasera de la escena. La sincronización del contenido del componente y el flujo de eventos están hechos automáticamente por el motor ZK.

Además de un modelo simple y ricos componentes, ZK también soporta un lenguaje de marcado llamado ZUML. ZUML, como XHTML, le permite a los desarrolladores diseñar interfaces de usuario sin programar. ZUML, con espacios de nombres XML, integra en una sola pieza un conjunto diferente de etiquetas en la misma página. Actualmente ZUML soporta 2 conjuntos de etiquetas, XUL y HTML. Usar ZK incluye tiempo de estudio del lenguaje ZUML y sus particularidades, lo cual implica una desventaja para el uso de este framework.

El lenguaje ZUML permite a los desarrolladores incrustar expresiones EL[42] y códigos script en sus lenguajes favoritos como: Java, JavaScript, Ruby y Groovy. A diferencia del JavaScript incrustado en HTML, ZK ejecuta todo código script incrustado en el servidor.

El mecanismo basado en AJAX construido por ZK consta de 3 partes: el cargador ZK, el motor ZK AU y el motor del cliente ZK. En la figura se muestra el funcionamiento de este framework a través de un diagrama:

[42] EL (Expression Language): Facilita el acceso a los datos almacenados en una aplicación en los componentes JavaBeans.

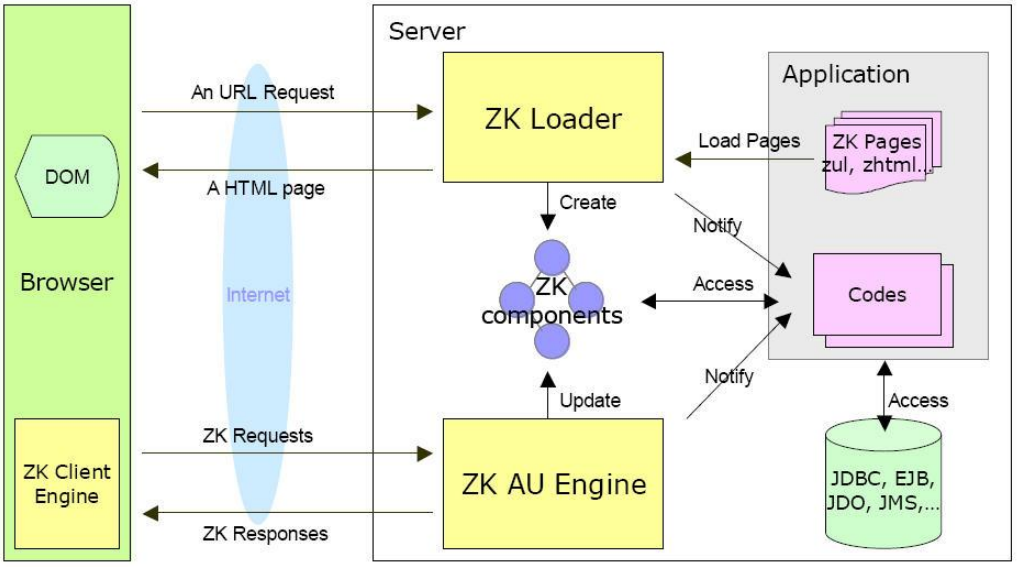


**Figura 2.2-2: Arquitectura de ZK [29]**

### 2.2.2.1. Librería de componentes

Dentro del gran número de componentes de ZK hay algunos que poseen funcionalidades muy usadas hoy en día en las aplicaciones web, que por tanto resultan útiles y permiten reducir tiempo de desarrollo. El componente *Captcha* es usado para saber quién trata de autenticarse en la aplicación, si es un humano o una máquina. Proponen además, un componente usado como editor de HTML-online, llamado *FCKeditor*. Poseen un componente paginador llamado *Paging*, muy útil a la hora de hacer consultas extensas, que te permite organizar formularios que conserven los datos a través de varias vistas y luego enviar la petición al servidor con la información de todas. Otro componente interesante es el *Splitter* de este framework, útil a la hora del trabajo con los marcos en las aplicaciones web. Este framework se destaca por ser uno de los que más componentes ha implementado.

### 2.2.2.2. Compatibilidad

Una de las desventajas de ZK es la poca compatibilidad con otros frameworks, que trae como consecuencia que todas las funcionalidades necesarias para la implementación de una aplicación, deban ser encontradas dentro del mismo ZK.

ZK es compatible con Internet Explorer (6.0 y 7.0), Mozila Firefox (2.0 y 3.0) y con el explorador Safari.

#### 2.2.2.3. Configuración

Para usar ZK se deben ubicar en el directorio *lib* de la aplicación web los siguientes archivos *jar*: *bsh.jar, commons-fileupload.jar, commons-io.jar, dojoz.jar, fckez.jar, Filters.jar, gmapsz.jar, jcommon.jar, jfreechart.jar, timelinez.jar, zcommon.jar, zcommons-el.jar, zhtml.jar, zk.jar, zkex.jar, zkmax.jar, zkplus.jar, zml.jar, zul.jar, zweb.jar*. Si se fueran a usar los componentes ZK se agrega, además del fichero de la implementación JSF que se use, el fichero zuljsf.jar.

Agregar ZK a una aplicación web implica adicionar los servlets, los escuchadores y el filtro opcional que se define en el fichero *web.xml*:

- *DHtmlLayoutServlet*: servlet que carga las páginas ZUML cuando el servidor Web recibe peticiones de URL enviadas por los usuarios (parámetro obligatorio).
- *DHtmlUpdateServlet*: servlet que maneja peticiones AJAX asíncronamente y automáticamente (parámetro obligatorio).
- *HttpSessionListener*: escuchador usado para limpiar la memoria cuando una sesión HTTP es destruida (parámetro obligatorio).
- *DHtmlLayoutFilter*: filtro diseñado para procesar páginas dinámicas generadas por otros servlets, dígase JSP o JSF. Le permite a los diseñadores agregar interfaces ricas escritas en cualquier tecnología.
- *InterpreterServlet*: servlet usado para procesar archivos DSP. DSP es similar a JSP como tecnología, pero difiere en que es interpretado en tiempo de ejecución, no requiere un compilador de Java. Además, se pueden distribuir páginas DSP en archivos *jar*. ZK está distribuido de esta forma. No se puede incluir código Java en páginas DSP. Las acciones DSP aunque extensibles a través de archivos TLD, son diferentes a las etiquetas de JSP.

```
<servlet>
    <description>ZK loader for ZUML pages</description>
      <servlet-name>zkLoader</servlet-name>
      <servlet-class>org.zkoss.zk.ui.http.DHtmlLayoutServlet</servlet-class>
      <init-param>
            <param-name>update-uri</param-name>
            <param-value>/zkau</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
            <servlet-name>zkLoader</servlet-name>
            <url-pattern>*.zul</url-pattern>
```

```
</servlet-mapping>
<servlet-mapping>
            <servlet-name>zkLoader</servlet-name>
            <url-pattern>*.zhtml</url-pattern>
</servlet-mapping>
<servlet-mapping>
            <servlet-name>zkLoader</servlet-name>
            <url-pattern>*.svg</url-pattern>
</servlet-mapping>
<servlet-mapping>
            <servlet-name>zkLoader</servlet-name>
            <url-pattern>*.xml2html</url-pattern>
</servlet-mapping>
<servlet-mapping>
            <servlet-name>zkLoader</servlet-name>
            <url-pattern>/zk/*</url-pattern>
</servlet-mapping>

<servlet>
    <description>The asynchronous update engine for ZK</description>
      <servlet-name>auEngine</servlet-name>
      <servlet-class>org.zkoss.zk.au.http.DHtmlUpdateServlet</servlet-class>
</servlet>
<servlet-mapping>
            <servlet-name>auEngine</servlet-name>
            <url-pattern>/zkau/*</url-pattern>
</servlet-mapping>

<filter>
   <filter-name>zkFilter</filter-name>
      <filter-class>org.zkoss.zk.ui.http.DHtmlLayoutFilter</filter-class>
            <init-param>
                  <param-name>extension</param-name>
                  <param-value>html</param-value>
            </init-param>
            <init-param>
                  <param-name>compress</param-name>
                  <param-value>true</param-value>
            </init-param>
</filter>
<filter-mapping>
            <filter-name>zkFilter</filter-name>
            <url-pattern>/test/filter.dsp</url-pattern>
</filter-mapping>
<filter-mapping>
            <filter-name>zkFilter</filter-name>
            <url-pattern>/test/filter2.dsp</url-pattern>
</filter-mapping>
<servlet>
      <servlet-name>dspLoader</servlet-name>
      <servlet-class>org.zkoss.web.servlet.dsp.InterpreterServlet</servlet-class>
            <init-param>
                  <param-name>class-resource</param-name>
                  <param-value>true</param-value>
            </init-param>
```

```
</servlet>
<servlet-mapping>
            <servlet-name>dspLoader</servlet-name>
            <url-pattern>*.dsp</url-pattern>
</servlet-mapping>
```

ZK utiliza para su funcionamiento un grupo de servlets propios que provoca que no se integre completamente al ciclo de vida de JSF, ya que procesa las peticiones en su flujo y luego se comunica con JSF. El framework ZK, similar a WebGalileoFaces, tiene una configuración trabajosa, lo cual constituye una desventaja para el desarrollo de aplicaciones que usan sus componentes.

#### 2.2.2.4. Aplicación en la construcción de las vistas

Como se menciona antes, utilizar el framework ZK requiere un tiempo para que los desarrolladores asimilen el lenguaje ZUML. Aunque mantiene las propiedades esenciales para los componentes de interfaz web, sus tags son nuevos y por tanto desconocidos si no se ha trabajado con ellos antes. No requiere directivas especiales en las páginas para el uso de los componentes. A continuación se muestra un ejemplo de código e imagen del componente *Captcha*.

```
<window title="captcha demo" border="normal">
  <vbox>
      <captcha id="cpa" length="5" width="200px" height="50px"/>
        <label id="val"/>
            <button label="Regenerate" onClick="cpa.randomValue();
            val.value=cpa.value;"/>
            <hbox>Assign one: <textbox onChange="cpa.value = self.value;
            val.value=cpa.value;"/>
            </hbox>
       <zscript>val.value=cpa.value;</zscript>
  </vbox>
</window>
```

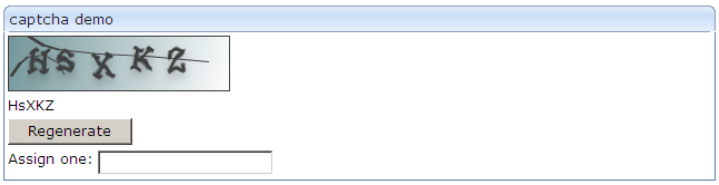


**Figura 2.2-3: Componente Captcha [30]**

#### 2.2.2.5. Integración a los componentes estándares de JSF

Todos los componentes de ZK tienen propiedades que le permiten funcionar con AJAX, sin embargo el framework no muestra facilidades para agregarle estas propiedades a los componentes propios de JSF.

### 2.2.3. QuipuKit

Con el framework de validación QuipuKit se pueden usar validadores estándar de JSF del lado del cliente, asignándolos a cualquier componente de JSF y de la librería de QuipuKit. De la misma manera se pueden usar validadores propios de QuipuKit que trabajan igualmente bien, tanto del lado del servidor como del lado del cliente. El framework de validación QuipuKit soporta una gran variedad de escenarios donde los datos deben ser validados, asegurando que las reglas de validación sean conocidas y los tipos de datos estén correctos en el cliente, así como la notificación inmediata de los posibles errores de los usuarios. Adicionalmente el framework provee un mecanismo flexible para configurar la presentación de los errores de validación por defecto de diferentes maneras.

#### 2.2.3.1. Librería de componentes

La ventaja principal de QuipuKit es el framework de validación que posee, el cual se integra muy bien a los componentes de JSF. Tiene la ventaja de poderse configurar tanto en el mismo código de la página como en el archivo *web.xml* a través de parámetros de contexto, lo que le permite aplicar las validaciones a toda la aplicación en dependencia del ámbito que se decida.

Entre los componentes más útiles de esta librería se encuentran el *Focus* y el *Scroll Position*, ambos conocidos como componentes no visibles. El siguiente código muestra sus posibles usos, tanto en la página como en los archivos de configuración, de manera que funcionen para toda la aplicación.

**Declaración en las páginas**:

```
<h:form id="form1">
  <q:focus focusedComponentId="subview1:input1"/>
  <f:subview id="subview1">
    <h:inputText id="input1"/>
  </f:subview>
</h:form>

<h:form>
  <q:scrollPosition scrollX="0"
                    scrollY="700"/>
</h:form>
```

**Declaración en el *web.xml*:**

```
<context-param>
  <param-name>teamdev.jsf.autoSaveFocus</param-name>
  <param-value>true</param-value>
</context-param>

<context-param>
  <param-name>teamdev.jsf.autoSaveScrollPos</param-name>
  <param-value>true</param-value>
</context-param>
```

#### 2.2.3.2. Compatibilidad

Todos los componentes de QuipuKit soportan las últimas versiones de los navegadores Microsoft Internet Explorer, Mozilla Firefox, Opera y Apple Safari.

QuipuKit está preparado para funcionar en una aplicación de conjunto con otros frameworks, que tengan objetivos similares. Los componentes de QuipuKit no sólo pueden ser usados con la tecnología JSP, si no también con el Framework Facelets. La forma de definir los atributos de los componentes de QuipuKit para usar Facelets es la misma que se utiliza en JSP. Es además compatible con la tecnología Portlets.

Los componentes de QuipuKit también son altamente compatibles con el Frameworks Ajax4jsf.. El componente *HintLabel* de QuipuKit no puede ser recargado con Ajax4jsf a través de su *id*. Es necesario envolver el componente con el tag *<a4j: outputPanel>* para poder recargarlo. El siguiente ejemplo muestra algunos de los detalles a tener en cuenta para la integración, desde el punto de vista de la programación y utilización de las funcionalidades de los componentes.

```
<a4j:outputPanel id="tableWrapper">
        <q:hintLabel value="#{request.summary}"/>
</a4j:outputPanel>
```

Se debe envolver además, un componente de entrada con el componente *Message* dentro del tag *<a4j: outputPanel>* en función de hacer que las características del framework de validación de QuipuKit trabajen correctamente, todo esto si se quiere recargar un componente de entrada con Ajax4jsf.

Cuando recargamos los componentes *DataTable* o *TreeTable* a través de Ajax4jsf, el contenido de los *facet above* y *below* no son recargados, para ellos estos igualmente deben ser envueltos en el tag *<a4j:outputPanel>* y de esta manera el panel se recarga.

#### 2.2.3.3. Configuración

Para lograr incluir QuipuKit en un proyecto JSF se deben seguir una serie de pasos. Primeramente, se debe agregar la librería *quipukit.jar* y luego otros ficheros *jar* también necesarios para lograr todas las funcionalidades. Después, se debe configurar el archivo *web.xml* en la carpeta del *WEB-INF:*

```
<!-- FILTER FOR PROCESSING INTERNAL QUIPUKIT RESOURCES -->
<filter>
  <filter-name>ResourceFilter</filter-name>
  <filter-class>teamdev.jsf.util.ResourceFilter</filter-class>
</filter>

<!-- MAPPING FOR QUIPUKIT COMPONENTS FILTER -->
<filter-mapping>
  <filter-name>ResourceFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Este código coloca un filtro delante del servlet de JSF, de manera que todas las peticiones que viajan al servidor son procesadas por QuipuKit. QuipuKit entonces garantiza darle el comportamiento de AJAX necesario y administrar el envío de peticiones al servlet de JSF.

Opcionalmente si se quieren activar las validaciones de manera automática para todas las páginas, se agregan las siguientes líneas de código en el mismo archivo de configuración:

```
<context-param>
  <param-name>teamdev.jsf.validation.clientValidation</param-name>
  <param-value>onSubmit</param-value>
</context-param>
```

Para mejorar el rendimiento de la aplicación, se puede reducir el ámbito de los pedidos que son manejados por el filtro de QuipuKit. Por ejemplo, se puede utilizar el siguiente mapeo del *ResourceFilter* en el archivo de configuración *web.xml*:

```
<!-- MAPPING FOR QUIPUKIT COMPONENTS FILTER -->
<filter-mapping>
  <filter-name>ResourceFilter</filter-name>
  <servlet-name>Faces Servlet</servlet-name>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>

<filter-mapping>
  <filter-name>ResourceFilter</filter-name>
  <url-pattern>/qk_internalResource/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

De esta manera se garantiza que el filtro solo se aplica al servlet de JSF y se restringe a la dirección determinada en el <*url-pattern*>.

#### 2.2.3.4. Aplicación en la construcción de las vistas

Usar los componentes que brinda la librería de QuipuKit es relativamente sencillo. Se debe declarar la directiva *http://teamdev.com/quipukit* con la letra *q* como prefijo, configuración que garantiza que el componente entre en el funcionamiento del ciclo de vida de JSF. Ejemplo:

```
<%@taglib uri="http://teamdev.com/quipukit" prefix="q" %>
```

Por ejemplo, para agregar el componente *FoldingPanel* se debe usar la etiqueta <*q:foldingPanel>.*

```
<q:foldingPanel>
        <f:facet name=”caption”>
                <h:outputText value=”Panel for color selection”/>
        </f:facet>
        <h:outputText value=”Select color:”/>
        <q:dropDownField>
                <q:dropDownItems value=”#{ColorBean.items}”/>
        </q:dropDownField>
</q:foldingPanel>
```

#### 2.2.3.5. Integración a los componentes estándares de JSF

QuipuKit no permite agregar comportamiento AJAX a los componentes de JSF puros, por lo cual la integración de AJAX a JSF a través de dicho framework se vería limitada a usar los componentes de QuipuKit.

### 2.2.4. ICEFaces

ICEFaces provee un ambiente de presentación web para aplicaciones de JSF, que enriquece el framework estándar de JSF y el ciclo de vida con características basadas en AJAX. ICEFaces sustituye los *renderers*[43] de JSF basados en HTML con *renderers* Direct-to-DOM (D2D) e introduce un puente de AJAX ligero para enviar los cambios en la presentación del cliente y comunicar los eventos de la interacción del usuario al servidor residente en la aplicación JSF. La arquitectura básica de aplicaciones construidas con ICEFaces se muestra en la siguiente figura:

[43] renderers: Elementos que permiten generar una imagen a partir de un modelo.

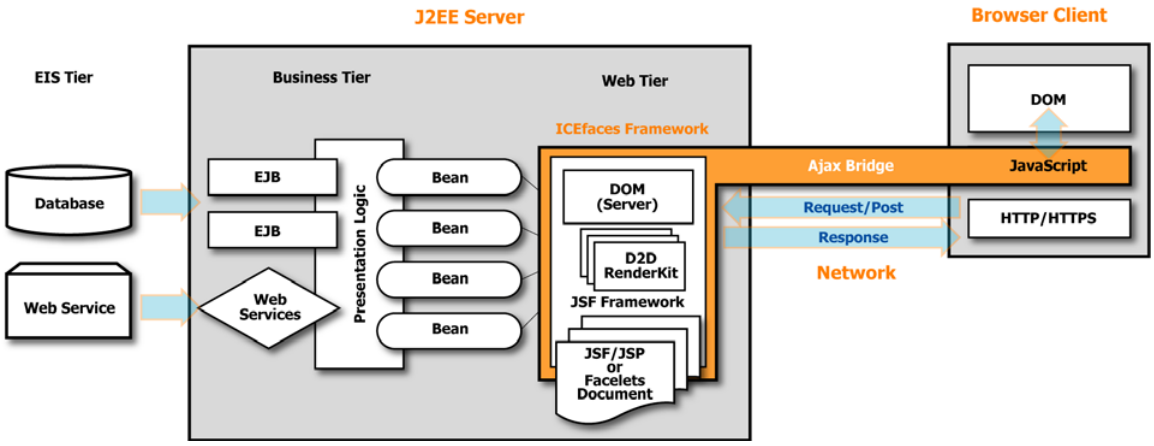


**Figura 2.2-4: Arquitectura de ICEFaces [4]**

ICEFaces tiene las siguientes características:

- Actualizaciones que no requieren refrescar la página completa para lograr los cambios de la presentación en la aplicación.
- Preservación del contexto del usuario durante la actualización de la página, incluyendo la posición de la barra de desplazamiento y el foco.

Más allá de estas características, ICEFaces introduce otras que los desarrolladores de JSF pueden incorporar:

- Procesamiento de formularios a través una técnica llamada *Partial Submit*, la cual envía automáticamente un formulario para procesarlo, basado en algún evento iniciado por el usuario. El envío automático es parcial, solo ocurre la validación parcial del formulario (los campos vacíos los marca como no requeridos). Usando este mecanismo, la aplicación puede reaccionar inteligentemente como interactúa el usuario con el formulario.
- Actualización de la presentación de manera asíncrona con el servidor iniciado. Las aplicaciones estándar de JSF solo pueden enviar cambios en la presentación como respuesta a un evento iniciado por el usuario. ICEFaces introduce un mecanismo de *triggers*[44] que permite a la lógica de la aplicación residente en el servidor hacer cambios en la presentación del cliente en respuesta a los cambios de estado de la aplicación.

[44] triggers: Clase que implementa una tarea que debe ser llevada a cabo, también conocido como disparador de eventos.

#### 2.2.4.1. Librería de componentes

La suite de componentes de ICEFaces provee un conjunto completo de los componentes más utilizados. Estos tienen características adicionales sobre los componentes de la implementación estándar de JSF, como son:

- Tecnología de *renderers* D2D que permite la actualización de interfaces de usuario para todos los componentes sin necesidad de refrescar la página completa.
- Soporte de atributos adicionales para características específicas de ICEFaces, como son *partialSubmit*, *effects*, *visibleOnUserRole* y otros.
- Soporte para la definición de estilos de los componentes a partir de CSS fáciles de personalizar.
- Soporte para efectos en los componentes del lado del cliente como *fading*, *expand*, *collapse* y otros.

Los atributos básicos de los componentes de ICEFaces están asociados con los tags de los componentes de JSF. Las propiedades específicas de ICEFaces, como el *partialSubmit* deben ser configuradas dentro del *tag* a través de atributos de JavaScript. Los *renderers* de los componentes de JSF no soportan la aplicación de estilos automáticos de ICEFaces mediante CSS.

#### 2.2.4.2. Compatibilidad

ICEFaces también se puede integrar con otros framework como Facelets. ICEFaces interpreta la sintaxis de JSP en un documento de JSF, pero no puede procesar estos documentos a través del ciclo compilación/ejecución estándar de JSP. En cambio, ICEFaces interpreta directamente los documentos de entrada y construye el árbol de componentes de JSF a través de los tags dados.

Es compatible con la librería JavaScript Sciptaculous y con la tecnología Portlets. Además con Internet Explorer 6.0 y Mozila FireFox (2.0 y 3.0).

#### 2.2.4.3. Configuración

En la mayoría de los casos el objetivo de adicionar ICEFaces a una aplicación JSF es transformar la aplicación completa a una aplicación de AJAX, pero hay muchas razones para sólo transformar algunas partes a ICEFaces. Para manejar algunas páginas con ICEFaces y otras con el mecanismo por defecto de JSF, se agrega al mapeo del servlet de ICEFaces la extensión *.iface* pero no se elimina

el de *Faces Servlet*" (servlet de JSF). Las páginas servidas a través del servlet de JSF se manejan sin ICEFaces. Para asegurarse que los *renderers* D2D son aplicados sólo a las páginas ICEFaces se incluye dentro de la aplicación web la librería *just-ice.jar*, preferentemente a *icefaces.jar*. Este *jar* contiene una versión de ICEFaces configurada con un *ViewHandler* que procesa sólo aquellas páginas con una extensión *.iface* y un *RenderKit* que no sustituirá los componentes estándar de JSF con los *renderers* D2D. En esta configuración las páginas son interpretadas según el *tag* que contengan, en el caso de la implementación estándar de JSF se utiliza *f:* y en el caso de ICEFaces se usa *ice:*

El archivo *iceface.jar* contiene un fichero *face-config.xml* que configura las extensiones de ICEFaces. Específicamente, el archivo de configuración registra los *renderers* del Direct-to-DOM.

El archivo *web.xml* de la aplicación debe incluir necesariamente el registro y el mapeo de los servlet que se van a emplear. El servlet de ICEFaces se configura de la siguiente forma:

```
<servlet>
        <servlet-name>Persistent Faces Servlet</servlet-name>
        <servlet-class>
                com.icesoft.faces.webapp.xmlhttp.PersistentFacesServlet
        </servlet-class>
        <load-on-startup> 1 </load-on-startup>
</servlet>
<servlet>
        <servlet-name>Blocking Servlet</servlet-name>
        <servlet-class>
                com.icesoft.faces.webapp.xmlhttp.BlockingServlet
        </servlet-class>
        <load-on-startup> 1 </load-on-startup>
</servlet>
```

Aquí se declaran los servlet de ICEFaces que usa el framework, pero no se elimina el servlet de JSF. Con el mapeo de los servlets se diferencia a partir de la extensión de los ficheros y la dirección, la localización de los archivos que debe procesar cada servlet. A continuación se muestra el mapeo de cada uno de ellos:

```
<servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jspx</url-pattern>
</servlet-mapping>
<servlet-mapping>
        <servlet-name>Persistent Faces Servlet</servlet-name>
        <url-pattern>*.iface</url-pattern>
</servlet-mapping>
<servlet-mapping>
        <servlet-name>Persistent Faces Servlet</servlet-name>
        <url-pattern>/xmlhttp/*</url-pattern>
</servlet-mapping>
```

```
<servlet-mapping>
        <servlet-name>Blocking Servlet</servlet-name>
        <url-pattern>/block/*</url-pattern>
</servlet-mapping>
```

El escuchador de contexto también se configura para ICEFaces, al no usar los que trae JSF. Se agrega en el *web.xml* como sigue:

```
<listener>
        <listener-class>
                com.icesoft.faces.util.event.servlet.ContextEventRepeater
        </listener-class>
</listener>
```

Hay una serie de parámetros de contexto que pueden ser configurados en la aplicación con el objetivo de darle mayores potencialidades. A continuación se muestran algunos:

- Las actualizaciones síncronas o asíncronas con el servidor pueden ser apagadas o encendidas usando el parámetro de contexto *com.icesoft.faces.synchronousUpdate*. Igualmente esto se configura en el archivo *web.xml*:

```
<context-param>
        <param-name>com.icesoft.faces.synchronousUpdate</param-name>
        <param-value>true/false</param-value>
</context-param>
```

- ICEFaces también da la posibilidad de abrir varias ventanas a la vez en una misma aplicación, propiedad que le da mayores posibilidades de navegación. Se configura con el parámetro de contexto *com.icesoft.faces.concurrentDOMViews*:

```
<context-param>
        <param-name>com.icesoft.faces.concurrentDOMViews</param-name>
        <param-value>true</param-value>
</context-param>
```

- También archivos de código JavaScript y CSS pueden ser comprimidos cuando son enviados al navegador, esto puede mejorar el tiempo en que se carga la aplicación. Se debe configurar para ello el parámetro de contexto *com.icesoft.faces.compressResource*:

```
<context-param>
        <param-name>com.icesoft.faces.compressResources</param-name>
        <param-value>true/false</param-value>
</context-param>
```

#### 2.2.4.4. Aplicación en la construcción de las vistas

Para utilizar los componentes en la programación de una página se hace de manera semejante a otros frameworks. El siguiente ejemplo muestra, además, la posibilidad de cambiar dinámicamente el tema de la hoja de estilo:

```
<ice:outputStyle href="#{styleBean.activeTheme}" rel="stylesheet"
        type="text/css" />
```

Se puede notar que para incluir los componentes ICEFaces en una página se utiliza el tag <ice: >, que se define en el encabezado de la misma. Entre los componentes más útiles de ICEFaces se encuentra el *richText,* del cual se muestra un ejemplo de código e imagen*:*

```
<ice:inputRichText id="iceInpRchTxt"
                           height="275"  width="600"
                           toolbar="#{inputRichTextBean.toolbarMode}"
                           value="#{inputRichTextBean.value}"
                           language="en"  skin="silver"
                           />
```

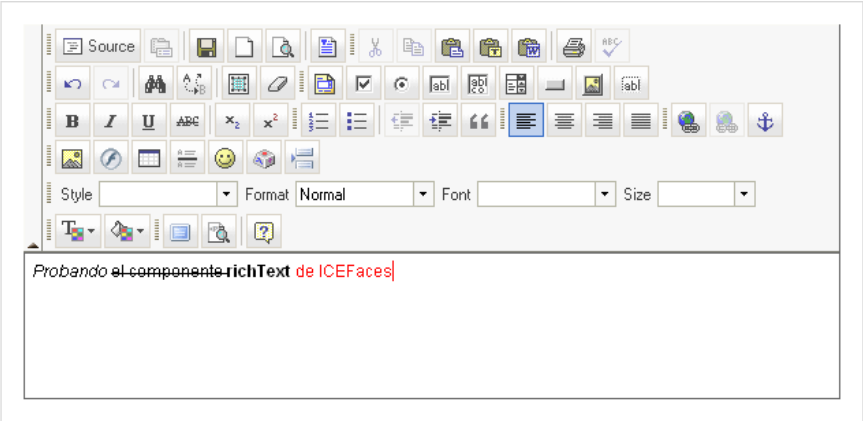


**Figura 2.2-5: Componente RichText [3]**

#### 2.2.4.5. Integración a los componentes estándares de JSF

ICEFaces aporta un grupo de componentes con propiedades útiles, tiene como ventaja el trabajo dinámico con el estilo de los componentes. ICEFaces crea componentes con propiedades AJAX, que heredan de las implementaciones de los componentes propios de JSF. Sin embargo, no le agrega características AJAX a los componentes estándares de JSF.

### 2.2.5. RichFaces

RichFaces se integra completamente al ciclo de vida de JSF como se muestra en la figura:

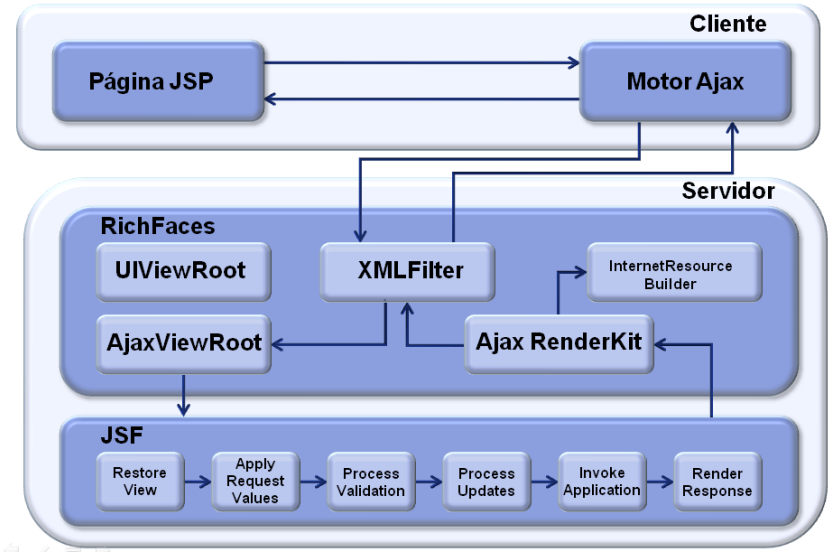


**Figura 2.2-6: Ciclo de vida de RichFaces [3]**

El motor de AJAX del lado del cliente implementado en JavaScript es invocado a partir de un evento JavaScript (ej: cuando un usuario da click en un botón). Un objeto XMLHttpRequest es creado y enviado al servidor. En el lado del servidor, el filtro de RichFaces intercepta la petición. Los datos transferidos son extraídos y la petición es procesada normalmente por JSF.

Cuando el ciclo de vida de JSF ha finalizado y la respuesta está lista, el filtro de RichFaces es invocado nuevamente. Codifica solamente la región de la petición y la envía en formato XML. Al finalizar, el motor de AJAX transforma en el cliente la información del XML en el DOM que será mostrado en la página.

#### 2.2.5.1. Librería de componentes

RichFaces cuenta con un gran número de componentes. Uno de los más útiles y que más ventajas le ofrece a RichFaces es *<a4j: support>*. El componente *<a4j: support>* adiciona soporte de AJAX a un componente existente de JSF. Este permite a un componente generar peticiones asíncronas y actualizar una parte de la página después de una respuesta del servidor.

Para usar este componente se debe poner el tag *<a4j: support>* anidado al componente que se desea agregar funcionalidad AJAX y especificar el evento correspondiente al componente que genera la petición AJAX. Ejemplo:

```
<h:inputText size="50" value="#{bean.text}" >
     <a4j:support event="onkeyup" reRender="rep"/>
</h:inputText>
<h:outputText value="#{bean.text}" id="rep"/>
```

#### 2.2.5.2. Compatibilidad

RichFaces trabaja con cualquier implementación de Sun JSF (1.1 o 1.2) y con la mayoría de las librerías de componentes JSF sin necesidad de configuraciones adicionales. RichFaces además, puede usarse con todas las versiones de Apache MyFaces (1.1.1 – 1.1.6) incluyendo librerías específicas como Tomahawk Sandbox y Trinidad. Sin embargo, hay muchas consideraciones que tener en cuenta a la hora de configurar aplicaciones con MyFaces y Rich Faces.

Una de las características más importantes de RichFaces es el soporte para el Framework Facelets. Cuando se trabaja con RichFaces no existen diferencias entre las distintas versiones de Facelets.

RichFaces también es compatible con los frameworks JBoss Seam y Facelets. Soporta además los servidores JBoss AS 4.0.4. y Sybase EAServer. Para ello solo se le debe poner el *load-on-startup*[45] del servlet de JSF en 0.

Soporta también Portlet y las librerías JQuery y Scriptaculous. Es compatible con los navegadores Konqueror, Safari, Firefox (2.0 y 3.0) y con Internet Explorer (6.0).

#### 2.2.5.3. Configuración

Para usar RichFaces en una aplicación JSF se deben incluir dentro de la carpeta *lib* los siguientes archivos *jar*: *richfaces-api-3.1.0.jar*, *richfaces-impl-3.1.0.jar*, *richfaces-ui-3.1.0.jar.*

En el archivo de configuración *web.xml* se debe agregar el filtro de RichFaces. También se puede agregar un parámetro de contexto para el *skin* de la página:

```
<context-param>
      <param-name>org.richfaces.SKIN</param-name>
      <param-value>blueSky</param-value>
```

[45] load-on-startup: parámetro de un servlet que dice en que momento este debe ser cargado al levantar la aplicación.

```
</context-param>
<filter>
      <display-name>RichFaces Filter</display-name>
      <filter-name>richfaces</filter-name>
      <filter-class>org.ajax4jsf.Filter</filter-class>
</filter>
<filter-mapping>
      <filter-name>richfaces</filter-name>
      <servlet-name>Faces Servlet</servlet-name>
      <dispatcher>REQUEST</dispatcher>
      <dispatcher>FORWARD</dispatcher>
      <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

Como se puede apreciar la configuración necesaria para agregar RichFaces a JSF es muy sencilla y se integra al servlet de JSF, ya que sólo consiste en poner un filtro delante de este.

#### 2.2.5.4. Aplicación en la construcción de las vistas

Para las páginas JSP se deben agregar las siguientes líneas de código:

```
<%@ taglib uri="http://richfaces.org/a4j" prefix="a4j"%>
<%@ taglib uri="http://richfaces.org/rich" prefix="rich"%>
```

Para el caso de las páginas xhtml:

```
<xmlns:a4j="http://richfaces.org/a4j">
<xmlns:rich="http://richfaces.org/rich">
```

Al igual que en los frameworks QuipuKit y ICEFaces los componentes, para poder usarse en las vistas, sólo necesitan estar precedidos por el *tag* que identifca al framework.

Un ejemplo de uno de sus componentes más útiles es el *PanelMenu,* utilizado para hacer más fácil la navegación a los usuarios desde un menú. A continuación se muestra el código y una imagen de cómo se implementa:

```
<rich:panelMenu>
      ...
      <rich:panelMenuItem submitMode="none"
         onclick="document.location.href='http://labs.jboss.com/jbossrichfaces/">
         <h:outputLink value="http://labs.jboss.com/jbossrichfaces/">
            <h:outputText value="RichFaces Home Page"></h:outputText>
         </h:outputLink>
      </rich:panelMenuItem>
...
</rich:panelMenu>
```

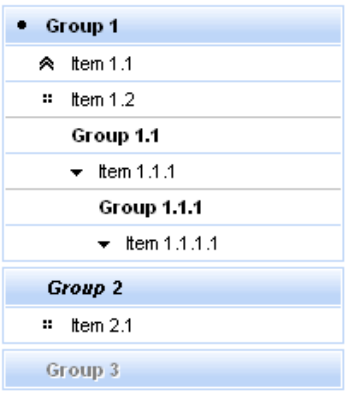


**Figura 2.2-7: Componente PanelMenu [22]**

#### 2.2.5.5. Integración a los componentes estándares de JSF

RichFaces permite integrarle AJAX a los componentes nativos de JSF usando el componente <*a4j:support*>. Este componente capta los eventos lanzados por un componente JSF y envía en peticiones AJAX lo que antes JSF enviaba en peticiones tradicionales, junto con los demás datos de la página. RichFaces es el único framework de los estudiados que logra esta propiedad, cuyo principal beneficio está en la posibilidad de usar todos los componentes de la implementación de JSF en una aplicación con características AJAX. En otros frameworks se debían sustituir los componentes, en este caso se adicionan nuevos y por tanto se adicionan más funcionalidades a JSF.

## *2.3. Facelets*

Como resultado de la profundización realizada en el estudio de los frameworks, resalta con gran significación que la mayoría de estos son compatibles con Facelets. Por esto, se decide agregar el presente epígrafe para describir y analizar el Framework Facelets.

Facelets es un motor de plantillas para JSF como lo es Tiles para el Framework Strut. Los motores de plantillas leen un fichero de texto, que contiene la presentación ya preparada en HTML (o meta información, por ejemplo XML, o cualquier contenido que sea texto), e inserta en él la información dinámica.

Los motores de plantillas suelen tener un pequeño lenguaje de script que permite generar código dinámico, como listas o cierto comportamiento condicional. Este lenguaje de script es absolutamente mínimo, solo lo necesario para posibilitar ese comportamiento dinámico. Las plantillas no se refieren

sólo a elementos simples de datos, si no que también pueden procesar objetos para mostrar sus miembros.

Facelets construye un árbol de componentes. Esto permite una gran reutilización, de forma que se pueden definir componentes como composición de otros componentes. Además, Facelets se ha creado teniendo en cuenta el ciclo de vida JSF, al contrario de JSP. Por tanto, si se usa Facelets sólo existe un servlet, el de JSF. De esta manera, se resuelve el problema de la independencia de los ciclos de vida de JSF y JSP y se facilita la integración de Facelets a JSF.

Facelets no es dependiente del contenedor JSP, lo que significa que una aplicación puede comenzar utilizando las nuevas características de JSF 1.2 sin esperar a que un contenedor tenga soporte para JSP 2.1. Por tanto no necesita un contenedor de servlet para definir y probar las vistas JSF.

Facelets sustituye el *ViewHandler*[46] de JSF por *FaceletsViewHandler*, el cual construye las vistas a partir de documentos XHTML. Esto posibilita un uso óptimo de AJAX pues una de las tecnologías que la componen es precisamente XHTML.
El *FaceletsViewHandler* solamente es invocado en la primera y la última fase del ciclo de vida de JSF (*RestoreView* y *RenderResponse)*. En la fase de *RestoreView,* existen dos posibilidades:

- La petición que llega al servidor es la primera proveniente de una página particular. En este caso una nueva vista es creada a partir del documento XHTML.
- La página actual ya fue pedida al menos una vez. En este caso la página ya existe en el UIViewRoot, por tanto delega en el ViewHandler de JSF.

La siguiente figura presenta el funcionamiento de Facelets integrado a JSF:

[46] ViewHandler: Es la clase que maneja las vistas de una aplicación, incide en las fases RestoreView y Render Response del ciclo de vida de JSF.

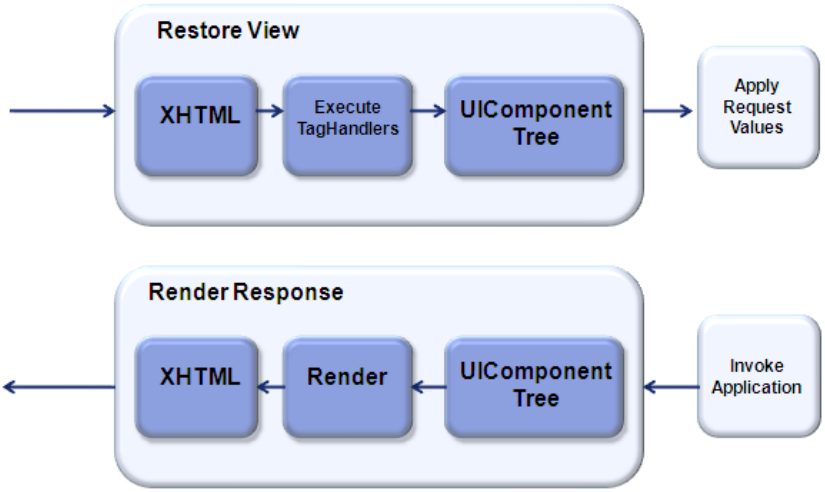


**Figura 2.3-1: Funcionamiento de Facelets [1]**

Las características más importantes de Facelets se pueden resumir del siguiente modo:

- Trabajo basado en plantillas.
- Fácil composición de componentes.
- Creación de etiquetas lógicas a la medida
- Funciones para expresiones
- Desarrollo amigable para el diseñador gráfico
- Creación de librerías de componentes.
- Facilita el diseño de páginas.

Facelets proporciona una librería de etiquetas para la construcción de las vistas, entre las cuales se encuentran:

| Etiqueta de plantilla | Etiqueta plantilla cliente | Otras etiquetas |
|---|---|---|
| <ui:insert/> | <ui:component/><br><ui:fragment/><br><ui:composition/><br><ui:decorate/> | <ui:debug/><br><ui:define/><br><ui:include/><br><ui:param/> |

| | | |
|---|---|---|
| | | <ui:remove/> |

- `<ui:insert>` declara las partes del documento que serán sobreescritas.
- `<ui:composition>` solo lo que este dentro podrá ser creado o editado con un editor visual.
- `<ui:define>` especifica la parte de la página que será creada.
- `<ui:include>` incluye en la página cualquier *tag composition* o *component*

Luego de estudiar el framework Facelets, se evidencian sus facilidades como complemento enriquecedor para la integración de AJAX a JSF.

## *2.4. RichFaces + Facelets en JSF*

A partir de los aspectos analizados en cada unos de los frameworks escogidos y según las necesidades específicas del proyecto Kainos se propone el uso de los frameworks RichFaces y Facelets para el desarrollo de la capa de presentación de la aplicación.

**¿Por qué RichFaces y Facelets?**

- RichFaces es un framework de código abierto.
- RichFaces se integra a JSF a través de un filtro que permite que todas las peticiones de AJAX se procesen por el servlet de JSF como una petición más, integrándose perfectamente al ciclo de vida de JSF.
- Facelets se integra a JSF a través de su primera y última fase, permitiendo la construcción y devolución de las vistas
- La configuración necesaria para lograr la integración de ambos frameworks es sencilla y no necesita de archivos adicionales de configuración.
- RichFaces permite agregar comportamiento AJAX a cualquier componente de la librería de componentes estándares de JSF, construir componentes personalizados con soporte AJAX incluido y brindar un amplio grupo de componentes implementados por el framework.
- RichFaces es compatible con la más reciente implementación de JSF (1.2) y se integra mejor a la implementación Sun JSF que a otras más conocidas como MyFaces.

- RichFaces soporta otros frameworks y librerías de componentes, de manera tal que de ser necesario un componente específico se puede incorporar.
- Facelets es una propuesta alternativa a la tecnología JSP pues evita el servlet de JSP.
- Facelets permite definir una plantilla sin tener que copiar el código HTML de la misma en todas las páginas de la aplicación y redefinir componentes de una forma muy sencilla.

La propuesta de integración de Facelets y RichFaces a JSF para la capa de presentación del proyecto Kainos, se muestra en la siguiente figura:

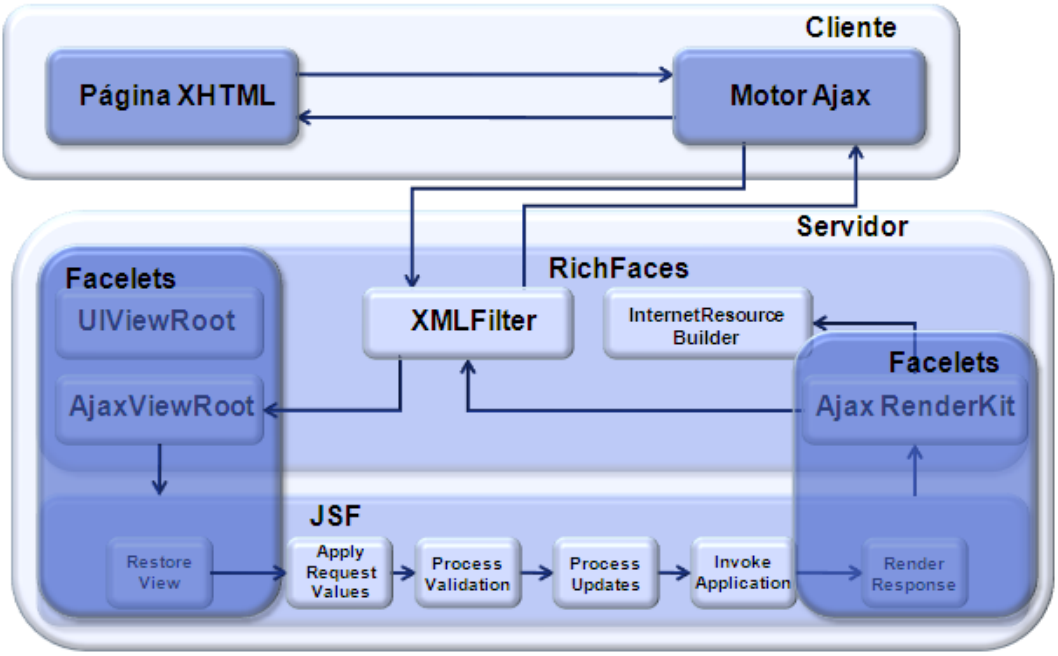


**Figura 2.4-1: RichFaces + Facelets en JSF**

# Capítulo 3: Mejora de la capa de presentación

En este capítulo se explica paso a paso la implementación llevada a cabo para mejorar la capa de presentación del proyecto Kainos a partir del uso de la propuesta definida en el Capítulo 2.

## *3.1. Revisión de la arquitectura*

Todas las aplicaciones están regidas por una arquitectura que se define en los inicios de la elaboración y va refinándose durante el avance del desarrollo de la misma. Una arquitectura que esté bien definida, que sea flexible y robusta, ayuda a la planificación de recursos y la asignación de tareas. Debido a esto, el trabajo de desarrollo puede ser particionado a través de los subsistemas y los esfuerzos de desarrollo individual pueden proceder en paralelo. También permite flexibilidad en el sistema pues facilita la ejecución de futuros cambios. Promueve la reutilización de componentes existentes como librerías de clases y de aplicaciones de terceros.

En este sentido se hizo una revisión de la arquitectura existente antes de llevar a cabo cualquier cambio en la capa de presentación, para alcanzar así un mayor entendimiento de esta y una clara separación entre las capas lógicas de la aplicación.

En la aplicación anterior se pudo identificar una inadecuada estructura organizacional en cuanto a paquetes, archivos de configuración y vistas. Los paquetes no tenían una división por capas lógicas, ya que el paquete que agrupaba a las clases de los servicios y el paquete de las clases de acceso a datos se ubicaban dentro del paquete del modelo. Además, los archivos de mapeo se encontraban en el paquete de las clases entidades cuando deben pertenecer al paquete de acceso a datos. En el caso de los archivos de configuración no existía una separación por módulos, esto dificultaba el trabajo simultáneo de varios desarrolladores, provocando conflictos en dichos archivos durante el trabajo con el repositorio. Las vistas, por su parte, no tenían una separación por módulos que permitiera una mejor organización.

El primer paso fue una reestructuración completa de los paquetes de la aplicación a partir de la configuración existente. Teniendo en cuenta que la aplicación cuenta con tres módulos, se estructuró en cuatro paquetes. El primer paquete llamado *core*, constituye el núcleo de toda aplicación pues agrupa paquetes de clases comunes para todos los módulos. Los demás, llamados *funcionamiento*, *estadística* y *seguridad,* se organizan para dar respuestas a las funcionalidades de cada módulo.

En el caso del paquete *core* tiene estructura interna descrita como sigue:

- ***reportes***: contiene el *datasource* que utilizan las clases que generan reportes.
- ***util***: contiene clases de utilidades necesarias para el correcto funcionamiento de la aplicación.
- ***web***: contiene un conjunto de paquetes y clases con diferentes utilidades para el correcto funcionamiento de la capa web, como los escuchadores, los *beans*, y un fichero con la personalización de los mensajes de error.

Los paquetes de los módulos se configuraron de la siguiente forma:

- ***dao***: contiene las clases interfaces y sus implementaciones garantizando el acceso a datos, así como los ficheros de mapeo de las mismas.
- ***modelo***: contiene las clases entidades de la aplicación. En este paquete se decidió eliminar las clases abstractas que se generaron en el mapeo de la base de datos, para lograr una mayor claridad en el desarrollo de la aplicación.
- ***servicio***: contiene las clases interfaces y sus implementaciones garantizando el acceso a todos los servicios que brinda la aplicación.
- ***web***: contiene los paquetes y las clases necesarias para el funcionamiento de la presentación de cada módulo, con una estructura muy similar al paquete web de *core*.

Siguiendo la misma idea de reestructuración de los paquetes de clases se reorganizó la estructura de la Web en siete grupos:

- ***components***: tiene la implementación de los componentes redefinidos, así como el XML donde se declara el tag con que se usan.
- ***css***: se encuentra el fichero *styles.css* que define las clases de estilos de las vistas y componentes de la aplicación.
- ***images***: se ubican las imágenes usadas en la aplicación.
- ***includes***: se encuentra la estructura del menú utilizado en la mayoría de las vistas.
- ***pages***: se ubican las páginas de la aplicación agrupadas por los módulos de la misma.
- ***templates***: se encuentra la plantilla utilizada en la aplicación.

- ***WEB-INF***: se ubican todos los ficheros de configuración de la aplicación dentro de una carpeta llamada *conf* que internamente está organizada según los módulos con que se cuenta.

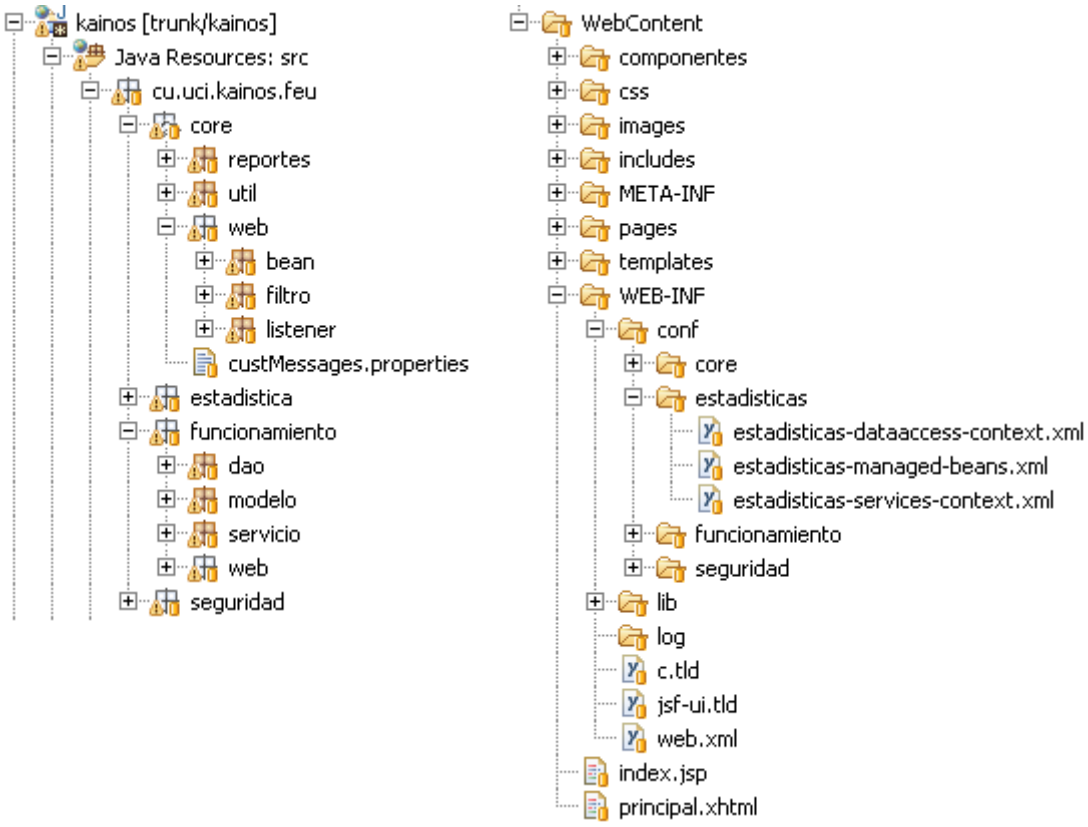


**Figura 3.1-1: Estructura de paquetes(izquierda)y de carpetas del *WebContent*(derecha)**

## *3.2. Configuración*

Para aplicar la propuesta definida se hicieron las siguientes configuraciones en el fichero *WEB-INF/web.xml*:

- Extensión de los clientes de plantillas:

```
<context-param>
      <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
      <param-value>.xhtml</param-value>
</context-param>
```

El parámetro *javax.faces.DEFAULT_SUFFIX* le dice a JSF que cuando reciba una petición busque los ficheros con extensión *.xhtml,* que es la extensión que se usó para trabajar con Facelets.

- Manejador de vistas de Facelets:

```
<context-param>
      <param-name>org.ajax4jsf.VIEW_HANDLERS</param-name>
      <param-value>com.sun.facelets.FaceletViewHandler</param-value>
</context-param>
```

El manejador de vistas de Facelets se ubica primero que los manejadores de JSF y RichFaces, de manera que sea el que utilice la aplicación.

- Redefiniendo componentes

```
<context-param>
      <param-name>facelets.LIBRARIES</param-name>
      <param-value>/componentes/arcmind.taglib.xml</param-value>
</context-param>
```

El parámetro *facelets.LIBRARIES* le dice a Facelets que en el fichero *arcmind.taglib.xml* están declarados los componentes redefinidos para la aplicación.

- Aplicando estilo de RichFaces

```
<context-param>
      <param-name>org.richfaces.SKIN</param-name>
      <param-value>ruby</param-value>
</context-param>
```

El parámetro *org.richfaces.SKIN* le dice a RichFaces que todos sus componentes usen el estilo *ruby*, definido por el mismo framework.

- Ficheros JavaScript y CSS

```
<context-param>
      <param-name>org.richfaces.LoadScriptStrategy</param-name>
      <param-value>ALL</param-value>
</context-param>

<context-param>
      <param-name>org.richfaces.LoadStyleStrategy</param-name>
      <param-value>ALL</param-value>
</context-param>
```

Los parámetros *org.richfaces.LoadScriptStrategy* y *org.richfaces.LoadStyleStrategy* le dicen a RichFaces que todos los ficheros JavaScript y CSS serán descargados en el cliente al inicio de la aplicación.

- Filtro de RichFaces

```
<filter>
      <display-name>RichFaces Filter</display-name>
      <filter-name>richfaces</filter-name>
      <filter-class>org.ajax4jsf.Filter</filter-class>
      <init-param>
            <param-name>log4j-init-file</param-name>
            <param-value>\\WEB-INF\\conf\\core\\log4j.xml</param-value>
      </init-param>
</filter>

<filter-mapping>
      <filter-name>richfaces</filter-name>
      <servlet-name>Faces Servlet</servlet-name>
      <dispatcher>REQUEST</dispatcher>
      <dispatcher>FORWARD</dispatcher>
      <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

La aplicación agrega así un filtro al servlet de JSF, lo que permite usar el framework RichFaces.

Con las configuraciones anteriores la aplicación está lista para usar todas las ventajas que brindan los frameworks RichFaces y Facelets.

## *3.3. Aplicando Facelets*

En este epígrafe se describe la implementación de cada una de las ventajas que nos brinda este framework a la capa de presentación de la aplicación.

### 3.3.1 Regiones editables

Facelets permite definir una plantilla general para la aplicación y dentro de ella un conjunto de regiones editables. En este caso se definen tres regiones editables principales. La primera se utiliza para mostrar el nombre del usuario registrado en la aplicación, la segunda para el menú dinámico y la tercera para el contenido principal sobre el que se esté trabajando. El siguiente código y la imagen muestran lo explicado anteriormente.

```
<div id="user">
    <ui:insert name="user"> Usuario </ui:insert>
</div>
```

```
<div id="div-body">
    <ui:insert name="menu"> Menú </ui:insert>

    <ui:insert name="content"> Contenido </ui:insert>
</div>
```

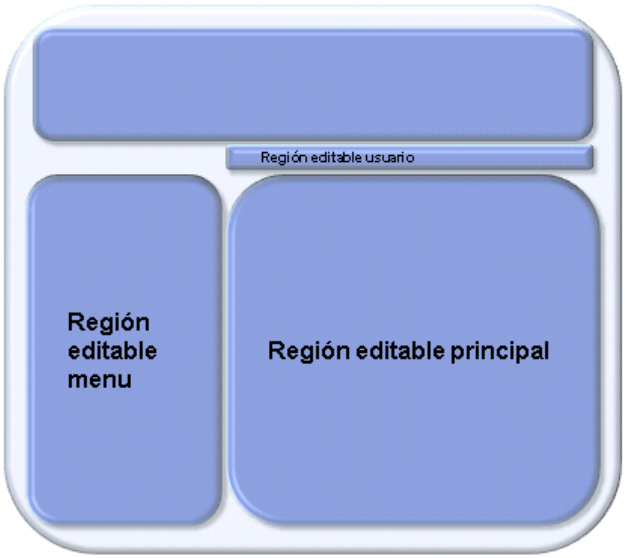


**Figura 3.3-1: Plantilla principal de la aplicación**

Es decir, dentro del código XHTML de la plantilla, usando el tag *<ui: insert>* se definen tres regiones lógicas, en las cuales se incluirán todos los contenidos definidos en las páginas de la aplicación. De esta forma se pueden agregar tantas regiones editables como se necesite. Esto permite que en las nuevas páginas sólo se encuentre el conjunto de componentes necesarios en ellas, sin necesidad de repetir en cada una todo el código HTML de la plantilla.

### 3.3.2 Formas de navegación

En el fichero *faces-navigation-rules.xml* de la aplicación se definen las reglas de navegación, tal y como se debe hacer cuando se usa JSF. Con la incorporación de RichFaces y Facelets no es necesario usar siempre una regla de navegación para ir de una página a otra.

RichFaces permite organizar el flujo de la página dentro del componente *<a4j: include>*. Este es el escenario típico del comportamiento de un Wizard[47]. El nuevo contenido es incluido dentro del área del componente. Este contenido sigue el flujo definido en el fichero *faces-navigation-rules.xml* de la

[47] Wizard: Asistente de interfaz de usuario que guía paso por paso, a través de cuadros de diálogo para cumplir una tarea específica.

aplicación. No importa que la página incluida no posea su propio *<f: view>* siempre que se use Facelets. Es importante aclarar que es necesario tener componentes AJAX dentro del *<a4j: include>* para navegar dentro de las páginas Wizard, de lo contrario la página sería recargada completamente.

Se utiliza el componente *<a4j: include>* dentro del *tag <ui: define>.* Esto permite asignarle la nueva vista contenida dentro del *<a4j: include>* al *<ui: define>,* responsable de entregarle al *tag <ui: insert>* dicha vista para que sea incorporada a la plantilla de la página. De esta forma, el tránsito de una página a otra de la aplicación se hace siguiendo este flujo, cambiando solamente la región definida por el valor del *<a4j: include>.* A continuación se muestra el *<ui: define>* que define la región principal de la aplicación:

```
<ui:define name="content">
      <div class="info" style="width: 485px; float: right;">
            <a4j:form id="panel">
                  <a4j:include viewId="#{currentBean.current}" />
            </a4j:form>
      </div>
</ui:define>
```

Siguiendo esta lógica se definen las reglas de navegación, para el flujo inicial de la aplicación, en el fichero *faces-navigation-rules.xml.* Dicho flujo consiste en las tres primeras vistas de la aplicación y termina cuando el usuario se registra en el sistema.

RichFaces incorpora además el atributo *reRender* a todos sus componentes para la optimización de las peticiones AJAX. Este atributo es de suma importancia pues permite definir cuál o cuáles áreas de la página serán actualizadas como resultado de la interacción con la respuesta AJAX. El valor de este atributo es un id o una lista de id de los componentes del DOM de la página. Para encontrar los componentes en el árbol de componentes él utiliza el algoritmo *UIComponent.findComponent()* definido en el API de JSF 1.2, por tanto uno puede definir lo rápido que será encontrado el componente en cuestión si se le menciona con más precisión en qué parte del árbol está. Además, es posible usar JSF EL como valor del atributo y podría ser una propiedad de los tipos de datos *Set*, *Collection*, *Array* o simplemente *String*.

Aprovechando estas ventajas de los componentes, se puede navegar simplemente especificándole al atributo *reRender* el id de la parte de la página que se desea actualizar.

```
<rich:panelMenuItem icon="disc"
      action="#{currentBean.updateCurrent}"
      reRender="panel" label="#{child.label}">
            <f:param name="current" value="#{child.name}" />
</rich:panelMenuItem>
```

Para hacer dinámico el flujo entre las vistas de la aplicación se implementa la clase llamada *CurrentBean,* que es la responsable de guardar el valor de la vista actual. Tiene el método *UpdateCurrent,* encargado de actualizar la variable con el valor de la vista hacia la que se va a navegar.

```
public String updateCurrent()
{
FacesContext context = FacesContext.getCurrentInstance();
setCurrent(context.getExternalContext().getRequestParameterMap().get("current"));
return null;
}
```

### 3.3.3 Redefiniendo componentes

Facelets además permite redefinir componentes con el objetivo de reutilizar código, de esta forma las páginas se hacen más cortas, lo que facilita un mayor entendimiento de las mismas.

Los nuevos componentes se crean de manera muy sencilla; el fichero XML que define el espacio de nombre de este, el fichero XHTML y el nombre del componente. El espacio de nombre se utiliza como directiva en la página donde es usado. El XHTML implementa el nuevo componente a partir de componentes conocidos con las configuraciones deseadas. El nombre es el *tag* que se utiliza en la programación de las vistas.

El siguiente código muestra la declaración de uno de los componentes *secret* redefinidos para la aplicación:

```
<namespace>http://www.arc-mind.com/jsf</namespace>
<tag>
     <tag-name>secret</tag-name>
     <source>secret.xhtml</source>
</tag>
```

A continuación se muestra la redefinición del componente:

```
<c:if test="${empty required}">
      <c:set var="required" value="true" />
</c:if>

<div class="enlinea" id="space_top"  style="#{estilo}" >
      <h:outputText styleClass="label" id="#{id}Label" value="#{fieldName}"/>
      <h:inputSecret id="#{id}" value="#{entity}" required="true"
                           styleClass="combo">
              <ui:insert />
              <f:validateLength  minimum="4" maximum="20"/>
      </h:inputSecret>
</div>
<rich:message for="#{id}" styleClass="msg" />
```

## *3.4. Aplicando RichFaces*

A continuación se explican cada uno de los componentes que se utilizan en la aplicación, permitiendo agregarle comportamiento AJAX a la capa de presentación del proyecto Kainos, así como la personalización del *skin* para la aplicación.

### 3.4.1 <rich: panelMenu>

Permite definir un menú en línea vertical dentro de una página. Este posibilita organizar el acceso a todas las vistas de la aplicación pues anteriormente los hipervínculos de la navegación se encontraban distribuidos en cada una de las páginas. Es, por tanto, el componente que facilita la navegabilidad dentro del sitio.

Este componente, para su funcionamiento, integra otros que no cumplen ningún objetivo por separado. El componente *<rich: panelMenuGroup>* es utilizado para definir un grupo expandible de opciones dentro del *<rich: panelMenu>* o de él mismo. Además, el componente *<rich: panelMenuItem>* es usado para definir una opción simple dentro de una lista desplegable.

Entre sus características fundamentales se encuentran:

- Alto nivel de personalización de la interfaz
- Diferentes modos de envíos de eventos
- Soporta diferente contenido dentro de cada *<rich: panelMenuItem>*
- Tiene íconos predefinidos y soporta la personalización los mismos
- Soporta la opción de deshabilitar

Este componente se construye dinámicamente según el usuario que se registre en el sistema. Para ello se usan los *tags* perteneciente a JSTL como se muestra en el siguiente código:

```
<rich:panelMenu … rendered="#{usuarioSessionBean.logueado}" >
  <c:forEach items="#{menuBean.groups}" var="grupo">
    <rich:panelMenuGroup label="#{grupo.label}" …>
      <c:forEach items="#{grupo.children}" var="grupito">
        <rich:panelMenuGroup label="#{grupito.label}" …>
          <c:forEach items="#{grupito.children}" var="child">
                  <rich:panelMenuItem          action="#{currentBean.updateCurrent}"
                  reRender="panel" label="#{child.label}">
                        <f:param name="current" value="#{child.name}" />
                  </rich:panelMenuItem>
          </c:forEach>
```

```
        </rich:panelMenuGroup>
      </c:forEach>
    </rich:panelMenuGroup>
  </c:forEach>
</rich:panelMenu>
```

### 3.4.2 <a4j: include>

Permite actualizar áreas determinadas de una página después de una petición AJAX, acorde a las reglas de navegación definidas en los ficheros de configuración de JSF, e implementar formularios en paginado en modo AJAX. Este componente permite definir el área principal de navegación tal y como se explica en el subepígrafe 3.3.2.

### 3.4.3 <rich: panel>

Es un panel personalizable que se dibuja como un rectángulo con bordes definidos y que puede tener encabezado o no. Este componente permite enmarcar el contenido de cada una de las vistas. De esta manera se logran homogenizar todas las páginas de la aplicación haciendo sutil la transición entre ellas.

Sus características principales son:

- Alto nivel de personalización de la interfaz
- Soporte para cualquier contenido dentro del componente

### 3.4.4 <a4j: outputPanel>

Es usado para agrupar componentes en un área AJAX. Se entiende como un área AJAX a la región delimitada por un componente que permite la comunicación asíncrona con el servidor de todos sus componentes.

Su uso es opcional, en RichFaces es posible indicar cualquier id de un componente existente en una vista de componentes para definir áreas de actualización. Para mejorar el rendimiento, RichFaces actualiza solamente el árbol de componentes incluido en el *<a4j: outputPanel>*.

Permite actualizar grupos de componentes dentro de una vista, invocando solamente el id del *<a4j: outputPanel>* que contiene a los mismos.

```
<a4j:outputPanel id="datos">
  <div class="enlinea">
    <a4j:region >
      <h:outputText styleClass="label" value="Sexo" />
      <h:selectOneMenu id="sexo" value="#{modificarCaracterizacionBean.idSexo}" …>
                        <a4j:support event="onchange" reRender="boolEstaEmb/>
       </h:selectOneMenu>
    </a4j:region>
  </div>
  <rich:message for="sexo" styleClass="msg" />
</a4j:outputPanel>
```

### 3.4.5 <a4j: support>

Es uno de los componentes más importantes de este framework y una de las razones por las que se decide usar RichFaces.

Su ventaja principal es que permite adicionar comportamiento AJAX a cualquier componente existente de JSF. Permite generar peticiones asíncronas si los eventos lo demandan y la actualización parcial del contenido de la página después de la respuesta del servidor.

Generalmente se usa asociado a los componentes *<h: selectOneMenu>* que abundan en la aplicación y constantemente tienen que actualizar información del servidor.

```
<h:selectOneMenu id="brigadasD" value="#{trasladoBean.idBrigadaSeleccionada}"
                 valueChangeListener="#{trasladoBean.buscarEstudiantes}"
                 styleClass="combo">
            <f:selectItem itemValue="-seleccione-" />
            <f:selectItems value="#{trasladoBean.brigadasDisponibles}" />
            <a4j:support event="onchange" reRender="caracterizaciones/>
</h:selectOneMenu>
```

### 3.4.6 <a4j: region>

Define un área que se decodifica en el servidor después del envío de la petición AJAX. Es utilizada para la manipulación de componentes que se envían al servidor. Ayuda a reducir la cantidad de datos procesada por el servidor.

Posibilita que cada uno de los componentes de una vista haga peticiones AJAX obteniendo la información solicitada sin afectar las validaciones del resto de los componentes de la misma. Es decir, en una misma vista se cuenta con varios componentes y sus validaciones correspondientes, cuando se hace una petición AJAX. Si el componente que la envía no está dentro de un *<a4j: region>,* provoca que se activen los mensajes de error de los demás componentes. Posibilita además, definir una región

específica para que se pueda mostrar el estado de una petición AJAX en combinación con el componente *<a4j: status>.*

### 3.4.7 <a4j: status>

Genera elementos para mostrar el estado de la petición AJAX que se está procesando. Hay dos estados posibles: una petición AJAX en proceso o finalizada. El componente cuenta con un atributo *for* que debe apuntar al id de un contenedor de AJAX (*<a4j: region>*).

```
<a4j:status id="commonstatus"  for="b8" startText="Cargando..." />

<a4j:region id="b8">
      …
</a4j:region>
```

### 3.4.8 <a4j: commandButton>

Es muy similar al componente original de la librería HTML de JSF (*<h: commandButton*>). La diferencia radica en que el evento *onclick* del botón genera una petición AJAX y permite actualización dinámica de la página luego de llegada una respuesta. Este componente no necesita el soporte de AJAX que brinda *<a4j: support>* porque ya lo tiene incorporado.

Los formularios de cada una de las vistas de la aplicación que terminen con este componente garantizan que se chequeen todas las validaciones de los componentes de la página antes de que la petición se envíe al servidor.

```
<a4j:commandButton value="Trasladar"
    action="#{trasladoBean.RealizarTraslado}" styleClass="boton2"
    style="margin-left: -0.7em;" reRender="panel"/>
```

### 3.4.9 <a4j: actionparam>

Combina las funcionalidades de los componentes *<f: param>* y *<f: actionListener* > de la librería Core de JSF.

Permite pasar datos por parámetros para hacer llamadas a otras páginas a través de peticiones AJAX.

```
<rich:panelMenuItem icon="disc" label="Salir"
      action="#{usuarioSessionBean.desloguearse}"
      reRender="panel, panelMenu, panelUser">
      <a4j:actionparam name="current" value="pages/inicio.xhtml"
                       assignTo="#{currentBean.current}" />
</rich:panelMenuItem>
```

### 3.4.10 <rich: calendar>

Es utilizado para crear elementos de calendario en una página.

Sus características principales son:

- Altamente personalizable
- Representación en forma de popup
- Soporte para deshabilitarlo
- Celdas personalizables
- Sustituciones grandes basadas en personalización con barras de herramientas
- Posicionamiento ágil definido por el usuario

Posee un atributo *popup* que puede tomar valores *true* o *false*, si es verdadero el calendario se presenta en la página como un campo *input* y un botón; de lo contrario se presenta como una ventana más en la página.

El atributo *ondateselected* permite disparar un evento después de seleccionada la fecha, este se puede vincular con *<a4j: support>*.

Además permite mostrar y administrar la hora. Para ello es necesario definir la hora en el patrón de la fecha (d/M/yy HH:mm), el cual garantiza el formato que se muestra en el componente.

Posibilita obtener la fecha y la hora en un mismo campo y obtener esa propiedad en el *bean* como un tipo *Date*, lo cual hace más legible el código.

```
<rich:calendar id="fecha" value="#{nuevaReunionBean.fecha}"
         required="true" datePattern="MM/dd/yyyy HH:mm"
         styleClass="combo"/>
```

Como JSF no cuenta con un componente de este tipo, anteriormente estaba incluido con un código JavaScript, con muchas menos funcionalidades y facilidades para su trabajo.

### 3.4.11 <rich: message> y <rich: messages>

El componente *<rich: message>* es usado para mostrar un mensaje en un componente específico.

Sus principales características son:

- Altamente personalizable

- Peticiones basadas en AJAX
- Tooltip para mostrar los detalles de una parte del mensaje

Su comportamiento es similar al *<h: message>* de la librería HTML de JSF, su diferencia radica en que el componente es actualizado automáticamente después de una petición AJAX sin necesidad de usar un *<a4j: outputPanel>* y se le puede adicionar un marcador al mensaje.

El componente *<rich: messages>* tiene un comportamiento muy parecido al *<rich: message>*, su diferencia radica en que es usado para mostrar todos los mensajes de los componentes de una vista. Posee el atributo *globalOnly* que si toma valor verdadero posibilita captar sólo mensajes del contexto de la aplicación y obviar los mensajes de error de los componentes de la vista. La función principal de ambos componentes es que permiten hacer validaciones sencillas del lado del cliente.

*<rich: message>* posibilita hacer las validaciones necesarias para cada uno de los componentes correspondientes.

*<rich: messages>* por su parte, posibilita captar los mensajes que generados por los *beans* de respaldo.

```
<rich:messages globalOnly="true" styleClass="msg" />

            <a4j:region>
                  <h:selectOneMenu id="estI"
                                    …
                  </h:selectOneMenu>
            </a4j:region>
            <rich:message for="estI" styleClass="msg" />
```

### 3.4.12 <a4j: form>

Es muy similar al componente original de la librería HTML de JSF (*<h: form>*). La única diferencia radica en la posibilidad del envío de peticiones AJAX al servidor. Este componente se utiliza para enmarcar vistas de la aplicación.

### 3.4.13 <rich: modalPanel>

Implementa una ventana de diálogo. Todas las operaciones de la ventana principal de la aplicación son bloqueadas mientras está activada la ventana. Las operaciones de abrir y cerrar esta ventana son utilizadas a través de código JavaScript en el cliente.

Sus características principales son:

- Altamente personalizable
- Soporte para operaciones *drag-and-drop* y ajustes de tamaño
- Posibilidad de restaurar el estado previo del componente en la página anterior

Posibilita bloquear la ventana principal para informar al usuario sobre el estado de algunos recursos de la aplicación.

```
<rich:modalPanel id="#{mp}" autosized="true"  zindex="2000" left="300" top="100">
            <f:facet name="header">
                  <h:outputText value="Información" />
            </f:facet>
            <f:facet name="controls">
                  <h:graphicImage  value="/images/close.png"  style="cursor:pointer"
                  onclick="Richfaces.hideModalPanel('#{mp}')" />
            </f:facet>
            <h:graphicImage value="/images/EnConstruccion.gif"/>

</rich:modalPanel>
```

### 3.4.14 <rich: suggestionBox>

Adiciona la capacidad de sugerir a cualquier campo de texto el posible contenido según la tecla que se accione. Cuando una tecla es presionada se genera una petición AJAX que busca en la base de datos todos los posibles resultados que comiencen con ella, mostrando las opciones en forma de popup debajo del campo de texto.

Sus características principales son:

- Procesa los posibles valores mediante peticiones AJAX sin necesidad de escribir directamente código JavaScript en la página
- Es posible dibujar una tabla en forma de popup con los valores que se sugieren
- Altamente personalizable
- Soporte para la navegación con el teclado
- Fácil configuración para colecciones de datos
- Configuración de restricciones para las peticiones AJAX

Este se utiliza sobretodo para los componentes de entrada de texto en las páginas de búsqueda, facilitando la entrada de datos al usuario.

```
<rich:suggestionbox for="#{for}" tokens=",["
            rules="#{suggestionBean.rules}"
            suggestionAction="#{suggestionBean.autocomplete}" var="result"
            fetchValue="#{result.text}" rows="#{suggestionBean.intRows}"
            first="#{suggestionBean.intFirst}"
            minChars="#{suggestionBean.minchars}"
            shadowOpacity="#{suggestionBean.shadowOpacity}"
            border="#{suggestionBean.border}" width="#{suggestionBean.width}"
            height="#{suggestionBean.height}"
            shadowDepth="#{suggestionBean.shadowDepth}"
            cellpadding="#{suggestionBean.cellpadding}">
            <facet name="nothingLabel"> <h:outputText value="Empty" /> </facet>
            <h:column>
                  <h:outputText value="#{result.text}" />
            </h:column>
</rich:suggestionbox>
```

### 3.4.15 <rich: datascroller> y <rich: dataTable>

*<rich: datascroller>* provee la funcionalidad de *scrolling* para tablas, mediante peticiones AJAX. Puede limitar el máximo de cantidad de datos a mostrar en la tabla, con el uso del atributo *maxPages*. Provee dos grupos de controles para el cambio rápido de páginas en la tabla:

- Números de las páginas para ir de una a otra directamente
- Controles de cambio rápido: *first*, *last*, *next*, *previous*, *fastforward*, *fastrewind*

*<rich: dataTable>* es similar a *<h: dataTable>*, su diferencia radica en su soporte para peticiones AJAX. El soporte AJAX es posible porque fue creado basado en el componente *<a4j: repeat>* y como resultado es posible hacer actualizaciones parciales.

Sus características principales son:

- Posiblidad de insertar componentes complejos como *colGroup* y *subTable*.
- Posibilidad de actualizar un conjunto limitado de cadenas con AJAX.

En conjunto posibilitan crear una tabla para mostrar hipervínculos a los datos solicitados después de una búsqueda y paginarlos en función de la cantidad.

```
<rich:dataTable id="listarCaracterizaciones>
                                …
      <f:facet name="footer">
              <rich:datascroller for="listarCaracterizaciones" maxPages="5" />
      </f:facet>
</rich:dataTable>
```

### 3.4.16 Utilizando skinnability

El *skinnability* de RichFaces permite personalizar el esquema de colores y otras propiedades de estilo de sus componentes de tres formas diferentes:

- Parámetros *skin* definidos en el framework.
- Clases CSS predefinidas para componentes.
- Clases de estilo del desarrollador.

Aprovechando estas posibilidades se modificó el fichero llamado *ruby.skin.properties* que es uno de los *skin* que trae por defecto el framework, personalizándolo para la aplicación.

# CONCLUSIONES

La combinación de los frameworks RichFaces y Facelets es la variante más indicada para integrar AJAX a JSF en la capa de presentación del proyecto Kainos y mejorar la interacción entre el usuario y la aplicación.

La implementación de esta combinación permite optimizar el ciclo de vida de las peticiones en la aplicación del proyecto Kainos y lograr una mayor reutilización de código.

Como resultados adicionales de este trabajo, se obtienen:

- Sistematización de las tecnologías que, mediante la integración de AJAX a JSF, permiten lograr interfaces de usuario más amigables.
- Reorganización de la arquitectura del proyecto Kainos en cuanto a estructura de paquetes, archivos de configuración y vistas.

# RECOMENDACIONES

- Utilizar la propuesta definida e implementada, para su aplicación y estandarización en futuras aplicaciones del proyecto Kainos.
- Integrar la propuesta definida al framework ArBaWeb como alternativa para la capa de presentación. Este es un framework creado en la UCI, que permite el rápido desarrollo de software para aplicaciones basadas en JEE.
    - Adicionar soporte al plugin ArBaWeb Integrator Tools.

# REFERENCIAS BIBLIOGRÁFICAS


1. **Bankinter, Fundación de la Innovación.** *Web 2.0 El negocio de las redes sociales.* 2007.
2. **Gallardo, David, McGovern, Robert.** *Eclipse In Action: A Guide for Web Developers.* 2003.
3. **Geary, David, Horstmann, Cay.** *Core JavaServer™ Faces, Second Edition.* s.l.: Prentice Hall, 2007. 0-13-173886-0.
4. **Javalobby.** *Ajax: a new approach to web application.* [En línea] 18 de febrero de 2005. [Citado el: 29 de abril de 2008.] http://www.javalobby.org/articles/ajax/.
5. **JCP.** The Java Community Process(SM) Program. *JSRs: Java Specification Request -Li.* [En línea] Mayo de 2006. [Citado el: 20 de abril de 2008.] http://jcp.org/en/home/index.
6. **Johnson, R. E., Foote, Brian.** *Designing Reusable Classes.* 1988.
7. **Kaiser.** *Software Paradigms.* 2005.
8. **Microsystems, Sun.** Sun Microsystems. *JavaServer Pages Overview.* [En línea] [Citado el: 7 de mayo de 2008.] http://java.sun.com/products/jsp/overview.html.
9. **Olson, Steven Douglas.** *Ajax on Java.* s.l.: O'Reilly, 2007. 0-596-10187-2.
10. **Spring.** *Springframework.org.* [En línea] 2006. [Citado el: 20 de abril de 2008.] http://www.springframework.org/.
11. **The Apache Software Foundation**. *Apache Tomcat.* [En línea] 2007. [Citado el: 2 de Mayo de 2008.] http://tomcat.apache.org/.

12. **Wikipedia**, la enciclopedia libre. *Aplicación web.* [En línea] [Citado el: 25 de Mayo de 2008.] http://es.wikipedia.org/wiki/Aplicaci%C3%B3n_web.

# BIBLIOGRAFÍA


1. **ABU, MZE, POS, RBO, AMA.** *EL4J Reference Documentation Version 1.3 Incremental Improvements for Spring.* Suiza: ELCA Informatique SA, 2008.
2. **Bankinter, Fundación de la Innovación.** *Web 2.0 El negocio de las redes sociales.* 2007.
3. **Component Suit Showcase**. *Rich Text.* [En línea] ICEsoft Technologies Inc., 2008. [Citado el: 15 de Mayo de 2008.] http://component-showcase.icefaces.org/component-showcase/showcase.iface.
4. ***Developer's Guide***. s.l.: ICEsoft Technologies, Inc., 2007.
5. **Gallardo, David, McGovern, Robert.** *Eclipse In Action: A Guide for Web Developers.* 2003.
6. **Geary, David, Horstmann, Cay.** *Core JavaServer™ Faces, Second Edition.* s.l.: Prentice Hall, 2007. 0-13-173886-0.
7. **Hightower, Richard.** IBM. *Facelets fits JSF like a glove.* [En línea] 21 de Febrero de 2006. [Citado el: 7 de mayo de 2008.] http://www.ibm.com/developerworks/java/library/j-facelets/.
8. **Hookom, Jacob.** Facelets - JavaServer Faces View Definition Framework. *Facelets - JavaServer Faces View Definition Framework Developer Documentation.* [En línea] 2008. [Citado el: 20 de mayo de 2008.] https://facelets.dev.java.net/nonav/docs/dev/docbook.html.
9. **Javalobby.** *Ajax: a new approach to web application.* [En línea] 18 de febrero de 2005. [Citado el: 29 de abril de 2008.] http://www.javalobby.org/articles/ajax/.
10. **JCP.** The Java Community Process(SM) Program. *JSRs: Java Specification Request -Li.* [En línea] Mayo de 2006. [Citado el: 20 de abril de 2008.] http://jcp.org/en/home/index.

11. **Johnson, R. E., Foote, Brian.** *Designing Reusable Classes.* 1988.

12. **JSCAPE, SoftAspects.** *WebGalileo Faces Developer Guide.* 2007.

13. **JSF AJAX Component Library Feature Matrix**. [En línea] 4 de Junio de 2008. [Citado el: 4 de Junio de 2008.] http://www.jsfmatrix.net/.

14. **Kaiser.** *Software Paradigms.* 2005.

15. **Mann, Kito D.** *JavaServer Faces in Action.* s.l. : Manning Publications, 2005.

16. **Mateu, Carles.** *Desarrollo de aplicaciones web.* Catalunya : Eureca Media, SL, 2004. 84-9788-118-4.

17. **Microsystems, Sun.** Sun Microsystems. *JavaServer Pages Overview.* [En línea] [Citado el: 7 de mayo de 2008.] http://java.sun.com/products/jsp/overview.html.

18. **Olson, Steven Douglas.** *Ajax on Java.* s.l.: O'Reilly, 2007. 0-596-10187-2.

19. **Pérez, Javier Eguíluz.** *Introducción a AJAX*. 2007.

20. **Pimentel, Luis Alberto González, Pérez, Iósev Rivero.** *ArBaWeb: arquitectura base sobre la web.* 2007.

21. ***QuipuKit Developer's Guide***. s.l.: TeamDev Ltd., 2007.

22. ***RichFaces Developer Guide***. s.l.: Red Hat, 2007.

23. **Ruiz, Yenivet Paula, González, Yanaisy Galbán.** *Tecnología Ajax para el desarrollo Web.* 2007.

24. **Spring.** *Springframework.org.* [En línea] 2006. [Citado el: 20 de abril de 2008.] http://www.springframework.org/.

25. **Sun.** Java en castellano. *Introducción a la Tecnología JavaServer Faces.* [En línea] [Citado el: 20 de Mayo de 2008.] http://www.programacion.com/java/tutorial/jsf_intro/.

26. **The Apache Software Foundation**. *Apache Tomcat.* [En línea] 2007. [Citado el: 2 de Mayo de 2008.] http://tomcat.apache.org/.

27. **WebGalileoFaces Demostration Application**. *Demostration Application WebGalileoFaces Open Sources JSF Components.* [En línea] SoftAspects, JSCAPE, 2007 . [Citado el: 15 de mayo de 2007.] http://support.softaspects.com:9080/webgalileofaces/.

28. **Wikipedia**, la enciclopedia libre. *Aplicación web.* [En línea] [Citado el: 25 de Mayo de 2008.] http://es.wikipedia.org/wiki/Aplicaci%C3%B3n_web.

29. ***ZK Developer's Guide***. s.l.: Potix Corporation, 2007.

30. **ZK™ Simply Ajax and Mobile**. *ZK - #1 Ajax and Mobile Framework.* [En línea] Potix Corporation, 2008. [Citado el: 15 de mayo de 2008.] http://www.potix.com/zkdemo/userguide/.

# GLOSARIO DE TÉRMINOS

**API** (Interfaz de Programación de Aplicaciones): Conjunto de especificaciones para comunicarse con una aplicación, normalmente para obtener información y utilizarla en otros servicios. Una API representa una interfaz de comunicación entre componentes software. Se trata del conjunto de llamadas a bibliotecas que ofrecen acceso a servicios desde los procesos y representa un método para conseguir abstracción en la programación, generalmente entre los niveles o capas inferiores y los superiores del software. Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general, por ejemplo, para dibujar ventanas o íconos en la pantalla.

**Beans**: Son simples clases de Java con sus métodos de acceso. Se usan para conservar el estado del usuario y los datos de los componentes. Usualmente implementan métodos de validaciones y manejadores de eventos que son invocados por la aplicación desde sus componentes.

**Cocoon** (Apache Cocoon): Framework de desarrollo web que se enfoca en la publicación de XML y XSLT y está construido usando el lenguaje de programación Java y basado en Spring. Esta construido alrededor del concepto de desarrollo basado en componentes.

**CRUD** (Create-Read-Update-Delete): Conocido como el padre de todos los patrones de capa de acceso a datos. Describe que cada objeto debe ser creado en la base de datos para que sea persistente. Una vez creado, la capa de acceso debe tener una forma de leerlo para poder actualizarlo o simplemente borrarlo.

**CSS** (Hojas de Estilo en Cascada): Lenguaje formal usado para definir la presentación de un documento escrito en HTML o XML (y por extensión en XHTML). El W3C (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los navegadores. La idea del desarrollo de CSS es separar la estructura de un documento de su presentación.

**CVS (**Concurrent Version System ó Concurrent Versioning System): Sistema de control de versiones. Mantiene el registro de todo el trabajo y los cambios en los ficheros (código fuente principalmente) que forman un proyecto y permite que distintos desarrolladores (potencialmente situados a gran distancia) colaboren.

**DOM** (Modelo en Objetos para la representación de Documentos): Modelo computacional a través de la cual los programas y scripts pueden acceder y modificar dinámicamente el contenido, estructura y estilo de los documentos HTML. Su objetivo es ofrecer un modelo orientado a objetos para el tratamiento y manipulación de las páginas. El DOM es una API para acceder, añadir y cambiar dinámicamente contenido estructurado en documentos con lenguajes como JavaScript.

**DTD** (Definición de Tipo de Documento): Descripción de estructura y sintaxis de un documento XML. Su función básica es la descripción del formato de datos, para usar un formato común y mantener la consistencia entre todos los documentos que utilicen la misma DTD. De esta forma, dichos documentos, pueden ser validados, conocen la estructura de los elementos y la descripción de los datos que trae consigo cada documento, y pueden además compartir la misma descripción y forma de validación dentro de un grupo de trabajo que usa el mismo tipo de información.

**FTP** (Protocolo de transferencia de archivos): Protocolo de transferencia de archivos entre sistemas conectados a una red TCP basado en la arquitectura cliente-servidor, de manera que desde un equipo cliente nos podemos conectar a un servidor para descargar archivos desde él o para enviarle nuestros propios archivos independientemente del sistema operativo utilizado en cada equipo.

**GUI** (Interfaz Gráfica de Usuario): Tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Habitualmente las acciones se realizan mediante manipulación directa para facilitar la interacción del usuario con la computadora.

**HTML** (Lenguaje de Marcado de Hipertexto): Lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

**HTTP** (Protocolo de Transferencia de Hipertexto): Protocolo usado en las transacciones de la Web. Define la sintaxis y la semántica que utilizan los elementos software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

**HTTPS**: Protocolo de red basado en el protocolo HTTP, destinado a la transferencia segura de datos de hipertexto. El sistema HTTPS utiliza un cifrado basado en las Secure Socket Layers (SSL) para crear un canal cifrado (cuyo nivel de cifrado depende del servidor remoto y del navegador utilizado por el cliente) más apropiado para el tráfico de información sensible que el protocolo HTTP. Cabe mencionar que el uso del protocolo HTTPS no impide que se pueda utilizar HTTP.

**Java**: Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

**JEE**: Versión empresarial de Java, que después de J2EE 1.4 es llamada JEE 5.0; destacando así los cambios significativos de los frameworks de peso ligero especificados en los estándares empresariales de Java. JEE incluye varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc y define cómo coordinarlos. Configura algunas especificaciones únicas para componentes. Estas incluyen Enterprise JavaBeans, servlets, portlets, JavaServer Pages y varias tecnologías de servicios web. Otros beneficios añadidos radican, por ejemplo, en que el servidor de aplicaciones puede manejar transacciones, seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados. Es decir, los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes que en tareas de mantenimiento de bajo nivel.

**JSON** (Notación de Objetos para JavaScript): Formato ligero para el intercambio de datos, es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML. La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX. Una de las ventajas de JSON sobre XML como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un analizador semántico de JSON.

**JSTL**: Tecnología proporcionada por Sun Microsystems que extiende de JSP proporcionando cuatro librerías de etiquetas (Tag Libraries), con utilidades ampliamente utilizadas en el desarrollo de páginas web dinámicas. Estas librerías de etiquetas extienden de la especificación de JSP. Su API nos permite además desarrollar nuestras propias librerías de etiquetas.

**PDA** (Asistente Digital Personal): Computador de mano originalmente diseñado como agenda electrónica con un sistema de reconocimiento de escritura. Hoy día se puede usar como una computadora doméstica (ver películas, crear documentos, juegos, correo electrónico, navegar por Internet, reproducir archivos de audio, etc.)

**Plugins**: Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica. Es una forma de expandir programas de forma modular, de manera que se puedan añadir nuevas funcionalidades sin afectar a las ya existentes ni complicar el desarrollo del programa principal.

**Portlets**: Componentes modulares de interfaz de usuario gestionados y visualizados en un portal web. Los portlets producen fragmentos de código de marcado que se agregan en una página de un portal.

Típicamente, una página de un portal se visualiza como una colección de ventanas de *portlet* que no se solapan, donde cada una de estas muestra un *portlet*. Por lo tanto un *portlet* (o colección de *portlets*) se asemeja a una aplicación web que está hospedada en un portal.

**Renderers**: Elementos que permiten generar una imagen a partir de un modelo. Son los responsables de mostrar los componentes de interfaz de usuarios y trasladar los datos de entrada del usuario al los valores del componente. Pueden ser diseñados para trabajar con uno o más componentes y un componente puede ser asociado a más de un Render.

**RIA**: (Aplicaciones Ricas en Internet) – Formas avanzadas de que un usuario interactúe con una aplicación o página web, ofreciéndole funciones y nuevas posibilidades útiles intentando al mismo tiempo mantener la simplicidad aparente. Los entornos RIA no producen recargas de página, ya que desde el principio se carga toda la aplicación y sólo se produce comunicación con el servidor cuando se necesitan datos externos como datos de una base de datos o de otros ficheros externos.

**Skinnability**: Propiedad que permite personalizar una serie de elementos gráficos que, al aplicarse sobre un determinado software, modifican su apariencia externa.

**ViewHandler**: Es la clase que maneja las vistas de una aplicación, incide en las fases RestoreView y Render Response del ciclo de vida de JSF. Permite a los clientes abrir, manipular y disponer de vistas. También coordina las dependencias entre vistas y organiza su actualización.

**Wizard**: Asistente de interfaz de usuario que guía paso por paso, a través de cuadros de diálogo para cumplir una tarea específica.

**XHTML** (Lenguaje Extensible de Marcas de Hipertexto): Versión XML más avanzada del lenguaje HTML que se utiliza para la creación y visualización de páginas web. XHTML sirve únicamente para transmitir la información que contiene un documento, dejando para hojas de estilo (como las hojas de estilo en cascada) su aspecto y diseño en distintos medios (computadoras, PDAs, teléfonos móviles, impresoras...) y para JavaScript su comportamiento.

**XML** (Lenguaje de Marcas Extensible): Metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una manera de definir lenguajes para diferentes necesidades. XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

**XMLHttpRequest** (Lenguaje de Marcado Extendido/Protocolo de Transferencia de Hipertexto): También referida como XMLHTTP, es una interfaz empleada para realizar peticiones HTTP y HTTPS a servidores WEB. Para los datos transferidos se usa cualquier codificación basada en texto, incluyendo: texto plano, XML, JSON, HTML y codificaciones particulares específicas. La interfaz se presenta como una clase de la que una aplicación cliente puede generar tantas instancias como necesite para manejar el diálogo con el servidor.

**XSD** (XML Schema Definition): Concebida como alternativa a las DTD. Es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML.

**XSLT**: Estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML. Las hojas de estilo XSLT (aunque el término de hojas de estilo no se aplica sobre la función directa del XSLT) realizan la transformación del documento utilizando una o varias reglas de plantilla: unidas al documento fuente a transformar, esas reglas de plantilla alimentan a un procesador de XSLT, el cual realiza las transformaciones deseadas colocando el resultado en un archivo de salida o, como en el caso de una página web, directamente en un dispositivo de presentación, como el monitor de un usuario.

**XUL** (Lenguaje basado en XML para la Interfaz de Usuario): Aplicación de XML a la descripción de la interfaz de usuario en el navegador Mozilla. XUL no es un estándar. La mejor fuente para encontrar material de referencia sobre XUL son páginas especializadas así como libros técnicos. La principal ventaja de XUL es que aporta una definición de interfaces GUI simple y portable. Esto reduce el esfuerzo empleado en el desarrollo de software.