UNIVERSISDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

“Implementación de Aplicaciones Framework de Código Abierto en Java con JavaServer Faces”

TRABAJO ESCRITO EN LA MODALIDAD DE “TESIS”

PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

PRESENTA:

MIGUEL ANGEL SÁNCHEZ HERNÁNDEZ

ASESOR: M. en C. JESÚS HERNÁNDEZ CABRERA

MÉXICO, 2007.

A mis Padres y Hermano por brindarme apoyo durante todo el transcurso de mi carrera, que al igual que yo fueron constantes en esfuerzos y me motivaron de una u otra forma para seguir adelante y no desistir en la lucha por conseguir mis objetivos personales, y en esta última etapa a mi Esposa.

Con Cariño:

A mi Padre Cipriano Sánchez García que es mi mejor Amigo.

Agradecimiento:

A mi asesor M. en C. Jesús Hernández Cabrera

Estimación y Afecto:

Profesores, Compañeros y Amigos

ÍNDICE

I

Introducción

Se puede decir que a lo largo de la historia han existido períodos en que han aparecido tecnologías desorganizadoras, las cuales han revolucionado la vida cotidiana y han obligado a las comunidades científicas, políticas y económicas a reconsiderar sus prácticas. Internet es la tecnológica que recientemente ha cambiado la forma en que las personas interactúan entre sí y la forma en que hacen negocios como transacciones, correo exprés, etc.

Tradicionalmente, los sistemas corporativos se diseñaban utilizando el modelo cliente-servidor, en el cual los sistemas clientes solicitaban procesamiento de los sistemas servidores, sin embargo, los sistemas empresariales estaban a su vez evolucionando, y se tiene un nuevo modelo conocido como servidores web que está reemplazando en forma gradual el modelo cliente-servidor en las corporaciones.

En el modelo de servidores web, los programadores de aplicaciones ensamblan aplicaciones a partir de un conjunto de componentes de procesamiento conocidos como servidores web, cada servicio web es independiente de los demás servicios web y de las aplicaciones. Las aplicaciones cliente se comunican con una aplicación servidora, la cual interactúa con los servidores web necesarios.

Java Enterprise Edition (J2EE), es una tecnología que da soporte para crear servicios web, por lo cual está muy cercana a Internet. Esto se debe a que permite crear una infraestructura de software robusta, la cual hace que la construcción de aplicaciones basadas en la web sea rentable y que su utilización sea fiable en cualquier sistema de misión crítica.

Se dará un vistazo en forma general pero concreta sobre las tecnologías de Servlet y JSP, así como analizaremos el Framework de Struts y culminaremos con el Framework de JavaServer Faces que este es lo último sobre el desarrollo de aplicaciones web, pero antes daremos un repaso sobre que es un Framework y el Patrón Modelo-Vista-Control (MVC).

1

Framework

1.1 ¿Qué es un Framework?

Un Framework es un conjunto extensible de objetos para funciones relacionadas, es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Un Framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado, además de otros componentes de software para así poder desarrollar y unir los diferentes componentes de un proyecto.

También éste puede ser definido como un conjunto de clases[1], generalmente algunas de ellas abstractas, y la relación que se establecen entre ellas en donde se obtiene un diseño abstracto para la solución de un conjunto de problemas.

En general, un Framework es:

- Un conjunto cohesivo de interfaces y clases que colaboran para proporcionar los servicios de la parte central e invariable de un subsistema lógico.
- Contiene clases concretas y especialmente abstractas que definen las interfaces a las que hay que ajustarse, interacciones de objetos en los que participan, y otras invariantes.
- Normalmente (aunque no necesariamente) requiere que el usuario del Framework defina subclases de las clases del Framework existentes para utilizar, adaptar y extender los servicios del Framework.
- Tiene clases abstractas que podrían contener tanto métodos abstractos como concretos.
- Si las clases definidas por el usuario (por ejemplo, nuevas clases) recibirán mensajes desde las clases predefinidas del Framework. Estos mensajes normalmente se manejan implementando métodos abstractos en las superclases.

[1] Las clases son una descripción abstracta de un grupo de objetos que comparten características (atributos) y operaciones (métodos).

1.2 Reutilización de Frameworks

En general como se mencionó, los Frameworks son construidos en base a lenguajes orientados a objetos, ya que éstos permiten una modulación de los componentes, además de una optimización y reutilización del código.

La mayoría de casos implementan uno o más patrones[2] de diseño de software que aseguran la escalabilidad del producto en cuestión.

El Framework captura las decisiones de diseño comunes a un tipo de aplicación logrando establecer un modelo común con todas ellas, asignando responsabilidades y estableciendo colaboraciones entre las clases que forman el modelo. La reutilización se produce cuando se hace la instanciación de un Framework.

Los Frameworks ofrecen un alto grado de reutilización (mucho más que con clases individuales), en consecuencia, si una organización está interesada en incrementar su grado de reutilización de software, entonces debería enfatizar en la creación de Frameworks.

Se sabe que los Frameworks de persistencia son un conjunto de tipos de propósito general, reutilizable y extensible, que proporciona funcionalidad para dar soporte a los objetos persistentes. Un servicio de persistencia (o subsistema) realmente proporciona el servicio y se creará con un Framework de persistencia. Por ejemplo, un servicio de persistencia tiene que traducir los objetos a registros y guardarlos en una base de datos, y traducir los registros a objetos cuando los recuperamos de la base de datos.

1.3 Puntos Calientes (hot-spots) de un Framework

Estos puntos calientes o Hot-Spots son las clases o los métodos abstractos [3] que pueden ser implementados o puestos en ejecución por lo que se llega a la conclusión que los Frameworks no son ejecutables.

[2] Un patrón de diseño es un set de metodologías probadas para resolver problemas comunes en el diseño de aplicaciones.
[3] Los métodos abstractos actúan como reservas de espacio para los métodos que se implementan en las subclases.

Para poder generar un Hot-Spots se debe hacer una instanciación[4] del Framework poniendo el código especifico de la aplicación en ejecución para cada punto caliente.

Una vez que se tiene la instancia de estos puntos calientes el Framework utilizará estas clases o métodos abstractos usando la repetición de la llamada o callback [5], el código del usuario del servicio declara que desea la ocurrencia a un determinado evento. Entonces el código del proveedor del servicio realiza la repetición de la llamada con el código del usuario del servicio al momento de ocurrir ese determinado evento.

1.4 Puntos Congelados (Frozen-Spots) de un Framework

Pero en los Frameworks algunas características no son mutables ni tampoco pueden ser alteradas, estos puntos inmutables constituyen el núcleo o Kernel de un Framework, éstos se conocen como puntos congelados o Frozen-Spots, a diferencia de los puntos calientes los puntos congelados son pedazos de código puestos en ejecución ya en el Framework que llaman a uno o más puntos calientes. El kernel será constante y presentará la parte de instancia de cada Framework.

Éste se puede pensar como una analogía de un motor, que requiere potencia y a diferencia de un motor tradicional un motor Framework se le puede proporcionar muchas entradas de potencia, y cada entrada de potencia es un punto caliente y éstos a su vez deben ser accionados para que el motor funcione. Por lo que los generados de potencia son el código especifico de la aplicación que se deben conectar a los puntos calientes.

El código agregado de la aplicación será utilizado por el código del Kernel del Framework y el motor empezará a correr hasta que estén conectadas todas las entradas, esto se muestra en la figura 1.

[4] Es el echo de producir o crear objetos de manera que puedan usarse en un programa.
[5] Acto de repetir la autentificación del número de usuario en caso de reconexión

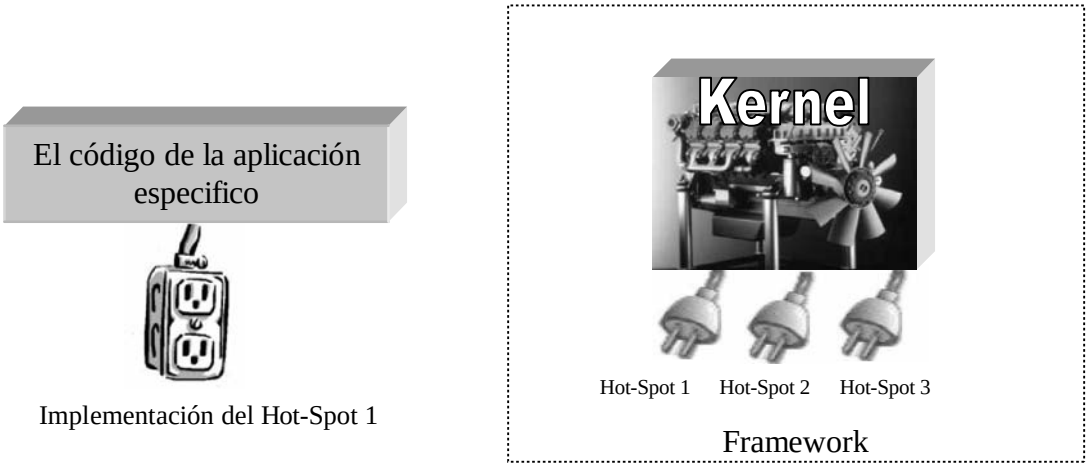


Figura 1.1 ¿Cómo se puede utilizar una analogía de un motor para una Framework?

1.5 La Instanciación y Documentación

Se pueden clasificar los Frameworks según su extensibilidad como caja blanca, negra o gris.

Caja Blanca: También llamados Frameworks con arquitectura de configuración-conducidos en donde su instanciación es solamente posible a través de nuevas clases, estas clases y el código se pueden introducir en el marco por herencia o composición. Esto lleva a que uno debe de entender muy bien el Framework para poder generar una aplicación.

Caja Negra: Se producen instancias a través de escrituras o scripts [6] de configuración, después de hacer la configuración una herramienta de instanciación se encarga de crear las clases y el código fuente, con esto se logra que el usuario no requiera conocer a detalle el funcionamiento interno del Framework.

Caja Gris: Estas tienen las características de ambas cajas.

[6] Son un conjunto de comandos escritos en un lenguaje interpretado para automatizar ciertas tareas de una aplicación.

La capacidad de poder crear sistemas ejecutables desde un Framework dependerá de los mecanismos de instanciación y la documentación del mismo. Si éste está pobremente documentado pocos usuarios lo utilizaran.

1.6 Costo del Dominio y la Experiencia

Un Framework se crea para generar una aplicación de dominio específico, para este propósito una de las fases del desarrollo del Framework es el análisis del dominio, éste procura descubrir los requisitos del dominio y los posibles requerimientos a futuro.

Durante el análisis del dominio los puntos calientes y los puntos congelados se destapan parcialmente, se intenta caracterizar el tamaño y la complejidad del dominio elegido. Si el dominio es demasiado grande se desperdiciara tiempo en recolectar y evaluar la información y sus recursos.

Por lo que el tiempo de elaboración del Framework y su costo serán excesivos (persona-año), por otro lado si se elige un dominio demasiado estrecho, se reduce la aplicabilidad del Framework y las aplicaciones generadas serán similares para justificar el esfuerzo de la construcción del mismo. Por eso es importante tener en cuenta cuáles son los puntos calientes más importantes y cuáles son superfluos.

Cuando se habla de un Framework bien diseñado es cuando se tiene una documentación sólida y resuelve los requerimientos del dominio. Un Framework que persevera el concepto de encapsulación orientada al objeto y una interfaz pública bien definida tiene mucha probabilidad de compatibilidad a nivel de interfaz a través de actualizaciones y revisiones. La evaluación del diseño de un Framework y/o su implementación no es directa, se debe considerar la experiencia del diseñador, la complejidad del proceso de instanciación, los requerimientos resueltos, y no resueltos y también la actualización del mapa de la ruta de trabajo, que contiene los planes para poner al día el Framework, ya que éstos indican las nuevas versiones o su compatibilidad entre ellas.

1.7 Ventajas de utilizar un Framework

Tomemos esta frase de Héctor Berlioz para explicar un Framework, “El tiempo es un gran profesor, pero desgraciadamente mata a todos sus alumnos”, constantemente tenemos que hacer modificaciones a nuestras aplicaciones para que se adapten a las nuevas herramientas técnicas y con el poco tiempo que se cuenta impide a la aplicación mantenerla actualizada, en cambio si utilizo un Framework tendré los siguientes beneficios:

- Programar rápidamente: No se tiene que reinventar la rueda, sólo saberla ocupar.
- Estructura: Se tiene una estructura de directorios y archivos generales.
- Ahorro de trabajo: Añadir nuevas funcionalidades a las aplicaciones.
- Consistencia: La estructura que provee el Framework es igual a cada parte de tu aplicación.
- Escribir mejor código: Como el Framework tiene orden el programador debe seguir ese orden en su aplicación.
- Utilizar lo último de herramientas técnicas: Por ejemplo, con sólo actualizar el Framework que se ocupa y añadir las actualizaciones, uno podía ocupar AJAX[7] en sus aplicaciones si el Framework contiene esta tecnología.
- Soporte gratuito: Tutoriales y listas de correos gratuitos para ocupar el Framework.
- Programadores actualizando el Framework: Gente que constantemente actualiza y revisa el Framework.

Como ejemplo pongamos a un GUI (Interfaz de Usuario Grafico), como es el AWT(Herramientas de Ventanas Abstractas) o Swing de Java. El Framework Swing de GUI de Java proporciona muchas clases e interfaces para las funciones principales de la GUI. Los desarrolladores pueden incluir elementos gráficos especializados creando subclases de las clases de Swing y redefiniendo ciertos métodos. También se pueden conectar diversos compartimientos

[7] **AJAX**, acrónimo de ***A**synchronous **J**avaScript **A**nd **XML*** (JavaScript y XML asíncronos, donde XML es un acrónimo de e**X**tensible **M**arkup **L**anguage), es una técnica de desarrollo web para crear aplicaciones interactivas, estas se ejecutan en el cliente, es decir, en el navegador del usuario, y mantiene comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad en la misma.

de respuesta a los eventos en las clases de los elementos gráficos predefinidos (como un botón de JButton).

1.8 Desventajas de un Framework

Si no está bien documentado nunca se podrá dominar el Framework que se intenta ocupar en nuestra aplicación, porque el periodo de aprendizaje llevará mucho tiempo, atrasando el desarrollo, otro problema es que no se esté constantemente revisando y actualizando el Framework para que se adapte a las nuevas herramientas técnicas, o por último, que no haga lo que se supone que realiza dando como consecuencia una infinidad de errores que no serán de nuestra aplicación, sino del mismo Framework.

2

El Patrón Modelo-Vista-Controlador

2.1 ¿Qué es un Patrón en forma General?

Para comenzar, intentaremos establecer una definición básica del concepto de patrón. De la misma forma que sucede con otros conceptos de la informática (y como es el caso de la arquitectura), no es fácil establecer una definición taxativa y definitiva del concepto de patrón.

Fue por los años 1994, que apareció el libro "Design Patterns: Elements of Reusable Object Oriented Sofware" escrito por los ahora famosos Gang of Four (GoF, que en español es la pandilla de los cuatro) formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Ellos recopilaron y documentaron 23 patrones de diseño aplicados usualmente por expertos diseñadores de software orientado a objetos. Desde luego que ellos no son los inventores ni los únicos involucrados, pero fue el inicio, después de la publicación de ese libro que empezó a difundirse con más fuerza la idea de patrones de diseño.

De hecho, en el capítulo 2 del libro "**Pattern Hatching**", John Vlissides (miembro del GoF) hace hincapié en la dificultad de esta tarea. En las diversas obras de referencia podemos encontrar diferentes definiciones.

En el libro "**The Timeless Way of Building**" (obra de referencia donde se plantea por primera vez la teoría de los patrones aplicada a la Arquitectura Civil y Urbanismo), el Arquitecto Christopher Alexander define al patrón de la siguiente manera:

"Cada patrón es una regla de 3 partes, que expresa una relación entre un contexto, un problema y una solución. Como un elemento en el mundo, cada patrón es una relación entre un contexto, un sistema de fuerzas que ocurren repetidamente en ese contexto y una configuración espacial que permite que esas fuerzas se resuelvan entre sí.".

"Como elemento de un lenguaje, un patrón es una instrucción que muestra como puede ser usada esta configuración espacial una y otra vez para resolver el sistema de fuerzas, siempre que el contexto lo haga relevante."

Continuando con la definición anterior, podemos agregar otro párrafo de Alexander, citado en la obra de referencia "Design Patterns" en la sección "¿Qué es un patrón de diseño?":

"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma."

Como reflexión final, podemos añadir que un patrón de diseño no es una solución en sí misma, sino la documentación de la forma en que construyeron soluciones a problemas similares en el pasado, lo cual permite una mejor gestión de la experiencia y transferencia de conocimientos.

2.2 Patrones de Software

De la misma forma que hemos buscado una definición para el concepto general de patrones en la literatura existente, haremos lo propio con los patrones de software. Una definición de este tipo de patrones que goza de amplia aceptación en la comunidad del software es la dada por Richard Gabriel en "**A Timeles Way of Hacking**":

"Cada patrón es una regla de 3 partes, que expresa una relación entre un contexto, un sistema de fuerzas que ocurren repetidamente en ese contexto y una configuración de software que permite que se resuelvan esas fuerzas."

Como podemos fácilmente observar, esta definición es una adaptación al mundo del software de la definición dada por Christopher Alexander, citada en la sección anterior.

Otra definición complementaria, más enfocada en el ámbito técnico, es la que se da en el libro de Microsoft "**Enterprise Development Reference Architecture**":

"Una descripción de un problema recurrente que ocurre en un contexto determinado y, basada en un conjunto de fuerzas, recomienda una solución. La solución es usualmente un mecanismo simple: una colaboración entre dos o más clases, objetos, servicios, procesos, threads, componentes o nodos que trabajan juntos para resolver el problema identificado por el patrón."

Finalmente, no podemos dejar de citar la frase con que comienza el libro "**Pattern Oriented Software Architecture, Volume 1**", que incluimos a continuación:

"Los patrones le ayudan a construir sobre la experiencia colectiva de ingenieros de software experimentados. Éstos capturan la experiencia existente y que ha demostrado ser exitosa en el desarrollo de software, y ayudan a promover las buenas prácticas de diseño. Cada patrón aborda un problema específico y recurrente en el diseño o implementación de un sistema de software."

2.3 Características de un buen Patrón

Documentar buenos patrones puede ser una tarea muy difícil, por lo que un buen patrón:

- Resuelve un problema: Los patrones capturan soluciones, no principios o estrategias abstractas.
- Es un concepto probado: Capturan soluciones, no teorías o especulaciones. En el caso del "Design Patterns", el criterio para decidir si algo era un patrón o no, era que éste debía tener al menos 3 implementaciones reales.
- La solución no es obvia: Muchas técnicas de resolución de problemas (como los paradigmas o métodos de diseño de software) intentan derivar soluciones desde principios básicos. Los mejores patrones generan una solución a un problema indirectamente (un enfoque necesario para los problemas de diseño más difíciles).
- Describe una relación: Los patrones no describen módulos sino estructuras y mecanismos.
- Tiene un componente humano significante: El software sirve a las personas. Los mejores patrones aplican a la estética y a las utilidades.

2.4 Niveles de Patrones

En "**Pattern Oriented Software Architecture, Volume 1**", se define una taxonomía como la que se muestra en la figura 2.1. Los patrones en cada grupo varían respecto a su nivel de detalle y abstracción.:

Figura 2.1: Patrones según el nivel de detalle

Es importante destacar que esta división no es absoluta ni totalmente comprensiva, dado que no incluye a otros patrones de amplia aceptación y utilización como los de análisis, integración de aplicaciones, pruebas, etc.

2.5 Patrones de Diseño

Analizamos estos Patrones porque en ellos se encuentra el Patrón MVC(Modelo-Vista-Control), que se explicará más adelante. Un patrón de diseño describe una estructura recurrente de componentes que se comunican para resolver un problema general de diseño en un contexto particular. Nomina, abstrae e identifica los aspectos clave de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizable. Identifica las clases e instancias participantes, sus roles y colaboraciones y la distribución de responsabilidades. Tiene, en general, 4 elementos esenciales (los cuales se explican en gran detalle en el libro "**Design Patterns**"):

- Nombre: Permite describir, en una o dos palabras, un problema de diseño junto con sus soluciones y consecuencias. Al dar nombres a los patrones estamos incrementando nuestro vocabulario de diseño, lo cual nos permite diseñar y comunicarnos con un mayor

nivel de abstracción (en lugar de hablar de clases individuales nos referimos a colaboraciones entre clases).

- § Problema: Describe cuándo aplicar el patrón. Explica el problema y su contexto. A veces incluye condiciones que deben darse para que tenga sentido la aplicación del patrón.
- § Solución: Describe los elementos que constituyen el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño o implementación en concreto, sino que es más bien una plantilla que puede aplicarse en muchas situaciones diferentes.
- § Consecuencias: son los resultados, así como ventajas e inconvenientes de aplicar el patrón.

Los patrones de diseño no son dogmas que deben ser aceptados. Qué es y qué no es un patrón de diseño, es una cuestión que depende del punto de vista de cada uno y del nivel de abstracción en que se trabaje.

2.6 Clasificación de los Patrones de diseño

El patrón de diseño es una solución en un alto nivel, éstos solucionan problemas que existen en muchos niveles de abstracción. Hay patrones que abarcan las distintas etapas del desarrollo, desde el análisis hasta el diseño y desde la arquitectura hasta la implementación.

Muchos diseñadores y arquitectos de software han definido el término de patrón de diseño de varias formas que corresponden al ámbito al cual se aplican los patrones, luego, se dividieron los patrones en diferentes categorías de acuerdo a su uso.

Los diseñadores de software extendieron la idea de patrones de diseño al proceso de desarrollo de software. Debido a las características que proporcionaron los lenguajes orientados a objetos (como herencia, abstracción y encapsulamiento) les permitieron relacionar entidades de los lenguajes de programación a entidades del mundo real fácilmente, los diseñadores empezaron a aplicar esas características para crear soluciones comunes y reutilizables para problemas frecuentes que exhibían patrones similares.

El grupo de GoF clasificó los patrones en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

Creacionales: Los Patrones creacionales tratan con las formas de crear instancias de objetos. El objetivo de estos patrones es abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.

Estructurales: Los patrones estructurales describen cómo las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.

Comportamiento: Caracterizan el modo en que las clases y objetos interactúan y se reparten la responsabilidad. Atañen a los algoritmos y a la asignación de responsabilidades entre objetos.

2.7 Historia del Patrón Modelo-Vista-Controlador (MVC)

El patrón de diseño MVC (pertenece a la categoría de comportamiento) fue inventado en Xerox Parc al parecer por TrygveReenskaug, en donde su primer aspecto público fue originalmente por el lenguaje SmallTalk-80, durante mucho tiempo no había información pública acerca del MVC.

El primer documento público de MVC fue "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk -80", por Glenn Krasner publicado en septiembre de 1988.

El patrón de diseño MVC cumple con uno de los enunciados más antiguos de la programación distribuida, separar la presentación del acceso a datos y la lógica de negocios. Este patrón MVC se ocupa principalmente en aplicaciones web, donde su vista son páginas HTML y un código que proveerá los datos dinámicos de la página.

2.8 División de MVC

Conforme se incrementan las necesidades de modificar una aplicación, se hace inminente modificar el código que se tiene y si no se tiene una clara visión de cómo está estructurado el código, se llega a que éste se torne indescifrable e impredecible debido a una mezcla de funcionalidades que surgen.

A través de MVC se hacen las siguientes divisiones:

- Modelo: Se centra con las funcionalidades relacionadas con el modelo de datos, que es el acceso y manipulación de los depósitos informativos que pueden ser una Base de Datos y Archivos, así como sus reglas de negocio.

- Vista: Se toca el aspecto visual / gráfico que será empleado en la aplicación en cuestión, se crea un formato para interactuar creando la interfaz de usuario.

- Controlador: Se encarga de coordinar las acciones que son llevadas entre Vista y Modelo, éste responderá a los eventos y usualmente a las acciones del usuario ocasionando cambios en el Modelo y posiblemente en la Vista.

2.9 Diferencias entre modelos Convencionales y MVC

Basándonos en un esquema básico de programa se tiene una entrada, se procesa y se obtiene una salida como se muestra en la figura 2.2.

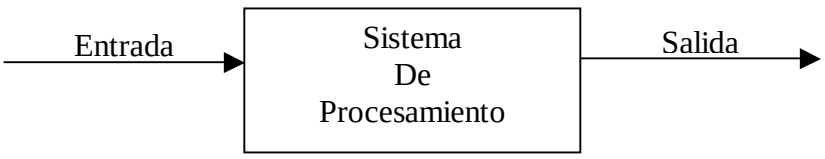


Figura 2.2: Esquema básico de programa

En el patrón MVC el proceso se lleva acabo en sus 3 componentes.

1.- El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo el usuario pulsa un botón).
2.- El Controlador recibe (por parte de los objetos de la interfaz-Vista) la notificación de la acción solicitada por el usuario. El Controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos.
3.- El Controlador accede al Modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario, los Controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifican su extensión.
4.- El Controlador delega a los objetos de Vista la tarea de desplegar la interfaz de usuario. La Vista obtiene sus datos del Modelo para generar la interfaz apropiada para donde se reflejan los cambios en el Modelo. El Modelo no debe tener conocimiento directo sobre la Vista, dejando que el Controlador envíe los datos del Modelo a la Vista.
5.- La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente, tal como se muestra en la figura 2.3.

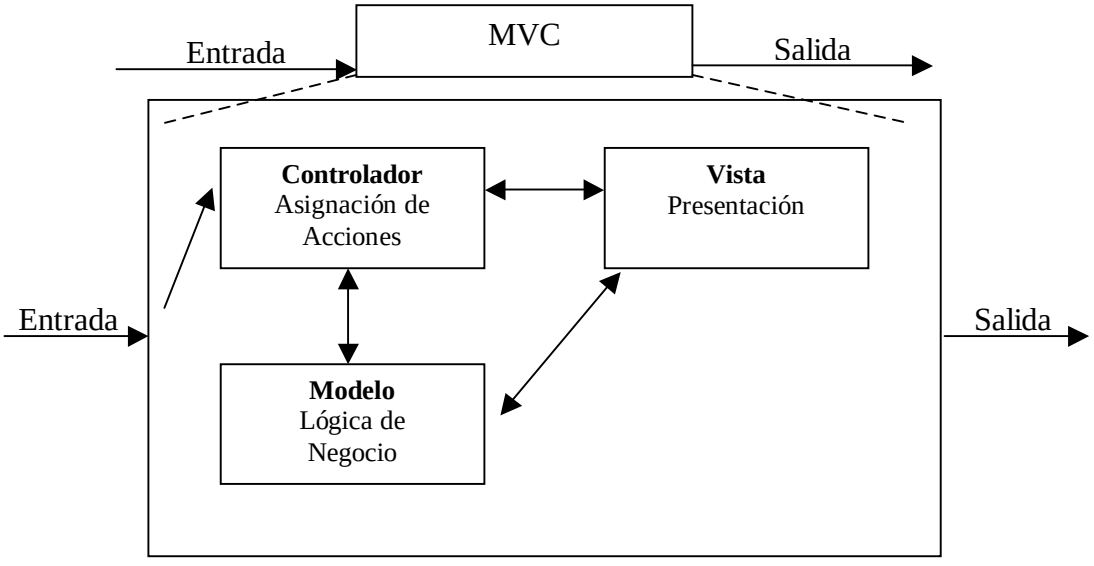


Figura 2.3: Diagrama de interacción del patrón MVC

Un ejemplo análogo que podemos tomar para saber cómo funciona el Controlador es como una operadora de teléfono que obtiene una petición y une 2 líneas. Para que el Controlador funcione adecuadamente éste debe tener un registro entre las órdenes que le llegan y la lógica de negocios que le corresponde la figura 2.4 explica este concepto.

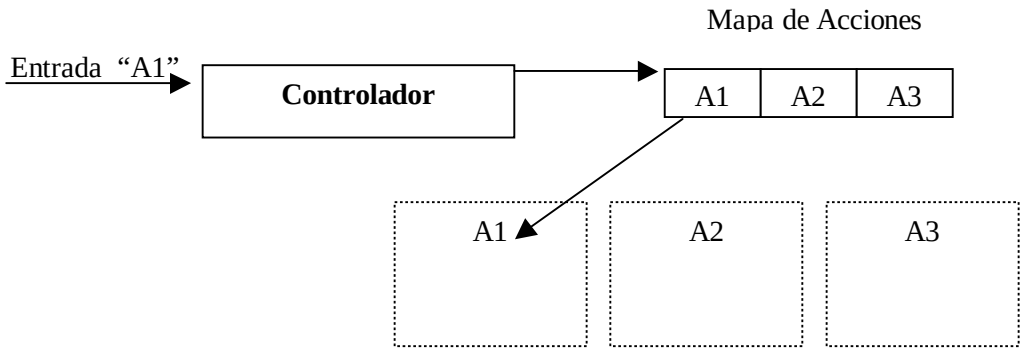


Figura 2.4: Analogía del funcionamiento del Controlador

El término "modelo" es posiblemente una expresión poco afortunada, porque es frecuente pensar que un modelo es la representación de un concepto abstracto. Los constructores de coches y de aviones crean modelos para simular coches y aviones reales. Pero esa analogía nos induce rápidamente a un error cuando se está pensando en el patrón Modelo-Vista-Controlador. En el patrón de diseño, el Modelo sirve como almacén del contenido completo, y la Vista ofrece una representación visual (completa o parcial) del contenido. Quizá se tendría una analogía mejor pensando en un modelo que posa para un artista. Es cuestión del artista examinar el modelo y crear una Vista. Dependiendo del artista, esta Vista podría ser un retrato formal, una pintura impresionista o un dibujo cubista que muestre las extremidades en contorsiones extrañas.

2.10 Relación entre los Componentes MVC

La relación Vista Controlador está herméticamente relacionada, cada caso de Vista es asociado a un único caso del Controlador, el cual es considerado como una estrategia que la Vista usa para la entrada, la Vista también es responsable de crear a las nuevas Vistas y Controladores.

Es lógico que la Vista y el Controlador estén fuertemente relacionados, porque la entrada y salida de una aplicación están muy ligados. Dentro de la GUI[1] del patrón MVC, la Vista y el Controlador simplemente están dentro de un objeto, a esto se le llama Vista del Documento. El Controlador es responsable de seleccionar vistas, y la Vista delega esta responsabilidad al Controlador. En http la respuesta es por parte del Controlador, el proceso por el Modelo y la visualización resultante por la Vista. Por su parte la Vista depende del Modelo, los cambios del Modelo están en paralelo con los cambios de la Vista, es muy difícil lograr una separación estricta entre el Modelo y la Vista. Y por último, tenemos el Controlador con el Modelo, el Controlador depende del Modelo, los cambios de la interfaz del Modelo pueden requerir los cambios en paralelo con el Controlador. La figura 2.5 muestra el diagrama se secuencias entre los objetos de Modelo, Vista y Controlador.

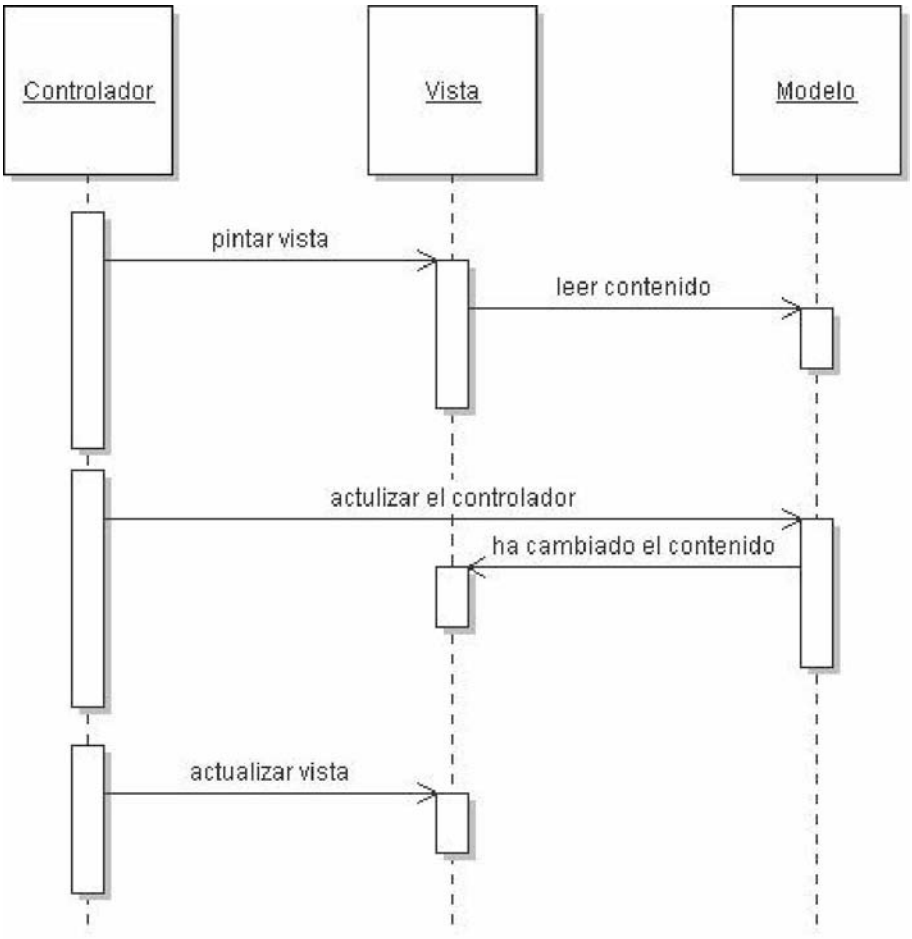


Figura 2.5: Interacciones entre los objetos de Modelo, Vista y Controlador

[1] Graphical User Interface (interfaz gráfica de usuario), es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos (iconos, ventanas, etc.) para representar la información y acciones disponibles en la interfaz.

2.11 Ejemplo típico de la analogía MVC

En el libro The Timeles Way of Building (Oxford University Press, 1979) se presenta esta analogía sobre el modelo MVC.

A todo el mundo le gustan los asientos de ventanilla, los ventanales y las ventana grandes con un alféizar bajo y sillas cómodas junto a ellas... una habitación que no tenga un lugar así rara vez nos permite sentirnos cómodos o completamente relajados.

Si la habitación no tiene una ventana que sea un lugar para sentarse, una persona que esté en la habitación sentirá dos tendencias contrapuestas:

1.- Querrá sentarse y ponerse cómoda.
2.- Se sentirá atraída hacia la luz.

Evidentemente, si los sitios cómodos, esto es, los sitios en que uno quiere sentarse, están lejos de la ventana, no habrá forma de solucionar este conflicto.

Por tanto: en toda habitación en que uno vaya a pasar un cierto tiempo durante el dia, hay que hacer que al menos una de las ventanas sea “un lugar junto a la ventana”. Todos los patrones del cátalo de este libro y los catálogos de patrones de software, respetan un formato concreto. Primero el patrón describe el contexto, que es una situación que da lugar a un problema de diseño. A continuación se explica el problema, que normalmente es un conjunto de fuerzas opuestas. Por último, la solución muestra una configuración que equilibra estas fuerzas.

En el patrón de “un lugar junto a la ventana”, el contexto es una habitación en la cual se pasa una cierta cantidad de tiempo a lo largo del día. Las fuerzas contrapuestas son que deseamos sentarnos y estar cómodos y que nos sentimos atraídos hacia la luz. La solución es crear “un lugar junto a la ventana”.

El patrón MVC, el contexto es un sistema de interfaz de usuario que presenta información y recibe entradas del usuario. Hay varias fuerzas. Puede haber múltiples representaciones visuales de los mismos datos, que será preciso actualizar a la vez. La representación visual puede cambiar, por ejemplo, para adaptarse a distintos estándares de aspecto y sensación. Los mecanismos de interacción pueden cambiar, por ejemplo, para admitir órdenes habladas, la solución es distribuir las responsabilidades en tres componentes interactivos por separado: el Modelo, la Vista y el Controlador.

El patrón MVC es más complejo que el patrón de "un lugar junto a la ventana", y es necesario mostrar con cierto detalle la forma de hacer que esta distribución de responsabilidades sea operativa.

2.12 Ventajas y Desventajas del Patrón MVC

Una de las ventajas es que un Modelo puede tener múltiples Vistas, cada una de las cuales mostrará una parte o aspecto distinto del contenido completo. Por ejemplo, un editor de HTML puede ofrecer dos Vistas simultaneas del mismo contenido, una visión WYSIWYG[2] y una visión de marcadores en bruto (tags HTML).

Cuando el modelo se actualiza a través del controlador de una de sus Vistas, el Modelo comunica a las dos Vistas asociadas la presencia de cambios, cuando las Vistas reciben la notificación, se refrescan a sí mismas automáticamente.

El Controlador maneja los eventos de entrada del usuario, como pueden ser clics y las teclas; Decide si debe traducir estos eventos a cambio en el Modelo o en la Vista. Por ejemplo, si el usuario pulsa la tecla de un carácter en un cuadro de texto, el Controlador llamará a la orden "insertar carácter" del Modelo, y entonces el Modelo indica a la Vista que se actualice a sí misma. La Vista nunca llega a saber por qué ha cambiado el texto.

[2] Es un acrónimo de What You See Is What You Get (lo que ves es lo que tienes), se aplica a los procesadores de texto con formato que permiten escribir un documento viendo directamente el resultado final, frecuentemente el resultado impreso.

Pero si el usuario pulsa una tecla de cursor, entonces el Controlador puede indicar a la Vista que se desplace. Desplazar a la Vista no afecta al texto subyacente, así que el Modelo nunca llega a saber que se ha producido este evento.

Otra ventaja es que se puede tener una interfaz de usuario sustituible, diferentes Vistas y Controladores se pueden cambiar para proporcionar una interfaz alterna al usuario, por ejemplo, los mismos datos del Modelo se pueden exhibir en una gráfica de barras, gráfica de pastel o una hoja de balance.

También porque MVC exige que el uso de una interfaz dentro de una aplicación esté estructurado en una jerarquía de objetos y define una relación estándar entre estos objetos, las versiones genéricas de estos objetos son posibles. Generalmente se llaman los componentes de la interfaz a utilizar y no hay ambiente moderno del GUI sin un complemento completo de ellas que combinan generalmente la Vista y el Controlador en un solo objeto, los componentes promueven la reutilización y reducen la necesidad de subclases especiales, éstos se conocen como visiones pluggable en la literatura de MVC.

Se asegura que el cambio se propague en las Vistas ocasionando que se puede obtener una vista del estado actual del Modelo. Y por último, con MVC se puede hacer una prueba rápida del núcleo de la aplicación, logrando así, tener una vista general de su funcionamiento.

Pero también tiene desventajas, los cambios que se hagan en el Modelo ocasiona que se requieran hacer los mismo cambios en paralelo en la Vista y posiblemente como adicional algunos en el Controlador, ocasionando que el código se haga más difícil y dificultando posibles actualizaciones.

Un aspecto importante de los patrones de diseño es que llegan a formar parte de nuestra cultura. Los programadores de todo el mundo saben cuando se habla MVC, de esta manera, los patrones se convierten en una forma eficiente de hablar sobre problemas de diseño. Por supuesto, los patrones sólo deben tomarse como una orientación y no como una religión. Ningún patrón es aplicable en todas las situaciones. Los Modelos son fáciles de separar, y cada componente de la

interfaz de usuario posee una clase de Modelo. Pero las responsabilidades de la Vista y del Controlador no siempre están separadas de una forma tan clara, y se distribuyen entre cierto número de clases diferentes.

3

Servlets y JSPs en Java

3.1 Introducción a las tendencias actuales de Internet.

En la actualidad la mayoría de las aplicaciones que se utilizan en entornos empresariales están construidas en torno a una arquitectura cliente-servidor, en el cual una o varias computadoras (de una potencia considerable) son los servidores, que proporcionan servicios a un número grande de clientes, conectados a través de la red. Los clientes suelen ser PCs de propósito general. Con el auge de la Internet, la arquitectura cliente-servidor ha adquirido mayor relevancia, ya que tiene el mismo principio básico de World Wide Web, un usuario mediante un browser [1] (cliente) solicita un servicio (páginas HTML, etc), a un computador que hace de servidor.

Los servidores HTTP[2] se limitaban a enviar una página HTML cuando el usuario lo requería directamente o daba un clic a un enlace, la inactividad de este proceso era mínima, ya que el usuario podía pedir ficheros, pero no enviar sus datos personales de modo que fueran almacenados en el servidor u obtuvieran una respuesta personalizada.

El servidor HTTP como mero servidor de fichero HTML el concepto ha ido evolucionando en dos direcciones complementarias:

1.- Añadir más dinamismo al servidor
2.- Añadir más dinamismo al cliente

[1] Es un software que permite al usuario recuperar y visualizar documentos de hipertexto, comúnmente descritos en HTML, desde servidores web, de todo el mundo a través de Internet.
[2] El protocolo de transferencia de hipertexto (*HTTP, HyperText Transfer Protocol*) es el protocolo usado en cada transacción de la Web. El hipertexto es el contenido de las páginas web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceder a una página web, y la respuesta de esa web, remitiendo la información que se verá en pantalla. También sirve el protocolo para enviar información adicional en ambos sentidos, como formularios con mensajes y otros similares.

La forma más viable para añadir inteligencia a los clientes (a las páginas HTML) han sido Javascript y los applets de Java. Javascript es un lenguaje sencillo, interpretado, cuyo código fuente se introduce en la página HTML por medio de etiquetas o tags, su nombre se deriva de cierta similitud sintáctica con Java.

Los applets de Java tienen más capacidad de añadir inteligencia a las páginas HTML que se visualizan en el browser, ya que estas son verdaderas clases de Java, que se cargan y se ejecutan en el cliente.

Por el lado de añadir inteligencia a los servidores HTTP, la primera tecnología que se utilizó ha sido los programas CGI (Common Gateway Interface), unida a los formularios HTML. Un formulario HTML permite de alguna manera invertir el sentido del flujo de la información. Complementando algunos campos con caja de texto, botones de opción y de selección, donde el usuario define sus preferencias o enviar datos al servidor.

La manera que los programas CGI reciben los datos y los procesan es porque cada formulario lleva incluido un campo llamado Action con el que se asocia el nombre del programa del servidor, en donde el servidor arranca dicho programa y le pasa los datos adjuntos que vienen en el formulario. Se tiene 2 maneras de pasar los datos de un formulario al programa CGI:

1.- Por medio de una variable de entorno del sistema operativo del servidor de tipo String (método GET).
2.- Por medio de un flujo de caracteres que llegan a través de la entrada estándar (stdin o System.in), que está asociada al teclado (método POST).

En estos 2 casos la información introducida por el usuario en el formulario llega en la forma de una única en cadena de caracteres, en donde el nombre del campo de formulario se asocia con el valor asignado por el usuario, y en donde los espacios en blanco y caracteres especiales se han sustituido por secuencia de caracteres de acuerdo con una determinada codificación.

Lo primero que tiene que hacer el programa CGI es decodificar esta información y separar los valores de los distintos campos, con esto ya puede realizar su tarea específica que sería escribir en

un fichero, en una base de datos o realizar una búsqueda de una información solicitada etc. El programa CGI termina enviando la información al cliente, en una página HTML en la que puede dar a conocer las tareas realizadas o si se tiene una dificultad para realizar dicha acción. Cuando se envía esta página HTML al cliente es a través de la salida estándar (stdout o System.out), esta página tiene que ser construida elemento por elemento, no sólo se transmite el contenido sino también los tags correspondientes de acuerdo a las reglas del lenguaje.

Estos programas CGI casualmente están escritos en lenguajes como Perl y C/C++, es importante resaltar que estos procesos tienen lugar en el servidor, esto puede ocasionar un problema, ya que teniendo varios clientes haciendo solicitudes al servidor, éste puede llegar a bajase el rendimiento o incluso a saturarse, hay que tener presente que cada vez que se recibe un requerimiento se arranca una nueva copia del programa CGI.

3.2 ¿Qué es un Servlets en Java?

Los Servlets son la respuesta de la tecnología Java a la programación CGI, son programas que se ejecutan en un servidor Web y construyen páginas Web. La construcción de páginas Web al vuelo es útil por las siguientes razones:

- Las páginas Web están basadas en datos enviados por el usuario, se tiene como ejemplos las páginas de resultado de los motores de búsqueda y las páginas que procesan pedidos desde sitio de comercio electrónico.
- Los datos cambian frecuentemente, se tienen las páginas sobre el informe del tiempo, o las páginas con cabecera de noticias, en donde éstas podían estar construidas dinámicamente, una manera seria devolviendo una página previamente construida y luego actualizarla.
- Las páginas Web que usan información desde una base de datos, tienen las páginas on-line que listan los precios actuales y el número de los artículos.

Por lo que se puede concluir que un Servlet es un programa escrito en Java que se ejecuta en un marco de un servicio de red, (un servidor HTTP, por ejemplo) que recibe y responde a las peticiones de uno o más clientes.

3.3 Ventajas de los Servlets con los CGI

Los Servlets de Java son más eficientes, fáciles de usar, poderosos, portables y baratos que el CGI tradicional y muchas otras tecnologías del tipo CGI, por lo tanto se tienen las siguientes ventajas:

- Eficiencia: Con CGI tradicional, se arranca un nuevo proceso para cada solicitud HTTP, si el programa CGI hace una operación relativamente rápida, la sobrecarga del proceso de arrancada puede dominar el tiempo de ejecución. Con los Servlet, la máquina Virtual de Java permanece arrancada, y cada operación es manejada por un thread[3] de Java de peso ligero, no un pesado proceso del sistema operativo, si en un programa CGI se tiene N peticiones simultáneas para el mismo programa CGI, el código de éste se cargará N veces en memoria y con los Servlets hay N threads pero sólo una copia de clase Servlet, estos también tienen más alternativas como optimización de los cálculos previos, mantener la conexión de la base de datos abiertas, etc.
- Conveniencia: Si se sabe Java no es necesario aprender otro lenguaje, los Servlets tiene una gran infraestructura para análisis automático y decodificación de datos de formularios HTML, leer y seleccionar cabeceras HTTP , manejar cookies[4], seguimiento de sesiones etc.
- Potencia: Los Servlets nos permiten hacer cosas que serían imposibles en los programas CGI normales, éstos pueden hablar directamente con el servidor Web, con esto se simplifica las operaciones que se necesitan para buscar imágenes y otros datos

[3] Los hilos o thread permiten dividir un programa en dos o más tareas que corren simultáneamente, por medio de la multiprogramación. En realidad, este método permite incrementar el rendimiento de un procesador de manera considerable. En todos los sistemas de hoy en día los hilos son utilizados para simplificar la estructura de un programa que lleva a cabo diferentes funciones.

[4] Cookie es un fragmento de información que se almacena en el disco duro del visitante de una página web a través de su navegador, a petición del servidor de la página. Esta información puede ser luego recuperada por el servidor en posteriores visitas. Al ser el protocolo HTTP incapaz de mantener información por sí mismo, para que se pueda conservar información entre una página vista y otra (como login de usuario, preferencias de colores, etc), ésta debe ser almacenada, ya sea en la url de la página, en el propio servidor, o en una *cookie* en el ordenador del visitante.

almacenados en situaciones comunes. Los Servlets también pueden compartir datos entre ellos, haciendo cosas útiles como almacenes de conexiones de datos fáciles de implementar, ellos logran mantener información de solicitud en solicitud, simplificando las operaciones como seguimiento de sesión y el cache de cálculos anteriores.

- § Portable: Como están escritos en Java, éste tiene un API bien estandarizado, consecuentemente si se tiene un Servlet escrito es un servidor I-Planet Enterprise, se puede ejecutar sin hacer una modificación en un servidor Apache, Microsoft IIS o WebStart, éstos son soportados directamente o mediante plug-in[5] en la mayoría de los servidores Web.
- § Barato: Hay un número de servidores Web gratuitos o muy baratos que son buenos para el uso personal o el uso de sitios Web de bajo nivel, sin embargo con la excepción de Apache que es gratuito, la mayoría de los servidores Web comerciales son relativamente caros, pero (una vez que tengamos uno) añadirle soporte para Servlets es gratuito o muy barato.

3.4 Generalidades y Arquitectura de los Servlets

Nuestra discusión sobre redes se enfoca en ambos lados de una relación cliente-servidor. Este modelo de comunicación tipo petición-respuesta es la base de las vistas de nivel superior sobre redes en Java. Un Servlet extiende la funcionalidad de un servidor, como un servidor Web. Los paquetes[6] javax.servlet y javax.servlet.http proporcionan las clases e interfaces para definir a los Servlets. Muchos desarrolladores sienten que los Servlets son la solución correcta para las aplicaciones que utilizan bases de datos en forma intensiva, las cuales se comunican con los denominados clientes delgados (aplicaciones que requieren de un soporte mínimo del lado cliente). El servidor es responsable del acceso a la base de datos. Los clientes se conectan al servidor utilizando los protocolos estándar disponibles en la mayoría de las plataformas cliente. Por lo tanto, el código de la lógica de presentación para generar contenido dinámico puede

[5] Un plugin (o plug-in) es un aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica, generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal. Los plugins típicos tienen la función de reproducir determinados formatos de gráficos, reproducir datos multimedia, codificar/decodificar emails, filtrar imágenes de programas gráficos. Se utilizan como 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.

[6] Un paquete contiene un grupo de clases e interfaces relacionadas que pueden importarse en un programa en Java.

escribirse una vez y residir en el servidor para que los clientes lo utilicen, lo cual permite a los programadores crear clientes delgados eficientes.

Sun Microsystems, a través del proceso de la comunidad de Java (Java Community Process), es responsable del desarrollo de las especificaciones para los Servlets, la implementación de estos estándares se encuentra bajo desarrollo por la fundación de software Apache, como parte del proyecto Jakarta. El objetivo del Proyecto Jakarta es proporcionar soluciones de servidores, con calidad comercial, basadas en la Plataforma Java, que se desarrolla en forma abierta y cooperativa.

Internet ofrece muchos protocolos. El protocolo HTTP (Protocolo de Transferencia de Hipertexto), que forma la base de World Wide Web, utiliza URLs (Localizadores uniformes de recursos) para localizar recursos en Internet. Los URLs comunes representan archivos o directorios, y pueden representar tareas complejas como búsquedas de bases de datos en Internet.

Un Servlet se utiliza comúnmente cuando una pequeña porción del contenido que se envía al cliente es texto estático o marcas, de hecho algunos Servlets no producen contenido, en vez de ello, realizan una tarea a beneficio del cliente y después invocan a otros Servlets.

El servidor que ejecuta un Servlet se le conoce como contendor de Servlets o motor de Servlets. Cuando un Servlet devuelve sus resultados al cliente normalmente lo hace en forma de un documento HTML, XHTML o XML para mostrarlo en un navegador, pero puede devolverse también en otros formatos, como imágenes o datos binarios.

3.5 La Interfaz Servlet y el Ciclo de Vida de un Servlet

Desde el punto de vista arquitectónico, todos los Servlets deben implementar a la interfaz Servlet, los métodos de la interfaz Servlet son invocados por el contenedor de Servlets. Esta interfaz define cinco métodos que son:

Método	Descripción
void init(ServletConfig config)	El contendor de Servlets llama a este método una vez durante el ciclo de ejecución de un Servlet, para inicializarlo. El argumento tipo ServletsConfig es proporcionado por el contenedor de Servlets que ejecuta al Servlets.
ServletConfig getServletConfig()	Este método devuelve una referencia a un objeto que implementa a la interfaz ServletConfig, este objeto proporciona el acceso a la información de configuración del Servlet, como sus parámetros de inicialización y el objeto ServletContext del Servlet, el cual proporciona el acceso a su entorno (es decir, al contenedor de Servlets en el cual se ejecuta ese Servlet).
String getServletInfo()	Este método es definido por un programador de Servlets para que devuelva una cadena que contiene información sobre un Servlet, como el autor y la versión.
void service (ServletRequest petición, ServletResponse respuesta)	El contendor de Servlets llama a este método para responder a la petición que un cliente hace al Servlet.
void destroy()	Este método de “limpieza” se llama cuando un Servlet es terminado por su contenedor de Servlets, los recursos utilizados por el Servlets, como un archivo abierto o una conexión abierta a una bases de datos, deben designarse aquí.

El ciclo de vida de un Servlet empieza cuando el contenedor de Servlets lo carga en memoria, generalmente en respuesta a la primera petición que recibe el Servlet. Antes de que éste pueda encargarse de esta petición, el contenedor de Servlets invoca al método init de este Servlet, una vez que init termina de ejecutarse, el Servlet puede responder a su primera petición. Todas las peticiones son manejadas por un método del Servlet llamado service, el cual recibe la petición, la procesa y envía una respuesta al cliente. Cada nueva petición generalmente resulta en un nuevo subproceso de ejecución (creado por el contenedor de Servlets) en el que se ejecuta el método service. Cuando el contenedor de Servlets termina el Servlet, se hace una llamada al método destroy para ese Servlet y así liberar sus recursos.

Los paquetes de Servlets definen dos clases abstract que implementan a la interfaz Servlet. La clase GenericServlet del paquete javax.servlet y la clase HttpServlet del paquete javax.servlet.http. Estas clases proporcionan implementaciones predeterminadas de todos los

métodos de Servlet. La mayoría de los Servlets extienden a GenericServlet o a HttpServlet, y redefine a algunos de sus métodos o a todos.

Se documentará el seguimiento de una aplicación utilizando la clase HttpServlet, la cual define capacidades de procesamiento mejoradas para los Servlets que extienden la funcionalidad de un servicio Web. El método clave en cada Servlet es service, el cual recibe tanto a un objeto ServletRequest como a un objeto ServletResponse. Estos objetos proporcionan acceso a los flujos de entrada y salida que permiten al Servlet leer datos del cliente y enviar datos al cliente. Estos flujos pueden estar basados en bytes o basados en caracteres. Si ocurren problemas durante la ejecución de un Servlet, se lanza una excepcion ServletException o IOExeption para indicar el problema.

Los Servlets pueden implementar la interfaz de seguimiento SingleThreadModel para indicar que sólo un subproceso de ejecución puede enviar al método service en una instancia especifica en un Servlet, en un momento dado. Cuando un Servlet implementa a SingleThreadModel , el contenedor de Servlets puede crear varias instancias del Servlet para que éste pueda manejar peticiones múltiples en paralelo. En este caso, tal vez se necesite proporcionar un acceso sincronizado a los recursos compartidos utilizados por el método service.

3.6 La Clase HttpServlet

Los Servlets basados en Web generalmente extienden a la clase HttpServlet. Esta clase redefine el método service para diferencias entre las peticiones típicas recibidas de un navegador Web cliente. Como se mencionó anteriormente los dos tipos de peticiones de HTTP más comunes, y también llamadas como métodos de petición, son Get y Post.

Recordemos que Get obtiene (o recupera) la información de un servidor, éstas se utilizan comúnmente para recuperar un documento HTML o una imagen. Una petición Post publica (o envía) datos en un servidor. Su utilización es para enviar información, como autenticación o los datos de un formulario que recopila la entrada del usuario, a un servidor.

La clase HttpServelet define los métodos doGet y doPost para responder a las peticiones Get y Post de un cliente, respectivamente. Estos métodos se llaman mediante el método service, el cual llama cuando llega una petición al servidor. El método service determina primero el tipo de petición y después llama método apropiado para manejar esa petición.

Los métodos doGet y doPost reciben como argumentos un objeto HttpSerletRequest y un objeto HttpServletResponse, los cuales permiten la interacción entre cliente y el servidor. Los métodos de HttpServletRequest facilitan el acceso a los datos que se proporcionan como parte de la petición y también los métodos de HttpServletResponse facilitan el devolver los resultados del Servlet al cliente Web.

3.7 La Interfaz HttpServletRequest

Como se mencionó, doGet o doPost de un objeto HttpServlet recibe un objeto que implementa a la interfaz HttpServletRequest. El servidor Web que ejecuta el Servlet crea un objeto HttpServletRequest y lo pasa al método service del Servlet, que a su vez, lo pasa a doGet o a doPost. Este objeto contiene la petición del cliente. Se proporciona una gran variedad de métodos para permitir al Servlet procesar la petición del cliente. Algunos de estos métodos son de la interfaz ServletRequest que es extendida por HttpServletRequest; se presentan algunos métodos clave pero si se quiere ver la lista completa de los métodos de HttpServletRequest en la siguiente dirección, se puede consultar:

java.sun.com/j2ee/j2sdkee/techdocs/api/javax/Servlet/http/HttpServletRequest.html

Método	Descripción
String getParameter(String nombre)	Obtiene el valor de un parámetro enviado al Servlet como parte de una petición get o post el argumento nombre representa el nombre del parámetro.
Enumeration getParameterNames()	Devuelve los nombre de todos los parámetros enviados al Servlet como parte de una petición post.
String[ ] getParameterValues(String nombre)	Para un parámetro con múltiples valores, este método devuelve un arreglo de cadenas que contiene los valores para parámetros de Servlet específico.

Cookie [ ] getCookies( )	Devuelve un arreglo de objetos Cookie almacenados en el cliente por el servidor. Los objetos Cookie pueden utilizarse para identificar clientes en forma única con el servidor.
HttpSession getSession (bolean crear)	Devuelve un objeto HttpSession asociado con la sesión de navegación actual del cliente. Este método puede crear un objeto HttpSession (argumento true) si no existe uno para el cliente. Los objetos HttpSession se utilizan en forma similar a los objetos Cookie para identificar a los clientes en forma única.

3.8 La interfaz HttpServletResponse

Toda llamada doGet o doPost para un objeto HttpServlet recibe un objeto que implementa a la interfaz HttpServletResponse. El servidor Web que ejecuta el Servlet crea un objeto HttpServletResponse y lo pasa al método service del Servlet, que de nuevo lo pasa a doGet o a doPost. Este objeto proporciona una variedad de métodos que permiten al Servlet formular la respuesta al cliente. Algunos de estos métodos son de la interfaz ServletResponse, la interfaz que se extiende por HttpServletResponse, se muestran algunos de estos métodos, para mayor consulta vea la siguiente dirección:

java.sun.com/j2ee/j2sdkee/techdocs/api/javax/Servlet/http/HttpServletResponse.html

Método	Descripción
Void addCookie(Cookie cookie)	Se utiliza para gregar un objeto Cookie al encabezado de la respuesta al cliente. Para determinar si se van a guardar objetos Cookie en el cliente, se compruban la edad máxima del objeto Cookie y si la opción para almacenar objetos Cookie está habilitada en el cliente.
ServletOutputStream getOutputStrem ( )	Obtiene un flujo de salida basada en bytes para enviar datos binarios al cliente
PrintWriter getWriter( )	Obtiene un flujo de salida en bytes para enviar datos binarios al cliente.
void setContenType (String tipo)	Especifica el tipo MIME de la respuesta al navegador. El tipo MIME ayuda al navegador a determinar cómo mostrar los datos (o, posiblemente que otra aplicación se ejecutará para procesar los datos). Por ejemplo, el tipo MIME “text/html” indica que la respuesta es un documento de HTML, por lo que el navegador muestra la página de HTML.

3.9 Una aplicación Web

Antes de poder desplegar un Servlet en un navegador necesitamos explicar algunos cosas, para mostrar las aplicaciones Web en el equipo local ocuparemos loopback, esto es para cuando nosotros no tengamos conexión de red y queramos aprender los conceptos de programación de redes. En los ejemplos que propongo localhost, indica que el servidor en el que está instalado el Servlet se está ejecutando en el equipo local.

El nombre de host del servidor va seguido por :8080, especificando el número de puerto TCP[7] en el que el servidor Tomcat (se especifica en la sección 3.13), espera las peticiones de los clientes. Los navegadores Web asumen el puerto TCP 80 de manera predeterminada como el puerto de servidor en el que los clientes hacen las peticiones, pero el servidor Tomcat espera las peticiones de los clientes en el puerto TCP 8080.

Esto permite a Tomcat ejecutarse en la misma computadora que una aplicación de servidor Web estándar, sin afectar la habilidad de esta aplicación de servidor Web para manejar las peticiones. Si no especificamos explícitamente el numero de puerto en el URL, el Servlet nunca recibirá nuestra petición y se mostrará un mensaje de error en el navegador.

Los puertos en este caso no son puertos físicos de hardware a los cuales se conectan cables, en vez de ello, son ubicaciones lógicas identificadas con valores enteros que permiten a los clientes pedir distintos servicios en el mismo servidor. El número de puerto especifica la ubicación lógica en donde un servidor espera y recibe conexiones de clientes, a esto se le conoce como punto de negociación (handshake point). Cuando un cliente se conecta a un servidor para pedir servicio, el cliente debe especificar el número de puerto para ese servicio, de no ser así, la petición del cliente no puede procesarse. Los números de puerto son enteros positivos con valores de 65,535, y hay conjuntos separados de estos números de puerto para los protocolos TCP y UDP[8], muchos

[7] El Protocolo de Control de Transmisión (TCP en sus siglas en inglés, Transmission Control Protocol que fue creado entre los años 1973 - 1974) es uno de los protocolos fundamentales en Internet. Muchos programas dentro de una red de datos compuesta por ordenadores pueden usar TCP para crear *conexiones* entre ellos, a través de las cuales enviarse datos. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron.

[8] User Datagram Protocol (UDP) es un protocolo del nivel de transporte basado en el intercambio detagramas. Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera. Tampoco tiene confirmación, ni control de flujo, por lo que los paquetes pueden adelantarse unos a otros; y tampoco sabemos si ha llegado correctamente, ya que no hay confirmación de entrega o de recepción.

sistemas operativos reservan los números de puerto menores a 1024 para los servicios del sistema (como los servidores de correo electrónico y World Wide Web).

En general, estos puertos no deben especificarse como puertos de conexión en sus propios programas de servidor. El termino número de puerto reconocido se utiliza a menudo cuando se describen los servicios populares en Internet, como los servidores Web y los servidores de correo electrónico. Por ejemplo, un servidor Web espera que los clientes hagan peticiones en el puerto 80 de manera predeterminada. Todos los navegadores Web conocen este número como el puerto reconocido en un servidor Web, en donde se realizan las peticiones de documentos HTML Por lo tanto, cuando se escribe un URL en un navegador Web, éste se conecta normalmente al puerto 80 en el servidor. De esta manera similar, el servidor Tomcat utiliza el puerto 8080 como su número de puerto, por lo que las peticiones que hagan a Tomcat de páginas Web o para invocar Servlet deben especificar que el servidor Tomcat espera esas peticiones en el puerto 8080, como se mencionó anteriormente.

El cliente puede acceder al Servlet sólamente si éste se encuentra instalado en un servidor que pueda responder a las peticiones del Servlet. En algunos casos, el soporte para Servlets está integrado directamente en el servidor Web, por lo que no requiere de una configuración especial para manejar las peticiones de Servlets, de lo contrario hay que integrar un contenedor de Servlets con un Servidor Web.

3.10 Instalación de Herramientas de desarrollo de Java JDK

Las versiones más completas y actualizadas de Java 2 Estándar Edition (J2SE) están disponibles en Sun Microsystems (http://java.sun.com/downloads/), para Solaris, Linux y Windows. Existen versiones en distintas fases de desarrollo para Macintosh y muchas otras plataformas, pero estas versiones son licenciadas y distribuidas por los fabricantes de estas plataformas.

La abreviatura JDK (Java Development Kit) denota a las Herramientas de desarrollo de Java, de forma ligeramente sorprendente, las versiones de 1.2 a la 1.4 recibían el nombre de Java SDK (Software Development Kit). Todavía se hallarán referencias ocasionales al término antiguo.

Se observa el término J2SE, éste denota “Java 2 Standard Edition”, a diferencia de J2EE “Java 2 Enterprise Edition” y de J2ME “Java 2 Micro Edition”.

El termino “Java 2” se acuñó en 1988 cuando el personal de mercadotecnia de Sun pensó que un incremento decimal del numero de versión no comunicaba adecuadamente los tremendos avances habidos en la JDK 1.2. Sien embargo, como inspiración les llegó después de la publicación del software, decidiendo mantener este número de versión para el juego de herramientas de desarrollo. Las versiones posteriores se denominarían 1.3, 1.4 y 5.0. Sin embargo, la plataforma cambió de nombre y pasó de ser “Java” a llamarse “Java 2”.

Por lo tanto, se tiene Java 2 Standard Edition Development Kit versión 5 o J2SE 5. Una vez descargado (en la dirección http://java.sun.com/javase/downloads/index.jsp) el JDK, siga las instrucciones de instalación dependiendo del sistema que se utiliza, en nuestro caso Windows XP.

El procedimiento de configuración ofrece un valor predeterminado para el directorio de instalación que contiene el número de versión de JDK, como jdk1.5 que vamos a ocupar.

3.11 Instalación Enterprise Edition J2EE

Sun también da la opción de bajar juntos JDK 5 con J2EE, pero que es concretamente la plataforma J2EE. El primer punto J2EE es una especificación y no un producto. Sun es el responsable de la difusión de las distintas versiones de la plataforma, la cual incluye los siguientes elementos.

- Una especificación de la plataforma y especificaciones individuales para cada API que compone J2EE. Éstas se presentan como documentos PDF, que describen las características de los servicios que proveen y la definición de las interfaces Java que integran las respectivas APIs. Cada versión de J2EE incluye versiones especificas de cada una de las API. Por ejemplo, la versión 1.4 de J2EE incluye la versión 2.4 de Java Servlets.

- Un conjunto de tests mediante los cuales los proveedores que implementan los Servicios de Aplicaciones J2EE se pueden desplegar sobre distintos Servicios de Aplicaciones.

Según la definición de Sun, "La Plataforma J2EE simplifica el desarrollo de aplicaciones Enterprise", basándolas en componentes estándar y moduladores, proveyendo un conjunto completo de servicios a esos componentes y manejando muchos de los aspectos propios del comportamiento de las aplicaciones de forma automática, sin necesidad de programación explícita.

Para comprender claramente esta definición dada, es necesario aclarar algunos términos usados habitualmente en el ámbito de J2EE y que participan en ella:

- Enterprise Aplication: Hace referencia a aquellas aplicaciones que tienen fuertes requerimientos no funcionales, como ser multitier[9], que ya se esta dejando el modelo cliente-servidor y tiende a un modelo de n capas más avanzado. Por ejemplo un modelo de tres capas, el cliente no hace llamadas a una base de datos. En lugar de hacer esto, se llama a una capa de software intermedia (middleware) que a su vez efectúa las consultas en la base de datos. Este modelo separa la presentación visual (en el cliente) de la lógica de negocio (en la capa intermedia) y de los datos puros (en la base de datos). Por tanto, se hace posible acceder a los mismos datos y a las mismas reglas de negocio desde múltiples clientes, soportando un alto grado de concurrencia y acceso multiusuario, proveer acceso seguro y transaccional a la funcionalidad brindada, y otorgar características de alta disponibilidad. En general, estas aplicaciones deben ser resistentes tanto a factores internos como poder agregar nuevas funcionalidades fácilmente como externos por ejemplo soportar varias plataformas.
- Componente: Es un elemento de software que implementa un conjunto de interfaces definidas.
- Servicios: Los componentes que conforman una aplicación se ejecutan dentro de un recipiente que les brinda un conjunto de servicios, como control de acceso, seguridad y

[9] Consiste en dividir los componentes primarios de la aplicación, progrmarlos por separado y luego unirlos sea en tiempo de ejecución o en el mismo código.

soporte, para que participen en transacciones. Los recipientes son parte del Application Server.

- En forma automática: Muchos de los servicios que brindan los recipientes no requieren que los componentes hagan uso explícito de ellos, ni que sea necesario escribir código para usarlos. Por el contrario, muchos de estos servicios se usan mediante especificaciones declarativas y el recipiente se encarga de manejarlos de acuerdo con esta especificación.

Una vez explicado a grandes rasgos J2EE, podemos bajar en el siguiente enlace a JDK y a JEE juntos e instalarlos en nuestro equipo.

http://java.sun.com/javase/downloads/index.jsp

3.12 Instalar NetBeans

En ciertas ocasiones Sun pone a disposición del público paquetes que contienen tanto JDK, JEE, como un entorno integrado de desarrollo (IDE). El IDE en distintos momentos de su existencia, ha recibido los nombres de Forte, Sun ONE Studio, Sun Java Studio y NetBeans todos ellos han evolucionado por separado.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio 2000 y continúa siendo el patrocinador principal de los proyectos.

El NetBeans es un entorno de desarrollo, una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe, además, un número importante de módulos para extender el IDE NetBeans. El IDE NetBeans es un producto libre y gratuito sin restricciones de uso. Se puede descargar en el siguiente enlace, se ocupará la versión 5.5 para Windows XP.

http://www.netbeans.org/

Se puede ver que también podemos bajar Netbeans 5.5 con Java Enterprise Edition Application Server 9.0, Proporciona una serie integrada y flexible de herramientas y servicios, necesaria para diseñar, desarrollar, desplegar y gestionar el suministro de arquitecturas orientadas a servicios (SOA) de nueva generación. El paquete incluye la versión más reciente de Sun Java System Application Server Enterprise Edition, una importante actualización del servidor de aplicaciones de Sun, que incorpora el rendimiento y las funcionalidades de la más reciente plataforma Java empresarial conocida como Java 2 Platform Edition (J2EE) 1.5, además de alta disponibilidad y soporte. El paquete también incluye Sun Java System Web Server. Se puede aprovechar y así sólo bajar JDK y después Netbeans con Java EE Application Server, ya que como se puede ver ya trae a J2EE integrado. Si sólo se quiere el IDE no hay problema, porque en su instalación se indica dónde está el JDK y automáticamente reconoce la ubicación de J2EE para poderlo usar.

3.13 Instalar un Servidor Apache Tomcat

Tomcat es una implementación completamente funcional de los estándares de Servlets 2.4, incluye un servidor Web, por lo que puede usarse como un contenedor de pruebas independiente para Servlets, tiene la habilidad de manejar las peticiones de Servlets recibidas por servidores Web populares como el servidor Apache HTTP de la fundación de software Apache o el servidor Microsoft Internet Information Services (IIS). Tomcat está integrado en la implementación de referencia Java 2 Enterprise Edition 5 de Sun Microsystems.

Para descargar una versión de Tomcat dirigirse a la siguiente dirección:

http://tomcat.apache.org

en donde hay una variedad de archivos comprimidos. La implementación completa de Tomcat está contenida en los archivos que empiezan con el nombre apache-tomcat-5.5.20, en nuestro caso, bajaremos esta versión que se encuentra en un archivo exe, para Windows. Pero antes debemos tener instalado JDK y el JEE. Una vez instalado Tomcat, y que esté corriendo, se abre una ventana del navegador y se introduce el siguiente URL.

http://127.0.0.1:8080

Con el cual nos tiene que presentar la siguiente pantalla:

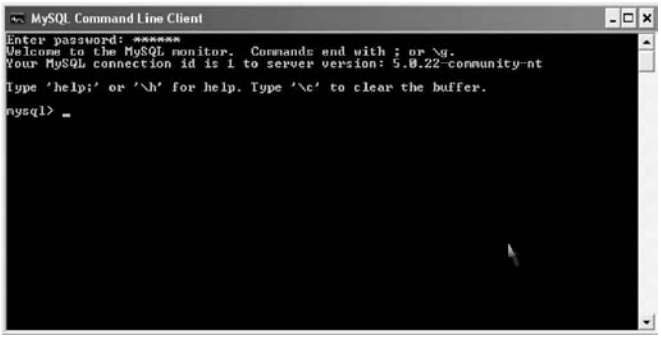
3.14 Instalación de MySQL

Como se mencionó, muchos desarrolladores sienten que los Servlets son la solución correcta para las aplicaciones que utilizan bases de datos en forma intensiva, para los ejemplos que se ocuparán, se necesitará un sistema gestor de base de dato relacional, seguiremos la pauta de seguir ocupando software libre.

MySQL es un sistema cliente-servidor de administración de base de datos relacionales diseñado para el trabajo, tanto en los sistemas operativos Windows como en los sistemas UNIX / LINUX . Además, determinadas sentencias de MySQL pueden ser embebidas en código Java y HTML para diseñar aplicaciones Web dinámicas que incorporan la información de las tablas de MySQL a páginas Web.

Primero comenzamos descargando de la página http://www.mysql.com la versión más actual de MySQL, que en este caso es la 5 y para el sistema operativo de Windows XP, se descomprime el archivo y se ejecuta su instalador, seguimos los pasos del hechicero del instalador. Recuerden que para tener comunicación es importante recordar que lo haremos en el puerto 3306. Para comprobar que esta instalado, ejecutamos el programa “MySQL Command Line Client” que se encuentra Inicio à Todos los programas à MySQL à MySQL Server 5 à MySQL Command Line

Cliente. Introducimos el password que proporcionamos en la instalación como root y se tiene que ver la siguiente pantalla:



3.15 Despliegue de una aplicación Web

Los Servlets y sus archivos de soporte se despliegan como parte de aplicaciones Web, las cuales generalmente se despliegan en el subdirectorio webapps de Tomcat. Una aplicación Web tiene una estructura de dirección conocida, en la que residen todos los archivos que forman parte de la aplicación. Esta estructura de directorios puede ser creada por el administrador del servidor en el directorio webapps, o toda la estructura de directorios completa puede archivarse en un archivo de fichero de aplicación WAR (Web Archive), que se coloca en el directorio webapps, entonces cuando el servidor Tomcat empiece a ejecutarse, extraerá el contenido del archivo WAR en la estructura de subdirectorio apropiada de webapps.

La estructura de directorios de aplicaciones Web contiene un directorio raíz de contexto(el directorio de nivel superior para toda una aplicación Web), y varios subdirectorios que se describen a continuación.

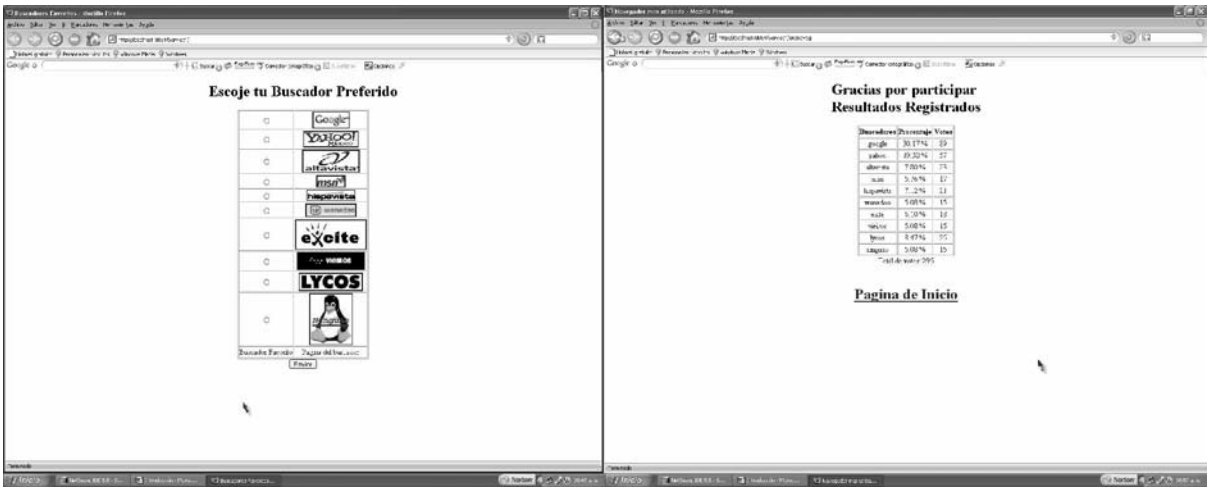
Directorio	Descripción
Raíz de contexto	Este es el directorio raíza para la aplicación Web, todos los documentos HTML, JSPS (que se explican mas adelante), Servlets y archivos de soporte como imágenes y archivos de clases residen en este directorio o sus subdirectorios. El nombre de este directorio es especifico por el creador de la aplicación Web. Para proporcionar la estructura en una aplicación

	Web, los subdirectorios pueden colocarse en el directorio raíz de contexto. Por ejemplo, si una aplicación utiliza muchas imágenes, podría colocar un subdirectorio llamado imágenes en este directorio.
WEB-INF	Este directorio contiene el descriptor de despliegue de la aplicación Web (web.xml)
WEB-INF/classes	Contiene los archivos de clases del Servlet y demás archivos de clases de soporte que utilizan en una aplicación Web. Si las clases son parte de un paquete, la estructura de directorio completa del paquete empezaría aquí.
WEB-INF/lib	Contiene archivos de fichero de Java (JAR). Los archivos JAR pueden contener archivos de clases de Servlets y demás archivos de clases de soporte que utilizan en una aplicación Web.

Después de configurar, el directorio raíz de contexto, debemos configurar nuestra aplicación Web para manejar las peticiones. Esta configuración se lleva a cabo en un descriptor de despliegue, el cual se almacena en un archivo llamado web.xml. El descriptor de despliegue especifica varios parámetros de configuración como el nombre usado para invocar el Servlet (es decir, el alias), una descripción del Servlet, el nombre de clase completamente calificado del Servlet, y una asociación de Servlet (es decir, la ruta o rutas que hacen que el contenedor de Servlets invoque al Servlet). Pero todo este trabajo de crear la estructura de directorios de una aplicación Web ya lo hacen automáticamente los IDE, que en este caso ocuparemos a NetBeans para crear nuestras aplicaciones Web.

3.16 Explicación del Servlet ServConexion para peticiones Get,Post

El código fuente de la aplicación Web se encuentran en la página 45. El programa que ponemos de ejemplo de un Servlet, utiliza un formulario sencillo donde el usuario escoge cuál es el navegador favorito que ocupa, se despliega una página html (Inicio2.html), con cajas de diálogo para votar por el navegador o buscador, una vez señalado el navegador se cuenta con un botón para poder hacer el envió de datos y que los pueda procesar el Servlet (ServConexion.java).



Archivo Inicio2.html que muestra cómo puede votar el usuario, y página html generada cuando se hace la consulta a la base de datos

También se tiene imágenes que nos llevarán a la página oficial del buscador. El envío de los datos del formulario (peticiones Post), los procesará el Servlet mediante el método doPost (líneas 84 a 169), las peticiones Get para el enlace de las páginas oficiales, se procesan en el método doGet (líneas 39 a 80).

Se irán registrando cada voto que se hace en una base de datos (navegadores), la cual cuando se despliegue el resultado de cuáles navegadores se utilizan más, será gracias a que se hará una consulta a dicha base de datos, y se construirá una página html con una tabla para poder desplegar dichos datos. También la base de datos guardará el registro de cada voto dado por el usuario.

3.17 Explicación del Descriptor de Despliegue (web.xml)

En este descriptor de despliegue ocupamos las siguientes etiquetas, de la línea 1 a 2 se especifica el tipo de documento para el despliegue de la aplicación Web, la etiqueta web-app (líneas 2 a 43) define la configuración de los Servlets. El elemento servlet-name (líneas 3 a 26) describe el Servlet, servlet-name (línea 4) es el nombre que elegimos para el Servlet (ServConexion), servlet-class (línea 5) especifica el nombre de la clase completamente calificado del Servlet compilado (ServConexion).

Dado que los Servlets inicializan mediante el método init, que sobrescribimos en ServConexion (líneas 20 a 37), como se mencionó, el método init es llamado sólamente una vez durante la vida de un Servlet, antes de aceptar cualquier petición de los clientes.

El método init toma un argumento ServletConfig y lanza una excepción ServletException. El argumento proporciona información al Servlet acerca de sus parámetros de inicialización, es decir, los parámetros que no están asociados con una petición, sino que se pasan al Servlet para inicializar su estado. Estos parámetros se especifican en este descriptor de despliegue, cada parámetro aparece en un elemento init-param, el elemento description nos proporciona una descripción del argumento que queremos pasar, este fragmento de código del descriptor de despliegue muestra lo que queremos decir.

```
<init-param>
        <description>Describe el nombre de la base de datos</description>
        <param-name>nombreBD</param-name>
        <param-value>jdbc:mysql://192.168.1.65:3306/navegadores</param-value>
</init-param>
```

Se tiene a param-name que es el nombre del parámetro que ocuparemos para hacer la referencia en el Servlet, y param-value que es el contenido que tiene este parámetro. En este caso el parámetro se llama nombreBD, y su valor indica que se utilizará una conexión de base de datos con Java (jdbc) y se ocupará MySQL (mysql) que se encuentra en el la dirección 192.168.1.65, el puerto es 3306 y el nombre de la bases de datos es navegadores, se da parámetros iniciales para el controlador, el nombre de la base de datos, nombre de usuario y su clave (líneas 6 a 25).

El elemento servlet-mapping (líneas 27 a 35), especifica los elementos servlet-name y url-pattern. La URL ayuda al servidor a determinar cuáles peticiones se envían al Servlet (en navegación y redirección). Con el elemento session-config (líneas 35 a 39), hacemos las configuraciones de cada sesión que se empiece, en este caso sólo ocupamos para indicar tiempo fuera de la sesión con session-timeout (líneas 36 a 38). Por ultimo, señalamos los archivos de inicio con welcome-file-list (líneas 40 a 43), y con el elemento welcome-file (línea 41) el archivo de inicio que es Inicio2.html.

3.18 Explicación del Código Fuente ServConexion.java

En las líneas 2,3 se importan los paquetes *javax.servlet* y *javax.servlet.http*, éste último proporciona la superclase *HttpServlet* para los Servlets que manejan get y post de HTTP. Esta clase implementa a la interfaz *javax.servlet.Servlet* y agrega métodos que soportan peticiones del protocolo HTTP. Las clase *ServConexion* extiende a *HttpServlet* (línea 13).

La superclase HttpServlet proporciona los métodos doGet y doPost para responder a las peticiones get, post, pero su funcionalidad predeterminada es indicar un error "Método no permitido". Se declara una referencia Connection (línea 15) para manejar la conexión a la base de datos y una referencia Statement para actualizar la base de datos en cuanto a los votos y obtener los resultados completos de la encuesta.

El Servlet puede obtener los valores de los parámetros de inicialización mediante el método init (líneas 20 a 37), como se desea registrar el controlador que se ocupará y su parámetro está indicado en el descriptor de despliegue (web.xml), se invoca al método getInitParameter de ServletConfig (línea 24), el cual recibe una cadena que representa el nombre del parámetro, lo mismo se hace para obtener el nombre de la base de datos, usuario y password (líneas 26 a 28). Para cualquier excepción, se lanza una variable UnavailableException para indicar que el Servlet no se encuentra disponible en este momento (líneas 33 a 36).

El método doGet (líneas 39 a 82) recibe dos argumentos, un objeto que implementa a la interfaz HttpServletRequest (request) para representar la petición del cliente y un objeto que implementa a la interfaz HttpServletResponse (response) para representar la respuesta del Servlet.
Si no se puede manejar la petición del cliente, se lanza una excepción ServletException y si ocurre un problema durante el procesamiento de los flujos de datos manda una excepción IOException (líneas 39 a 40).

La línea 43 utiliza el método getWriter del objeto response para obtener una referencia al objeto PrintWriter, que permite al servidor enviar contenido al cliente. Los parámetros se pasan como pares nombre/valor. En la línea 44 obtenemos información que se pasa al Servlet como parte de la petición del cliente.

El método getParameter del objeto response recibe el nombre del parámetro como un argumento y devuelve el valor String correspondiente, o null si el parámetro no forma parte de la petición.

El documento Inicio2.html línea 24 tenemos <a href="redireccion?pagina=yahoo" target="_blank"> que nos servirá para enlazar con el Servlet, en el descriptor de despliegue se hace la referencia en el elemento servlet-mapping a través de:

```
<servlet-mapping>
        <servlet-name>ServConexion</servlet-name>
        <url-pattern>/redireccion</url-pattern>
</servlet-mapping>
```

Cuando el usuario selecciona una imagen el navegador anexa lo siguiente

redirección ? pagina=yahoo

El separador ? separa la cadena, los datos que se pasan como parte de la petición get, del resto del URL, los pares nombre/valor se pasan con el nombre y el valor separador por un =. Si hay más de un par nombre/valor, cada par se separa por un &.

Con el método getParameter, le decimos que obtenga el valor del parámetro página, y lo guardamos en la variable de referencia ubicación (línea 44). Con esto sólo comparamos las cadenas correspondientes para hacer la redirección de URLs, del método sendRedirect de la clase HttpServletResponse (líneas 45 a 63).

Si el usuario escoge que ninguno, entonces se genera una página html indicando que se sugiere que pruebe alguno y se hace un enlace de nuevo a la página principal (líneas 64 a 78). Esta página se crea escribiendo cadenas mediante el método println del objeto out, este método imprime un carácter de nueva línea después de su argumento String. Al desplegar la página Web, el navegador no utiliza el carácter de nueva línea, en vez de ello, el carácter de nueva línea aparece en el código fuente de HTML. Por último, en la línea 80 se cierra el flujo para completar la página.

El método doPost (líneas 84 a 155), procesa el voto del usuario al igual que doGet recibe dos argumentos, un objeto que implementa a la interfaz HttpServletRequest (request) para representar la petición del cliente y un objeto que implementa a la interfaz HttpServletResponse (response) para representar la respuesta del Servlet.

Si no se puede manejar la petición del cliente se lanza una excepción ServletException y si ocurre un problema durante el procesamiento de los flujos de datos manda una excepción IOException (líneas 84 a 85). En la línea 87 se utiliza el método setContentType del objeto response para especificar el tipo de contenido de los datos que se van a enviar como respuesta al cliente. Esto permite al navegador cliente comprender y manejar el contenido. El tipo del contenido se conoce como el tipo MIME (Extensión multipropósito de correo Internet) de los datos, el tipo de contenido es text/html para indicar que la respuesta es un documento html.

También obtenemos una referencia al objeto PrintWriter para mandar el contenido al cliente, en el descriptor de despliegue se hace la referencia a través de lo siguiente:

```
<servlet-mapping>
        <servlet-name>ServConexion</servlet-name>
        <url-pattern>/encnaveg</url-pattern>
 </servlet-mapping>
```

Y en el archivo Inicio2.html la línea <form action="encnaveg" method="POST"> ayuda para que se cumpla dicha referencia, y esto se logra también gracias a la etiqueta
<input type="radio" name="voto" value="1"/>, el valor obtenido se transforma a un entero (línea 94), que se ocupará para ir registrando cuántos usuarios han estado votando. De las líneas 97 a 140 se procesan los datos enviados, primero incrementando una unidad cada vez que se hace un voto al navegador preferido en la base de datos (línea 98 a 99), mediante lenguaje sql, segundo se suma el total de votos mediante otro query, y se pide ese resultado y se convierte a entero (líneas 102 a 105), como tercer paso se pide el contenido de las columnas Nombre,Votos, e Id para poderlas desplegar en una tabla (líneas 107 a 155), si se produce un error en la base de datos, esto se cacha desde las líneas 156 a 169 gracias a SQLException, y se produce una página html diciendo que ocurrió un error en la base de datos que intenten más tarde.

Por último, se tienen 2 métodos getServletInfo para dar una descripción del Servlet, y destroy que se ocupa para destruir al Servlet para liberar recursos, que en este caso se utiliza para cerrar la conexión a la base de datos. A continuación se muestran las imágenes generadas por la aplicación Web, así como su código de ServConexion, y de la página Inicio2.html junto con la configuración del descriptor.

Código de ServConexion.java

```
 1 import java.io.*;
 2 import javax.servlet.*;
 3 import javax.servlet.http.*;
 4 import java.text.*;
 5 import java.sql.*;
 6
 7
 8 /**
 9  *
10  * @author Sanchez Hernandez Miguel Angel
11  * @version 1.0
12  */
13 public class ServConexion extends HttpServlet {
14     //campos para estableces la conexion
15     private Connection conexion;
16     private Statement instruccion;
17
18     /*Registrando el controlador,nombre de la B.D
19      *usuario y clave*/
20     public void init(ServletConfig config)
21     throws ServletException{
22         //Abriendo la conexion a la base de Datos
23         try{
24             Class.forName(config.getInitParameter("controlador"));
25             conexion=DriverManager.getConnection(
26                 config.getInitParameter("nombreBD"),
27                     config.getInitParameter("usuario"),
28                     config.getInitParameter("clave"));
29
30             //Crear objeto Statement para consultar la BD
31             instruccion=conexion.createStatement();
32         }
33         catch(Exception exp){
34             exp.printStackTrace();
35             throw new UnavailableException(exp.getMessage());
36         }
37     }
38     //procesando las peticiones Get
39     protected void doGet(HttpServletRequest request,
40             HttpServletResponse response)
41     throws ServletException, IOException {
42         //processRequest(request, response);
43          PrintWriter out = response.getWriter();
44         String ubicacion=request.getParameter("pagina");
```

```
45      if (ubicacion!=null){
46          if(ubicacion.equals("google"))
47              response.sendRedirect("http://google.com.mx");
48          else if(ubicacion.equals("yahoo"))
49              response.sendRedirect("http://mx.search.yahoo.com");
50          else if(ubicacion.equals("altavista"))
51              response.sendRedirect("http://es.altavista.com");
52          else if(ubicacion.equals("msn"))
53              response.sendRedirect("http://www.msn.com.mx");
54          else if(ubicacion.equals("hispavista"))
55              response.sendRedirect("http://www.hispavista.com.mx");
56          else if(ubicacion.equals("wanadoo"))
57              response.sendRedirect("http://busca.wanadoo.es");
58          else if(ubicacion.equals("exite"))
59              response.sendRedirect("http://www.excite.es");
60          else if(ubicacion.equals("vieiros"))
61              response.sendRedirect("http://buscador.vieiros.com");
62          else if(ubicacion.equals("lycos"))
63              response.sendRedirect("http://www-es.lycos.com");
64          else{
65              out.println("<html>");
66              out.println("<head>");
67              out.println("<title>Pagina Invalida</title>");
68              out.println("</head>");
69              out.println("<body>");
70              out.println("<h1> Le sugerimos que pruebe algunos");
71              out.println("de los navegadores que le presentamos</h1>");
72              out.println("<br/><h1>Gracias por participar</h1>");
73              out.println("<a href="+
74                      "\"/Servlet4/Inicio2.html\">");
75              out.println("<br/><h1>Pagina de Inicio</h1></a>");
76              out.println("</body>");
77              out.println("</html>");
78          }
79      }
80      out.close();
81
82  }
83  //Procesando las peticiones Post
84  protected void doPost(HttpServletRequest request, HttpServletResponse response)
85  throws ServletException, IOException {
86      //processRequest(request, response);
87      response.setContentType("text/html;charset=UTF-8");
88      PrintWriter out = response.getWriter();
89      DecimalFormat dosDigitos=new DecimalFormat("0.00");
90      //obteniendo la respuesta del cliente
91      out.println("<html>");
92      out.println("<head>");
93      //leer la respuesta actual del cliente
94      int valor=Integer.parseInt(request.getParameter("voto"));
95      String consulta;
96      //procesando los votos y mostrar los resultados
97      try{
98          consulta="Update navegador Set Votos=Votos+1 "+
99                  "Where id="+valor;
100         instruccion.executeUpdate(consulta);
101         //obteniendo el totol de las respuestas
102         consulta="Select sum(Votos) from navegador";
103         ResultSet TotalRS=instruccion.executeQuery(consulta);
```

```
104         TotalRS.next();
105         int total=TotalRS.getInt(1);
106         //obteniendo todos los resultados
107         consulta="Select Nombre,Votos,Id from navegador "+
108               "Order by Id";
109         ResultSet resultadosRS=instruccion.executeQuery(consulta);
110         out.println("<title>Navegador mas utlizado</title>");
111         out.println("</head>");
112
113         out.println("<body><center>");
114         out.println("<p><h1>Gracias por participar");
115         out.println("<Br/>Resultados Registrados</h1></p>");
116
117         //procesar los resultados
118         int votos;
119         out.println("<Table id=\"Resultado\" border=\"1\" cellspacing=\"1\" cellpadding=\"1\">");
120         out.println("<Thread>");
121         out.println("<Tr><Th>Buscadores</Th>"+
122                     "<Th>Procentaje</Th>"+
123                     "<Th>Votos</Th></Tr>");
124         out.println("</Thread>");
125         out.println("<TBody>");
126
127         while (resultadosRS.next()){
128            out.println("<tr>");
129            out.println("<td><center>");
130            out.print(resultadosRS.getString(1));
131            out.println("</center></td>");
132            votos=resultadosRS.getInt(2);
133            out.println("<td><center>");
134            out.print(dosDigitos.format((double)votos/total*100));
135            out.print(" %");
136            out.println("</center></td>");
137            out.println("<td><center>");
138            out.println(votos);
139            out.println("</center></td></tr>");
140         }
141         resultadosRS.close();
142         out.println("</TBody>");
143         out.println("</Table>");
144         out.print("Total de votos: ");
145         out.print(total);
146         out.println("<br />");
147         out.println("<a href="+
148                  "\"/Servlet4/Inicio2.html\">");
149         out.println("<br /><h1>Pagina de Inicio</h1></a>");
150
151         out.println("</center>");
152         out.println("</body>");
153         out.println("</html>");
154         out.close();
155      }
156      catch(SQLException exp){
157         exp.printStackTrace();
158         out.println("<html>");
159         out.println("<head>");
160         out.println("<title>Error</title>");
161         out.println("</head>");
162         out.println("<body>");
```

```
163             out.println("<h1>Ocurrio un error en la B.D</h1>");
164             out.println("<br/>Intente más tarde");
165             out.println("</body>");
166             out.println("</html>");
167             out.close();
168         }
169     }
170 
171     /** Returns a short description of the servlet.
172      */
173     public String getServletInfo() {
174         return "Short description";
175     }
176     // cerrar instrucciones de B.D cuando termine el servlet
177     public void destroy(){
178         //tratar de cerrar instrucciones y conexiones B.D
179         try{
180             instruccion.close();
181             conexion.close();
182         }
183         catch(SQLException exp){
184             exp.printStackTrace();
185         }
186     }
187 }
```

Tags de la página Inicio2.html

```
 1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
 2 <html>
 3   <head>
 4     <title>Buscadores Favoritos</title>
 5   </head>
 6   <body>
 7       <center><h1>Escoje tu Buscador Preferido</h1>
 8           <form action="encnaveg" method="POST">
 9               <table align="Center" border="1" cellspacing="1" cellpadding="1">
10                   <thead>
11                       <tr>
12                           <th><center><input type="radio" name="voto" value="1"/></center></th>
13                           <th><a href="redireccion?pagina=google" target="_blank"><center>
14                                   <img src="imagenes/google.gif" width="75" height="32" alt="google"/>
15                               </center>
16                           </a>
17                           </th>
18                       </tr>
19                   </thead>
20                   <tbody>
21                       <tr>
22                           <td><center><input type="radio" name="voto" value="2"/></center></td>
23                           <td><a href="redireccion?pagina=yahoo" target="_blank"><center>
24                                   <img src="imagenes/yahoo2.gif" width="114" height="31" alt="yahoo2"/>
25                               </center>
26                           </a>
27                           </td>
28                       </tr>
29                       <tr>
30                           <td><center><input type="radio" name="voto" value="3"/></center></td>
31                           <td><a href="redireccion?pagina=altavista" target="_blank"><center>
```

```
32                      <img src="imagenes/altavista.gif" width="118" height="45" alt="altavista"/>
33                    </center>
34                  </a>
35                </td>
36              </tr>
37              <tr>
38                <td><center><input type="radio" name="voto" value="4" /></center></td>
39                <td><a href="redireccion?pagina=msn" target="_blank"><center>
40                      <img src="imagenes/msn.gif" width="58" height="22" alt="msn"/>
41                    </center>
42                  </a>
43                </td>
44              </tr>
45              <tr>
46                <td><center><input type="radio" name="voto" value="5" /></center></td>
47                <td><a href="redireccion?pagina=hispavista" target="_blank"><center>
48                      <img src="imagenes/hispavista.gif" width="100" height="20" alt="hispavista"/>
49                    </center>
50                  </a>
51                </td>
52              </tr>
53              <tr>
54                <td><center><input type="radio" name="voto" value="6" /></center></td>
55                <td><a href="redireccion?pagina=wanadoo" target="_blank"><center>
56                      <img src="imagenes/wanadoo.gif" width="104" height="25" alt="wanadoo"/>
57                    </center>
58                  </a>
59                </td>
60              </tr>
61              <tr>
62                <td><center><input type="radio" name="voto" value="7" /></center></td>
63                <td><a href="redireccion?pagina=exite" target="_blank"><center>
64                      <img src="imagenes/excite.gif" width="148" height="60" alt="excite"/>
65                    </center>
66                  </a>
67                </td>
68              </tr>
69              <tr>
70                <td><center><input type="radio" name="voto" value="8" /></center></td>
71                <td><a href="redireccion?pagina=vieiros" target="_blank"><center>
72                      <img src="imagenes/vieiros.gif" width="141" height="35" alt="vieiros"/>
73                    </center>
74                  </a>
75
76                </td>
77              </tr>
78              <tr>
79                <td><center><input type="radio" name="voto" value="9" /></center></td>
80                <td><a href="redireccion?pagina=lycos" target="_blank"><center>
81                      <img src="imagenes/lycos.jpg" width="132" height="35" alt="lycos"/>
82                    </center>
83                  </a>
84                </td>
85              </tr>
86              <tr>
87                <td><center><input type="radio" name="voto" value="10" /></center></td>
88                <td><a href="redireccion?pagina=ninguno" target="_blank"><center>
89                      <img src="imagenes/ninguno.jpg" width="89" height="106" alt="ninguno"/>
90                    </center>
```

```
                    </a>
                  </td>
                </tr>
                <tr>
                  <td><center>Buscador Favorito</center></td>
                  <td><center>Pagina del buscador</center></td>
                </tr>
              </tbody>
            </table>
             <input type="submit" value="Enviar"/>
          </form>
      </center>
   </body>
</html>
```

Configuración de web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <servlet>
      <servlet-name>ServConexion</servlet-name>
      <servlet-class>ServConexion</servlet-class>
      <init-param>
        <description>Describe el nombre de la base de datos</description>
        <param-name>nombreBD</param-name>
        <param-value>jdbc:mysql://192.168.1.65:3306/navegadores</param-value>
      </init-param>
      <init-param>
        <description>Controlador de mysql</description>
        <param-name>controlador</param-name>
        <param-value>com.mysql.jdbc.Driver</param-value>
      </init-param>
      <init-param>
        <description>Nombbre del usuario</description>
        <param-name>usuario</param-name>
        <param-value>miguel</param-value>
      </init-param>
      <init-param>
        <description>password del usuario</description>
        <param-name>clave</param-name>
        <param-value>1234</param-value>
      </init-param>
      </servlet>
   <servlet-mapping>
      <servlet-name>ServConexion</servlet-name>
      <url-pattern>/encnaveg</url-pattern>
   </servlet-mapping>
   <servlet-mapping>
      <servlet-name>ServConexion</servlet-name>
      <url-pattern>/redireccion</url-pattern>
   </servlet-mapping>
   <session-config>
      <session-timeout>
        30
      </session-timeout>
   </session-config>
```

```
40    <welcome-file-list>
41        <welcome-file>Inicio2.html</welcome-file>
42        </welcome-file-list>
43 </web-app>
```

3.19 Introducción a JSP (JavaServer Page)

JavaServer Page es una extensión de la tecnología de los Servlets. Los JSP simplifican la producción de contenido Web dinámico. Permiten a los programadores de aplicaciones Web crear contenido dinámico mediante la reutilización de componentes predefinidos y mediante la interacción con los componentes, utilizando secuencias de comandos del lado del servidor. Los programadores de un JSP pueden reutilizar JavaBeans y crear bibliotecas de marcas personalizadas que encapsulen una funcionalidad compleja y dinámica.

Las bibliotecas de marcas personalizadas permiten incluso a los diseñadores Web que no están familiarizados con Java mejorar las páginas Web con poderosas herramientas de procesamiento y contenido dinámico.

3.20 Ventajas de JSP

- Contra Active Server Page (ASP), es una tecnología similar a la de Microsoft, primero la parte dinámica está escrita en Java, no en Visual Basic, por eso es mucho más poderosa y fácil de usar, y es portable a otros sistemas operativos y servidores Web.
- Contra los Servlets, JSP no nos da nada que no pudiéramos hacer en un Servlet, pero es más conveniente escribir y modificar HTML normal, que tener que hacer un billón de sentencias println que generen HTML, además separando el formato del contenido podemos poner diferentes personas en diferentes tareas, unos diseñando HTML y dejando espacio para que los programadores inserten JSP para el contenido dinámico.
- Contra Server-Side Includes (SSI), esta es una tecnología ampliamente soportada que incluye piezas definidas externamente dentro de una página Web estática. JSP nos permite usar Servlets en vez de un programa separado para generar las partes dinámicas, además, SSI está diseñado para inclusiones sencillas, no para programas reales que usen formularios de datos hagan conexiones a base de datos etc.

- Contra Javascript, éste puede generar HTML dinámico en el cliente, tiene una capacidad útil, pero sólo maneja situaciones donde la información dinámica está basada en el entorno del cliente, con la excepción de los Cookies, el HTTP y el envío de formularios no están disponibles con Javascript y como se ejecutan en el cliente no se puede acceder a los recursos en el lado del servidor, como base de datos, catálogos, información de precios etc.

3.21 Generalidades acerca JSP

Los JSP como se mencionó, nos permiten separar la parte dinámica de nuestras páginas Web de HTML estático, simplemente escribimos el HTML regular, usando cualquier herramienta de construcción de paginas Web, encerramos el código de las partes dinámicas en unas etiquetas especiales, las cuales las mayoría empiezan <% y terminan con %>.

Se le dará una extensión .jsp , y normalmente se instalará en el mismo sitio que una página html normal, aunque lo que escribamos se parezca a un fichero normal HTML en vez de un Servlet, detrás de la escena, la página JSP se convierte en un Servlet normal, donde el HTML estático simplemente se imprime en el stream de salida estándar asociado con el método service del Servlet. Las clases e interfaces especificas para la programación con JSP se encuentran en los paquetes *javax.servlet.jsp* y *javax.servlet.jsp.tagext*.

Hay cuatros componente clave de los JSP que son:

- **Directivas:** son mensajes para el contenedor de JSPs, que permiten al programador especificar configuraciones de página, incluir contenido de otros recursos y especificar bibliotecas de marcas personalizadas para usarlas en un JSP.
- **Acciones:** encapsulan la funcionalidad en marcas predefinidas que los programadores pueden incrustar en un JSP. A menudo, las acciones se realizan con base en la información que se envíe al servidor como parte de una petición de cliente específica.
- **Scriptlets o elementos de secuencia de comandos:** permiten a los programadores insertar código de Java que interactúe con los componentes de un JSP y probablemente

con otros componentes de la aplicación Web, para realizar el procesamiento de las peticiones.

§ **Bibliotecas de marcas:** son parte del mecanismo de extensión de marcas (tags) que permiten a los programadores crear marcas personalizadas, dichas marcas permiten a los diseñadores manipular el contenido de un JSP.

A continuación se muestra una tabla con el sumario de sintaxis para JSP.

Elemento JSP	Sintaxis	Interpretación
Expresión JSP	<%= expresión %>;	La expresión es evaluada y situada en la salida
Scriptlet JSP	<% code %>	El código se inserta en el método service
Declaración JSP	<%! code %>	El código se inserta en el cuerpo de la clase del Servlet, fuera del método service
Directiva page JSP	<%@ page att= “val” %>	Dirige al motor Servlet sobre la configuración general
Directiva include JSP	<%@ include file = “url” %>	Un fichero del sistema local se incluirá cuando la página se traduzca a un Servlet
Comentario JSP	<%-- comment --%>	Comentario ignorado cuando se traduce la página JSP en un Servlet
Acción jsp:include	<jsp:include page “relative URL” flush=”true” />	Incluye un fichero en el momento en que la página es solicitada
Acción jsp:useBean	<jsp:useBean att=val> ... </jsp:useBean>	Encuentra o construye un Java Bean
Acción jsp:setProperty	<jsp:setProperty att=val*/>	Selecciona las propiedades del bean, directamente o designando el valor que viene desde un parámetro de la petición
Acción jsp:getProperty	<jsp:getProperty name=”propertyName” value=”val”/>	Recupera y saca las propiedades del Bean
Acción jsp:forward	<jsp:forward page = “relative URL”	Reenvía la petición a otra página
Acción jsp:plugin	<jsp:plugin attribute= “value” > ... </jsp:plugin>	Genera etiquetas Object o EMBED apropiadas al tipo del navegador, pidiendo que se ejecute un applet usando el java plugin

Acción: jsp:param	<jsp:param>	Se utiliza con las acciones include,forward y plugin para especificar pares nombre/valor adicionales de información para que sean utilizados por estas acciones

Un JSP se ve como un documento estándar HTML o XHTML porque incluye marcado de los mismos, a este marcado se le conoce como datos de plantilla fija o texto de plantilla fija. Los datos de plantilla fija a menudo ayudan a que un programador decida si va a utilizar un Servlet o un JSP. Los programadores tienden a utilizar JSPs cuando la mayoría del contenido que se envía al cliente está compuesto por datos de plantilla fija y sólo una pequeña porción del contenido se genera en forma dinámica mediante código Java. Comúnmente se utilizan Servlets cuando sólo una pequeña porción del contenido que se envía al cliente está compuesta por datos de plantilla fija.

De hecho, algunos Servlets no producen contenido, en vez de ello, realizan una tarea a beneficio del cliente y después invocan a otros Servlets y JSPs, que son intercambiables. Al igual que los Servlets, los JSPs por lo general se ejecutan como parte de un servidor Web, el componente del servidor que los ejecuta se conoce como contenedor de JSPs.

Cuando un servidor habilitado para usar JSP recibe la primera petición de un JSP, el contenedor de JSPs traduce ese JSP en un Servlet de Java que maneja la petición actual y las peticiones futuras de ese JSP. Si hay errores al compilar el nuevo Servlet, éstos se convierten en errores en tiempo de traducción. El contenedor de JSPs coloca las instrucciones de Java que implementan las petición del JSP en el método _jspService, en tiempo de traducción.

Si el nuevo Servlet se compila apropiadamente, el contenedor de JSPs invoca al método _jspService para procesar la petición. Los JSP pueden responder directamente a la petición, o pueden invocar a otros componentes de la aplicación Web para que le ayuden a procesarla petición. Cualquier error que ocurra durante el procesamiento de la petición se denomina error en tiempo de petición.

El mecanismo de petición/respuesta y el ciclo de vida de un JSP son iguales que los de un Servlet. Un JSP puede definir los métodos jspInit y jspDestroy similares a los métodos init y destroy del Servlet , que son invocados por el contenedor de JSPs al inicializar y terminar un JSP respectivamente.

3.22 Objetos Implícitos

Los objetos implícitos proporcionan a los programadores el acceso a muchas herramientas de Servlets, dentro del contexto de un JSP, los objetos implícitos tienen cuatro alcances: aplicación, página, petición y sesión. Los JSP y la aplicación contenedora de Servlets poseen objetos con alcance de aplicación. Cualquier Servlet o JSP puede manipular a dichos objetos. Los objetos con alcance de página existen sólamente en la página que los define. Cada página tiene sus propias instancias de los objetos implícitos con alcance de página. Los objetos de alcance de petición existen durante el tiempo de vida de la petición, por ejemplo un JSP puede procesar parcialmente una petición y después reenviarla hacia otros Servlet o JSP para seguir procesándola. Los objetos con alcance de petición quedan fuera de alcance cuando se completa el procesamiento de la petición mediante una respuesta al cliente. Los objetos con alcance de sesión existen durante toda la sesión de navegación del cliente. En la siguiente tabla se describen los objetos implícitos y sus alcances, además de los objetos implícitos en JSP y sus alcances.

Objeto implícito	Alcance	Descripción
application	Aplicación	Este objeto javax.servlet.ServletContext representa el contenedor en el que se ejecuta el JSP
config	Página	Este objeto javax.Servlet.ServletConfig representa las opciones de configuración del JSP, al igual que con los Servlets, las opciones de configuración pueden especificarse en un descriptor de aplicación Web
exception	Página	Este objeto java.lang.Throwable representa la excepción que se pasa a la página de error de JSP, este objeto está disponible sólamente en una página de error de JSP

out	Página	Este objeto javax.servlet.jsp.JspWriter escribe texto como parte de la respuesta a una petición. Este objeto se utliza en forma implícita con extensiones y acciones de JSP para insertar contenido de cadena en una respuesta
page	Página	Este objeto java.lang.Object representa la referencia this para instancia actual del JSP
pageContext	Página	Este objeto de javax.servlet.jsp.CpageContext oculta los detalles de implementación del Servlet y contenedor de JSPs subyacentes, y proporciona a los programadores de JSPs el acceso a los objetos implícitos que se describen en esta tabla
response	Página	Este objeto representa la respuesta al cliente y generalmente es una instancia de una clase que implementa a HttpServletResponse, si se utiliza el protocolo distinto a http, este objeto es una instancia de una clase que implementa a javax.Servlet.ServletResponse
request	Petición	Este objeto representa la petición del cliente. El objeto normalmente es una instancia de una clase que implementa a HttpServletRequest. si se utiliza el protocolo distinto a http, este objeto es una instancia de una clase que implementa a javax.Servlet.ServletRequest
session	Sesión	Este objeto javax.servlet.http.httpSession representa la información de sesión del cliente, si es que se ha creado dicha sesión, este objeto está disponible solamente en páginas que participen en una sesión

3.23 Elementos de Scripts en JSP

Los elementos de scripts nos permiten insertar código Java dentro del Servlet que se generará desde la página JSP actual; hay tres formas:

- Expresiones de la forma <%= expresión %> que son evaluadas e insertadas en la salida.
- De la forma <% código %> que se insertan dentro del método service del Servlet
- Declaraciones de la forma <%! código %> que se insertan en el cuerpo de la clase del Servlet, fuera de cualquier cuerpo existente.

La primera forma <%= código %> es evaluada, convertida a string e insertada en la página, esta evaluación se realiza durante la ejecución (cuando se solicita la página), y así se tiene total acceso a la información sobre la solicitud, por ejemplo, se puede mostrar la fecha y hora.

```
<%= new java.útil.Date( ) %>
```

Para simplificar estas expresiones, hay un número de variables predefinidas que podemos usar, las más importantes son:

- request, el HttpServletRequest
- response, el HttpServletResponse
- session, el HttpSession asociado con el request (si existe)
- out, el PrintWriter (una versión del buffer del tipo JspWriter) para enviar la salida al cliente

Por ejemplo, si deseamos tener la dirección del host

```
<%= request.getRemoteHost ( ) %>
```

Si se requiere hacer algo más complejo que insertar una simple expresión, ocupamos la segunda forma, los scriptlets de JSP nos permiten insertar código arbitrario dentro del método servlet que será construido al generar la página, lo cual se tiene la siguiente forma <% código % >, por ejemplo:

```
<% if (Math.random() < 0.5 ) { %>
    Menor a <B> .5 </B>
<% } else { %>
     Mayor a <B> .5 </B>
  <% } %>
```

La tercera forma nos permite una declaración JSP para definir métodos y campos que serán insertados dentro del cuerpo principal de la clase servlet (fuera del método service que procesa la petición), que tiene la siguiente forma <%! código %>, como las declaraciones ninguna salida, normalmente se usan en conjunción con expresiones JSP o scriplets, los campos se convierten en campos de instancia de la clase servlet, y los métodos en miembros de la clase que representa a la

JSP traducida, por lo que los JSPs no deben almacenar información de estado del cliente en variables de instancia, en vez de ello deben utilizar el objeto implícito session.

3.24 Acciones estándar

Estas acciones proporcionan a los que implementan JSPs el acceso a varias de las tareas más comunes que se realizan en un JSP, como incluir contenido de otros recursos, reenviar peticiones hacia otros recursos e interactuar con JavaBeans. Los contenedores de JSPs, procesan las acciones en tiempo de petición. Las acciones están delimitadas por las marcas <jsp:acción> y </jsp:acción>, en donde acción es el nombre de la acción estándar. En casos en los que no aparece nada entre las marcas de inicio y fin, puede usarse la sintaxis de elemento vacío <jsp:acción /> de XML.

Los JavaServer Pages soportan dos mecanismos de inclusión, la acción <jsp:include> y la directiva include. La acción <jsp:include> permite incluir contenido dinámico en un JSP, en tiempo de petición (no en tiempo de traducción, como es el caso de la directiva include). Si el recurso incluido cambia entre peticiones, la siguiente petición al JSP que contenga la acción <jsp:include> incluirá el nuevo contenido del recurso. Por otro lado, la directiva include copia en el JSP una vez, en tiempo de traducción de JSP, si el recurso incluido cambia, el nuevo contenido no se reflejará en el JSP que utilizó la directiva include, a menos que se vuelva a compilar esa JSP. Sus atributos son los siguientes.

Atributo	Descripción
page	Especifica la ruta URL realtiva del recurso a incluir. El recurso debe formar parte de la misma aplicación Web
flush	Especifica si el búfer debe vaciarse después de realizar la inclusión.

La acción <jsp:forward> permite a un JSP reenviar el procesamiento de la petición a un nuevo recurso distinto. El procesamiento de la petición por parte de la JSP original termina tan pronto como el JSP reenvía la petición. Éste sólo tiene un atributo page que especifica el URL, relativo del recurso (en la misma aplicación Web) hacia el que deberá reenviarse la petición.

Un JSP cuando utiliza la acción <jsp:useBean> permite manipular un objeto de Java. Esta acción crea un objeto de Java o localiza uno existente para utilizarlo en JSP. Si no se especifican los atributos class y beanName, el contendor de JSPs trata de localizar un objeto existente del tipo especificado en el atributo type. Al igual que los objetos implícitos de JSP, los objetos que se especifican con la acción <jsp:useBean> tienen alcance tipo page, request, session o application, el cual indica en qué parte de una aplicación Web pueden utilizarse. Los objetos con alcance tipo page pueden ser utilizados sólamente por la página en la que están definidos. Varias páginas de JSP pueden potencialmente acceder a los objetos en otros alcances, por ejemplo, todas las JSPs que procesan una sola petición pueden acceder a un objeto con alcance tipo request. Sus atributos son los siguientes

Atributo	Descripción
id	El nombre para manipular el objeto de Java con las acciones <jsp:setProperty> y <jsp:getProperty>, una variable de este nombre también se declara para usarse en los elementos de secuencia de comandos de JSP. El nombre que se especifica aquí es susceptible al uso de mayúsculas y minúsculas
scope	El alcance en el cual el objeto de Java puede utilizarse, como page, request, session o application. El alcance predeterminado es page
class	El nombre completamente calificado de la clase del objeto de Java
beanName	El nombre de un bean que puede utilizarse con el método instantiate de la clase java.beans.Beans para cargar un JavaBean en memoria.
type	El tipo de JavaBean. Este puede ser el mismo tipo que el atributo class, una superclase de este tipo o un interfaz implementada por este tipo. El valor predeterminado es el mismo que para el atributo class. Se produce una exepción ClassCastException si el objeto de Java no es del tipo especificado con el atributo type

3.25 Directivas

Las directivas son mensajes para el contenedor de JSPs, que permiten al programador especificar configuraciones de página (como la página de error), incluir de otros recursos y especificar bibliotecas de marcas personalizadas para usarlas en un JSP. Las directivas (delimitadas por <%@ y %>) se procesan en tiempo de traducción. Por lo tanto, las directivas no producen ningún tipo de salida inmediata, ya que se procesan antes de que el JSP acepte cualquier petición.

La directiva page especifica las configuraciones globales del JSP en el contenedor de JSPs, puede haber muchas directivas page, siempre y cuando sólo haya una ocurrencia de cada atributo. La única excepción a esto es el atributo import, el cual puede usarse repetidamente para importar los paquetes de Java que se utilicen en el JSP, sus atributos son los siguientes:

Atributo	Descripción
language	El lenguaje de secuencias utilizando en el JSP, hasta el momento, el único valor permitido en este atributo es java
extends	Especifica la clase a partir de la cual se heredará el JSP traducida, este atributo debe ser un nombre de clase completamente calificado
import	Especifica una lista separada por comas de los nombres completamente calificados de los tipos y/o paquetes que se utilizan en el JSP actual
session	Especifica si la página participa en un sesión. Los valores para este atributo son true que participa en una sesión que son los valores predeterminado o false, cuando la página forma parte de una sesión, el objeto implícito session está disponible para ser utilizado en esa página. En caso contrario, session no está disponible y si se utiliza en el código de la secuencia de comandos, se genera un error en tiempo de traducción
buffer	Especifica el tamaño del búfer de salida utilizado con el objeto implícito out. El valor de este atributo puede ser none cuando no se usa búfer, o puede ser un valor de 8kb que es el prederterminado.

autoFlush	Cuando se establece en true valor predeterminado, indica que el búfer de salida utilizado con el objeto implícito out debe vaciarse automáticamente al llenarse el búfer, si se establece en false, se produce una excepción si el búfer se desborda, el valor de este atributo debe ser true si el atributo buffer se establece none
isThreadSafe	Especifica si la página es segura para usar subprocesos. Si es true valor predeterminado, la página se considera como segura para usar subprocesos y puede procesar varias peticiones al mismo tiempo. Si es false el servlet que representa a la página implementa a la interfaz java.lang.SingleThreadModel y ese JSP sólo puede procesar una petición a la vez
info	Especifica una cadena de información que describe la página. Esta cadena es devuelta por el método getServletInfo del servlet que representa a un JSP traducido. Este método puede invocarse a través del objeto implícito page del JSP
errorPage	Cualquier excepción en la página actual que se atrape, se envía a la página de error para su procesamiento. El objeto implícito exception de la página de error hace referencia a la excepción original
isErrorPage	Especifica si la página actual es una página de error que será invocada en respuesta a un error en otra página. Si el valor del atributo es true, se crea el objeto implícito exception y hace referencia a la excepción original que ocurrió. Si es false que es la opción predeterminada, cualquier uso del objeto exception en la página producirá un error en tiempo de traducción
contentType	Especifica el tipo MIME de los datos en la respuesta al cliente. El tipo predeterminado es text/html

Por otro lado, la directiva include incluye el contenido de otro recurso una vez, en tiempo de traducción del JSP. Esta sólo tiene un atributo que es file, el cual especifica el URL del recurso a incluir. Como se mencionó, la diferencia entre la directiva include y la acción <jsp:include> se nota sólamente si cambia el contenido incluido. Por ejemplo, si la definición de un documento de XHTML cambia después de ser incluido mediante la directiva include, las futuras invocaciones

de un documento del JSP mostrarán el contenido original del documento XHTML, no en el documento. En contraste, la acción <jsp:include> se procesa en cada petición del JSP que utilice la acción <jsp:include>.

3.26 Bibliotecas de Marcas (tags)

Los JSP están compuestos por varias marcas, todos ellos encapsulan de una manera sencilla, por ejemplo la marca <jsp:useBean> que crea una instancia de una clase definida como un JavaBean, esto hace que nosotros podamos separar la lógica de presentación de la del proceso, y consigue unos JSPs más claros, y lo más importante el que escribe un JSP no tiene que tener conocimientos avanzados en programación para crear JSP de funcionalidades más complejas.

Con los Custom Tag se pueden añadir nuevas marcas a las ya existentes en JSP, para encapsular comportamientos más complicados. Se tiene dos tipos de marcas, las que no tienen cuerpo y las que si tienen, por ejemplo <img src=”cruz azul.jpg”>, no tiene cuerpo ni marca al final, <title>Inicio en marcas</title> se tiene un cuerpo y marca al final. Los componentes que componen a los Custom Tag son:

- Tag Handler class: Clase que define la acción de la marca.
- Tag Library Descriptor (TLD): Mapeo de los elementos XML a manejadores de clase.
- JSP: los que utilizan los Custom Tag.

Tenemos tres formas para implementar una marca, se implementa una interfaz o extender de dos clases, si se quiere implementar una interfaz, se tiene que tener los siguientes requerimientos:

- setPageContext: Este método lo invoca el motor JSP para definir el contexto.
- setParent: Lo invoca el motor JSP para pasarle un manejador de marca a la marca padre.
- getParent: Devuelve la instancia de la marca padre.
- doStartTag: Se procesa la marca de apertura.
- doEndTag: Se ejecuta después de doStartTag.
- release: Lo invoca el motor JSP para hacer limpieza.

Si sólo la marca tiene una funcionalidad mínima, lo que podemos hacer es sólo heredar de la clase TagSupport, para sobrescribir los métodos que necesitamos. Pero si nuestra marca necesita tener cuerpo, entonces heredamos de la clase BodyTagSupport, con esta clase también sobrescribimos los métodos que necesitamos.

El archivo TLD, los elementos que necesitamos para poder ocupar nuestras marcas creadas son los siguientes:

Sintaxis	Interpretación
<taglib>	à Se empieza a definir la marca
<tlibversion>1.0</tlibversion>	à Versión de la librería
<jspversion>1.1</jspversion>	à Especificación JSP
<shortname>nombre</shortname>	à Nombre Corto
<uri></uri>	à Ubicación del recurso
<info>Etiquetas de ejemplo</info>	à Especificar que hace la marca
<tag>	
<name>holamundo</name>	à Nombre de la marca
<tagclass>HolaMundoTag</tagclass>	à La clase que implementa la marca
<bodycontent>empty</bodycontent>	à Si tiene cuerpo la marca, en este caso no
<info>Saludo</info>	à Información de la marca
</tag>	
</taglib>	

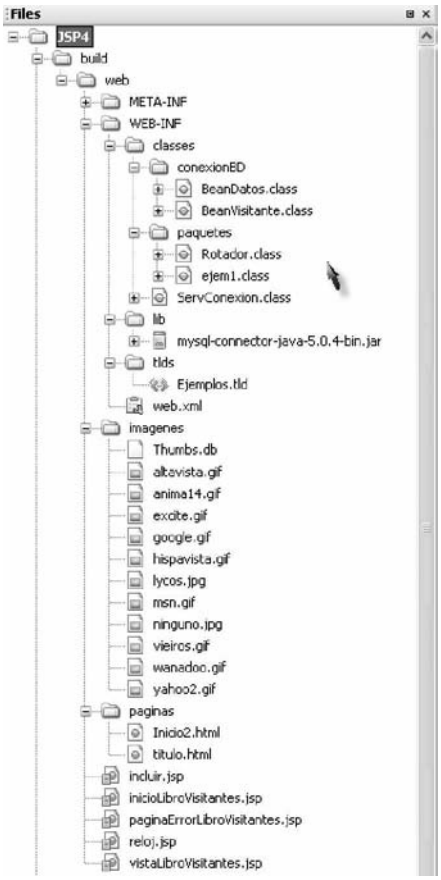
Cabe mencionar que el elemento <bodycontent> cuando se necesita que lleve cuerpo ya sea texto, html o otras etiquetas se indica JSP, y otro tipo de contenido es tagdependent, esto no afecta la interpretación de la etiqueta, sólo es informativo.

3.27 Ejemplo de una Aplicación Web con JSP

Lo que vamos hacer en este ejemplo es un libro de visitantes en donde el usuario ingresa su nombre de pila, apellido y correo electrónico, estos datos serán almacenados en una base de datos utilizando MySQL, con esto podemos consultar estos datos y poderlos desplegar en una tabla.
Dado que podemos utilizar Servlet en el contexto de un JSP, vamos a incluir el ejemplo anterior del Servlet en esta aplicación sin afectar su funcionamiento, los componentes clave que utilizamos de los JSPs son:

- Acciones: <jsp:useBean>,<jsp:include>,<jsp:getProperty>,<jsp:setProperty>,<jsp:forward>.
- Directivas: <@page ..>.
- Scriptlets: <% ..%>, <%=expresión %>
- Bibliotecas de marca: Se crea una marca que se llama ejem1 y se encuentra en el paquete "paquetes".
- Servicios: Conexión a una base de Datos de MySQL a través de JDBC, esta base de datos se llama "visitantes", y contiene una sola tabla llamada "visitante" con los campos nombrePila, apellidoPaterno e email. También se ocupa la base de datos "navegadores" para que funcione correctamente el Servlet (ServConexion.java).

La siguiente imagen muestra la estructura de cómo queda nuestro directorio en Netbeans:



Señalemos que estamos utilizando JDBC por lo que utilizamos el controlador de MySQL (mysql-connector-java-5.0.4-bin.jar), en la carpeta (tlds) se guarda el descriptor de librerías de marcas, se tiene otra carpeta (imágenes) donde se almacenan todas las imágenes que necesitamos, la carpeta (páginas) guarda la página Inicio2.html que se ocupó en el ejemplo anterior del Servlet, titulo.html solo tiene texto que se incluirá como título en la aplicación.

Se tiene un paquete "conexionBD" que guarda dos JavaBeans, BeanDatos y BeanVisitante, el primero nos ayuda a establecer la conexión a la base de datos, así como registrar y recuperar los datos de MySQL, el segundo sólo sirve para almacenar los datos recogidos del formulario para su posterior almacenamiento en MySQL.

El paquete "paquete", tiene dos clases Rotador.java y ejem1.java, Rotador.java sirve para hacer un efecto rotativo de imágenes en el título, cada vez que se inicializa una petición al JSP, ejem1.java es la implementación de la interfaz necesaria para crear nuestra marca personal que vamos a ocupar. Antes de explicar los JPs que se ocupan, primero explicaremos JavaBeans que tenemos, si nos basamos en el patrón MVC, éstos caen en la parte de Modelo.

BeanVisitante nos sirve para guardar los datos proporcionados por el usuario en el formulario, para luego después vaciarlos en la base de datos, su código es el siguiente:

```
1 package conexionBD;
 2 public class BeanVisitante {
 3    private String nombrePila, apellidoPaterno, email;
 4    // establecer el nombre de pila del visitante
 5    public void setNombrePila( String nombre )
 6    {
 7       nombrePila = nombre;
 8    }
 9    // obtener el nombre de pila del visitante
10     public String getNombrePila()
11     {
12        return nombrePila;
13     }
14     // establecer el apellido paterno del visitante
15     public void setApellidoPaterno( String nombre )
16     {
17        apellidoPaterno = nombre;
18     }
19     // obtener el apellido paterno del visitante
20     public String getApellidoPaterno()
21     {
```

```
22      return apellidoPaterno;
23    }
24    // establecer la dirección de email del visitante
25    public void setEmail( String direccion )
26    {
27      email = direccion;
28    }
29    // obtener la dirección de email del visitante
30    public String getEmail()
31    {
32      return email;
33    }
34 }
```

Después se tiene BeanDatos, éste proporciona conexión a la base de datos, obtención de los datos, poder guardar los datos y por último cerrar la conexión para liberar el recurso; su programación es:

```
1 package conexionBD;
 2 import java.io.*;
 3 import java.sql.*;
 4 import java.util.*;
 5 public class BeanDatos {
 6    private Connection conexion;
 7    private Statement instruccion;
 8    /** Creates a new instance of BeanDatos */
 9    public BeanDatos()throws Exception{
10       //cargando la clase controladora
11       Class.forName("com.mysql.jdbc.Driver");
12       //establecer la conexion con la BD
13       conexion=DriverManager.getConnection("jdbc:mysql://localhost:3306/visitantes",
14            "miguel","1234");
15       //creando Statement para la consulta BD
16       instruccion=conexion.createStatement();
17    }
18    //devolver un objeto ArrayList de objetos BeanDatos
19    public List getListaVisitantes() throws SQLException{
20       List listaVisitantes=new ArrayList();
21       //obtener la lista de los titulos
22       ResultSet resultados = instruccion.executeQuery(
23            "SELECT nombrePila, apellidoPaterno, email FROM visitante" );
24
25       // obtener datos de la fila
26       while ( resultados.next() ) {
27          BeanVisitante visitante = new BeanVisitante();
28          visitante.setNombrePila( resultados.getString( 1 ) );
29          visitante.setApellidoPaterno( resultados.getString( 2 ) );
30          visitante.setEmail( resultados.getString( 3 ) );
31          listaVisitantes.add( visitante );
32       }
33       return listaVisitantes;
34    }
35    // insertar un visitante en la base de datos del libro de visitantes
36    public void agregarVisitante( BeanVisitante visitante ) throws SQLException {
37       instruccion.executeUpdate( "INSERT INTO visitante ( nombrePila, " +
```

```
38            "apellidoPaterno, email ) VALUES ( '" + visitante.getNombrePila() + "', '" +
39            visitante.getApellidoPaterno() + "', '" + visitante.getEmail() + "' )" );
40    }
41    // cerrar instrucciones y terminar la conexión a la base de datos
42    protected void finalize() {
43       // tratar de cerrar la conexión a la base de datos
44       try {
45          instruccion.close();
46          conexion.close();
47       }
48       // procesar posible excepcion SQLException en operación de cierre
49       catch ( SQLException excepcionSQL ) {
50          excepcionSQL.printStackTrace();
51       }
52    }
53 }
```

Rotador.java nos sirve para que cuando se haga una petición a nuestro JSP cambia una imagen en el título por cada petición.

```
1 package paquetes;
 2 public class Rotador {
 3   //arreglo que guarda los nombre de las imagenes a mostrar
 4    private String imagenes[]={"imagenes/anima14.gif",
 5    "imagenes/google.gif","imagenes/excite.gif","imagenes/lycos.jpg"};
 6    //para cambiar el indice del arreglo
 7    private int indice=0;
 8    //para obtener la imagen
 9    public String getImagen(){
10       return imagenes[indice];
11    }
12    //cambia el indice del arreglo
13    public void siguiente(){
14       indice=(indice+1)%imagenes.length;
15    }
16 }
```

ejem1.java herede de la interfaz TagSupport los métodos necesarios para crear nuestra marca personal, sólo regresa una cadena (Mostrar Libro de Visitantes), que podemos ocupar en cualquier JSP dentro de nuestra proyecto.

```
1 package paquetes;
 2 import java.io.*;
 3 import javax.servlet.jsp.*;
 4 import javax.servlet.jsp.tagext.*;
 5 public class ejem1 extends TagSupport{
 6     public int doStartTag()throws JspException{
 7         try{
 8             pageContext.getOut().print("Mostrar Libro de Visitantes");
 9         }
10          catch(IOException e){
```

```
11            throw new JspException("Error de E/S"+
12                  e.getMessage());
13        }
14        return SKIP_BODY;
15     }
16     public int doEndTag() throws JspException{
17        return EVAL_PAGE;
18     }
19 }
```

Una vez explicado la Parte de Modelo, vamos a analizar cada JSPs que se tiene, estos hacen la parte de Vista, y algunos de Vista / Controlador al mismo tiempo.

El JSP reloj.jsp se muestra la utilización del elemento <%=expresión%> que es evaluada e insertada al mismo tiempo, y a su vez se utiliza <% .... %> que en este caso es para darle formato a la salida de fecha y hora según la configuración local del usuario.

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 4 "http://www.w3.org/TR/html4/loose.dtd">
 5 <table>
 6    <tr>
 7       <td style="background-color: black;">
 8          <p class ="grande" style="color: cyan; font-size: 3em;
 9          font-weight:bold;">
10             <%
11             java.util.Locale confRegional=request.getLocale();
12             java.text.DateFormat formato=
13                 java.text.DateFormat.getDateTimeInstance(
14                 java.text.DateFormat.LONG,
15                 java.text.DateFormat.LONG,confRegional);
16             %>
17             <%=formato.format(new java.util.Date())%>
18          </p>
19       </td>
20    </tr>
21 </table>
```

Como necesitamos desplegar un formulario, el JSP inicioLibroVisitantes.jsp hace esa función, primero configuramos la página con la directiva page, en la línea 6, cualquier excepción que nose atrape, se envía a la página “paginaErrorLibroVisitantes.jsp”, con las acciones de <jsp:useBean ..> utilizamos los beans que ocuparemos (líneas 8-11), “visitante” es el identificador para el objeto de la clase BeanVisitante, “datosVisitante“ identifica al objeto de la clase BeanDatos.

A la línea 28 le decimos que queremos todas las propiedades de “visitante”, las líneas 30-64 sirven para desplegar un formulario, cuando se llenan el formulario la petición la reenvía al mismo JSP, entonces se ejecuta la sentencia else (líneas 65-71) para mandar los datos ala base de datos (visitantes), línea 66 , una vez llenado los datos, sólo falta mostrarlos, para eso ocupamos la acción <jsp:forward> y el JSP encargado de esta tarea es vistaLibroVisitantes.jsp, este JSP (inicioLibroVisitantes.jsp) tiene la parte de Vista y Control al mismo tiempo en el modelo MVC, su código es el siguiente:

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 4 "http://www.w3.org/TR/html4/loose.dtd">
 5 <%-- configuración de página --%>
 6 <%@ page errorPage = "paginaErrorLibroVisitantes.jsp" %>
 7 <%-- beans utilizados en esta JSP --%>
 8 <jsp:useBean id = "visitante" scope = "page"
 9             class = "conexionBD.BeanVisitante" />
10 <jsp:useBean id = "datosVisitante" scope = "request"
11              class = "conexionBD.BeanDatos" />
12 <html xmlns = "http://www.w3.org/1999/xhtml">
13    <head>
14       <title>Inicio de sesión del libro de visitantes</title>
15       <style type = "text/css">
16          body {
17          font-family: tahoma, helvetica, arial, sans-serif;
18          }
19          table, tr, td {
20          font-size: .9em;
21          border: 3px groove;
22          padding: 5px;
23          background-color: #dddddd;
24          }
25       </style>
26    </head>
27    <body>
28       <jsp:setProperty name = "visitante" property = "*" />
29       <% // iniciar scriptlet
30       if ( visitante.getNombrePila() == null ||
31            visitante.getApellidoPaterno() == null ||
32            visitante.getEmail() == null ) {
33       %> <%-- terminar scriptlet para insertar datos de plantilla fija --%>
34       <form method = "post" action = "inicioLibroVisitantes.jsp">
35          <p>Escriba su nombre de pila, apellido paterno y dirección
36          de e-mail para registrarlo en nuestro libro de visitantes.</p>
37          <table>
38             <tr>
39                <td>Nombre de pila</td>
40                <td>
41                   <input type = "text" name = "nombrePila" />
42                </td>
```

```
43          </tr>
44          <tr>
45            <td>Apellido paterno</td>
46            <td>
47              <input type = "text" name = "apellidoPaterno" />
48            </td>
49          </tr>
50          <tr>
51            <td>Email</td>
52            <td>
53              <input type = "text" name = "email" />
54            </td>
55          </tr>
56          <tr>
57            <td colspan = "2">
58              <input type = "submit" value = "Enviar" />
59            </td>
60          </tr>
61        </table>
62      </form>
63      <% // continuar scriptlet
64      }  // fin de instrucción if
65      else {
66        datosVisitante.agregarVisitante( visitante );
67      %> <%-- terminar scriptlet para insertar acción jsp:forward --%>
68      <%-- reenviar para mostrar el contenido del libro de visitantes --%>
69      <jsp:forward page = "vistaLibroVisitantes.jsp" />
70      <% // continuar scriptlet
71      }  // fin de instrucción else
72      %> <%-- fin de scriptlet --%>
73    </body>
74 </html>
```

Toca el turno del JSP vistaLibroVisitantes.jsp, este sólo se encarga de mostrar en una tabla la lista de los datos de los visitantes ya registrados, la configuración de este JSP empieza cargando la página de excepciones (paginaErrorLibroVisitantes.jsp), el paquete de utilerías, así como el paquete completo de “conexionBD”, líneas 6-8, nombramos un identificador “datosVisitante” para poder ocupar el objeto de la clase BeanDatos. En las líneas 42-44 obtenemos los valores de la base de datos, para desplegarlos hacemos un ciclo con While, y se van añadiendo los datos en cada fila de la tabla (líneas 45-57), por último hacemos un enlace de nuevo a la página de inicio (incluir.jsp) línea 61, el código es el siguiente:

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 4 "http://www.w3.org/TR/html4/loose.dtd">
 5 <%-- configuración de página --%>
 6 <%@ page errorPage = "paginaErrorLibroVisitantes.jsp" %>
 7 <%@ page import = "java.util.*" %>
 8 <%@ page import = "conexionBD.*" %>
 9 <%-- objeto BeanDatosInvitado para obtener lista de visitantes --%>
```

```
<jsp:useBean id = "datosVisitante" scope = "request"
             class = "conexionBD.BeanDatos" />
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title>Lista de visitantes</title>
    <style type = "text/css">
      body {
      font-family: tahoma, helvetica, arial, sans-serif;
      }

      table, tr, td, th {
      text-align: center;
      font-size: .9em;
      border: 3px groove;
      padding: 5px;
      background-color: #dddddd;
      }
    </style>
  </head>
  <body>
    <center>
      <p style = "font-size: 2em;">Lista de visitantes</p>
      <table>
        <thead>
          <tr>
            <th style = "width: 100px;">Apellido paterno</th>
            <th style = "width: 100px;">Nombre de pila</th>
            <th style = "width: 200px;">Email</th>
          </tr>
        </thead>
        <tbody>
          <% // iniciar scriptlet
          List listaVisitantes = datosVisitante.getListaVisitantes();
          Iterator iteradorListaVisitantes = listaVisitantes.iterator();
          BeanVisitante visitante;
          while ( iteradorListaVisitantes.hasNext() ) {
            visitante = ( BeanVisitante ) iteradorListaVisitantes.next();
          %> <%-- terminar scriptlet; insertar datos de plantilla fija --%>
          <tr>
            <td><%= visitante.getApellidoPaterno() %></td>
            <td><%= visitante.getNombrePila() %></td>
            <td>
              <a href = "mailto:<%= visitante.getEmail() %>">
              <%= visitante.getEmail() %></a>
            </td>
          </tr>
          <% // continuar scriptlet
          } // fin de instrucción while
          %> <%-- fin de scriptlet --%>
        </tbody>
      </table>
      <a href="incluir.jsp">Pagina de inicio</a>
    </center>
  </body>
</html>
```

Para capturar todos los errores posibles, el JSP paginaErrorLibroVisitantes.jsp tiene esa funcionalidad, su configuración se hace indicando en la directiva page el atributo isErrorPage=”true”, con esto especificamos que la página actual es de error y que fue invocada en respuesta a un error de otra página, se importan los paquetes de utilerías y sql de java, (líneas 6-8), de las líneas 24-33 se da informes de por qué se produjo ese error, ya que tenemos acceso al objeto implícito “exception“ del JSP,se hace un enlace a la página de inicio(incluir.jsp), línea 34, las líneas de código son las siguiente:

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 4 "http://www.w3.org/TR/html4/loose.dtd">
 5 <%-- configuración de página --%>
 6 <%@ page isErrorPage = "true" %>
 7 <%@ page import = "java.util.*" %>
 8 <%@ page import = "java.sql.*" %>
 9 <html xmlns = "http://www.w3.org/1999/xhtml">
10    <head>
11       <title>¡Error!</title>
12       <style type = "text/css">
13          .rojoGrande {
14          font-size: 2em;
15          color: red;
16          font-weight: bold;
17          }
18       </style>
19    </head>
20    <body>
21       <p class = "rojoGrande">
22          <% // scriptlet para determinar el tipo de la excepción
23          // y enviar a la salida el comienzo del mensaje de error
24          if ( exception instanceof SQLException )
25          %>
26          Una excepción SQLException
27          ocurrió al interactuar con la base de datos del libro de visitantes.
28       </p>
29       <p class = "rojoGrande">
30          El mensaje de error fue:<br />
31          <%= exception.getMessage() %>
32       </p>
33       <p class = "rojoGrande">Por favor intente de nuevo más tarde</p>
34       <a href="incluir.jsp">Pagina de Inicio</a>
35    </body>
36 </html>
```

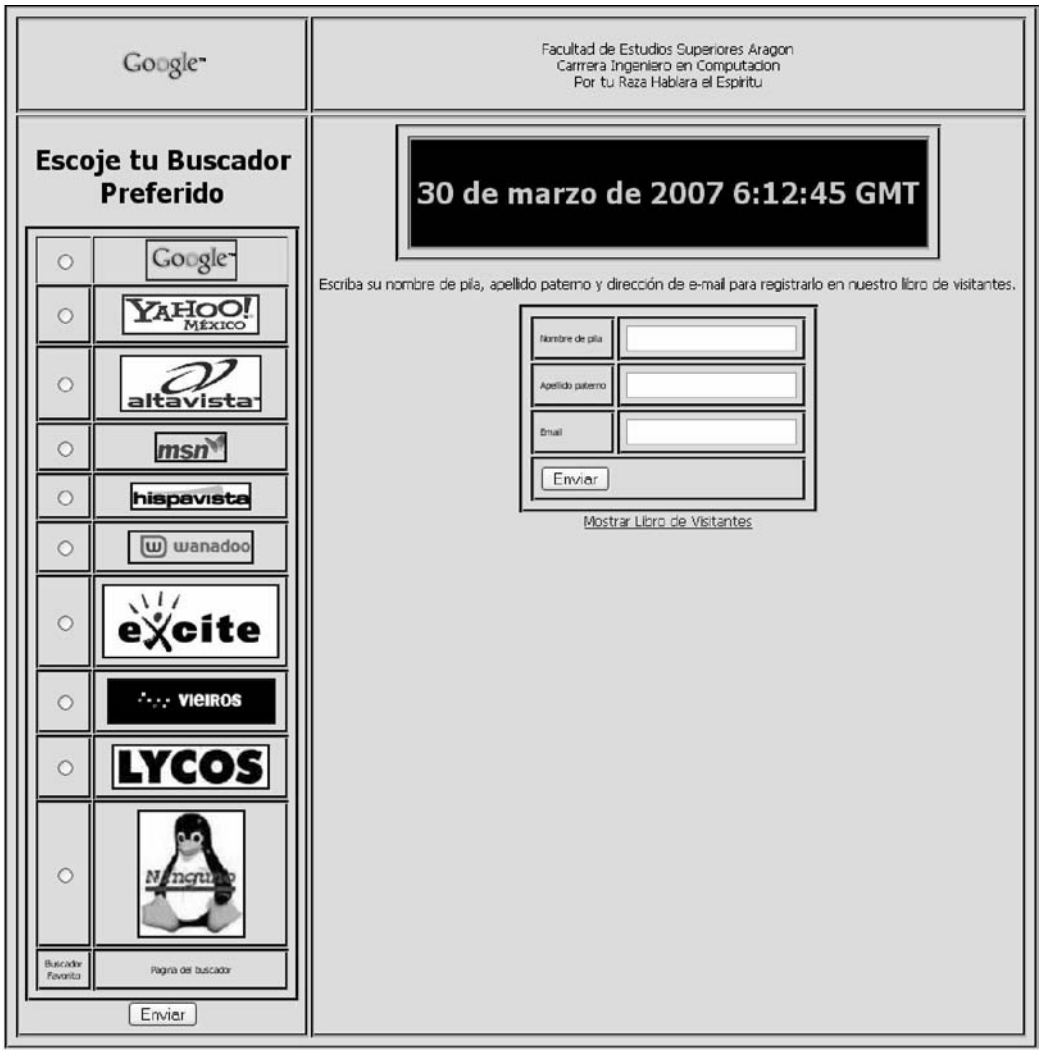
Una vez diseñado todo sólo falta incluirlo en un solo JSP, incluir.jsp tiene esta función, aquí cargamos nuestra marca que hicimos para esto utilizamos la directiva taglib, que como prefijo le

nombramos “ejemplos”, línea 3, y sólo ocuparemos un solo bean, que lo llamamos “rotador”, línea 7, para cambiar su índice de “rotador” llamamos al método “siguiente” línea 22, para obtener la cadena del nombre del archivo de la imagen que cambiara por cada petición,es a través de la acción <jsp:getProperty >, línea 29-30, se incluyen páginas html, titulo.html que sólo tiene texto, y Inicio2.html, que con ésta última ya podemos ocupar todo el ejemplo anterior del Servlet ServConexion.java (líneas 33,39), los JSPs añadidos son el del reloj.jsp, inicioLibroVisitantes.jsp y una referencia a vistaLibroVisitantes.jsp (líneas 43-45), es aquí en la línea 46, donde ocupamos nuestra marca creada que devuelve una cadena (Mostrar Libro de Visitante), que sirve como texto para que el usuario haga el enlace al JSP, las líneas del código son:

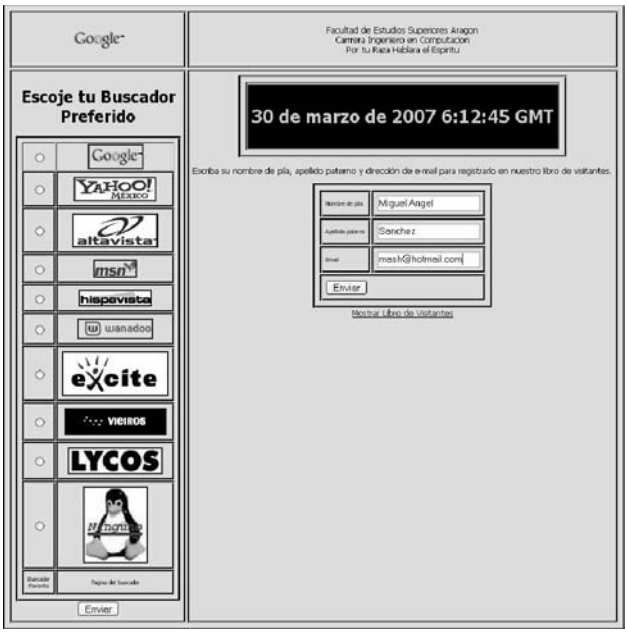
```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <%@ taglib uri="/WEB-INF/tlds/Ejemplos.tld" prefix="ejemplos"%>
 4 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 5 "http://www.w3.org/TR/html4/loose.dtd">
 6 <%-- bean utilizado en esta JSP --%>
 7 <jsp:useBean id="rotador" scope="session" class="paquetes.Rotador"/>
 8 <html>
 9   <head>
10     <title>Uso de jsp:include</title>
11     <style type="text/css">
12       body{
13       font-family: tahoma,helvetica,arial,sans-serif;
14       }
15       table,tr,td{
16       font-size: .9em;
17       border: 3px groove;
18       padding: 5px;
19       background-color: #dddddd;
20       }
21     </style>
22     <% rotador.siguiente();%>
23   </head>
24   <body>
25     <center>
26       <table>
27         <tr>
28           <td style="width: 160px; text-align: center">
29             <img src= "<jsp:getProperty name="rotador"
30                                property="imagen"/>" title="Anuncio"/>
31           </td>
32           <td><center>
33               <jsp:include page="paginas/titulo.html" flush="true"/>
34             </center>
35           </td>
36         </tr>
37         <tr>
```

```
38            <td style="width: 160px">
39              <jsp:include page="paginas/Inicio2.html" flush="true"/>
40            </td>
41            <td style="vertical-align: top">
42              <center>
43                <jsp:include page="reloj.jsp" flush="true"/>
44                <jsp:include page="inicioLibroVisitantes.jsp"/>
45                <a href="vistaLibroVisitantes.jsp">
46                  <ejemplos:salida1/>
47                </a>
48              </center>
49            </td>
50          </tr>
51        </table>
52      </center>
53    </body>
54 </html>
```

Para que no tengamos problemas con el Sevlet, el descriptor de archivo debe ser igual al ejemplo del Servlet, se muestran imágenes del funcionamiento de la aplicación web con los JSP.

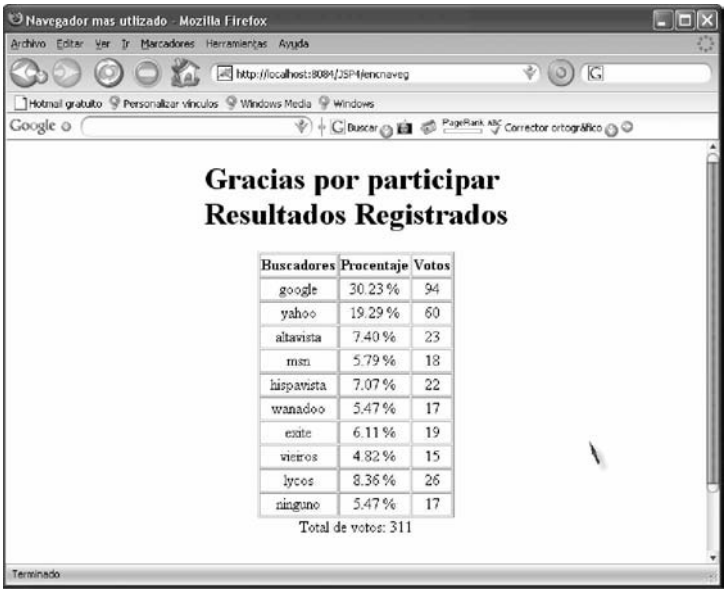


JSP incluir.jsp que agrupa los demas JSPs ocupados

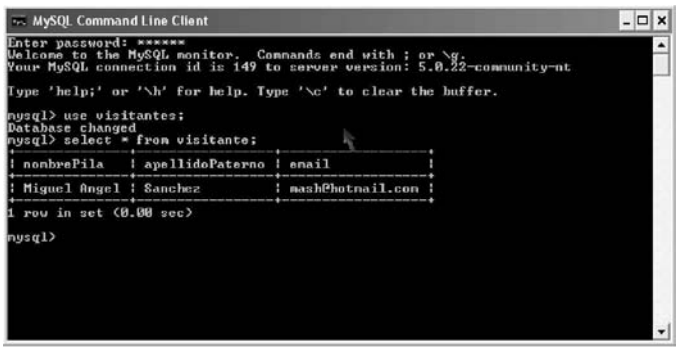


Registro de un usuario

Mostrando la lista de Visitantes



Funcionando del Servlet en un JSP



Comprobación de los datos guardados en la Base de Datos visitantes

4

Struts

4.1 Introducción

Realizar aplicaciones web con Servlets Java implica escribir interminables sentencias “println” para poder enviar datos en formato HTML al navegador, una tarea tediosa para los programadores, Luego aparecieron los JSP páginas que permitian mezclar HTML con código Java y, de esta manera mantener las ventajas de los Servlets.

El problema entonces, radicaba en que una página web con extenso código Java era un tanto difícil de comprender para quienes debían realizar la interfaz de usuario. Por estas razones fue necesario mejorar el proceso y calidad del desarrollo de software.

Mediante la aplicación del patrón de diseño MVC se consigue una mayor calidad en el desarrollo de aplicaciones y una alta reducción de los tiempos requeridos para su creación y mantenimiento. Pero, lamentablemente, el problema que surgía era cada grupo de desarrollo hacia las cosas de manera diferente, con lo cual se producía un alto costo a la hora de incorporar un desarrollador nuevo al proyecto o de realizar reformas y/o agregados.

Una vez que se mejoró la calidad en el desarrollo de software, la meta fue tratar de estandarizar el proceso, por este motivo, surge en el mercado informático varios Frameworks que pretenden alcanzar esto, este es el caso de Struts.

4.2 Patrón de diseño MVC

En este tipo de ámbito se intenta seguir algunas pautas para conseguir un desarrollo estructurado (o casi estructurado) de una aplicación. Para lograr este objetivo se trata de utilizar el patrón de

diseño MVC que consiste, básicamente, en dividir las aplicaciones en tres partes denominadas capas:

- Modelo
- Vista
- Controlador

El Modelo es la parte de la aplicación que se ocupa de manejar la lógica de negocio; es la capa que debe desarrollar el programador.

La Vista es la capa de presentación de la información; en el caso de las páginas web, serían el HTML, JSP y demás tecnologías para tratar los datos. En esta capa no se debería mezclar lógica de negocio.

El Controlador se encarga del flujo de la aplicación y asigna una respuesta a una petición. Es el que sabe quién tiene que hacer las cosas, pero no sabe cómo las hace.

En un proyecto web, cuando llega una petición al Controlador, éste dispara una acción del Modelo; éste le devuelve los resultados al Controlador, que redirecciona a la vista que corresponde.

4.3 ¿Qué es Struts Framework?

Es un Framework para el desarrollo de aplicaciones web bajo el paradigma MVC, la primera versión de Struts fue creado por Craig R. McClanahan y donado a la ASF (Apache Software Fundation) en el año 2000. Al ser Struts proyecto de código abierto (open source), está en constante evolución. Actualmente son muchísimos los colaboradores de todas partes del mundo que aportan código e ideas para futuras versiones del proyecto.

Struts Framework intenta normalizar el proceso de desarrollo web siguiendo pautas estándar, y trata de simplificar muchas de las tareas más comunes en este tipo de aplicaciones para lograr que el posterior mantenimiento se realice en el menor tiempo posible.

Actualmente, es uno de los frameworks MVC más usados, debido al tiempo que lleva en el mercado y a su facilidad de uso. Un programador que tenga poca o nada de experiencia podría empezar a utilizarlo en muy poco tiempo. Para obtener más información se puede consultar el sitio oficial del proyecto Struts.

www.struts.apache.org

4.4 ¿Para qué sirve?

Como todo Framework separa la capa de Vista, del Controlador (Servlet provisto por Struts) de la aplicación y del Modelo de negocios. El flujo de la aplicación se programa a través de un archivo XML, mientras que para la interfaz normalmente se utilizan páginas JSPs y un amplio conjunto de tags (Tag Libraries) predefinidos que provee el Framework. Por lo que Struts está diseñado para ser independiente de la Vista, y aunque JSP es la opción más usada y para la que trae mayor soporte de funcionalidad, podemos usar otros frameworks para implementar la capa de la Vista, entre ellos tenemos Velocity(un sistema similar a JSP pero basado en platillas), Cocoon(un framework de desarrollo web) y Stxx(una extensión de Struts que implementa Model2X).

Al separar las distintas capas de la aplicación, es posible trabajar paralelamente con profesionales especializados en cada una de ellas, con el fin de obtener mejores tiempos de desarrollo y una mayor calidad. De este modo Struts consigue que el programador se concentre sólo en el desarrollo de aplicaciones complejas, ayudándose con los elementos que ofrece el Frameworks.

4.5 Características de Struts

Struts nos va a ayudar mucho a desarrollar aplicaciones web, Algunas de las utilidades que provee son:

- Controlador proporcionado por el framework.
- Configuración del flujo de la aplicación desde un archivo .xml.
- Desarrollo en componentes de acuerdo con el paradigma MVC, separando la lógica del contenido.

- Utilidades para mapear JavaBean con formularios (mediante ActionForm) y cargar formulario con el contenido del bean.
- Internacionalización de la presentación. En base al lenguaje del navegador, Struts automáticamente puede mostrar el contenido apropiado.
- Herramienta para la validación de los datos de los formularios web.
- Manejo de excepciones, podemos definir manejadores de excepciones a nivel global o por acción, de esta forma, cuando ocurra una excepción podremos realizar acciones de contingencia, registrar lo ocurrido para depuración, etc.
- Subclases de Action, no tenemos que limitarnos a extender la clase Action para cada acción que deseemos, Struts provee varias subclases con ciertas funcionalidades.
- Soporte para DataSources, en el archivo de configuración podemos declarar uno o más elementos data-source para definir accesos a bases de datos y utilizarlos desde acciones o Servlets.
- Plugins, se puede definir plugins que se enchufen en la aplicación para realizar diversas tareas en la inicialización o terminación de la misma.
- Upload de archivos, provee un API sencilla para manejar casos en los que el usuario envía archivos al servidor como parte de un formulario.
- Reusabilidad de componentes de Vista usando Struts Tiles.

4.6 Desarrollo de aplicaciones con Struts

Sabiendo cómo desarrollar aplicaciones con Servlets y con JSP, veamos en qué se diferencia el desarrollo al utilizar Struts. Siguiendo con la temática del paradigma MVC, el desarrollo de aplicaciones web se reduce a escribir estos componentes.

Modelo: Objetos por lo general JavaBeans, aunque pueden ser objetos de cualquier tipo que contienen la información y la lógica de lo que queremos representar. Los objetos del modelo idealmente deben estar completamente desacoplados del entorno web. Esto incrementa las oportunidades de poder reutilizarlos en otros entornos. En concreto, no deberíamos depender de ninguna clase específica de Struts o Servlets.

Vista: La vista consiste en generar páginas HTML o crear componentes que las generen como páginas JSP. Es importante recordar que Struts no se limita a usar JSP como capa de presentación, ni siquiera tenemos que generar HTML como resultado, podríamos generar y devolver al usuario cualquier tipo de documento, si así lo quisiéramos.

Control: Escribir estos componentes consiste en desarrollar objetos Action, ActionForm y asociarlos entre sí en el archivo de configuración. Los objetos ActionForm no se limitan a replicar las propiedades del formulario al cual estarán, sino que también desarrollan su validación. Los objetos Action son los encargados de procesar los pedidos e indicarle a la Vista a mostrar como resultado.

4.7 Estructura de Struts

Struts se basa, principalmente en tres elementos fundamentales: ActionForm Bean, Action Bean y un par de archivos de configuración.

Los ActionForm Bean son clases que extienden ActionForm (org.apache.struts.action. ActionForm) y que implementan los métodos get y set para cada una de las properties de un formulario web.

Estos poseen dos métodos muy importantes para el desarrollo de una aplicación, el método reset, en el cual se configuran los valores de las properties; el método validate, en el que se desarrollan las validaciones de los datos definidos en el formulario. Ambos métodos son opcionales, pero su uso ayuda a mantener las tres capas bien separadas.

Tanto los ActionForm como los Action deben ser definidos en un archivo de configuración llamado struts-config.xml, como así también los mapeos de las URLs de nuestra aplicación, en cuanto al control de errores Struts ofrece un mecanismo para enviar y/o mostrar mensajes de error.

4.8 Funcionamiento de los Struts

Si hubiera que realizar una aplicación web que leyera datos de un formulario y, luego, los enviara al servidor para procesarlos, se necesitarían más archivos de lo normal al hacerlo con Struts, la creación de más archivos es el costo que se debe pagar por realizar una buena programación y mantener cada capa bien separada de otra.
Lo que hay que tener presente es que Struts nunca se llega a una página de la capa de presentación directamente, sino que se debe invocar una acción (Action) que debe estar mapeada en el archivo struts-config.xml.

Cuando se llega una petición desde una JSP, pasa primero por el Controlador (Servlet), el cual se encargará de leer el archivo struts-config.xml y de obtener los mapeos a la URL o al Action correspondiente. En el caso de mapear con un Action, el Controlador envía la información al que corresponda, y éste último accede a las properties del formulario web a través de los métodos de acceso gets definidos en el ActionForm, validando previamente esa información con el método validate.

Una vez realizadas las acciones y definidas las properties que se mostrarán en la JSP de respuesta (a través de los sets establecidos en el ActionForm) se vuelve a pasar por el Controlador, que otra vez, lee el archivo struts-config.xml para obtener la URL y realizar el forward a la JSP correspondiente. En la siguiente figura 4.1 se muestra el funcionamiento de los Struts.

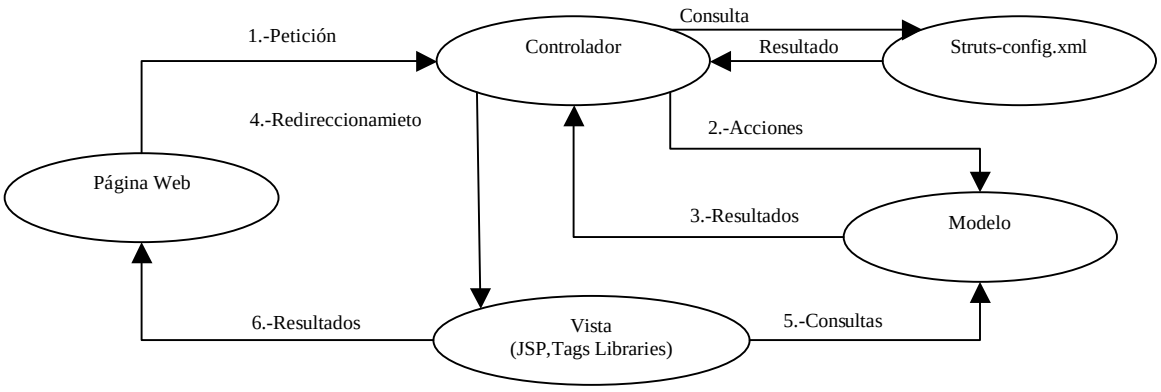


Figura 4.1: Funcionamiento de Struts

4.9 Interactuando con el usuario (ActionForms)

Usualmente vamos a desarrollar aplicaciones web que interactúen con usuarios, aunque también sería válido crear aplicaciones que simplemente muestren contenido (dinámico) que vaya cambiando con el correr del tiempo, en general queremos que el usuario pueda interactuar con la aplicación, de forma tal que podemos brindarle la información específica que necesita.

En la programación web con páginas HTML, el usuario tiene diversas formas de interactuar con la aplicación. Cuando necesitamos que el usuario ingrese datos, usamos los formularios. Al ser los formularios la herramienta fundamental para recibir información del usuario, las aplicaciones web trabajan con ellos intensamente, Struts provee funcionalidad para ayudarnos a simplificar la tarea de escribir formularios y, sobre todo, para mantener su estado entre pedidos.

Es muy común en sitios web tener formularios de registro donde se pide una gran cantidad de datos, como nombres, clave, confirmación de clave, dirección, teléfono etc. Supongamos (como suele suceder) que tenemos ciertas restricciones sobre ese formulario, el campo clave y confirmación de clave deben contener el mismo valor, el nombre de usuario no debe existir ya en nuestra base de datos, etc. Imaginemos que el usuario ingresa algunos datos y envía el formulario pulsando el botón correspondiente. Estos datos llegan al servidor y son recibidos. El servidor, al procesar los datos, encuentra que el campo de correo electrónico, que es una campo requerido, no contiene datos. Entonces agrega el mensaje del error correspondiente y devuelve el mismo formulario al usuario para que corrija el error.

Siguiendo estos pasos aquí descritos, todo parece estar correctamente; sin embargo el usuario recibirá el mensaje de error y el formulario todo en blanco, deberá entonces volver a completar todos los datos además de realizar la corrección, algo largo y tedioso. Por lo que debemos asegurarnos de devolver el formulario cargado con los datos que el usuario ya ingresó, y esto se llama mantenimiento del estado de los formularios. Struts, entre otras cosas, provee lo necesario para no tener que implementar nosotros la funcionalidad de completar los campos que el usuario ya había ingresado al devolver un formulario.

ActionForm representa un formulario en una página web. Las acciones en Struts (implementadas en la clase Action) rara vez interactúan directamente con HttpServletRequest o HttpServletResponse.

Implementar un ActionForm es sencillo, simplemente debemos crear un JavaBean que extienda la clase org.apache.struts.action.ActionForm y definir sus propiedades y métodos de acceso como cualquier otro JavaBean.

Una vez definido el ActionForm, debemos declararlo en el Struts-config y asociarlo con acciones. Al asociar un ActionForm con acciones, le estamos diciendo a Struts que cuando llegue un pedido para una acción asociada, intente completar los datos del ActionForm con los datos del pedido, y ese ActionForm es el que recibirá el método execute de la clase Action.

4.10 Configurando un ActionForm

Una vez que tenemos defenido un ActionForm, debemos asociarlo con determinadas acciones para que sea instanciado y cargado por Struts cuando dichas acciones sean accedidas. Los forms dentro del struts-config se defienen como una seria de tags (marcas) form-bean encerrados dentro de un tag form-beans, por ejemplo:

```
<form-beans>
        <form-bean ...>
        <form-bean ...>
        <form-bean ...>
</form-beans>
```

Cada elemento form-bean puede tener tres atributos:

- § className define la clase que ha de configurar al bean que estamos definiendo. (subclase de org.apache.struts.config.FormBeanConfig). Puede ser útil usar esta condifuración cuando tenemos varios beans iguales. En vez de configurarlos uno por uno en el struts-config, creamos una clase FormBeanConfig y se la pasamos a cada bean para que se configure, aunque por lo general es más práctico configurarlos en el struts-config.

- name es el nombre que le damos a este bean.
- type es el nombre de la clase que instanciará el bean.

Sólo falta asociar ActionForm con Action en el archivo de configuración struts-config a estos se le llama declarar una acción (más adelante se hablará de Actions en Struts, solo se comenta aquí para no perder la asociación entre Action y ActionForm) que use ActionForm. Esto se hace dentro del tag <action-mappings> </action-mappings> y se usa el tag <action ...> </action> por ejemplo:

```
<action-mappings>
        <action .....>
        </action>
</action-mappings>
```

4.11 Método reset de ActionForm

Dentro de ActionForm se define un método importante que es reset, que su encabezado es public void reset(ActionMapping map, HttpServletRequest req), este método es invocado por Struts antes de reponer las propiedades del Form, ¡este método no tiene sentido!. Dado un form con propiedades, se llama a un método que reinicialaza sus propiedades, volviéndolas a su valor por defecto, para luego establecerle los valores que vienen del cliente. Entonces ¿para qué volverlas a un valor por defecto si en definitiva se van a reescribir?, en la práctica hay un caso en el que este método es necesario.

Si tenemos un form guardado en sesión (esto es, declarado en la acción que lo utiliza con scope=”session” se hablara más adelante) y este formulario tiene algún checkbox (casilla de selección binaria), es necesario reestablecer el valor de las propiedades asociadas con los checkboxes. Esto se debe a que un checkbox, que tiene dos estados, sólo envía su estado cuando está marcado. Cuando se encuentra desmarcado, no envía nada al servidor. Bajo este funcionamiento, si el form es reutilizado y el valor de una propiedad booleana asociada con un checkbox es verdadera, nunca se va a cambiar a falso aunque la opción esté desmarcada en el navegador, simplemente porque no se recibirá información alguna sobre esa propiedad y mantendrá su estado.

4.12 Validación de ActionForm

Por lo general, al interactuar con el usuario, necesitamos validar lo que éste ingresa en los formularios. Algunos datos son requeridos y el usuario no puede completarlos, otros tienen un rango determinado que no puede excederse, otros deben pertenecer a cierto dominio, etc.
La clase ActionForm posee un método apropiado para validar los datos ingresados por el usuario su encabezado es public ActionErrors validate(ActionMapping map, HttpServletRequest req).

Este método puede ser invocado cada vez que se realiza un pedido. Debemos prever una implementación particular para cada uno de los ActionForms que creemos. ActionErrors es una clase que encapsula mensajes de error. Cada mensaje de error (una instancia de la clase ActionMessage) puede ser global o estar asociado con un campo en particular. Si no hay mensajes de error, podemos devolver un objeto ActionErrors vacío o bien null. Es conveniente devolver todos los errores juntos y no frenar al encontrar el primer error, si no, el usuario tendría que hacer las correcciones una por una.

Es muy simple agregar un mensaje de error a un ActionErrors, invocando el método:
Add(String property,ActionMessage message). En el primer parámetro es el nombre de la propiedad a la cual está asociada el mensaje de error. Si queremos un mensaje de error global usamos la constante ActionErrors.GLOBAL_MESSAGE.

Los objetos ActionMessage fueron diseñados para trabajar con internacionalización y externalización de cadenas de caracteres, a fin de no tener que embeber texto en el código. Como se dijo, el método validate puede ser invocado con cada pedido. La invocación o no del método es configurable para cada acción. Podemos desear que algunas acciones validen el contenido y otras que no, siempre refiriéndonos al mismo ActionForm. Para configurar las acciones con la validación, debemos establecer los valores de dos atributos en cada elemento action del struts-config, validate define esta acción y efectúa la validación del form (valores true/false), input define la página a donde redirigir al usuario cuando la validación falla (se explica input en Action).

4.13 Configurar Struts con MessageResources

Al respecto de textos externalizados, Struts necesita que definamos los archivos donde hallar estos textos (en varios idiomas), basta con agregar la siguiente línea en struts-config.xml, luego de los action-mappings.

<message-resources parameter=”MessageResources”/>

Esta línea indica que Struts debe buscar los mensajes en una archivo de nombre MessageResource.properties. Struts busca el archivo en las ubicaciones de class-path, por lo que una buena recomendación es ubicar este archivo en la carpeta WEB-INF/src, un error muy común es ubicar el archivo MessageResources junto con las clases en la carpeta WEB-INF/classes. Esta carptea está en el classpath, pero a menudo es borrada o regenerada, y junto con ella puede ir este archivo.

4.14 Alternativas de ActionForm

A medida que nuestra aplicación web va creciendo, también lo hace la cantidad de clases ActionForm. Si usamos la funcionalidad de Struts para validación (Struts Validator, un componente con funcionalidad para validar automáticamente nuestros formularios), entonces nuestras clases se convierten en objetos JavaBean, que son meramente un repositorio de propiedades con métodos acceso y que no aportan funcionalidad. En estos casos, es conveniente buscar una alternativa, en vez de tener que crear una clase con sus propiedades y acceso para cada formulario que necesitemos.

Si no aportamos funcionalidad a los form-bean, éstos terminan cumpliendo con el papel de un diccionario o mapa; una colección de campos con sus respectivos valores, Struts provee la clase DynaActionForm, que se comporta exactamente de esta manera, ahorrándonos el trabajo de escribir forms por doquier. Para usar este tipo de form-bean simplemente necesitamos definirlos en el archivo struts-config y declarar sus propiedades, por ejemplo:

```
<form-bean
        name=”datosForm” type=”org.apache.struts.action.DynaActionForm”>
        <form-property name=”nombre” type=”java.lang.String”/>
        <form-property name=”edad type=”java.lang.Integer”/>
        <form-property ...../>
</form-bean>
```

La desventaja que se tiene es que se pierde la validación de ActionForm. Si bien la clase DynaActionForm permite tipos primitivos en sus propiedades, si se intenta asignarles un valor nulo obtendremos una excepción, por lo que es conveniente usar siempre objetos.
Las opciones del elemento form-property son:

- § className: nombre de la clase que configura este elemento. Es un atributo que funciona igual en la definición de un form-bean. La superclase que deben extender las clases de definición de propiedades es org.apache.struts.config.FormPropertyConfig.
- § initial: valor inicial que tomará esta propiedad (representada por una cadena de texto). Si omitimos este valor, la propiedad será creada usando el constructor sin argumentos de la clase.
- § name: es el nombre de la propiedad.
- § size: es el número de elementos a crear si el tipo de esta propiedad es un arreglo y no se especificó un valor inicial de creación.
- § Type: es el nombre de la clase que instanciará el bean.

Los arreglos son necesarios cuando el usuario tiene que elegir una cantidad de ítem de una lista múltiple. Lo bueno de DynaActionForm es que no necesitamos modificar nada de código para acceder a ellos en las páginas JSP. Si estamos migrando viejos ActionForm a DynaActionForm, sí tenemos que modificar ligeramente la forma de acceder a las propiedades. El método para obtener los valores de los campos en un DynaActionForm es:

```
public Object get(String name)
```

El parámetro corresponde al nombre de la propiedad a acceder y el resultado es el valor de dicha propiedad. La ventaja principal de usar DynaActionForm es prescindir de escribir una clase completa para cada formulario web que necesitamos.

Sin embargo, este cambio no es gratis. Entre sus desventajas tenemos:

- El método get devuelve un Object, lo cual implica que debemos cachar el resultado, al ser cachado dinámicamente, no se le puede predecir en tiempo de compilación, y si cometimos un error, nos enteraremos en tiempo de ejecución.
- Lo mismo ocurre con el nombre del campo. Podemos equivocarnos en el nombre y el compilador interpretará una llamada a función totalmente válida, sólamente al ejecutar nos enteraremos del error.

Pese a estas dificultades, utilizar DynaActionForm es muy conveniente cuando el formulario precisa validaciones simples que pueden ser implementadas mediante el uso del Struts Validator.

4.15 Actions en Struts

Se vio que un ActionForm es como un firewall entre el pedido que hace el cliente y el servidor, no deja pasar información indeseada, y la que pasa debe ser validada. La clase Action hace las veces de interfaces entre el pedido http que efectúa el cliente y las clases que han de efectuar la lógica subyacente. Se podría realizar todas las operaciones dentro del código de Action, pero esto implica mezclar responsabilidades, porque la lógica estará altamente atada al entorno web (y en particular a Struts) y no podemos reutilizar componentes de software.

Una buena práctica de programación web es programar los Action como adaptadores entre objetos propios del entorno web y de Struts (HttpServletRequest,ActionForward, etc.) y los JavaBeans. Deben recibir la información proveniente de las páginas, validarla, procesarla y enviársela a los JavaBeans para que hagan lo suyo. Finalmente envían el resultado a la vista, en donde ésta consultará a los beans su estado para mostrar al cliente una página web.

Struts reutiliza las instancias de las clases Action que definamos para manejar pedidos simultáneos, por eso debemos programar las Actions de forma segura a la ejecución concurrente, las variables (tanto de instancia como de clase) no deben usarse para mantener información del

estado actual del pedido, y el acceso a recursos externos debe estar sincronizado si dicho recurso lo requiere.

En general, es recomendable como buena práctica no usar directamente ningún tipo de variables (ni de instancia de clase) globales a la clase, y declarar todas las variables que serán usadas como locales dentro del método execute.

4.16 Configurar un Action

Estos son algunos atributos que tenemos para configurar un Action dentro del elementos action en struts-config:

- atribute: es el nombre que queramos darle al ActionForm para ser guardado en el pedido o en la sesión. Si no especificamos este atributo, se usará el nombre del ActionForm tal como aparece en el atributo name.
- forward: es la dirección del recurso (puede ser otro action, un JSP, un Servlet o cualquier otro recurso que pueda manejar el pedido) que se encargará de procesar este pedido.
- include: es la dirección del recurso que se ha de incluir en la respuesta que genera este pedido. La diferencia de funcionalidad entre estos dos últimos atributos es en cierta forma similar a los distintos tipos de inclusión que existen en las páginas JSP, usando forward estamos creando una acción que envía el pedido al recurso que especifiquemos, y este recurso se encargará de procesarlo y generar la respuesta, en cambio, si usamos include, estamos incluyendo en nuestra respuesta el resultado que genere el recurso especificado, pero no estamos derivando completamente el pedido a dicho recurso.
- type: nombre completo de la subclase de org.apache.struts.action.Action que procerará los pedidos que este elemento describe, en cada elemento action los atributos forward, include y type son excluyentes entre sí, sólo puede haber uno de ellos.
- input: dirección del recurso a invocar si el ActionForm asociado al elemento (de existir) reporta errores de validación.
- name: nombre del form-bean asociado (tal como fue definido dentro del elemento form-beans), si hubiere.

- path: dirección que este elemento procesará.
- parameter: parámetro de uso general, utilizando para pasar información extra al action, Por ejemplo podemos crear una clase llamada BDAction y que tome de este parámetro el nombre de la conexión a la base de datos, luego las acciones que requieran conexión a una base simplemente extetenderán BDAction.
- scope: contexto en donde reside el form-bean: request o session.
- unknown: marcador (sólo puede tomar valores true o false) usado para configurar la acción ejecutada por defecto. Si recibe un pedido que no está asociado con ninguna acción, el usuario recibirá un error de parte de Struts. Si declaramos una acción con el atributo unknown verdadero, será la acción por defecto en estos casos, esto es útil para mostrar al usuario un mensaje de error agradable o redirigirlo a una página de navegación. Sólo puede haber una acción con este atributo con valor true.
- validate: marcador (sólo toma valore true o false) usado para indicar si el form-bean debe ser valido antes de ser enviado al objeto Action y éste ejecutado, por defecto este atributo lleva valor true.

En todos los atributos en lo que hemos de especificar direcciones, éstas deben empezar con una barra (/), veamos unos ejemplos de configuraciones. En este se ocupa un elemento simple que asocia una acción directamente con una página JSP. Este tipo de configuración es útil para esconder las páginas JSP detrás de acciones, una buena práctica por temas de seguridad.

```
<action
        path="/formulario"
        forward="/prueba/formulario.jsp">
</action>
```

Similar a la anterior pero usando include, éste es de utilidad para pies de página o elementos que se repiten mucho, de nuevo escondiendo las páginas JSP de la interacción directa con el usuario.

```
<action
        path="/formulario"
        include="/prueba/formulario.jsp">
</action>
```

En éste declaramos que el bean no debe ser valido, sin embargo, estamos definiendo la página donde debe reenviarse en caso de que la validación falle, si bien la página no será accedida nunca, esto es sintácticamente correcta.

```
<action
        path="/formulario"
        name="forma"
        validate="false"
        input="forma.jsp"
        type="datos.DatosAction">
</action>
```

Este ejemplo sirve para manejar todos los pedidos que Struts reciba y no tengan asociados un elemento action, y redirigirlos a una página JSP donde se muestre un mensaje de error.

```
<action
        path="/"
        unknown="true"
        forward="/salidas/error.jsp">
</action>
```

4.17 ActionForward

Todo elemento action puede tener opcionalmente un conjunto de elementos que describan ActionForward, los cuales pueden ser accedidos desde el Action para comunicarle al controlador hacia dónde debe seguir el flujo de la acción. Un ActionForward es una clase que contiene información sobre hacia dónde debe enviar el pedido el controlador como resultado de una acción. Los atributos para definir un ActionForward son:

- name: nombre que queremos darle al ActionForward.
- path: dirección de destino.
- redirect: si este atributo toma valor true, entonces un nuevo pedido será emitido para acceder al destino de este ActionForward.

Al emitirse un nuevo pedido para el destino del ActionForward, estamos perdiendo todos los datos que el pedido actual tenía. Datos de formulario, datos guardaros por un Action, etc., se

pierden. Esto puede ser beneficioso o perjudicial. Si necesitamos algo de esta información para poder construir la Vista, obtendremos un error, o bien, la Vista estará incompleta. El hecho de emitir un nuevo pedido es útil para cuando el usuario efectúa una acción con efectos colaterales y no queremos que la repita por error. Un ejemplo sería que transfirió una cantidad de dinero entre cuentas de un banco, en este caso, la acción fue transferir una cantidad x de dinero entre cuenta A y cuenta B, que como resultado final tuvo un ActionForward a una página que confirmó la operación. Si el usuario, accidentalmente o no, presionara el botón de recargar página en el navegador, estaría reenviando el pedido de transferir nuevamente el efectivo, algo que dudosamente sea lo que él quiera. Si el ActionForward de la operación de transferencia hacia la página que muestra que se efectuó correctamente emite un nuevo pedido, entonces el último pedido será ir hasta la página de confirmación y no habría problemas si el usuario recargara la página. Ejemplo de ActionForward:

```
<forward name=”pedido” path=”/logica/pedidos.jsp” />
<forward name=”confirmacion” path=”/logica/confirma.jsp” redirect=”true” />
```

4.18 DispatchAction

DispatchAction llama al método según el valor de un parámetro del pedido, cuyo nombre es especificado por la propiedad parameter en el elemento action de strug-config, el ejemplo siguiente muestra su utilización.

Declaramos un acción en struts-config

```
<action
        path=”/cuentas”
        type=”entradas.accionCuentas”
        name=”formulario”
        scope=”session”
        parameter=”metodo”/>
```

Se define la clase

```
public class accionCuentas extends DispatchAction {
  public ActionForward cambiar(ActionMapping mapping, ActionForm  form,
      HttpServletRequest request, HttpServletResponse response)
      throws Exception {
         //acciones a realizar
  }
```

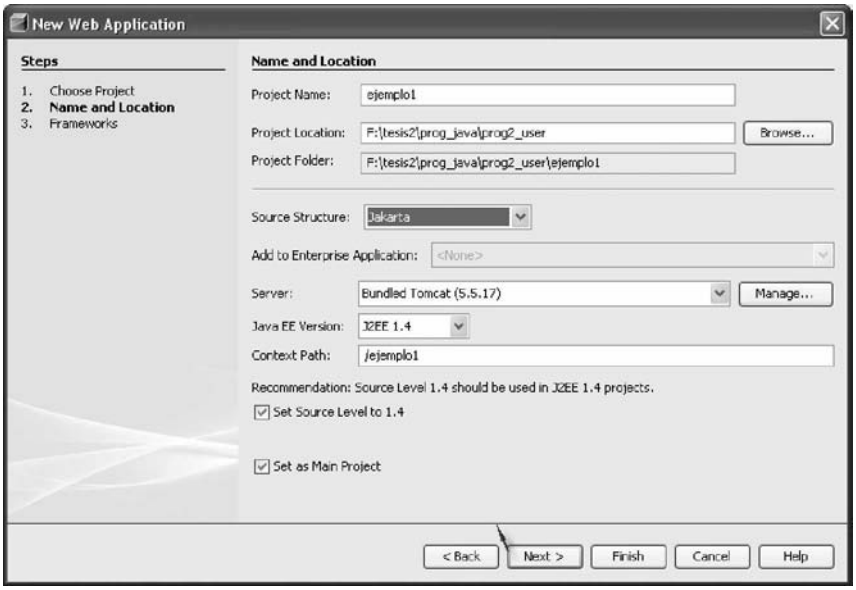
```
  public ActionForward mover(ActionMapping mapping, ActionForm  form,
      HttpServletRequest request, HttpServletResponse response)
      throws Exception {
         //acciones a realizar
       }
}
```

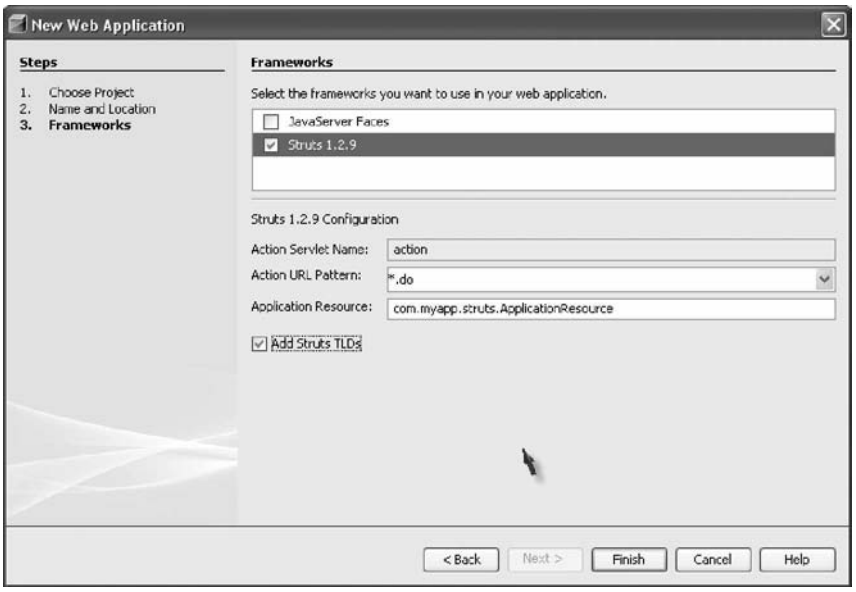
Supongamos que lo solicitamos de un JSP

```
<a href=”cuentas.do?metodo=cambiar ”>Cambiar</a>
<a href=”cuentas.do?metodo=mover ”>Mover</a>
```

4.19 Ejemplo Sencillo de un Struts

Con todo esto ya podemos realizar un ejemplo sencillo usando Struts, como primer paso abrimos nuestro NetBeans, nos vamos a File->New Proyect, escogemos en categoría como Web y en proyectos Web Application le damos next y seguimos la configuración de las imágenes siguientes:





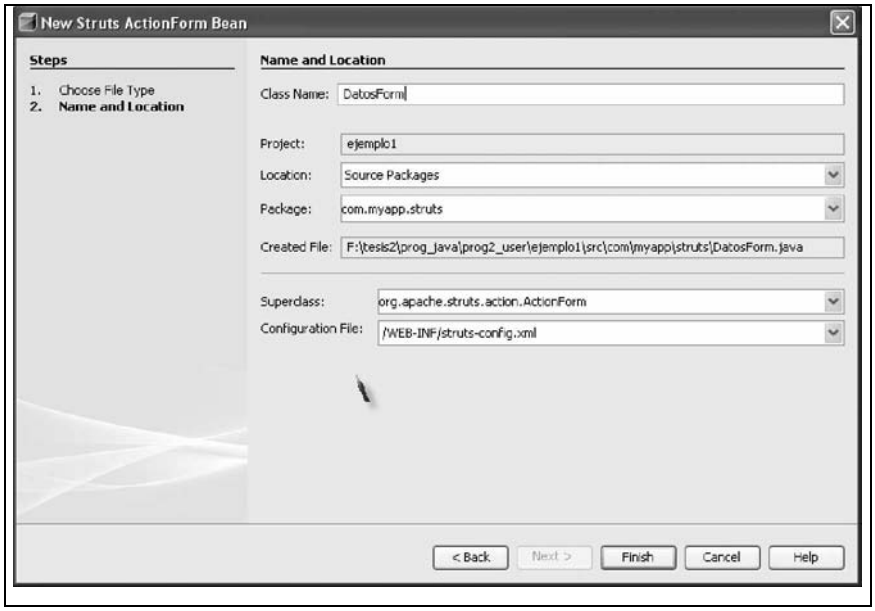
Recordemos que la configuración de una aplicación web se realiza mediante el archivo web.xml, residente en la carpeta WEB-INF, los elementos importantes a tratar aquí son los siguientes:

<servlet> <servlet-name>action</servlet-name> <servlet-class> org.apache.struts.action.ActionServlet</servlet-class> <init-param> <param-name>config</param-name> <param-value>/WEB-INF/struts-config.xml</param-value> </init-param> <init-param> <param-name>debug</param-name> <param-value>2</param-value> </init-param> <init-param> <param-name>detail</param-name> <param-value>2</param-value> </init-param> <load-on-startup>2</load-on-startup> </servlet>	à Definimos el Servlet de Struts à Indicamos a Struts dónde buscar el archivo de configuración à Nivel de detalle en los mensajes de depuración que escribe struts con el método ServletContext.log à Nivel de detalle en los mensajes de depuración que escribe Struts en System.out. à Indica que el Servlet a de cargarse al inicializar Tomcat
<servlet-mapping> <servlet-name>action</servlet-name> <url-pattern>*.do</url-pattern> </servlet-mapping>	Asociamos el Servlet de Struts a todos los pedidos que terminen con .do
<jsp-config> <taglib> <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri> <taglib-location>/WEB-INF/struts-bean.tld</taglib-location> </taglib> <taglib> <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri> <taglib-location>/WEB-INF/struts-html.tld</taglib-location> </taglib> <taglib> <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>	Descriptores de la librería de tags de Struts.

```
        <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>/WEB-INF/struts-nested.tld</taglib-uri>
        <taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
        <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
    </taglib>
</jsp-config>
```

Se define un Servlet que es el controlador de Struts y se asocia todo pedido que termine con .do a ese Servlet. Tradicionalmente, las aplicaciones Struts asocian esta extensión, pero podemos definir cualquier tipo que queramos. Notemos también que dicho Servlet toma un parámetro. Este parámetro es la ubicación del archivo de configuración de Struts, que contiene toda la información necesaria para el funcionamiento de la aplicación (struts-config.xml), finalmente están las declaraciones de taglibs. Struts provee una serie de tags muy útiles para usar en las páginas JSP (la Vista, se explicarán en el apartado de Vista), para poderlas usar por eso se declararon en el descriptor web.

Desarrollaremos para nuestro ejemplo, un formulario donde el usuario debe ingresar nombre, edad y si es fumador o no, nos vamos File à New File, en Categories señalamos Web, File Types escogemos Struts ActionForm Bean, y seguimos la configuración de la siguiente imagen



Borramos el código que nos proporciona Netbeans e introducimos el siguiente:

```
 1 package com.myapp.struts;
 2 import javax.servlet.http.HttpServletRequest;
 3 import org.apache.struts.action.ActionErrors;
 4 import org.apache.struts.action.ActionMapping;
 5 import org.apache.struts.action.ActionMessage;
 6 public class DatosForm extends org.apache.struts.action.ActionForm {
 7     private String nombre;
 8     private Integer edad;
 9     private boolean fumador;
10     public String getNombre(){
11         return nombre;
12     }
13     public void setNombre(String nombre){
14         this.nombre=nombre;
15     }
16     public Integer getEdad(){
17         return edad;
18     }
19     public void setEdad(Integer edad){
20         this.edad=edad;
21     }
22     public boolean isFumador(){
23         return fumador;
24     }
25     public void setFumador(boolean fumador){
26         this.fumador=fumador;
27     }
28     /*sobreescribimos el método reset para restablecer el valor de
29      *la propiedad fumador*/
30     public void reset(ActionMapping map,HttpServletRequest req){
31        fumador=false;
32     }
33     public DatosForm() {
34         super();
35     }
36
37     public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
38         ActionErrors errors = new ActionErrors();
39         if(nombre==null || nombre.trim().equals("")){
40             errors.add("nombre",new ActionMessage("falta ingresar nombre",false));
41         }
42         if(edad==null){
43             errors.add("edad",new ActionMessage(
44                     "falta ingresar edad",false));
45         }
46         else{
47             if(fumador && edad.intValue()<18){
48                 errors.add("fumador",new ActionMessage(
49                         "Pequeño fumador!",false));
50             }
51         }
52         return errors;
53     }
54 }
```

Nuestro ActionForm no es más que un JavaBean que extiende la clase ActionForm de Struts, se ocupa un Integer para representar la edad y un boolean para indicar su status de fumador. ¿Pero por qué Integer y no el primitivo int?. Recordemos que toda la comunicación cliente-servidor se realiza mediante el protocolo HTTP, que en definitiva transmite cadenas de caracteres, Struts recibe los datos e intenta hacer la conversión de tipos correspondientes entre los valores del pedido y las propiedades del ActionForm. Si en vez de Integer hubiésemos puesto Vector, por ejemplo, al enviar el formulario desde el navegador obtendríamos un error de tipo, en el campo de texto que le brindamos al usuario para que ingrese la edad, estamos esperando un número que represente una edad, pero el usuario puede ingresar cualquier texto. Si el usuario ingresa una cadena que no pueda ser transformada a un número, Struts devuelve el valor por defecto del objeto, en este caso, un número con valor cero. Esto funciona tanto para el primitivo int como para el objeto Integer, pero no podremos discernir si el usuario no ingresó valor o ingresó un valor correcto, o realmente ingreso el número 0.

Si usamos como tipo de datos el primitivo int, no tendríamos forma de saberlo, en cambio, utilizando Integer podemos indicarlo a Struts que, cuando no pueda realizar las conversiones de tipo, establezca null como valor de los objetos en lugar de su valor por defecto. De esta forma, si recibimos null como valor, podremos aseverar que hubo un error de conversión de tipo y que el usuario no ingresó un numero válido. Esta configuración se realiza asignando el parámetro de inicialización convertNull en el archivo web.xml, dentro de <servlet> .... </servlet> de la siguiente manera:

```
<init-param>
        <param-name>convertNull</param-name>
        <param-value>true</param-value>
</init-param>
```

Como se mencionó en ActionForm sólo falta asociarlo con una determinada acción que se hace dentro de struts-config a través del tag form-bean, pero esto ya lo hizo por nosotros Netbeans, y lo podemos constatar al revisar el archivo struts-config y tenemos que tener las siguientes líneas:

```
<form-beans>
    <form-bean name="DatosForm" type="com.myapp.struts.DatosForm"/>
</form-beans>
```

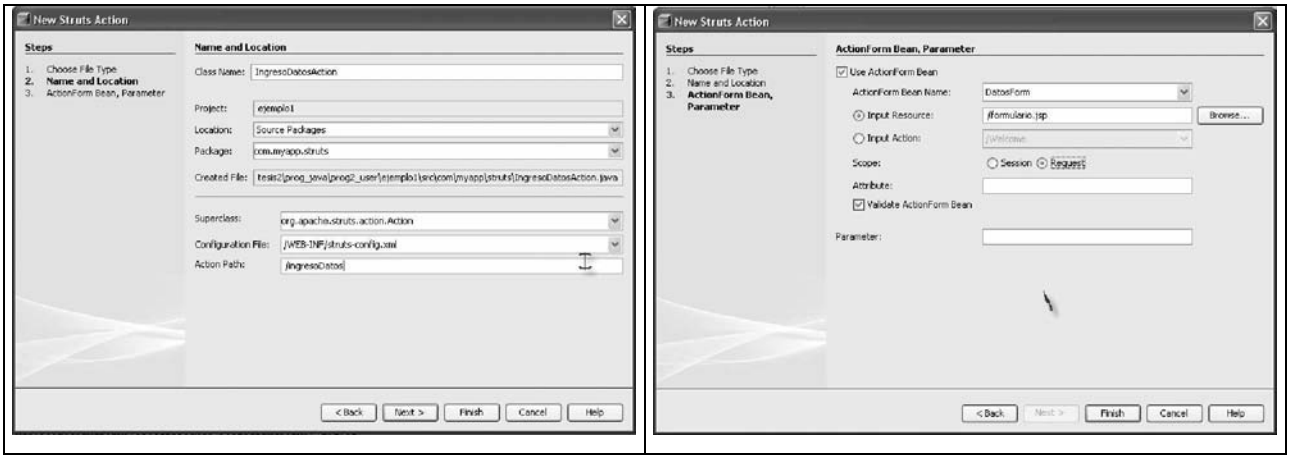
Ahora hagamos el punto de entrada de nuestra aplicación nos vamos a File à New File y escogemos que queremos un JSP, al cual lo llamaremos formulario, le dejamos los valores predeterminados que tiene, y le quitamos el código que da Netbeans y le ponemos lo siguiente:

```
1 <%@page contentType="text/html"%>
2 <%@page pageEncoding="UTF-8"%>
3 <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
4 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
5 "http://www.w3.org/TR/html4/loose.dtd">
6
7 <%-- Este formulario se enviara a la acción de ingresoDatos--%>
8 <html:form action="ingresoDatos">
9   Nombre: <html:text property="nombre" />
10   <%-- El mensaje de error si algo falla respecto al nombre --%>
11   <b><html:errors property="nombre" /></b>
12   <br/>
13   Edad: <html:text property="edad" />
14   <%-- El mensaje de error si algo falla respecto la edad --%>
15   <b><html:errors property="edad" /></b>
16   <br/>
17   fumador: <html:checkbox property="fumador" />
18   <%-- El mensaje de error si algo falla respecto ala condicion de fumar --%>
19   <b><html:errors property="fumador" /></b>
20   <br/>
21   <br/>
22   <html:submit/>
23 </html:form>
```

Hacemos lo mismo para crear la página de salida, la que le pondremos como nombre datos, los tags que ocupamos de Struts se explicarán en la parte de Vista, por el momento diremos que los tags html y bean nos sirven, para crear páginas html más fácilmente y para el manejo de JavaBeans respectivamente, solo muestra los datos introducidos en el formulario, el código es el siguiente:

```
1 <%@page contentType="text/html"%>
2 <%@page pageEncoding="UTF-8"%>
3 <%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
4
5 <bean:write name="DatosForm" property="nombre" /><br/>
6 <bean:write name="DatosForm" property="edad" /><br/>
7 <bean:write name="DatosForm" property="fumador" /><br/>
```

Ahora crearemos el Action para ActionForm creado, el cual le llamaremos IngresoDatosAction, nos vamos a File à New File, en Categories escogemos Web, y Files Type, Struts Action, le damos los siguientes parámetros como se muestran en las imágenes:



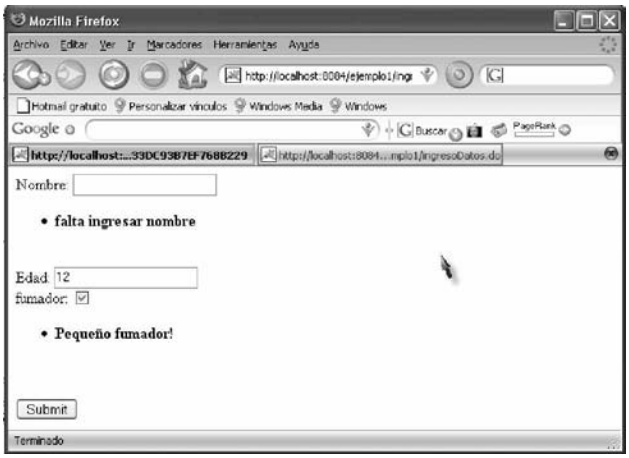
Con estos parámetros Netbeans llena los atributos de action en el archivo de struts-config, pero nosotros ocuparemos ActionForward para cambiar el flujo de la acción, por lo que el archivo struts-config debe quedar de la siguiente forma:

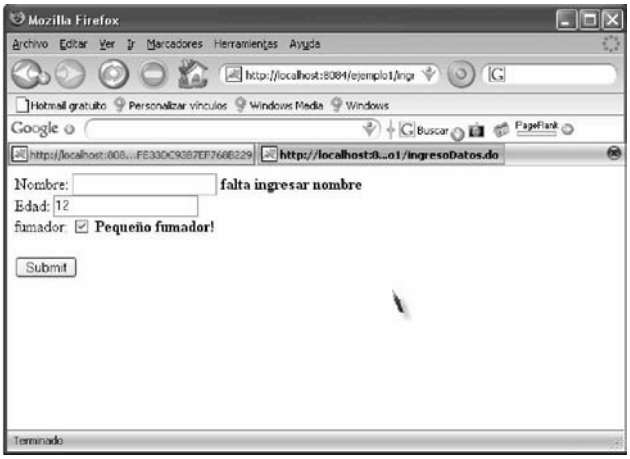
```
<action input="/formulario.jsp"
            name="DatosForm"
             path="/ingresoDatos"
             scope="request"
             type="com.myapp.struts.NewStrutsAction">
          <forward name="salida" path="/datos.jsp" />
</action>
```

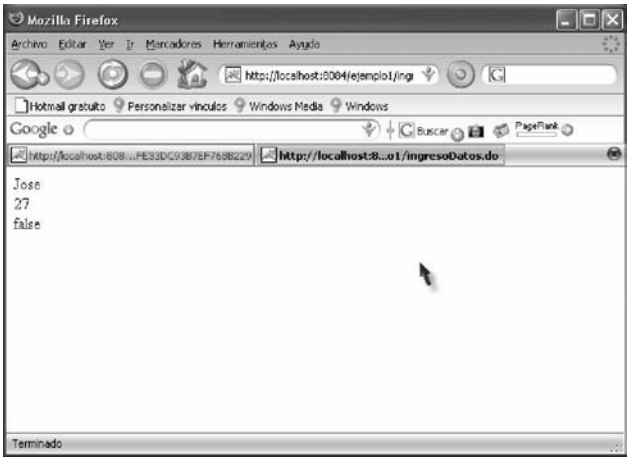
Típicamente sobrescribimos el método execute, que es el método llamado por Struts, en este ejemplo informamos al controlador que devuelva la vista de nombre salida. Ahora nos vamos a IngresoDatosAction.java y le ponemos el siguiente código:

```
1 package com.myapp.struts;
2 import javax.servlet.http.HttpServletRequest;
3 import javax.servlet.http.HttpServletResponse;
4 import org.apache.struts.action.Action;
5 import org.apache.struts.action.ActionForm;
6 import org.apache.struts.action.ActionMapping;
7 import org.apache.struts.action.ActionForward;
8 public class IngresoDatosAction extends Action {
9    public ActionForward execute(ActionMapping mapping, ActionForm  form,
10         HttpServletRequest request, HttpServletResponse response)
11         throws Exception {
12       /*No hacemos nada aqui, simplemente redirigimos a la pagina
13        *para la salida*/
14       return mapping.findForward("salida");
15    }
16 }
```

Sólo falta comentar que Netbeans proporciona un archivo donde configura los MessageResources, éste lo crea automáticamente con nuestro proyecto y lo llama ApplicationResources.properties, esto sólo cambia un poco nuestro ejemplo porque da un salto de línea y le agrega una viñeta a los mensajes de falta de ingreso de nombre o edad, podemos probar dejando el archivo struts-config con este archivo o podemos poner en el elemento <message-resources parameter=”MessageResources”>, a continuación se muestra las dos vistas de los formularios y la salida de éstos cuando se reciben los valores introducidos en el formulario







4.20 DownloadAction en Struts

Si se quiere manejar la descarga de archivos, la clase DownloadAction brinda esta funcionalidad, usar Struts en lugar de crear directamente un enlace al archivo que se descargará, nos sirve para no revelarle al usuario la ubicación del archivo que se descargará y para poder realizar comprobaciones (típicamente permisos), antes de permitir la descarga, además del hecho de que un enlace es estático y, usando este enfoque, la ubicación del archivo (incluso el contenido mismo del archivo) se puede obtener dinámicamente.

DownloadAction es una clase abstracta; nuestra subclase debe implementar el método abstracto getStreamInfo y, opcionalmente, getBufferSize, el primer método abstracto devuelve un objeto con la información sobre los datos a transferir, este objeto es de tipo DownloadAction.StreamInfo que es una interfaz. Los dos métodos que declara esta interfaz son:

- String getContentType() le informamos al navegador el tipo de archivo que estamos enviando.
- InputStream getInputStream() se define la fuente de los datos a enviar.

La clase DownloadAction provee dos clases estáticas que implementan esta interfaz, listas para ser usadas, si sólo nos fijamos de sus constructores que es lo único que se necesita para poderlas utilizar:

- DownloadAction.FileStreamInfo(String contentType,java.io.File file)
- DownloadAction.ResourceStreamInfo(String contentType, ServletContext context, String path)

En el primer caso proveemos el contentType y luego, un objeto File con el archivo que se descargará. Esto es útil para archivos fijos y no variables, ya que debemos proveer una dirección absoluta. En el segundo caso, además del contentType, debemos proveer el contexto del servlet y una dirección relativa. Este caso es más útil para devolver recursos de la aplicación web, ya que estamos limitados en las direcciones a usar.

Y por último, getBufferSize simplemente devuelve un entero con el tamaño del buffer que será usado por el Servlet al transferir la información del cliente.

4.21 LocaleAction

Esta acción nos permite cambiar el Locale del usuario, por defecto, Struts configura un Locale basándose en propiedades enviadas por el navegador. En muchos casos, el usuario no configuró su navegador para elegir su idioma o está en otra máquina, con esta acción podemos cambiar el objeto Locale que Struts guarda internamente, y si nuestra aplicación soporta internacionalización, cambiaremos el idioma en el que el usuario navega.

Esta acción se define normalmente en el struts-config. Al ejecutarse, busca en el pedido los parámetros language, country y page. Los valores de lenguaje y country definen el nuevo Locale, el parámetro page define la página a donde dirigirse luego del cambio de Locale. Si el parámetro page no existiera, se redirecciona el ActionForward de nombre success. Dado este funcionamiento, es altamente recomendable definir un ActionForward de salida con ese nombre si pensamos usar esta acción, aunque siempre pasemos el parámetro page, nos resguarda de posibles errores al hacer cambios o adaptaciones.

4.22 Servlet Filter

Son componentes de software que asociados a direcciones web, nos permiten interceptar tanto el pedido como la respuesta al pedido para realizar acciones acordes. Por ejemplo, un filtro no crea la respuesta al pedido sino que simplemente la modifica o realiza operaciones sobre ella o el pedido que la generó, son útiles para realizar tareas recurrentes, como pueden ser:

- Seguridad: un filtro revisa si el pedido está autorizado y, en caso negativo, lo bloquea o redirige.
- Auditoría: un filtro que lleve un registro de las veces que determinado recurso fue accedido puede tomar el tiempo que tardó en procesar, etc.
- Conversión: un filtro que transforme una respuesta genérica a un formato concreto.

- § Encriptación: un filtro que encripte la información que es recibida o enviada al cliente.
- § Comprensión: un filtro que comprima la información enviada al cliente para ahorrar ancho de banda.
- § Disparador: un filtro que, al accederse determinado recurso, dispare un evento.

Un filtro puede estar asociado con cero o más pedidos y un pedido puede tener asociados cero o más filtros, creando una cadena de filtros que se ejecutan antes de llamarse al Servlet, JSP o Struts. El proceso de crear filtros puede separarse en dos partes:

1. Desarrollar el filtro y las clases necesarias.
2. Asociar el filtro a los recursos.

La configuración de los filtros se realiza en el archivo web.xml, como se muestra en la tabla siguiente:

<filter> <filter-name>Primer filtro</filter-name> <filter-class> paquete.claseImplementa</filter-class> <init-param> <param-name>parametro1</param-name> <param-value>logica </param-value> </init-param> <init-param> <param-name>parametro2</param-name> <param-value>logica2</param-value> </init-param> <init-param> <param-name>detail</param-name> <param-value>2</param-value> </init-param> </filter>	à Declaramos instancia del filtro à Nombre del filtro. à Clase a instanciar que implementa la interfaz javax.Servlet.Filter. à Parámetro que será pasado al filtro al inicializarse à Parámetro que será pasado al filtro al inicializarse

Los filtros, para que tengan funcionalidad deben ser asociados como se dijo a Servlets, JSPs o Struts, esto se logra de la siguiente manera en el mismo archivo web.xml :

<filter-mapping> <filter-name>Primer Filtro</filter-name> <servlet-name>nombre del Servlet</Servlet-name> .	à Asociación entre un filtro o mas recursos . à Nombre del filtro. à Asociar un Servlet

. <url-pattern>recurso a asociar</url-pattern> . . <dispatcher>contexto</dispatcher> . . </filter-mapping>	à Asociar un conjunto de recursos à Otros contextos

También podemos configurar un filtro para que se ejecute en otros contextos. El elemento dispatcher indica a qué tipo de pedidos el filtro ejecutará, sus valores posibles son:

- REQUEST: el pedido fue efectuado directamente por el cliente (valor por defecto).
- FORWARD: el pedido fue redirigido desde otro recurso.
- INCLUDE: el pedido es resultado de una inclusión desde otro recurso.
- ERROR: el pedido es resultado de una redirección al haber un error.

Como se mencionó, para desarrollar un filtro sólo se debe implementar la interfaz javax.Servlet.Filter, la cual tiene tres métodos que son:

- void init(FilterConfig filterConfig) throws ServletException
- void destroy()
- public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException

El método init es llamado sólo una vez cuando se inicia el servidor y se instancia la clase, aquí es donde se pueden obtener los recursos necesarios para el filtro, a través de interfaz FilterConfig y de sus métodos:

- String getFilterName(): devuelve el nombre del filtro definido en el descriptor.
- String getInitParameter(String param): el nombre del parámetro definido en el descriptor de archivo, si no existe devuelve null.
- Enumeration getInitParameterNames(): devuelve una enumeración con todos los nombres de los parámetros que el filtro tiene.

En el método doFilter hacemos la lógica de lo que hace el filtro, y en el método destroy cerramos todos los recursos que utilizamos, ya que se llama este método cuando el servidor lo desecha, generalmente cuando el contexto es finalizado.

4.23 Segundo Ejemplo con Struts

Ahora ya disponemos de más herramientas para hacer una aplicación más completa, en este ejemplo se creara una aplicación web donde los alumnos podrán subir sus tareas a un servidor, los datos que serán necesarios serán guardados en una base de datos, sus datos que nos proporcionarán son:

- Matricula
- Nombre
- Correo
- Password

Una vez obtenidos, la base de datos (tareas) tiene un campo donde guarda el nombre del fólder en que se guardan sus tareas, el cual se llamará igual al nombre del alumno. Una vez registrado y que necesite subir sus tareas al servidor (Tomcat), tendrá que autentificarse el cual con sólo su número de matrícula y password que él mismo proporcionó; Tiene acceso para subir sus tareas, esta aplicación web tiene la funcionalidad de que si no proporciona correctamente sus datos se le avisa qué datos son incorrectos, no podrá seguir hasta que sean válidos dichos datos. Incluso, si él quiere acceder al recurso para poder subir sus datos sin estar autentificado, la misma aplicación lo manda a que se autentifique logrando así una seguridad de que sólo personas registradas puedan subir sus datos.

Para cumplir con estos requerimientos las características que usaremos de Struts son:

- ActionForms: en forma estáticos.
- Acciones: ActionForward y DispatchAction
- Filtros: utilizamos Servlet Filter

- Struts Tags: para ahorrar código
- Servicios: conexión con la BD a través de JDBC

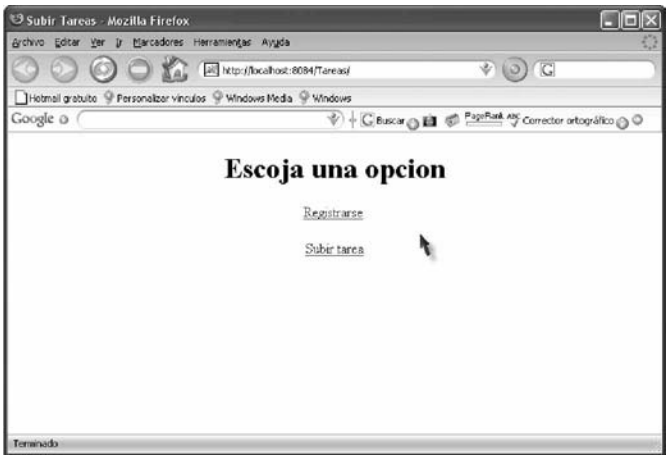
Empecemos con los servicios, ya que se necesita estar interactuando con la base de datos para registrar y consultar la información se creó una clase que hace esas funciones y guarda los datos en sus campos de dicha clase, la clase se llama Conexión y se encuentra en el paquete control.

```
1 package control;
 2 import java.io.*;
 3 import java.sql.*;
 4 import java.util.*;
 5 public class Conexion {
 6    private Connection conexion;
 7    private Statement instruccion;
 8    private String matricula;
 9    private String folder;
10     private String password;
11      /** Creates a new instance of Conexion */
12     public Conexion()throws Exception{
13        //cargando la clase controladora
14        Class.forName("com.mysql.jdbc.Driver");
15        //establecer la conexion con la BD
16        conexion=DriverManager.getConnection("jdbc:mysql://localhost:3306/tareas",
17                "root","mike97");
18     }
19     public Conexion(String matricula) throws Exception{
20        //cargando la clase controladora
21        Class.forName("com.mysql.jdbc.Driver");
22        //establecer la conexion con la BD tareas
23        conexion=DriverManager.getConnection("jdbc:mysql://localhost:3306/tareas",
24                "root","mike97");
25        //creando Statement para la consulta BD
26        instruccion=conexion.createStatement();
27        ResultSet resultados=instruccion.executeQuery("SELECT matricula,password,folder FROM alumnos"+
28                " where matricula=" + matricula );
29        while(resultados.next()){
30            this.matricula=resultados.getString(1);
31            this.password=resultados.getString(2);
32            this.folder=resultados.getString(3);
33        }
34         try {
35           instruccion.close();
36           conexion.close();
37        }
38        // procesar posible excepcion SQLException en operación de cierre
39        catch ( SQLException excepcionSQL ) {
40           excepcionSQL.printStackTrace();
41        }
42     }
43     public Connection getConexion(){
44         return conexion;
45     }
```

```
46
47    public String getmatricula(){
48       return matricula;
49    }
50    public String getpassword(){
51       return password;
52    }
53    public String getfolder(){
54       return folder;
55    }
56    public void cerrar(){
57       // tratar de cerrar la conexión a la base de datos
58      try {
59         instruccion.close();
60         conexion.close();
61      }
62      // procesar posible excepcion SQLException en operación de cierre
63      catch ( SQLException excepcionSQL ) {
64         excepcionSQL.printStackTrace();
65      }
66    }
67 }
```

El código de pantalla de inicio es un JSP (inicio.jsp), y sólo muestra dos enlaces, uno para registrarse y el otro para subir la tarea.

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 4   "http://www.w3.org/TR/html4/loose.dtd">
 5 <html>
 6    <head>
 7       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 8       <title>Subir Tareas</title>
 9    </head>
10    <body>
11       <center>
12          <h1>Escoja una opcion</h1>
13          <a href="./registro.jsp">Registrarse</a>
14          <br/>
15          <br/>
16          <a href="./login.jsp"> Subir tarea</a>
17       </center>
18    </body>
19 </html>
```



Para no perder la orientación sobre el flujo de la construcción de la aplicación pondremos primero la Vista (que es un JSP), después Modelo (ActionForm que representa un formulario) y por último el Control (Action que indicara hacia donde dirigirse), también se ponen los elementos necesarios para la configuración en el archivo struts-config.xml.

Si el usuario va hacia el registro, un JSP (registro.jsp) nos ayudará a obtener sus datos, su código es el siguiente:

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
 4 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 5 "http://www.w3.org/TR/html4/loose.dtd">
 6 <html>
 7   <head>
 8     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 9     <title>Registro</title>
10   </head>
11   <body>
12     <center>
13     <h1>Introdusca los datos</h1>
14     <html:form action="datos">
15       <kbd><b><i>Matricula:</i></b></kbd><html:text property="matricula" />
16       <b><html:errors property="matricula" /></b>
17       <br/>
18       <kbd><b><i>Nombre:</i></b></kbd><html:text property="nombre" />
19       <b><html:errors property="nombre" /></b>
20       <br/>
21       <kbd><b><i>Correo:</i></b></kbd><html:text property="correo" />
22       <b><html:errors property="correo" /></b>
23       <br/>
24       <kbd><b><i>Password:</i></b></kbd><html:password property="password" />
```

```
25          <b><html:errors property="password" /></b>
26          <br/>
27          <br/>
28          <html:submit value="Enviar"/>
29       </html:form>
30    </body>
31 </html>
```

Para representar el Modelo se utiliza la clase formulario1 (formulario1.java) que se extiende de ActionForm; su código es:

```
1 package control;
 2 import javax.servlet.http.HttpServletRequest;
 3 import org.apache.struts.action.ActionErrors;
 4 import org.apache.struts.action.ActionMapping;
 5 import org.apache.struts.action.ActionMessage;
 6 public class formulario1 extends org.apache.struts.action.ActionForm {
 7    private Integer matricula;
 8    private String nombre;
 9    private String correo;
10     private String password;
11     public Integer getmatricula(){
12        return matricula;
13     }
14     public void setmatricula(Integer matricula){
15        this.matricula=matricula;
16     }
17
18     public String getnombre(){
19        return nombre;
20     }
21     public void setnombre(String nombre){
22        this.nombre=nombre;
23     }
24     public String getcorreo(){
25        return correo;
26     }
27     public void setcorreo(String correo){
```

```
28      this.correo=correo;
29    }
30    public String getpassword(){
31      return password;
32    }
33    public void setpassword(String password){
34      this.password=password;
35    }
36    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
37      ActionErrors error = new ActionErrors();
38      if(nombre==null || nombre.trim().equals("")){
39        error.add("nombre",new ActionMessage("falta ingresar nombre",
40            false));
41      }
42      if(matricula == null){
43        error.add("matricula",new ActionMessage("falta ingresar matricula",
44            false));
45      }
46      if(correo==null || nombre.trim().equals("")){
47        error.add("correo",new ActionMessage("falta ingresar correo",
48            false));
49      }
50      if(password==null || password.trim().equals("")){
51        error.add("password",new ActionMessage("falta ingresar correo",
52            false));
53      }
54      return error;
55    }
56 }
```

La parte de Control (formulario1Accion.java), se encarga de hacer la conexión con la base de datos y proporcionar el query para insertar los datos, también se encarga de crear un fólder con el nombre del alumno, esto se ocupara para guardar las tareas de los alumnos. A continuación se presenta su código:

```
1 package control;
 2 import control.Conexion;
 3 import javax.servlet.http.HttpServletRequest;
 4 import javax.servlet.http.HttpServletResponse;
 5 import org.apache.struts.action.Action;
 6 import org.apache.struts.action.ActionForm;
 7 import org.apache.struts.action.ActionMapping;
 8 import org.apache.struts.action.ActionForward;
 9 import java.sql.*;
10 import java.io.File;
11 public class formulario1Accion extends Action {
12    public ActionForward execute(ActionMapping mapping, ActionForm  form,
13        HttpServletRequest request, HttpServletResponse response)
14        throws Exception {
15      formulario1 forma=(formulario1) form;
16      //obtener los datos de la BD
17      String matricula=forma.getmatricula().toString();
18      String nombre=forma.getnombre();
19      String correo=forma.getcorreo();
20      String password=forma.getpassword();
```

```
21      Connection conexion=null;
22      String query="insert into alumnos "+
23          "(matricula,nombre,correo,password,folder)"+
24          "values('"+matricula+"',"+
25                  "'"+nombre+"',"+
26                  "'"+correo+"',"+
27                  "'"+password+"',"+
28                  "'"+nombre+"')";
29      try{
30        //obtenemos la conexion por defecto
31        conexion=new Conexion().getConexion();
32        //Codigo JDBC
33        Statement pstm=conexion.createStatement();
34        //ejecucion
35        pstm.executeUpdate(query);
36      }
37      catch(SQLException e){
38        getServlet().log("conexion mala",e);
39      }
40      finally{
41        try{
42          conexion.close();
43        }
44        catch(SQLException e){
45          getServlet().log("conexion cerrada",e);
46        }
47      }
48      //creando el directorio del alumno
49      String directorio=getServlet().getServletContext().getRealPath("");
50      directorio=directorio+"/tareas";
51      File crearDirectorio=new File(directorio,nombre);
52      crearDirectorio.mkdir();
53      System.out.println(directorio);
54      return mapping.findForward("datos_correcto");
55    }
56 }
```

Cuando el usuario quiera subir sus tareas, entonces tendrá que autentificarse dando su cuenta y password, si no son válidos se despliega un mensaje indicando qué dato está mal; su código del JSP (login.jsp) e imagen se muestran a continuación:

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
 4 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 5    "http://www.w3.org/TR/html4/loose.dtd">
 6 <html>
 7     <head>
 8         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 9       <title>Seguridad</title>
10     </head>
11     <body>
12     <h1>Introduzca los datos para ingresar</h1>
13     <html:form action="login">
```

```
14        <kbd><b><i>Matricula:</i></b></kbd><html:text property="matricula" />
15        <b><html:errors property="matricula" /></b>
16        <kbd><b><i>Password:</i></b></kbd><html:password property="password" />
17        <b><html:errors property="password" /></b>
18        <br/>
19        <html:submit value="Enviar"/>
20      </html:form>
21 </html>
```

El ActionForm (formulario2.java) que se encarga del Modelo verifica los datos cada vez que se realiza la petición de entrar, si están correctos el alumno puede subir sus datos, si no, se pide que corrija; su código es el siguiente:

```
1 package control;
 2 import javax.servlet.http.HttpServletRequest;
 3 import org.apache.struts.action.ActionErrors;
 4 import org.apache.struts.action.ActionMapping;
 5 import org.apache.struts.action.ActionMessage;
 6 import control.Conexion;
 7 public class formulario2 extends org.apache.struts.action.ActionForm {
 8    private Integer matricula;
 9    private String password;
10     private static Integer compartirmatricula;
11     public static Integer getcompartir(){
12        return compartirmatricula;
13     }
14     public Integer getmatricula(){
15        return matricula;
16     }
17     public void setmatricula(Integer matricula){
18        this.matricula=matricula;
19        this.compartirmatricula=matricula;
20     }
21     public String getpassword(){
22        return password;
23     }
24     public void setpassword(String password){
```

```
25      this.password=password;
26    }
27    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request){
28      ActionErrors error = new ActionErrors();
29      if((matricula == null)){
30        error.add("matricula",new ActionMessage("falta ingresar matricula ",
31            false));
32      } else{
33        try {
34          Conexion conexion=new Conexion(matricula.toString());
35          if(conexion.getmatricula()==null){
36            error.add("matricula",new ActionMessage("matricula no valida ",
37                false));
38          }else{
39            if(!password.equals(conexion.getpassword())){
40              error.add("password",new ActionMessage("password no valido ",
41                  false));
42            }
43          }
44        } catch (Exception ex) {
45          ex.printStackTrace();
46        }
47      }
48      if(password==null || password.trim().equals("")){
49        error.add("password",new ActionMessage("falta ingresar password ",
50            false));
51      }
52      return error;
53    }
54 }
```

La Acción (formulario2Accion.java) consulta la base de datos, y corrobora con los datos proporcionados por los alumnos, si son correctos, mandamos un findForward llamado “acceso”, si no lo son, mandamos un findForward de “noacceso”; su código es el siguiente:

```
1 package control;
 2 import java.util.Comparator;
 3 import javax.servlet.http.HttpServletRequest;
 4 import javax.servlet.http.HttpServletResponse;
 5 import org.apache.struts.action.Action;
 6 import org.apache.struts.action.ActionForm;
 7 import org.apache.struts.action.ActionMapping;
 8 import org.apache.struts.action.ActionForward;
 9 public class formulario2Accion extends Action {
10    public ActionForward execute(ActionMapping mapping, ActionForm  form,
11        HttpServletRequest request, HttpServletResponse response)
12        throws Exception {
13      formulario2 forma=(formulario2) form;
14      String matricula=forma.getmatricula().toString();
15      String password=forma.getpassword();
16      Conexion conexionBD=new Conexion(matricula);
17      String matriculaBD=conexionBD.getmatricula();
18      String passwordBD=conexionBD.getpassword();
19      if(matricula.equals(matriculaBD)&&
```

```
20            password.equals(passwordBD)){
21          return mapping.findForward("acceso");
22       }
23       else{
24          return mapping.findForward("noacceso");
25       }
26    }
27 }
```

Una vez autentificado el alumno ya tiene derecho de subir las tareas, y esta autentificación durará durante toda la sesión, el JSP (subirDatos.jsp) es el encargado de presentar al alumno la interfaz para realizar la operación de poder subir sus tareas, el código e imagen es la siguiente:

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
 4 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 5 "http://www.w3.org/TR/html4/loose.dtd">
 6 <html:html>
 7     <center>
 8         <h1>Nombre del Archivo a subir</h1>
 9         <html:form action="subir" enctype="multipart/form-data" >
10             Archivo: <html:file property="nombreArchivo" /> <br/>
11             <html:submit value="Enviar" />
12         </html:form>
13     </center>
14 </html:html>
```

La parte del Modelo (datosForm.java) sólo la ocupamos para poder guardar la ubicación del archivo que se desea subir, su código es:

```
1 package control;
 2 import javax.servlet.http.HttpServletRequest;
 3 import org.apache.struts.action.ActionErrors;
```

```
 4 import org.apache.struts.action.ActionMapping;
 5 import org.apache.struts.action.ActionMessage;
 6 import org.apache.struts.upload.FormFile;
 7 public class datosForm extends org.apache.struts.action.ActionForm {
 8     private FormFile nombreArchivo;
 9     public FormFile getnombreArchivo(){
10         return nombreArchivo;
11      }
12      public void setnombreArchivo(FormFile nombreArchivo){
13         this.nombreArchivo=nombreArchivo;
14      }
15 }
```

En el Control (accionForm.java) se obtiene la ubicación del archivo, y se abre el flujo para poder guardarlo en su carpeta correspondiente del alumno, el código necesario es el siguiente:

```
1 package control;
 2 import java.io.FileOutputStream;
 3 import java.io.InputStream;
 4 import java.io.OutputStream;
 5 import javax.servlet.http.HttpServletRequest;
 6 import javax.servlet.http.HttpServletResponse;
 7 import org.apache.struts.action.Action;
 8 import org.apache.struts.action.ActionForm;
 9 import org.apache.struts.action.ActionMapping;
10 import org.apache.struts.action.ActionForward;
11 import org.apache.struts.upload.FormFile;
12 import control.Conexion;
13 import java.sql.*;
14 import java.io.File;
15 public class accionForm extends Action{
16     public ActionForward execute(ActionMapping mapping, ActionForm  form,
17             HttpServletRequest request, HttpServletResponse response)
18             throws Exception {
19         datosForm forma=(datosForm) form;
20         //obtencion de los parametros de la forma
21         FormFile parametros=forma.getnombreArchivo();
22         String contexto=parametros.getContentType();
23         String nombre=parametros.getFileName();
24         int tamano=parametros.getFileSize();
25         //ver si nos da la ubicacion de nuestro directorio
26         String basePath=getServlet().getServletContext().getRealPath("tareas");
27         //obtener el directorio del usuario
28         Conexion conexion=new Conexion(formulario2.getcompartir().toString());
29         basePath=basePath+"/"+conexion.getfolder();
30         //obtener el nombre del archivo
31         String archivoEnDisco=basePath+
32                 System.getProperty("file.separator","/")+nombre;
33         //realizando para obtener el flujo del archivo
34         InputStream in=parametros.getInputStream();
35         OutputStream bos=new FileOutputStream(archivoEnDisco);
36         //recuperando el archivo
37         int bytesRead=0;
38         byte[] buffer=new byte[8192];
39         while ((bytesRead=in.read(buffer,0,8192))!=-1){
```

```
40            bos.write(buffer,0,bytesRead);
41        }
42        bos.close();
43        in.close();
44        return mapping.findForward("guardar");
45    }
46 }
```

El alumno puede escoger subir otra tarea o salir del sistema, si señala subir otra tarea, se le manda al JSP (subirDatos.jsp), ya no es necesario que se autentifique de nuevo, y si escoge salir, se cierra su sesión y se liberan los recursos, el código del JSP (cerrar.jsp) es el siguiente:

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 4 "http://www.w3.org/TR/html4/loose.dtd">
 5 <html>
 6     <head>
 7         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 8         <title>Salida</title>
 9     </head>
10     <body>
11         <center>
12             <h1>Tarea Registrada</h1>
13             <a href="opciones.do?metodo=cerrar">Salir del sistema</a>
14             <a href="opciones.do?metodo=subir">Subir otra tarea</a>
15         </center>
16     </body>
17 </html>
```

Su parte de Control (cerrar.java), se ocupa DispatchAction porque sólo necesitamos llamar al método que cierre la sesión o mande al JSP (subirDatos.jsp), su código es el siguiente:

```
1 package control;
 2 import javax.servlet.http.HttpServletRequest;
 3 import javax.servlet.http.HttpServletResponse;
```

```
 4 import org.apache.struts.actions.DispatchAction;
 5 import org.apache.struts.action.ActionForm;
 6 import org.apache.struts.action.ActionMapping;
 7 import org.apache.struts.action.ActionForward;
 8 public class cerrar extends DispatchAction {
 9   public ActionForward cerrar(ActionMapping mapping, ActionForm  form,
10       HttpServletRequest request, HttpServletResponse response)
11       throws Exception {
12     request.getSession().invalidate();
13     return mapping.findForward("cerrado");
14   }
15   public ActionForward subir(ActionMapping mapping, ActionForm  form,
16       HttpServletRequest request, HttpServletResponse response)
17       throws Exception {
18     return mapping.findForward("subir");
19   }
20 }
```

Expliquemos ahora el archivo struts-config.xml, ya que aquí hacemos todas las configuraciones necesarias para poder enlazar todos los elementos de Struts que ocupamos. Primero utilizamos el tag form-bean:

```
<form-beans>
        <form-bean name="formulario2" type="control.formulario2"/>
        <form-bean name="datosForm" type="control.datosForm"/>
        <form-bean name="formulario1" type="control.formulario1"/>
 </form-beans>
```

Segundo, declaramos todas las acciones que usen los ActionForm, esto se logra con el tag action-mappings:

```
<action-mappings>
    <action
      path="/datos"
      name="formulario1"
      scope="session"
      validate="true"
      input="/registro.jsp"
      type="control.formulario1Accion">
      <forward name="datos_correcto" path="/opciones.do?metodo=cerrar" redirect="false" />
    </action>
    <action
      path="/subir"
      name="datosForm"
      scope="session"
      validate="false"
      input="/subirDatos.jsp"
      type="control.accionForm">
      <forward name="guardar" path="/cerrar.jsp" redirect="false"/>
    </action>
    <action input="/login.jsp"
        name="formulario2"
        path="/login"
```

```
            scope="session"
            type="control.formulario2Accion">
        <forward name="acceso" path="/subirDatos.jsp"/>
        <forward name="noacceso" path="/inicio.jsp" redirect="false"/>
    </action>
    <action input="/subirDatos.jsp"
            name="formulario2"
            parameter="metodo"
            path="/opciones"
            scope="session"
            type="control.cerrar">
        <forward name="cerrado" path="/inicio.jsp"/>
        <forward name="subir" path="/subirDatos.jsp"/>
    </action>
  </action-mappings>
```

Para lograr que las personas no puedan entrar a las páginas JSP, sin antes estar autentificadas, se hace un filtro,el cual debe configurarse en web.xml, su configuración es la siguiente:

```
<filter>
    <filter-name>Seguridad</filter-name>
    <filter-class>filtro.Seguridad</filter-class>
</filter>
<filter-mapping>
    <filter-name>Seguridad</filter-name>
    <url-pattern>/subirDatos.jsp</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>Seguridad</filter-name>
    <url-pattern>/cerrar.jsp</url-pattern>
</filter-mapping>
```

El nombre del filtro es Seguridad, los tags filter-mapping restringen los JSPs subirDatos, cerrar, la clase (Seguridad) que cambia el flujo de la aplicación está en el paquete filtro, su código es el siguiente:

```
1 package filtro;
 2 import java.io.*;
 3 import java.net.*;
 4 import java.util.*;
 5 import java.text.*;
 6 import javax.servlet.*;
 7 import javax.servlet.http.*;
 8 import javax.servlet.Filter;
 9 import javax.servlet.FilterChain;
10 import javax.servlet.FilterConfig;
11 import javax.servlet.ServletContext;
12 import javax.servlet.ServletException;
13 import javax.servlet.ServletRequest;
14 import javax.servlet.ServletResponse;
15 import javax.servlet.*;
```

```
16 import javax.servlet.http.*;
17 public class Seguridad implements Filter {
18    public void init(FilterConfig filterConfig) throws ServletException {
19       //objeto de inicializacion
20    }
21    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException,
ServletException {
22       HttpServletResponse respuesta=(HttpServletResponse) response;
23       respuesta.sendRedirect("./login.jsp");
24    }
25    public void destroy() {
26    }
27 }
```

Para mostrar la efectividad de la aplicación lo haremos correr desde el servidor Tomcat, para esto tomaremos el archivo Tareas.war, seguimos los siguientes pasos:

1. Iniciar el servidor Tomcat.
2. Damos clic en Tomcat Manager.
3. Proporcionamos el nombre de usuario (admin) y el password.
4. Le decimos la dirección del archivo war en Archivo WAR a desplegar.

Una vez hecho estos pasos, en cualquier explorador le damos la siguiente dirección:

http://localhost:8080/Tareas

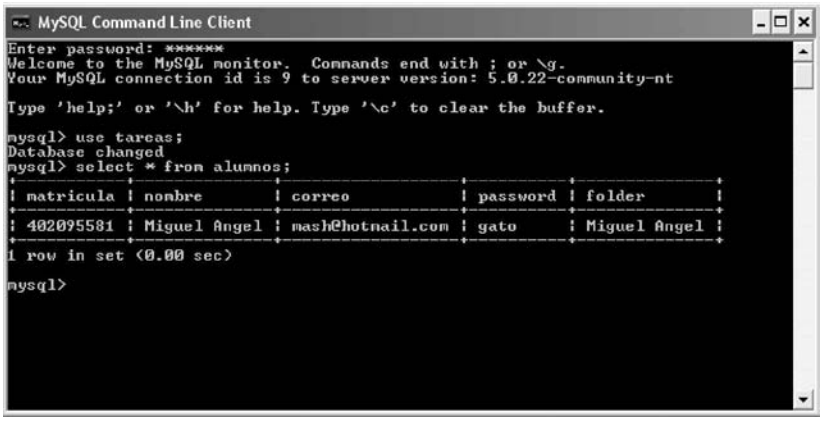
y se debe mostrar la pantalla de inicio de la aplicación como en la siguiente figura:

No se nos debe olvidar que también debe estar corriendo MySQL con la base de datos tareas y tiene que tener la tabla alumnos, con sus respectivos campos. Registremos un alumno y comprobemos que efectivamente guarda los datos en la base de datos tareas, los datos se muestra en la imagen siguiente:

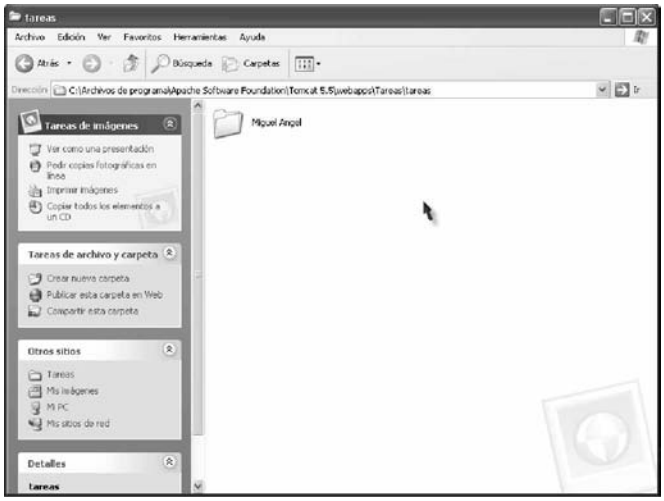
Abramos la consola de comandos de MySQL , y hacemos los siguientes pasos:

1. Proporcionamos el password de root.
2. Escribimos “use tareas;”.
3. Ahora “select * from alumnos;”.

En la imagen de abajo comprobamos que efectivamente sí se guardaron los datos.

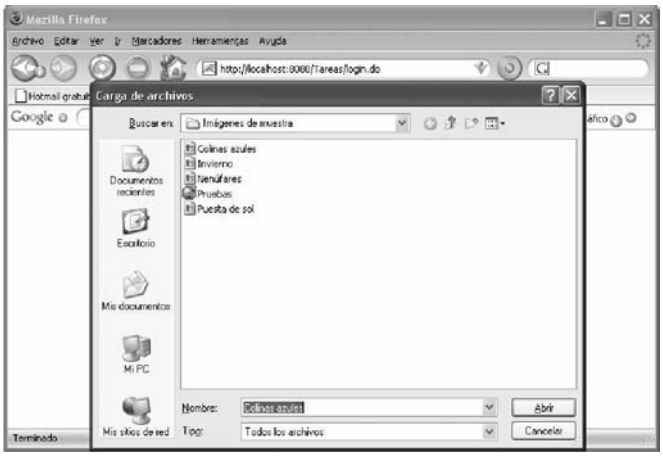


El campo fólder nos indica el nombre del fólder donde se guardaran las tareas del alumno, si nos vamos a la carpeta webapps de Tomcat y entramos a la carpeta de Tareas, dentro de estas tareas, encontraremos la carpeta creada por la aplicación llamada, en este caso Miguel Angel.

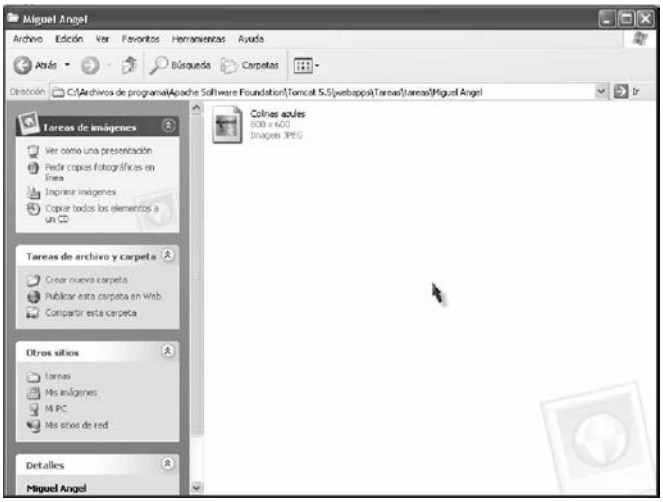


Comprobemos que las tareas se guardan en esa carpeta, le damos al enlace Subir tarea, si introducimos mal el nombre o el password, la aplicación nos indicará qué dato está mal.

Cuando los datos estén bien le proporcionamos el archivo a subir



Ahora comprobamos que realmente se encuentre este archivo en la carpeta del alumno



5

JavaServer Faces

5.1 ¿Por qué JavaServer Faces?

Actualmente existen dos técnicas de desarrollo para aplicaciones web:

- Un estilo de desarrollo rápido, usando un ambiente de desarrollo visual ofrecido por Microsoft en ASP.NET.
- El estilo de codificación de centro duro, escribiendo lotes de código que soportan un alto trazó de rendimiento a lado, ofrecido por J2EE.

Sabemos que J2EE es una atractiva plataforma, es muy escalable, tiene la facilidad de ser portable a múltiples plataformas, pero por otro lado ASP.NET hace la construcción fácil de una interfaz de usuario sin una programación tediosa, claro que los programadores quieren ambos: un trazo de alto rendimiento y una interfaz de usuario fácil para programar. La promesa de JavaServer Faces es brindar una interfaz de usuario rápida para el desarrollo de servicios en Java.

JavaServer Faces (JSF), al igual que Struts en cuanto a su objetivo de normalizar y estandarizar el desarrollo de aplicaciones con interfaz web. El Framework JSF es mucho más reciente que Struts, y tuvo en cuenta la experiencia de éste último para tomar lo mejor y reparar algunas de sus deficiencias.

5.2 ¿Qué es JSF?

Es un Framework que nos sirve para construir interfaces para aplicaciones Web, por lo que incluye:

- Un Conjunto de API´s para representar componentes gráficos y manejar su estado.

- Controlar eventos.
- Validar la entrada de datos.
- Definir la navegación por las distintas páginas de la aplicación.
- Dar soporte a la internacionalización.
- Accesibilidad por tener interfaces accesibles para todo el mundo.
- Una librería JSP personalizada para expresar una interfaz JSF en una página JSP.

En JSF se trabaja la capa de la Vista (la interfaz del usuario) de forma totalmente diferente de cómo se trabaja en la mayoría de los Frameworks. Sería muy similar al estilo de Swing o ASP.NET, donde la programación de la interfaz se hace a través de componentes y basado en eventos. Podemos concluir que JSF es una tecnología basada en especificaciones (JSR 127,252,276), realizado por le empresa Sun basado para aplicaciones Web, utilizando el patrón MVC dando un Framework totalmente compatible con la tecnología de Java.

5.3 Beneficios de JSF

La facilidad de empleo ha sido el objetivo principal de JSF, se define claramente una separación entre la lógica de la aplicación y la presentación, mientras que a su vez es mas fácil conectar la capa de presentación con el código de la aplicación. Se logra que cada miembro de un equipo de desarrollo de una aplicación Web, se dedique a su parte en el proceso de desarrollo de la aplicación, permitiendo así que se puedan unir todas las piezas cómodamente.

Como JSF ha sido desarrollada por JCP[1] que tiene miembros respetados entre la comunidad Java y vendedores de herramientas con lo que se garantiza un gran soporte a JSF en su estandarización.

También permite a los desarrolladores de componentes extender sus componentes de las clases para generar más componentes de librerías de marcas para clientes específicos. Como JSF es una especificación, nosotros podemos encontrar implementaciones de distintos fabricantes, esto no nos permite atarnos a un solo proveedor y podemos seleccionar al que más nos convenga, por

[1] Proceso de la comunidad de Java, establecida en 1988, permite a personas interesadas a involucrarse en la definición de futuras versiones y características de la plataforma en Java

ejemplo optar por MyFaces de la fundación apache, o JavaServer Faces de Sun que ocuparemos aquí.

5.4 Tecnología subyacente en JSF

Las aplicaciones Web de Java hablan en protocolo http vía Servlet, y emplean a los JSP para la visualización, por lo que JSF se apoya en:

- Protocolo de Transferencia de Hipertexto (http): JSF fue diseñado para que los desarrolladores se olviden que están utilizando este protocolo.
- Servlets: permite el desarrollo de aplicaciones Web sin necesidad de preocuparse por API de los Servlets.
- Portlest: Siempre ha existido la necesidad de que un software agregué información de distintas fuentes de datos en una interfaz fácil de usar, cada fuente de datos es normalmente mostrada en una región independiente dentro de la Web, con un compartimiento similar al de una ventana, cada una de estas regiones es un Portlet, JSF trabaja con el API Portlet, simplificando el uso de igual manera, como si se trabajara con Servlets.
- JavaBeans: Fácil creación y acoplamiento de esta tecnología en un JSF.
- JSP: Se pueden ocupar todas las cualidades de JSP dentro de un JSF.

5.5 Términos Fundamentales de JSF

JSF tiene su propio conjunto de términos, que forman la base conceptual de las características que proporciona. Es fácil darse una idea qué significado tiene cada uno de sus componentes, pero para hacer aplicaciones JSF es necesario comprenderlos mejor:

- Componente de interfaz de usuario: Es un objeto mantenido por el servidor, que nos da una funcionalidad específica para interactuar con el usuario. Estos componentes son un JavaBean con propiedades, métodos y eventos, que están organizados dentro de una vista, la cual es un árbol de componentes normalmente mostrados como una página.

- Renderer: Se encarga de mostrar los componentes de la interfaz de usuario, y traducir la entrada del usuario a valores de los componentes, éstos pueden ser diseñados para trabajar con uno o más componentes de interfaz de usuario y un interfaz de usuario puede ser asociado con muchos renders distintos.
- Validator: Se encarga de asegurar que el valor introducido por el usuario sea aceptable, uno o más validator pueden estar en un componente de usuario.
- Backing beans: Son JavaBeans especializados que coleccionan los valores de los componentes de interfaz gráfica e implementan métodos que capturan eventos, se pueden tener también referencia a componentes de interfaz de usuario.
- Converter: Convierte el valor de un componente en una cadena para mostrar, sólo un componente de interfaz de usuario se puede asociar a un converter.
- Eventos y captura de Eventos: JSF usa el modelo de JavaBeans de Evento/Captura. Los componentes de interfaz de usuario(y otros objetos) generan eventos, los que capturan los eventos se registran para manejar estos eventos.
- Messages: Información que se muestra de vuelta al usuario, en cualquier parte de la aplicación se puede generar información o mensaje de error que pueden ser mostrados al usuario.
- Navigation: La capacidad de moverse de una página a otra, JSF tiene un potente sistema de navegación que está integrado con captura de eventos específicos.

JSF tiene un alto grado de abstracción que permite una mayor separación entre el diseño y el código permitiendo dividir realmente el desarrollo de una aplicación en diseño y desarrollo, donde los desarrolladores sólo necesitan conocer poco la tecnología y se dejan de preocupar por la lógica de la aplicación, por ejemplo, cuando se construye una aplicación Web no es necesario conocer la diferencia entre un Post y un Get.

Incluye un conjunto de componentes y herramientas que facilitan la creación de widgets[2] comunes como caja de texto o botones, su codificación estándar (usados normalmente dentro de un JSP como tags) es definida por cada implementación de JSF.

[2] Es una pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de widgets, su función es dar acceso a funciones frecuentemente usadas y proveer información visual.

La comunicación entre los componentes, la entrada de datos y la validación, forman parte de la tecnología, es decir, no hay nada que implementar como en otras tecnologías. Su visualización de mensajes de validación y errores de conversión también están estandarizados, incluso soportan internacionalización.

Su vinculación con datos (data binding) en JSF puede ser realizada mediante JSF-managed bean (también llamado code-behind, similar al concepto de ASP.Net), por ejemplo se hace una pequeña modificación del lenguaje de expresiones JSTL[3] que es usado, si se escribe ${carro.motor} se codifica a #{carro.motor} donde carro tiene que se un JSF-managed bean y motor una de sus propiedades.

5.6 Ejemplo sensillo de un JSF

Para entender los términos antes mencionados, crearemos un ejemplo simple de un JSF utilizando Netbeans y explicaremos los componentes necesarios para contribuir al flujo de nuestra aplicación Web, en este ejemplo sólo le pediremos al usuario que introduzca su nombre y lo desplegaremos en un JSP dando la bienvenida a la utilización del Framework de JSF. Los pasos a seguir son:

1. Como primer paso arrancamos el Netbeans y le indicamos New Proyect->Web Application.
2. Le ponemos el nombre del proyecto en nuestro caso Ejemplo1, la estructura del código será Jakarta el servidor a ocupar Tomcat y lo demás lo dejamos como esta, le damos siguiente, y escogemos el tipo de Framework que utilizaremos que en este caso es JSF.
3. El nombre del Servlet será Faces Servlet y su clase que lo implementa es javax.faces.webapp.FacesServlet (ésta se registra en el descriptor de archivo web.xml), en Servlet URL Mapping lo cambiaremos a *.jsf, por ultimo le damos finalizar.

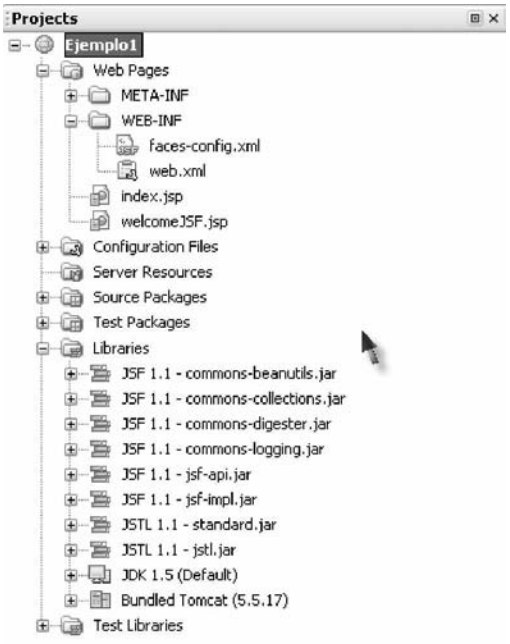
[3] JSTL (JSP Standard Tag Library), es un componente de las especificaciones de J2EE, y es controlado por Sun, no es mas que un conjunto de librerías de etiquetas simples y estándares que encapsulan las funcionalidad principal para escribir páginas JSP.

Para reconocer que se está ocupando JSF típicamente se ocupa la extensión *.jsf, pero podemos configurar de dos maneras, ya sea de manera como sufijo o prefijo, si lo queremos como sufijo, como es nuestro caso lo ponemos *.jsf, pero de manera de prefijo lo ponemos como estaba en la configuración por defecto en Netbeans /faces/*, para poder acceder a nuestra aplicación tenemos que indicar en la URL lo siguiente:

- Sufijo: http://localhost:8080:/Ejemplo1/index.jsf
- Prefijo: http://localhost:8080/Ejemplo1/faces/index.jsf

Hay que notar que si estamos ocupando a index.jsf como página de inicio, éste es un JSP común, pero como JSF está implementado por encima de la tecnología de Servlet, entonces el contenedor de Servlet hace el trazo del mapeo al JSP apropiado, incluso como se mencionó que la extensión más común es *.jsf también se ocupa *.faces y si queremos podemos poner la que nosotros deseemos, sólo que seguiremos el acuerdo que se ocupa y pondremos *.jsf.

En la siguiente imagen se muestra la estructura del directorio que nos genera Netbeans, y que ocuparemos para hacer las configuraciones correspondientes a nuestra aplicación Web.



Empecemos con web.xml, si lo analizamos Netbeans nos da los elementos necesarios en la configuración de nuestra aplicación web, aquí esta declarado el Servlet (Faces Servlet), y su clase que se ocupará(javax.faces.webapp.FacesServlet), y la URL(*.jsf) para hacer referencia a este Servlet, sólo quitaremos el elemento <welcome-file-list>, y nos vamos a pestaña de Projects le damos clic con el botón derecho en Ejemplo1 y le damos a Properties, y en Categories le damos a Run, y en la parte de Relative URL ponemos index.jsf, con esto indicamos que nuestra aplicación Web iniciará con esta página. Esta configuración es cuando hacemos pruebas con Netbeans y corremos desde el servidor que contiene, pero si lo hacemos desde nuestro servidor Tomcat no le ponemos nada al Relative URL, y tenemos que crear una página "index.html", en ella se pone lo siguiente:

```
<html>
  <head>
     <meta http-equiv="Refresh" content="0; URL=index.jsf">
    <title>Inicializando aplicación</title>
  </head>
  <body>
     <p>Espere mientras se inicia la aplicación</p>
  </body>
</html>
```

Esto se debe a que si nosotros damos en la URL del navegador http://localhost:8080:/Ejemplo1/ Tomcat no despliega la página de inicio que es "index.jsf", ya que él está buscando como inicio "index.html" o "index.jsp".

Ahora empecemos a trabajar con las bondades que nos da JSF, ocuparemos todos los términos antes mencionados.

Los mensajes son una parte fundamental en toda aplicación Web, como JSF se adapta a la internacionalización se puede configurar, para que dependiendo de la configuración del cliente en su navegador se muestre la aplicación al lenguaje que tiene predeterminado. Lo único que debemos hacer es un archivo con cadenas de caracteres, y adaptarnos al lenguaje de códigos ISO-639, para que esté más claro, seguimos los siguientes pasos:

1. En la pestaña de Projects y en Source Packages, creamos un nuevo paquete y le ponemos "mensajes".
2. Creamos un archivo "mensaje.properties", dentro de este paquete.

El formato que lleva este archivo es:

```
Titulo0= Ejemplo de JSF
Titulo1= Introduce tu nombre por favor
Titulo2= Hola
Titulo3= Bienvenido a JSF
Titulo4= JSF es una Tecnología amigable
Titulo5= Enviar
```

Si el navegador del cliente tiene como idioma principal el ingles, entonces de acuerdo al ISO-639, lo único que haremos es crear otro archivo con el siguiente nombre "mensaje_en.properties", si fuera Germano entonces sería "mensaje_de.propierties", y su formato del archivo es:

```
Titulo0= Example of JSF
Titulo1= Please enter your name
Titulo2= Hello
Titulo3= Welcome to JSF
Titulo4= JSF is a friendly Technology
Titulo5= To send
```

La configuración de nuestro JSF se hace en el archivo faces-config.xml, utilizaremos las siguientes etiquetas dentro de <faces-config>:

```
<application>
    <locale-config>
      <default-locale>es</default-locale>
      <supported-locale>en</supported-locale>
    </locale-config>
</application>
```

Estamos indicando que el lenguaje predeterminado es el español y que soportara el lenguaje de ingles, la salida de los mensajes en un idioma u otro lo proporciona el navegador del cliente.

Vamos a crear un JavaBeans, para esto creamos un paquete llamado Granos, y después le damos clic en el botón derecho del ratón y le indicamos que deseamos un JSF Managed Bean, de nombre le ponemos "Persona", y dejamos los parámetros que trae como predeterminado. Si nos fijamos en el archivo faces.config.xml trae la siguiente estructura:

```
<managed-bean>
    <managed-bean-name>Persona</managed-bean-name>
    <managed-bean-class>Granos.Persona</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Con esto ya podemos hacer uso de nuestro JavaBean en nuestra aplicación JSF, lo único que queremos de éste, es que guarde el nombre del cliente y no los pueda dar cuando lo necesitemos su código es el siguiente:

```
1 package Granos;
2 public class Persona {
3   private String nombre;
4   public String getNombre(){
5     return this.nombre;
6   }
7   public void setNombre(String nombre){
8     this.nombre=nombre;
9   }
10 }
```

Pasemos ahora al archivo index.jsp, para poder ocupar las tags de JSF, primero ocuparemos la etiqueta de la directiva de un JSP <%@ taglib .. > con ella incluimos los tags de JSF que ocuparemos, se añaden los tags de núcleo (core tags), que sirven para el renderer, y ocuparemos tags html (html tags), para ocupar elementos de HTML como son caja de texto y un botón para mandar la información. Cabe mencionar que todos los tags de JSF son contenidos dentro de la etiqueta f:view, ya que así le pondremos como prefijo en <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>, en lugar de utilizar una etiqueta form para formulario con formato HTML, se utilizará para JSF h:form, para un input en HTML se cambia la etiqueta a h:inputText, ya que también así se declaró como prefijo en <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>. Los valores del campo de entrada están restringidos al JavaBean (data binding) que ocupamos en este caso "Persona", por lo que cuando se introduzca el valor será almacenado en la variable "nombre", a través de la etiqueta siguiente <h:inputText value="#{Persona.nombre}"/>, para mostrar los mensajes según el idioma que esté configurado el navegador del usuario, primero debemos indicar donde se encuentran nuestros archivos de propiedades, para eso ocupamos <f:loadBundle basename="mensajes.mensaje" var= "msg" />, y solo se llama así <h:outputText value="#{msg.Titulo0}" />.

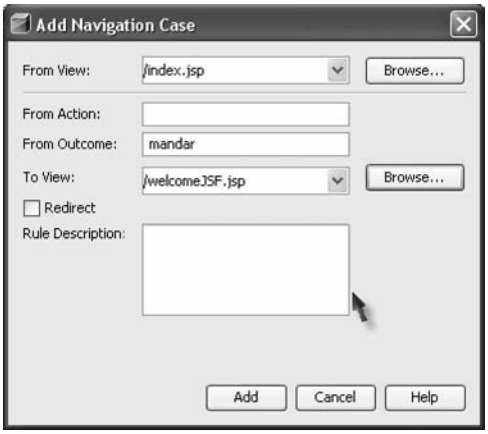
Sólo falta mencionar que añadimos un botón para mandar los datos, esto se hace con <h:commandButton value="#{msg.Titulo5}" action="mandar"/> el código queda de la siguiente manera:

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <%@ taglib  uri="http://java.sun.com/jsf/core" prefix="f" %>
 4 <%@ taglib  uri="http://java.sun.com/jsf/html" prefix="h" %>
 5 <html>
 6   <f:view>
 7     <f:loadBundle basename="mensajes.mensaje" var="msg"/>
 8     <head>
 9       <title><h:outputText value="#{msg.Titulo0}"/></title>
10     </head>
11     <body>
12       <center>
13         <h:form>
14           <h:outputText value="#{msg.Titulo1}"/>
15           <br/>
16           <h:inputText value="#{Persona.nombre}"/>
17           <br/>
18           <h:commandButton value="#{msg.Titulo5}" action="mandar" />
19         </h:form>
20       </center>
21     </body>
22   </f:view>
23 </html>
```

El archivo welcomeJSF sólo muestra el nombre del usuario que mandó y los mensajes de bienvenida, y su código es:

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
 4 <%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
 5 <html>
 6   <f:view>
 7     <f:loadBundle basename="mensajes.mensaje" var="msg"/>
 8     <head>
 9       <title><h:outputText value="#{msg.Titulo0}"/></title>
10     </head>
11     <body>
12       <center>
13         <h:outputText value="#{msg.Titulo2} " />
14         <h:outputText value="#{Persona.nombre}" />
15         <br/>
16         <h:outputText value="#{msg.Titulo3}"/>
17         <br/>
18         <h:outputText value="#{msg.Titulo4}"/>
19       </center>
20     </body>
21   </f:view>
22 </html>
```

Ya tenemos la parte de Modelo y Vista diseñados, sólo falta hacer la parte de Control, y esto como es lógico se lleva a cabo en el archivo faces-config.xml, haremos el flujo de nuestra aplicación estático, la parte dinámica se explicará más adelante. JSF ocupa elementos definidos para indicar su flujo, nos vamos al archivo faces-config.xml, le damos clic con el botón derecho en cualquier parte del código y le indicamos JavaServer Faces->Add Navigation Case, y seguimos la configuración de la siguiente imagen.

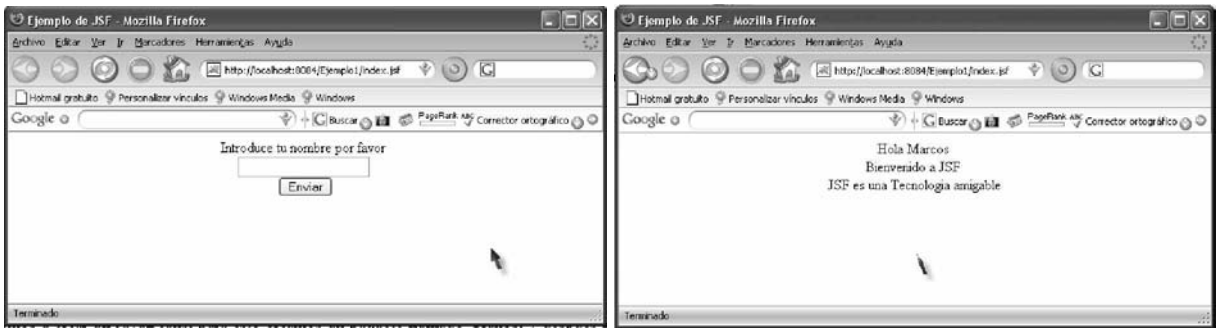


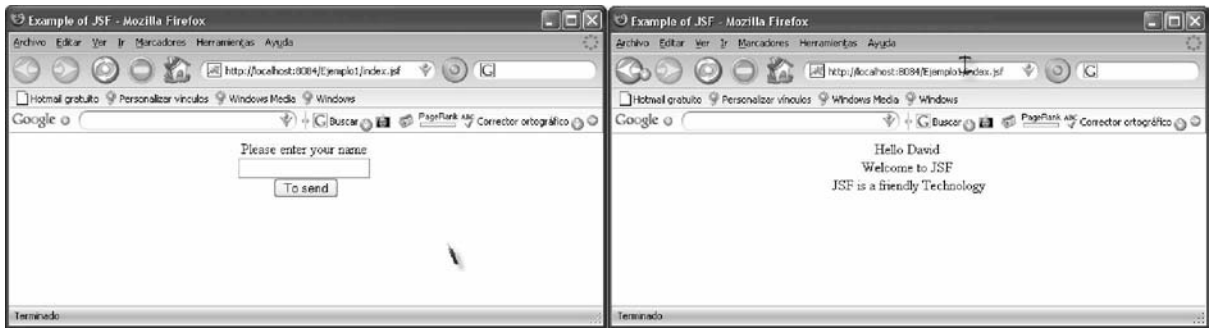
Esta venta es muy explícita indicando que es lo que deseamos hacer, primero indicamos que de la vista index.jsp tome el resultado de “mandar” (así le pusimos el nombre del elemento action del commandButton del tag JSF) a la vista welcomeJSF.jsp, le damos al botón Add, y nuestro archivo faces-config.xml queda de la siguiente manera.

```
1 <?xml version='1.0' encoding='UTF-8'?>
 2 <!DOCTYPE faces-config PUBLIC
 3   "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
 4   "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
 5 <!-- ========== FULL CONFIGURATION FILE ================================== -->
 6 <faces-config>
 7     <application>
 8       <locale-config>
 9         <default-locale>es</default-locale>
10          <supported-locale>in</supported-locale>
11       </locale-config>
12     </application>
13     <managed-bean>
14       <managed-bean-name>Persona</managed-bean-name>
15       <managed-bean-class>Granos.Persona</managed-bean-class>
16       <managed-bean-scope>request</managed-bean-scope>
17     </managed-bean>
18     <navigation-rule>
19       <from-view-id>/index.jsp</from-view-id>
```

```
20      <navigation-case>
21        <from-outcome>mandar</from-outcome>
22        <to-view-id>/welcomeJSF.jsp</to-view-id>
23      </navigation-case>
24    </navigation-rule>
25  </faces-config>
```

Con esto ya tenemos todo listo para correr nuestra aplicación Web, presionamos la tecla F6 a nuestro Netbeans y vemos las ventajas de JSF, a continuación se muestran las imágenes de la aplicación Web corriendo, las dos primeras son con la configuración del navegador con idioma en español y las dos últimas con ingles.





5.7 Navegación en JSF

Existen dos tipos de navegación en JSF: la navegación estática y dinámica. Analicemos la estática, considerando el ejemplo anterior el usuario introduce los datos y hace clic en el botón enviar, los datos y los cambios hechos por el usuario son transmitidos al servidor, en este momento la aplicación Web en el servidor analiza los datos de entrada del usuario y debe decidir que página JSF debe usar para dar la respuesta, el gestor de navegación es el responsable de

seleccionar la siguiente página, si analizamos el atributo que pusimos de la marca <h:commandButton value="#{msg.Titulo5}" action="mandar" /> y vamos al archivo faces-config.xml tenemos lo siguiente:

```
18    <navigation-rule>
19      <from-view-id>/index.jsp</from-view-id>
20      <navigation-case>
21        <from-outcome>mandar</from-outcome>
22        <to-view-id>/welcomeJSF.jsp</to-view-id>
23      </navigation-case>
24    </navigation-rule>
```

Aquí es donde estamos indicando hacia donde se debe dirigir el gestor de navegación que en este caso es a la página welcomeJSF.jsp, pero supongamos que deseamos un grupo múltiple de navegación y que varias páginas tengan un botón con el atributo "action" nombrado igual, y que se deban redireccionar a una sola página, la configuración debe ser de la siguiente forma.

```
<navigation-rule>
      <navigation-case>
        <from-outcome>mandar</from-outcome>
        <to-view-id>/welcomeJSF.jsp</to-view-id>
     </navigation-case>
</navigation-rule>
```

Otro caso que se podía presentar es que en una misma página tengamos varios botones y cada botón necesite dirigirse a una página en específico, por lo tanto así debe indicarse.

```
<navigation-rule>
     <from-view-id>/index.jsp</from-view-id>
     <navigation-case>
        <from-outcome>mandar</from-outcome>
        <to-view-id>/welcomeJSF.jsp</to-view-id>
      </navigation-case>

       <navigation-case>
        <from-outcome>mandar2</from-outcome>
        <to-view-id>/registro.jsp</to-view-id>
       </navigation-case>
</navigation-rule>
```

La navegación dinámica es necesaria porque en muchos casos en las aplicaciones Web no se puede utilizar una navegación estática, ya que algunas páginas no dependen de los botones si no de las respuestas que el usuario haya introducido en una caja de texto, pongamos un ejemplo, si el

usuario está contestando un cuestionario dependiendo de los resultados que vaya registrando la aplicación Web puede tener una acción en un página o en otra. Otro ejemplo común tenemos que ver si se registra bien el nombre de usuario y su password, si en la página donde se hace uso del botón para mandar los datos al servidor y se tiene la siguiente etiqueta:

```
<h:commandButton value=”Enviar” action=”#{Analizar.verificar}”/>
```

Aquí, se hace referencia a un bean con su clase “Analizar”, y esta clase tiene un método llamado “verificar”, este método va ser el encargado de regresar un String, dependiendo del análisis que haga, por ejemplo, éste se puede ver de la siguiente manera:

```
String verificar(){
    If(.....)
        return “valido”;
    else
        return “novalido”;
}
```

Entonces este método retorna dos Strings, válido y no válido, el gestor de navegación utiliza estos String para aparentar una regla de navegación combinada, y nuestro archivo faces-config.xml queda de la siguiente forma:

```
<navigation-rule>
        <navigation-case>
          <from-outcome>valido</from-outcome>
          <to-view-id>/inicio.jsp</to-view-id>
        </navigation-case>
        <navigation-case>
          <from-outcome>novalido</from-outcome>
          <to-view-id>/error.jsp</to-view-id>
        </navigation-case>
</navigation-rule>
```

En resumen los pasos que son llevados cuando el usuario da clic al botón “Enviar”.

- El bean especificado es recuperado.
- El método mencionado es llamado.
- La cadena como resultado es pasada al gestor de navegación.
- El gestor de navegación manda a la próxima página.

5.8 Etiquetas de JSF

Para poder realizar una buena aplicación Web con JSF se necesita una comprensión de las etiquetas que se tiene a disposición, se dividen en dos las de núcleo y HTML (core-HTML), que representan entre las dos 43 etiquetas disponibles. La mayoría de las páginas JSF tienen la siguiente estructura:

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
        <h:form>
        .....
        </h:form>
</f:view>
```

Como se puede ver, para el uso de las etiquetas se debe importar con la directiva taglib, se puede elegir cualquier prefijo que uno desee, pero por convención se pone "f" para core y "h" para HTML. A continuación se muestra una visión general de las etiquetas de núcleo.

Etiqueta	Descripción
view	Crea el nivel de vista
subview	Crea un sub-vista de la vista
facet	Añade una faceta a un componente
attribute	Añade un atributo (key/value) al componente
param	Añade un parámetro al componente
actionListener	Añade una acción oyente a un componente
valueChangeListener	Añade un valuechange oyente a un componente
converter	Añade un convertidor arbitrario a un componente
convertDateTime	Añade un convertidor de tiempo/Fecha a un componente
convertNumber	Añade un convertidor de numero a un componente
validator	Añade un validator a un componente
validateDoubleRange	Valida a un rango double a un componente
validateLength	Valida la longitud del valor de un componente
validateLongRange	Valida una longitud larga de valor de un componente
loadBundle	Carga los recursos de propiedades
selectitems	Especificar los items de selección de varios componentes
selectitem	Especificar el item de selección de varios componentes
verbaitem	Añade la marcación de una página JSF

Las etiquetas de JSF en HTML representan el flujo de la clases que los componen, una revisión general se muestra en la siguiente tabla.

Etiqueta	Descripción
form	Formulario en HTML
inputText	Línea simple de entrada de texto
inputTextarea	Múltiples líneas de texto de entrada
inputSecret	Entrada de texto de Password
inputHidden	Un campo escondido
outputLabel	Rotulo de otro componente de accesibilidad
outputLink	Un presentador en HTML
outputFormat	Semejante a outputText, pero organiza los mensajes
outputText	Línea simple de texto de salida
commandButton	Botón de tipo submit,reset o pushbutton
commandLink	Un vinculo que actua como botón
message	Para desplegar un mensaje reciente de un componente
messages	Despliega varios mensajes
graphicImage	Despliega una imagen
selectOneListBox	Selección simple de una caja de un listbox
selectOneMenu	Selección simple de un menú
selectOneRadio	Selección de un botón de radio
selectBoolenCheckbox	Un CheckBox
selectManyCheckbox	Selección de varios checkboxes
selectManyListbox	Múltiples selecciones de listbox
selectManyMenu	Múltiples selecciones de menus
panelGrid	Tabla en HTML
panelGroup	Dos o mas componentes que son colocados como uno
dataTable	Características de control en una tabla
column	Columna dentro de una tabla

Podemos agrupar las etiquetas de HTML de la siguiente manera:

- Entradas (input...).
- Salidas (output....).
- Mandato (commandButton and commandLink).
- Selecciones (checkbox,listbox,menu,radio).
- Diseño (panelGrid).
- Tabla de datos (dataTable).
- Errores y mensajes (message,messages).

Para poder identificar bien que hace cada uno, se sigue la relación componente/renderer, por ejemplo selectManyListbox, donde selectMany es el componente y Listbox es renderer, porque se transformara a un listbox.

Existen tres atributos comunes de etiquetas, que son compartidos entre los componentes de HTML múltiple:

- § Básicos.
- § HTML 4.0.
- § Eventos de DHTML.

Los atributos básicos de las etiquetas de HTML se muestran a continuación:

Atributo	Tipos de Componentes	Descripción
id	A (25)	Identificador de un componente
binding	A (25)	Referencia de un componente que puede se usando como apoyo en un bean.
rendered	A (25)	Un bolean, falso o verdadero, para no permitir el renderer.
styleClass	A (23)	Presentación para las hojas de estilo del nombre de la clase CSS.
value	I,O,C (19)	Para obtener el valor de un componente.
valueChangeListener	I (11)	Unir un método a otro método que responde a los cambios de valor.
converter	I,O (15)	Convertir al nombre de la clase.
validator	I (11)	Nombre de la clase validator, que es creado, para fijar a un componente.
required	I (11)	Es boleano, si es verdadero requiere un valor, para entrar el campo asociado.
A = Todos, I = Entrada, O = Salida C = Mandato, (n) = número de etiquetas con este atributo		

Se tiene que los atributos id, binding, son aplicables a todos las etiquetas de HTML que mencionan un componente, el primero es usado por los diseñadores de páginas, mientras que el segundo por los desarrolladores de Java. Mientras que value y converter son atributos que nos permiten especificar el valor del componente y querer transformar una cadena a un objeto o viceversa. Se tiene que validator, required y valueChangeListener están disponibles para las entradas, sus componentes están para validar valores y reaccionan a los cambios de esos valores, por último rendered y styleClass son atributos que afectan un componente respecto al renderer.

Ahora se enlista en la tabla los atributos de HTML 4.0.

Atributo	Descripción
accesskey (14)	Para definir una tecla combinada y darle el enfoque al elemento.
accept (1)	Separación de la lista de contenido para un formato.
accept-charset (1)	Espacios separados de una lista de caracteres codificados para un formato.
alt (4)	Texto alternativo para elementos no textuales para imágenes o applets.
border (4)	Para la anchura de los bordes.
chatset (3)	Codificación de calidad para un recurso vinculado.
coords (2)	Coordenadas de un elementos cuya forma es un circulo, rectángulo o polígono.
dir (18)	Dirección del texto.
disabled (11)	Deshabilitar el estado de un elementos de entrada o botón.
hreflang (2)	El lenguaje base de un recurso.
lang (20)	Lenguaje base de los atributos y texto de un elementos.
maxlength (2)	Máximo número de caracteres en un campo de texto.
readonly (11)	Estado de solo lectura en un campo de entrada, se puede seleccionar pero no editar.
rel (2)	Relación entre el documento en curso y un vinculo especificado, con la característica de href.
rev (2)	El vinculo contrario entre el documento en curso.
rows (1)	Número visible de filas en una área de texto.
shape (2)	Forma de una región.
size (4)	Tamaño de un campo de entrada.
style (23)	Información del estilo de linea.
tabindex (14)	Especificando el número de tabulaciones indexadas.
target (3)	El nombre de un recuadro con el que un documento fue abierto.
title (22)	El nombre que fue accionado para tener acceso a un elemento.
type (4)	Tipo de enlace.
width (3)	El ancho de un elemento.
(n) = numero de etiquetas con este atributo	

Por último se enlistan los eventos DHTML:

Atributo	Descripción
onblur (4)	Elemento que pierde el enfoque.
onchange (11)	Los cambios de valor del elemento.
onclick (17)	Botón del ratón que se hace clic sobre el elementos.
ondbclick (18)	Doble clic sobre el elemento.
onfocus (14)	Elemento que recibe el enfoque.

onkeydown (18)	La Tecla es presionada.
onkeypress (18)	La tecla es presionada y liberada posteriormente.
onkeyup (18)	La clave de la tecla es dada a conocer.
onmousedown (18)	Botón del ratón es presionado sobre el elemento.
onmousemove (18)	El ratón es movido sobre el elemento.
onmouseout (18)	El ratón deja el área del elemento.
onmouseover (18)	El ratón se mueve sobre un elemento.
onmouseup (18)	El botón del ratón es liberado.
onreset (1)	El formato es de reinicialización.
onselect (11)	Texto seleccionado dentro de un campo de entrada.
onsubmit (1)	El formato es presentado.
(n) = numero de etiquetas con este atributo	

Se muestran algunos ejemplos utilizando las etiquetas más comunes y algunos de sus atributos, sólo cabe mencionar que para ello se crea un Bean form y se registra en el archivo faces-config.xml, como lo hicimos en el ejemplo de JSF. Su código del Bean es:

```
public class form {
  private String testString="1234";
  private String lineas="linea 1\nlinea 2\nlinea 3";
  private boolean seleccion;
  public String getDatos(){
    return testString;
  }
  public String getLineas(){
    return lineas;
  }
  public void setSeleccion(boolean selec){
    this.seleccion=selec;
  }
  public boolean getSeleccion(){
    return seleccion;
  }
}
```

Ejemplo	Resultado
<h: inputText value"#{form.datos}" readonly="true"/>	1234
<h:inputSecret redisplay="true" value="#{form.datos}"/>	••••
<h:inputTextarea rows="2" cols="10" value="#{form.lineas}"/>	linea 1 linea 2 linea 3
<h:outputText value="Numeros #{form.datos}"/>	Numeros 1234
<h:graphicImage value="imagenes/rose.gif" />	

<h:outputLink value="http://google.com"> <h:outputText value="google"/> </h:outputLink>	google
<h:outputText value="Seleccionar "/> <h:selectBooleanCheckbox value="#{form.seleccion}"/>	Seleccionar ☑
<h:commandButton value="Enviar" action="mandar" />	Enviar

5.9 Mensajes

Durante el ciclo de vida de un JSF cualquier objeto puede generar un mensaje, y puede añadirlo a la Vista, típicamente estos mensajes son relacionados con los componentes y pueden mostrar conversiones o errores de validación. Aunque comúnmente los errores de mensajes son de tipo message, éstos se encuentran en cuatro variantes:

- § Información.
- § Advertencia.
- § Error.
- § Fatal.

Todo mensaje puede tener un resumen y un detalle de porqué se generó, por ejemplo un resumen sería “Invalida entrada”, y el detalle “el número de caracteres nos es válido”. Como se mencionó JSF ocupa dos etiquetas h:messages y h:message. La primera visualiza los mensajes en el contexto de JSF y su ciclo de vida, y podemos hacerlos globales, el segundo lo visualiza sólo para un componente en especial si se han generado muchos mensajes de acuerdo al contexto, h:message sólo muestra el último. Sus atributos son los siguientes:

Atributo	Descripción
for	El Id del componente que será desplegado solo se aplica a h:message.
errorClass	La clase de un CSS aplicar el error de mensaje.
errorStyle	La clase de un CSS aplicar el error de mensaje.
fatalClass	La clase de un CSS aplicar el error de mensaje.
fatalStyle	La clase de un CSS aplicar el error de mensaje.
globalOnly	Instrucción para desplegar globales mensajes aplicable solo a h:messages.
infoClass	Información de mensajes aplicados a clases CSS.
infoStyle	Información de mensajes aplicados de estilo a CSS.

layout	Especificación de mensaje de diseño, tabla o lista aplicable solo a h:messages.
showDetail	Un booleano que indica se los detalles de los mensajes son los indicados, false para h:messages, true para h:message predeterminado.
showSummary	Un booleano que determina si los resúmenes de los mensajes son para mostrar, true para h:messages, false para h:message predeterminado.
tooltip	Un booleno que detemina si los detalles de los mensajes son rendered dentro de tooltip, estos son únicamente rendered si showDetail y showSummary son true.
warnClass	Advertencias de mensajes en un clase CSS.
warnStyle	Advertencias de mensajes en un estilo de CSS.
biding,id,rendered, styleClass	Atributos Básicos.
style, title	HTML 4.

Si uno desea desplegar todos los mensajes en la parte de arriba de una página JSP, y aparte a lado de una etiqueta, por ejemplo un campo de texto, se puede hacer de la siguiente forma:

```
<h:form>
        <link href="styles.css" rel="stylesheet" type="text/css"/>
         <h:messages layout="table" errorClass=”errores”/>
         <h:inputText id="nombre" required="true"  value="#{form.datos}"/><br/><br/>
         <h:message for="nombre" errorClass=”errores”/>
         <h:commandButton value="Enviar" action="mandar" />
</h:form>
```

Sólo que aquí estamos señalando una clase CSS y que se encuentra definida en el archivo styles.css y su contenido es:

```
.errores{
font-style:italic;
}
```

5.10 Paneles

Durante la creación de salida de la Vista se utiliza los JSP, y para que estén alineados los componentes se utiliza muy frecuentemente tablas, este procedimiento es tedioso y pesado, pero JSF nos proporciona una etiqueta que nos ayuda a hacer este trabajo mas rápido, con h:panelGrid crea una tabla en HTML, con sólo especificar el número de columnas mediante el atributo columns, la sintaxis sería de la siguiente manera:

```
<h:panelGrid columns=”3”>
.....
</h:penelGrid>
```

Su funcionamiento es el siguiente, si nosotros especificamos tres columnas y nueve componentes, se genera tres filas y en ellas se ponen todos los componentes, lo podemos ver así, se crea una tabla de 3 x 3, pero si en cambio ponemos tres columnas y diez componentes se genera una tabla de 3 x 4 y en la última fila se inserta el último componente. Sus atributos son:

Atributo	Descripción
bgcolor	Color de fondo en la tabla.
border	Ancho del bordo de la tabla.
cellpadding	Relleno alrededor de la celda en la tabla.
cellspacing	Espacio entre las celdas de la tabla.
columnClasses	Lista de las CSS de las columnas.
columns	Numero de columnas en la tabla.
footerClass	Clases de CSS del pie de la tabla.
frame	Especificación del Frame circundante en la tabla que pueden ser dibujaos, sus valores validos son: none, above, below, hsides, vsides, lhs, box, border.
headerClass	Clase CSS para la cabecera de la tabla.
rowClasses	Lista de clases CSS de separación para las columnas.
rules	La especificación para dibujar líneas entre las celdas los valores validos son: groups, rows, columns, all.
summary	Resumen del propósito de la tabla y estructura usada.
binding, id, rendered, styleClass, value	Atributos básicos.
dir, lang, style, title, width	HTML 4.0.
Onclick, ondlclick, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup.	DHTML

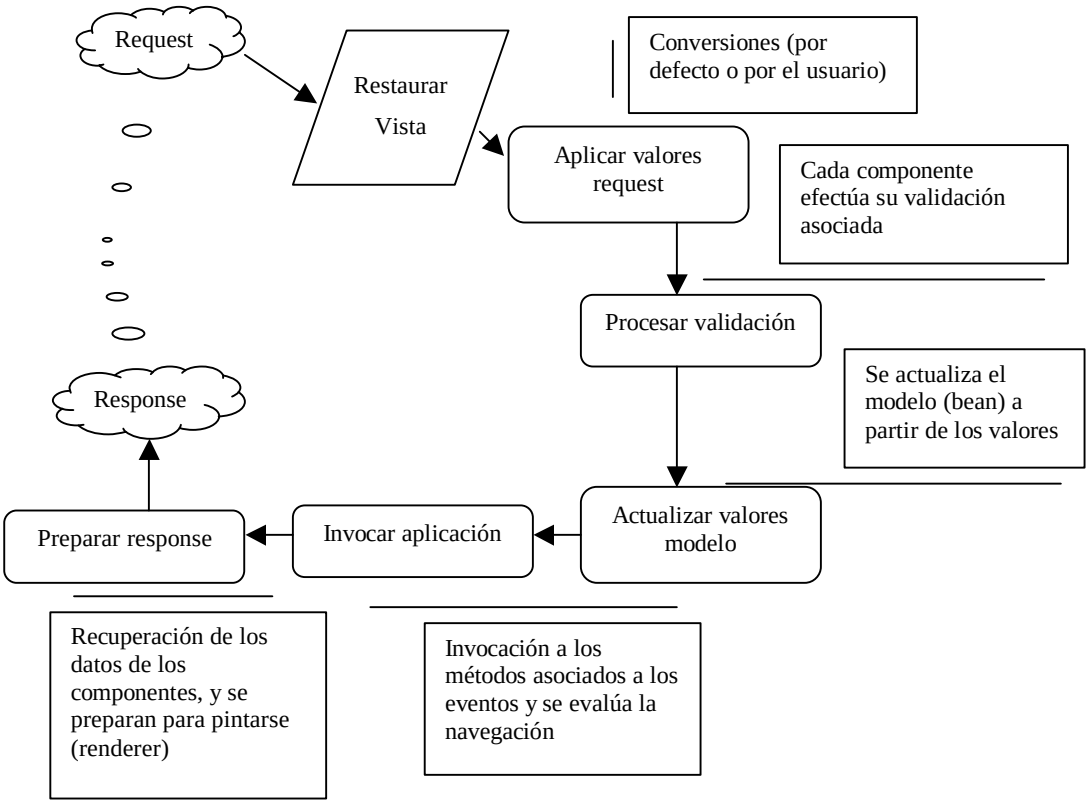
Uno puede especificar las clases CSS en diferentes partes de la tabla, cabecera, pie, filas y columnas. Los atributos columnClasses y rowClasses, especifican las lista de clases CSS en que se aplican para qué columnas o filas. Por ejemplo, para construir una posible especificación de

clases se puede hacer lo siguiente: rowClasses=”evenRows, oddRows” y columnClasses=”evenColumns,oddColumns”. La etiqueta h:panelGrid es utilizada en conjunto con h:panelGroup, con ella se pueden agrupar dos o más componentes en una misma celda. Un ejemplo seria así:

```
<h:panelGrid columns=”2”>
        ........
        <h:panelGroup>
                <h:inputText id=”nombre” value=”#{usuario.nombre}”/>
                <h:messge for=”nombre” />
        </h:panelGroup>
        .........
</h:panelGrid>
```

5.11 Conversión y Validación en JSF

JSF proporciona mecanismos para el formateo, conversión y validación de los componentes, estos mecanismos pueden ser extendidos para adaptarlos a nuestras necesidades, primero hablamos del formateo y conversión, por último la validación. Para entender mejor cómo JSF realiza estas maniobras de conversión y validación de los componentes de un formulario, analicemos el siguiente diagrama:



JSF proporciona dos mecanismos separados que nos ayudan a validar los valores introducidos por los usuarios a la hora de enviar los formularios:

- El primero es de conversión, y está definido por la interfaz javax.faces.convert.Converter, y sus múltiples implementaciones, las conversiones aseguran que el tipo de dato introducido por el usuario sea el correcto, entonces el tipo de cadena introducido en el formulario sea del tipo esperado por Java, y que está especificado en la propiedad correspondiente del bean. Los conversores son los que se encargan de hacer la transformación cadena à tipo Java y viceversa. JSF llama primero a los conversores antes de hacer las validaciones, y por lo tanto antes de aplicar los valores introducidos a las propiedades del bean. Si en el tipo de dato que es cadena no hay correspondencia al tipo de Java apropiado, el conversor lanza un ConversionException y el componente se marca como inválido.
- El segundo mecanismo es la validación y está definido por la interfaz javax.faces.validator.Validator y sus variados componentes, éste asegura que el dato introducido en el correspondiente componente es correcto según la lógica de la aplicación, este proceso ocurre antes que el Framework asigne los valores introducidos en el formulario a las propiedades del bean y justo después que se hayan aplicado las conversiones, si hay. Con esto aseguramos que los valores introducidos en un formulario de JSF tengan un valor correcto.

Todos los mensajes que se lancen en conversiones y validaciones se puede mostrar a través de la etiqueta h:message. Los conversores estándar que ofrece JSF se dividen en dos tipos, que son de tipo numéricos y de tipo Fecha. Disponemos de dos formas para indicar que el valor de un determinado componente será convertido y son:

1. Añadiendo el atributo converter="#{identificador_conversor}" al componente, los componentes que aceptan esto son inputText,i nputSecret, inputHidden y outputText, un ejemplo sería así:

```
<h:inputText converter="#{Integer}" value="bean.entero"/>
```

2. Incluyendo <f:convertNumber [lista de atributos]/> o <f:convertDateTime [lista de atributos] /> dentro de la invocación del componente quedaría de la siguiente manera:

```
<h:inputText value="#{bean.numeroDouble}">
  <f:convertNumber type="currency" />
</h:inputText>
```

De la primera forma los identificadores de conversor son BigDecimal, BigInteger, Number, Integer, Short, Byte, Character, Float, Double, Bolean y DateTime.

Los atributos disponibles para f:convertNumber son:

Atributo	Tipo
type	String
pattern	String
maxFractionDigits	Int
minFractionDigits	Int
maxIntegerDigits	Int
minIntegerDigits	Int
integerOnly	Boolean
groupingUsed	Boolean
locale	java.util.locale
currencyCode	String
currencySymbol	String

Sus correspondientes atributos para f:convertDateTime son:

Atributo	Tipo
type	String
dateStyle	String
timeStyle	String
pattern	String
locale	java.util.locale
timeZone	java.util.TimeZone

También nosotros podemos crear nuestro propio conversor, para hacer eso debemos seguir los siguientes pasos:

1. Crear una clase que implemente la interfaz javax.faces.convert.Converter.

2. Implementar el método getAsObject, que convierte el valor del componente que tengamos de la página (que es una cadena) a un objeto Java (que requiere la aplicación).
3. Implementar el método getAsString que convertira el objeto Java en una cadena requerida por la página.
4. Registrar el nuevo conversor en el contexto de JSF.
5. Ocuparlo en el componente para que conviertá el valor en nuestra página con la etiqueta <f:converter>.

Dado que JSF soporta la internacionalización nosotros podemos cambiar los mensajes de conversor que trae predeterminados, y estos se ajusten al lenguaje que tiene el usuario en su navegador, en el archivo faces-config.xml sólo agregamos el siguiente elemento:

```
<application>
    <message-bundle>mensajes.recursos</message-bundle>
</application>
```

En donde mensajes es una carpeta y recursos es un archivo de propiedades que contiene la cadena que se sustituye de acuerdo al mensaje del conversor y como se mencionó, para indicar el tipo de lenguaje sólo creamos otro archivo de propiedades que se llame recursos_en.properties, con esto estamos indicando que despliegue cadenas en ingles. Las claves para los mensajes estándar se muestran en la siguiente tabla:

Clave	Descripción
javax.faces.component.UIInput.CONVERSIÓN	Se ha producido un error de conversión.
javax.faces.component.UIInput.REQUIRED	Se debe introducir un valor obligatoriamente.
javax.faces.component.UISelectOne.INVALID	El valor introducido no es una opción válida.
javax.faces.component.UISelectMany.INVALID	El valor introducido no es una opción válida.
javax.faces.validator.NOT_IN_RANGE	El atributo especificado no aparece entre los valores esperados: {0} y {1}.
javax.faces.validator.DoubleRangeValidator.MAXIMUM	El valor supera el máximo permitido, que es {0}.
javax.faces.validator.DoubleRangeValidator.MINIMUM	El valor no alcanza el mínimo permitido, que es {0}.
javax.faces.validator.DoubleRangeValidator.TYPE	El valor no es del tipo correcto.
javax.faces.validator.LengthValidator.MAXIMUM	El valor supera el máximo permitido, que es {0}.
javax.faces.validator.LengthValidator.MINIMUM	El valor no alcanza el mínimo permitido, que es {0}.

javax.faces.validator.LongRangeValidator.MAXIMUM	El valor supera el máximo permitido, que es {0}.
javax.faces.validator.LongRangeValidator.MINIMUM	El valor no alcanza el mínimo permitido, que es {0}.
javax.faces.validator.LongRangeValidator.TYPE	El valor no es del tipo correcto.

Por ejemplo, si tenemos nuestro archivo recursos.properties y queremos cambiar el mensaje de error CONVERSIÓN sólo añadimos lo siguiente:

javax.faces.component.UIInput.CONVERSION= Datos no validos, corríjalos.

Y en el archivo recursos_en.properties añadimos la siguiente línea:

javax.faces.component.UIInput.CONVERSION= Not worth data, correct them.

Los validadores pueden realizar validaciones más genéricas durante la fase de validación en JSF, estos componentes implementan la interfaz javax.faces.validator.Validator, que contienen un único método que es validate( ), JSF contiene un conjunto de validadores estándar que se describen a continuación:

Etiqueta	Clase Validator	Atributos	Descripción
f:validateDoubleRange	DoubleRangeValidator	minimum,maximun	Valida que el componente sea mayor que un mínimo, y mayor que un máximo, el valor debe ser de tipo float.
f:validateLongRange	LongRangeValidator	minimum,maximun	Valida que el componente sea mayor que un mínimo, y mayor que un máximo, el valor debe ser de tipo entero.
f:validateLength	LengthValidator	minimum,maximun	Valida que la longitud del componente este entre un mínimo y un máximo.

Si nosotros queremos crear nuestro propio validador, necesitamos hacer los siguientes pasos:

1. Crear una clase que implemente la interfaz javax.faces.validator.Validator.
2. Implementar el método validate.
3. Registrar el nuevo validador en el contexto de JSF.
4. Insertarlo dentro del componente cuyo valor queremos validar a través de la etiqueta <f:validator>.

Se puede cambiar también los mensajes que trae predeterminado JSF de errores de validación, si ya se hizo una sola vez el procedimiento explicado con conversiones, automáticamente el de validaciones cambiará sus mensajes porque ocupa las mismas claves de mensajes estándar.

5.12 Tablas de datos

Las tablas de HTML son muy populares para colocar contenido en nuestra aplicación Web, para hacer esto sólo tenemos que ocupar la etiqueta h:dataTable, esta etiqueta interactúa con los datos para crear una tabla en HTML, su estructura debe ser la siguiente:

```
<h:dataTable value=”items” var=”identificador”>
        <h:column>
                <%--Componentes de la columna a la derecha--%>
                <h:outputitText value=”#{items.Nombre_de_Propiedad}”/>
        </h:column>
        <h:column>
                <%--siguientes componentes de la columna--%>
                <h:outputitText value=”#{items.Nombre_de_Propiedad}”/>
        </h:column>
        <%-- añadir mas columnas, si se necesitan--%>
</h:dataTable>
```

El atributo value representa los datos que la etiqueta h:dataTable puede interacturar, estos datos deben ser uno de los siguientes:

- Un arreglo.
- Una instancia de java.util.List.
- Una instancia de java.sql.Resulset.

- Una instancia de javax.Servlet.jsp.jstl.sql.Result.
- Una instancia de javax.faces.model.DataModel.

El nombre de los items para su ocupación es especificado por el atributo var, con esto podemos identificar donde queremos ir insertando cada elemento en la tabla. Su uso podría ser como sigue, construyamos primero un bean que guarde sólo el nombre y apellido.

```
public class Nombre {
  private String nombre;
  private String apellido;
  public Nombre(String nombre,String apellido) {
    this.nombre=nombre;
    this.apellido=apellido;
  }
  public void setNombre(String Nombre){this.nombre=Nombre;}
  public String getNombre(){return nombre;}

  public String getApellido() {return apellido;}
  public void setApellido(String Apellido){this.apellido=Apellido;}
}
```

Hagamos el bean que servirá como item para la tabla su código es el siguiente:

```
public class DatosTabla {
  private static final Nombre[] nombres= new Nombre[]{
    new Nombre("Miguel","Sanchez"),
    new Nombre("Mario","Aguirre"),
    new Nombre("Pablo","Trejo"),
    new Nombre("Carlos","Perez"),
  };
  public Nombre[] getNombre(){return nombres;}
}
```

En el archivo faces-config.xml lo damos de alta para su utilización:

```
<faces-config>
  <managed-bean>
    <managed-bean-name>tabladatos</managed-bean-name>
    <managed-bean-class>granos.DatosTabla</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
</faces-config>
```

Ahora en una página JSP su utilización de la etiqueta sería así:

```
<h:dataTable value="#{tabladatos.nombre}" var="nom">
        <h:column>
          <h:outputText value="#{nom.nombre}"/>
        </h:column>
        <h:column>
          <h:outputText value="#{nom.apellido}"/>
        </h:column>
</h:dataTable>
```

Notemos que el atributo value tiene los datos del item que se ocupará, es un arreglo de objetos de la clase Nombre, y la clase DatosTabla regresa ese arreglo de objetos a través de la variable estática constante nombres, y como se llamó en el archivo faces-config.xml “tabladatos” el bean, esa es la razón que se llame así en el atributo value, para su referencia en la página JSP se pone el nombre de “nom”, porque está indicado en el atributo var.

Las tablas de datos en JSF se pueden hacer las siguientes operaciones:

- Añadir Cabeceras y Pies de página.
- Incrustar componentes de JSF a las celdas.
- Editar las celdas de las tablas.
- Ponerles Estilos a las filas y Columnas.
- Insertar datos directos de una Base de Datos.
- Borrar Filas de las tablas.
- Insertar Filas en las tablas.
- Ponerles Barras de distribución.
- Barras de distribución con dispositivos de ventana.

5.13 Ventajas de JSF con Struts

Como podemos ver JSF nos ofrece todo lo necesario para la capa de Vista y así como para la capa del Controlador, sus ventajas ante Struts son:

- Flexibilidad en el Controlador y manejo de eventos.

- Navegación Dinámica.
- Desarrollo de páginas más rápido.
- Integración y Extensibilidad

En pocas palabras podemos decir que JSF mejoró todos los defectos que se tienen con Struts, ya que Craig McClanahan, líder de las especificaciones JSF es el creador de Struts.

6

Aplicación con JavaServer Faces

6.1 JDBC con JSF

Desde que Sun publicó su primera API en 1996 que permite la Conectividad de Bases de datos desde Java (JDBC), en donde permite a los programadores conectarse a una base de datos y efectuar después consultas o actualizaciones en ella, empleado el Lenguaje Estructurado de Consultas o SQL haciendo así más fácil el trabajo para el programador.

La gran ventaja que tenemos con respecto a otros entornos de programación con base de datos, es que los programas hechos con Java y JDBC son independientes de la plataforma y del fabricante, una misma aplicación puede funcionar en una máquina basada en NT, en un servidor Solaris o en cualquier artefacto dotado de una base de datos basado en la plataforma Java.

Como se ha visto a lo largo de la tesis se hizo comunicación utilizando JDBC con Servlets, JSPs y con Struts, y como era de esperarse JSF no es la excepción, incluso este Framework es más fácil de hacer el flujo de información que Struts gracias a su Navegación Dinámica que nos proporciona, evitando así hacer grandes esfuerzos para poder comunicar la parte del Modelo que lleva la lógica del negocio con la Vista, cuando queremos presentar los resultados obtenidos. Si nosotros hacemos un Bean que realice todo el proceso de conexión, consulta y modificación con solo declararlo en el manejador de Bean´s de JSF que es en el archivo faces-config.xml, tendremos todas las bondades que nos ofrece JDBC en nuestra aplicación.

6.2 Etiquetas en JSF

La mayoría de las etiquetas de JSP se encargan en generar el código HTML directo, como lo hacen también las etiquetas de Struts, pero JSF traza las etiquetas en un mapa a un componente de renderer que este se encarga de traducir a un componente de JSF, los renderers se ocupan

mucho en plug-and-play, con esto deja paso a que los desarrolladores puedan desplazar el renderer para dispositivos diferentes, con solo de hacer un TLD (Descriptor de librerías de etiquetas) e implementar los gestores de clases ya podemos hacer cualquier comunicación con dispositivos que deseemos.

6.3 Ericsson Mobile JSF Kit

Como se mencionó, la cualidad de renderer que tiene JSF nos da la posibilidad de la comunicación de diferentes dispositivos, MobileFases está basado en la arquitectura del Framework de JSF, utilizando sus etiquetas nosotros podemos hacer una página en JSF y poder visualizarla en nuestro dispositivo móvil que soporte WAP (Protocolo de Aplicaciones Inalámbricas).

Hoy día cada vez los desarrolladores tienen que cambiar sus aplicaciones Web para dispositivos móviles, la representación gráfica es uno de los problemas más grandes entre una aplicación Web y una móvil, esto se debe a que las aplicaciones móviles tienen que ver con diferentes dispositivos, ya que soportan diferentes lenguajes de marcado, tamaños diferentes de pantalla y manejo variado de imágenes.

El kit de Mobile JSF provee unas librerías basadas en JSF que solucionan esto, logrando así poder implementar aplicaciones Web que se pueden ver también en un móvil. Para descargar este kit se hace en el siguiente enlace.

http://www.ericsson.com/mobilityworld/sub/open/technologies/open_development_tips/tools/mobile_jsf_kit

Incluso existe un templete (plantilla) para Netbeans, que se puede bajar en el enlace de arriba, logrando hacer más fácil las aplicaciones Web para móviles. La principal diferencia es el lenguaje que soporta, una aplicación Web utiliza HTML o XHTML, como componentes de las páginas, pero WAP ocupa WML (Lenguaje de Marcado Inalámbrico), para navegadores con WAP 1.0, y XHTML-MP (XHTML-Perfil de Móvil), para navegadores WAP 2.0, XHTML-MP es similar a XHTML, pero WML es totalmente diferente a XHTML y HTML.

6.4 Instalación del Templete Mobile JSF para Netbeans

Una vez bajado el templete, los descomprimimos y hacemos los siguientes pasos en Netbeans.

- Dentro de Tools, señalamos Update Center.
- Señalamos Install Manually Downloaded Modules.
- En el botón add señalamos la ruta en donde se encuentra nuestro archivo *.nbm que se encuentra en la carpeta release en donde descomprimimos el archivo que bajamos.
- En la siguiente pantalla señalamos el plug-in y le damos next.
- Nos pedirán que se reinicie nuestro Netbeans, le decimos que si.

6.5 Navegador para Móvil

Necesitamos un navegador móvil para poder ver nuestros avances, para esto ocupamos el navegador de Openwave que se encuentra en el siguiente enlace.

http://developer.openwave.com/dl/Openwave_v70_Simulator.exe

Si no se puede bajar, es porque es necesario registrarse, con este emulador de un navegador móvil podemos ver nuestras aplicaciones móviles que vayamos creando, su funcionamiento se muestra en la imagen siguiente:

6.6 Aplicación Móvil con JSF

Pongamos un situación que se puede presentar, estamos en una institución educativa, y necesitamos ver los horarios de clases, si la institución cuenta con WAP y nuestro móvil tiene acceso, nosotros podemos ver los horarios fácilmente mediante una aplicación Móvil que este corriendo en un servidor, que soporte JSF como es Tomcat.

Lo que se va a ocupar es:

- El API de JDBC para la conexión a una base de datos que contiene la información de los horarios.
- El kit de Movile Faces de Ericsson.
- El emulador de Openwave para ver nuestros avances.
- El Framework de JSF.

Como primer paso creamos un proyecto nuevo en Netbeans con los siguientes pasos:

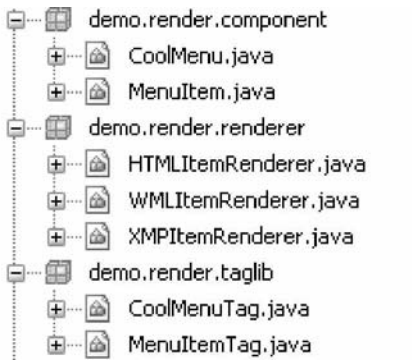
- File à New Proyect.
- En Categories señalamos Web, y en Projects Web Application.
- Nombre del proyecto le ponemos “horarios”, en Source Structure “Jakarta” y lo demás lo dejamos como esta.
- Señalamos el tipo de Framework que ocuparemos que es JavaServer Faces.
- Servlet URL Mapping ponemos *.jsf , y le damos finish.

Añadiremos las librerías de Mobile JSF, en la pestaña de Projects señalamos libraries y con el botón derecho del ratón ponemos add Library, son dos las librerías que incluiremos “MobiFaces” y “MobileFaces”, que contiene las etiquetas necesarias para el render al móvil que ocupemos.

También nos proporciona Movile JSF unas etiquetas especiales como ejemplo para poder poner menús en forma de imágenes, señalando las bondades que nos ofrece JSF con sus componentes, para utilizarlas hacemos lo siguiente:

- En la pestaña Projects señalamos Source Packages, le damos clic al ratón con el botón derecho y señalamos Java Package.
- Como nombre del paquete le ponemos lo siguiente “demo.render.component”.

Hacemos los mismo pasos añadiendo como nombre de los paquetes “demo.render.rendeder” y “demo.render.taglib”, una vez echo esto necesitamos el código fuente, para esto nos vamos al directorio donde se desempaqueto Mobile JSF, y en la carpeta de samples à CustomRenderer à src à java à com à ericsson à sn à mobilefaces. Encontramos los archivos del código fuente. Solo copiamos cada archivo .java en su respectiva carpeta en donde se encuentra nuestra aplicación Web “horarios”, tal como se muestra en la imagen siguiente:



Después en el editor de Netbeans abrimos cada archivo y le cambiamos la ruta de la ubicación de los paquetes, por ejemplo el archivo “CoolMenu.java” tiene como ruta de ubicación de paquete package com.ericsson.sn.mobilefaces.demo.render.component; lo cambiamos por package demo.render.component; porque ahora es su nueva ubicación, hacemos lo mismo para cada archivo, nota dependiendo en donde se encuentre el archivo cambia su ruta.

Necesitamos añadir un descriptor de librerías de etiquetas (TLD), este debe de estar ubicado en la carpeta WEB-INF, le damos clic con el botón derecho, escogemos New à File/Fólder à Web, buscamos la opción Tag Library Descriptor. Como nombre le ponemos coolmenu, pulsamos finish.

Con el TLD se da toda la relación necesaria para que funcionen las clases, su código es el siguiente:

```
1 <?xml version="1.0" encoding="UTF-8"?>
 2 <!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
 3 <!--Tag library define for cool menu items-->
 4 <taglib>
 5   <tlib-version>0.03</tlib-version>
 6   <jsp-version>1.2</jsp-version>
 7   <short-name>d</short-name>
 8   <uri>http://www.ericsson.com/sn/mobilefaces/demo</uri>
 9   <description>
10       This tag library for custom renderer demo.
11    </description>
12   <!--cool menu tag-->
13   <tag>
14     <name>coolMenu</name>
15     <tag-class>demo.render.taglib.CoolMenuTag</tag-class>
16     <description>
17           This cool menu main body to contain the menu items.
18     </description>
19     <attribute>
20       <name>columns</name>
21       <required>false</required>
22       <description>Control the layout of columns to display</description>
23     </attribute>
24   </tag>
25   <!--cool menu item tag-->
26   <tag>
27     <name>menuItem</name>
28     <tag-class>demo.render.taglib.MenuItemTag</tag-class>
29     <description>
30           This menu item for cool menu.
31     </description>
32     <attribute>
33       <name>itemHref</name>
34       <required>false</required>
35       <description>the href link url of this item</description>
36     </attribute>
37     <attribute>
38       <name>itemLabel</name>
39       <required>true</required>
40       <description>the label of this item</description>
41     </attribute>
42     <attribute>
43       <name>itemIcon</name>
44       <required>false</required>
45       <description>the icon image path of this item</description>
46     </attribute>
47     <attribute>
48       <name>disabled</name>
49       <required>false</required>
50       <description>disable this item or not</description>
51     </attribute>
52   </tag>
53 </taglib>
```

Mobile JSF necesita unos parámetros iniciales para que se configure, para esto necesita dos archivos xml, uno es devices.xml que identifica el dispositivo en donde se desplegara la aplicación, y el otro adaptors.xml como se desplegara la información , para utilizarlos se crea una carpeta en Web Pages con el nombre de “conf”, y hay van los dos archivos, su código del primero es:

```
1 <?xml version="1.0" encoding="UTF-8"?>
 2 <device-list>
 3   <!--
 4       Default Device. Use this one, if no other device can be matched.
 5       -->
 6   <device>
 7     <DEVICE_NAME>DefaultDevice</DEVICE_NAME>
 8     <USER_AGENT>default</USER_AGENT>
 9     <SCREEN_LEVEL>9</SCREEN_LEVEL>
10     <MARKUP>HTML_BASIC</MARKUP>
11     <SCRIPT>javascript</SCRIPT>
12     <IMAGE_SUPPORT>jpg;jpeg;bmp;gif;png;tiff;</IMAGE_SUPPORT>
13   </device>
14   <!--
15       InternetExplorer. Web browser with HTML language.
16       -->
17   <device>
18     <DEVICE_NAME>InternetExplorer</DEVICE_NAME>
19     <USER_AGENT>MSIE</USER_AGENT>
20     <SCREEN_LEVEL>9</SCREEN_LEVEL>
21     <MARKUP>HTML_BASIC</MARKUP>
22     <SCRIPT>javascript</SCRIPT>
23     <IMAGE_SUPPORT>jpg;jpeg;bmp;gif;png;tiff;</IMAGE_SUPPORT>
24   </device>
25   <!--
26       OpenWave-WML. WAP1 browser with WML language.
27       -->
28   <device>
29     <DEVICE_NAME>OpenWaveWML</DEVICE_NAME>
30     <USER_AGENT>OPWV-SDK-WML</USER_AGENT>
31     <SCREEN_LEVEL>5</SCREEN_LEVEL>
32     <MARKUP>WML</MARKUP>
33     <SCRIPT>wmlscript</SCRIPT>
34     <IMAGE_SUPPORT>png;gif;wbmp;</IMAGE_SUPPORT>
35   </device>
36   <!--
37       OpenWave-XMP. WAP2 browser with XHTML-MP language.
38       -->
39   <device>
40     <DEVICE_NAME>OpenWaveXMP</DEVICE_NAME>
41     <USER_AGENT>OPWV-SDK-XMP</USER_AGENT>
42     <SCREEN_LEVEL>5</SCREEN_LEVEL>
43     <MARKUP>XHTML_MP</MARKUP>
44     <SCRIPT>ecmascriptmp</SCRIPT>
45     <IMAGE_SUPPORT>png;gif;</IMAGE_SUPPORT>
```

```
46    </device>
47
48 </device-list>
```

El de adaptors.xml es:

```
1 <?xml version="1.0" encoding="UTF-8"?>
 2 <adaptor-list>
 3       <!--
 4       Image convertor for default size
 5       -->
 6       <image-convertor>
 7             <NAME>Default</NAME>
 8             <SCREEN>0</SCREEN>
 9             <RATE>0.3</RATE>
10       </image-convertor>
11       <!--
12       Image convertor for Middle size
13       -->
14       <image-convertor>
15             <NAME>Middle</NAME>
16             <SCREEN>5</SCREEN>
17             <RATE>0.3</RATE>
18       </image-convertor>
19       <!--
20       Image convertor for Middle size
21       -->
22       <image-convertor>
23             <NAME>Big</NAME>
24             <SCREEN>9</SCREEN>
25             <RATE>1</RATE>
26       </image-convertor>
27
28 </adaptor-list>
```

Para que estos parámetros inicialicen necesitamos declarar un descriptor de archivo que lo llamaremos mfaces-config.xml y este se ubica en la carpeta WEB-INF sus elementos que debe tener son:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <mfaces-config>
3   <!-- Configuration Files -->
4   <configuration-file>
5     <devices>conf/devices.xml</devices>
6     <media-adaptors>conf/adaptors.xml</media-adaptors>
7   </configuration-file>
8 </mfaces-config>
```

Para que sepa distinguir nuestra aplicación que archivo de configuración es para JSF y para Móvil JSF, en el descriptor web.xml se introducen los siguientes elementos.

```
<context-param>
      <param-name>javax.faces.CONFIG_FILES</param-name>
      <param-value>/WEB-INF/faces-config.xml</param-value>
   </context-param>

   <!--set the MobileFaces config file path-->
   <context-param>
      <param-name>ericsson.mobilefaces.CONFIG_FILE</param-name>
      <param-value>/WEB-INF/mfaces-config.xml</param-value>
   </context-param>

   <!--define the InitFilter for all MobileFaces application-->
   <filter>
      <filter-name>MobileFaces Init Filter</filter-name>
      <filter-class>com.ericsson.sn.mobilefaces.MobileFacesServletFilter</filter-class>
      <init-param>
        <description>Set to "true" to turn on the debug </description>
        <param-name>Debug</param-name>
        <param-value>true</param-value>
      </init-param>
   </filter>

   <filter-mapping>
      <filter-name>MobileFaces Init Filter</filter-name>
      <url-pattern>/*</url-pattern>
   </filter-mapping>
```

Por último sólo falta registrar nuestras nuevas etiquetas en el archivo de configuración faces-config.xml su contenido es:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC
"-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
"http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
  <!--Cool menu renderers for HTML page-->
  <render-kit>
    <!--Cool menu item renderer-->
    <renderer>
      <component-family>demo.render.MenuItem</component-family>
      <renderer-type>demo.render.MenuItem</renderer-type>
      <renderer-class>demo.render.renderer.HTMLItemRenderer</renderer-class>
    </renderer>
  </render-kit>

  <!--Cool menu renderers for XHTML-MP page-->
  <render-kit>
    <render-kit-id>XHTML_MP</render-kit-id>
    <!--Cool menu item renderer-->
    <renderer>
      <component-family>demo.render.MenuItem</component-family>
      <renderer-type>demo.render.MenuItem</renderer-type>
      <renderer-class>demo.render.renderer.XMPItemRenderer</renderer-class>
    </renderer>
```

```
  </render-kit>

  <!--Cool menu renderers for WML page-->
  <render-kit>
    <render-kit-id>WML</render-kit-id>
    <!--Cool menu item renderer-->
    <renderer>
      <component-family>demo.render.MenuItem</component-family>
      <renderer-type>demo.render.MenuItem</renderer-type>
      <renderer-class>demo.render.renderer.WMLItemRenderer</renderer-class>
    </renderer>
  </render-kit>


  <!--Cool menu component-->
  <component>
    <display-name>CoolMenu</display-name>
    <component-type>demo.render.CoolMenu</component-type>
    <component-class>demo.render.component.CoolMenu</component-class>
  </component>

  <!--Cool menu item component-->
  <component>
    <display-name>MenuItem</display-name>
    <component-type>demo.render.MenuItem</component-type>
    <component-class>demo.render.component.MenuItem</component-class>
  </component>
```

Ya podemos hacer uso de las etiquetas especiales para móviles, empecemos a crear la página inicial, hagamos un nuevo JSP llamado “menu.jsp”, y este lleva el siguiente código.

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <%@ taglib  uri="http://java.sun.com/jsf/core" prefix="f" %>
 4 <%@ taglib  uri="http://java.sun.com/jsf/html" prefix="h" %>
 5 <%@ taglib  uri="http://www.ericsson.com/sn/mobilefaces" prefix="m" %>
 6 <%@ taglib  uri="http://www.ericsson.com/sn/mobilefaces/demo" prefix="d" %>
 7 <f:view>
 8    <m:doctype/>
 9    <m:html>
10       <head>
11          <link href="styles.css" rel="stylesheet" type="text/css"/>
12          <title>Menu Principal</title>
13       </head>
14       <m:body>
15          <center>
16             <m:h2>Escoja la Carrera</m:h2>
17             <br/>
18             <h:outputText value="Computación" styleClass="emphasis"/><br/>
19             <d:coolMenu columns="1">
20                <d:menuItem itemHref="computacion.jsf" itemIcon="img/compu.jpg" itemLabel="Com" />
21             </d:coolMenu>
22             <br/>
23             <h:outputText value="Arquitectura" styleClass="emphasis"/><br/>
24             <d:coolMenu columns="1">
```

```
25              <d:menuItem itemHref="arquitectura.jsf" itemIcon="img/arq.jpg" itemLabel="Arq"/>
26            </d:coolMenu>
27          </center>
28        </m:body>
29    </m:html>
30 </f:view>
```

Las importantes diferencias que se encuentran aquí se muestran en la tabla siguiente:

JSP	Nuevo JSP
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">	<m:doctype/>
<html> & </html>	<m:html> & <m:html>
<body>& </body>	<m:body> & </m:body>
&	<m:h1> & </m:h1>
<%@taglib uri="http://www.ericsson.com/sn/mobilefaces" prefix="m" %>	Para poder ocupar las etiquetas.
<%@taglib uri="http://www.ericsson.com/sn/mobilefaces/demo" prefix="d" %>	Con esto ocupamos las nuevas etiquetas que traen como ejemplo Mobile JSF

Nosotros diseñamos nuestra parte de Vista en la página JSP normal con las etiquetas de JSF, pero ocupamos también las etiquetas para hacer menús con imágenes, éstas las traen como ejemplo Mobile JSF para que uno se dé cuenta de cómo podemos añadir nuevas etiquetas de acuerdo a nuestras necesidades, de las líneas 19 a la 21, se crea una columna con una imagen, para esto se crea un carpeta en Web Pages llamada “img”, donde pondremos las imágenes a mostrar, en el elemento <d:menuItem itemHref="computacion.jsf" itemIcon="img/compu.jpg" itemLabel="Com" /> estamos indicando donde se debe trasladar cuando el usuario seleccione la imagen “cumpu.jpg”, su enlace nos lleva a la página “computacion.jsf”, lo mismo pasa de la línea 24 a la 26. El código de la página computacion.jsf es el siguiente:

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <%@ taglib  uri="http://java.sun.com/jsf/core" prefix="f" %>
 4 <%@ taglib  uri="http://java.sun.com/jsf/html" prefix="h" %>
 5 <%@ taglib uri="http://www.ericsson.com/sn/mobilefaces" prefix="m"%>
 6 <f:view>
 7   <m:doctype/>
 8   <m:html>
 9     <title>Semestres</title>
10     <m:body>
11       <center>
12         <h:form id="f">
13           <h:outputText id="primero" value="Computacion" style="color: Red;
14                   font-style: italic"/>
15           <h:outputText value="Escoja el Semestre" style="color: Red;
```

```
16                      font-style: italic"/>
17            <p>
18              <h:messages/>
19            </p>
20            <p>
21              <h:selectOneListbox size="2" value="#{form.semestre}">
22                <f:selectItems value="#{form.semestres}"/>
23              </h:selectOneListbox>
24            </p>
25            <br/>
26            <br/>
27            <h:commandButton id="Enviar" type="submit" value="Enviar"
28                      action="#{form.revisarDatos}"
29                      actionListener="#{form.checarCarrera}"/>
30          </h:form>
31        </center>
32      </m:body>
33    </m:html>
34 </f:view>
```

Ahora el código de arquitectura.jsf:

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <%@ taglib  uri="http://java.sun.com/jsf/core" prefix="f" %>
 4 <%@ taglib  uri="http://java.sun.com/jsf/html" prefix="h" %>
 5 <%@ taglib uri="http://www.ericsson.com/sn/mobilefaces" prefix="m"%>
 6 <f:view>
 7   <m:doctype/>
 8   <m:html>
 9     <title>Semestres</title>
10     <m:body>
11       <center>
12         <h:form id="f">
13           <h:outputText id="arq" value="Arquitectura" style="color: Red;
14                      font-style: italic" />
15           <h:outputText value="Escoja el Semestre" style="color: Red;
16                      font-style: italic"/>
17           <p>
18             <h:messages/>
19           </p>
20           <p>
21             <h:selectOneListbox size="2" value="#{form.semestre}">
22               <f:selectItems value="#{form.semestres}"/>
23             </h:selectOneListbox>
24           </p>
25           <br/>
26           <br/>
27           <h:commandButton id="Enviar" type="submit" value="Enviar"
28                      action="#{form.revisarDatos}"
29                      actionListener="#{form.checarCarrera}" />
30         </h:form>
31       </center>
32     </m:body>
33   </m:html>
34 </f:view>
```

Estas dos páginas que son la Vista de la aplicación Web, solo muestran el título de lo que debe hacer el usuario y una lista para seleccionar una opción que es el semestre a consultar, y por ultimo un botón para llevar la información al servidor, notemos los siguientes puntos:

- Las líneas 13 de los dos JSP se les puso un identificador “id” a los componentes de texto de salida, esto nos sirve para poder saber que tabla de la base de datos quieren consultar, ya sea para la tabla de la carrera de computación o la de arquitectura.
- Líneas 21,23, sacamos una caja de lista, que solo se puede seleccionar una opción, que se guarda en la variable privada “semestre” del Bean “form” (veremos su código más adelante), ese indica el semestre que quieren consultar, el llenado de la caja de lista se hace a través de la etiqueta <f:selectItems value="#{form.semestres}"/>, que llama al método “getSemetres” del Bean y este regresa una Lista para rellenar la caja de lista.
- Líneas 27,29 indicamos que se envié la información, el atributo “action” es el encargado de llamar al método revisarDatos del Bean, con esto verificamos si por lo menos escogió un semestre, y el atributo “actionListener”, dispara un evento que a su vez llama al método “checarCarrera” con ello nosotros sabemos que base de datos tenemos que consultar, de acuerdo a la página donde esta el usuario, como JSF tiene la cualidad de ver que componentes estas activos según al contexto donde estemos, el “id” que le pusimos a la caja de texto de salida de cada página ya sea de arquitectura o computación, podremos preguntar a JSF que nos regrese el identificador del componente de la página activa, y con esto sabemos que consulta quieren realizar.

La última Vista que falta es la salida de datos de la consulta que se hace, el JSP que se encarga de esto se llama “resultados.jsp” su código es:

```
1 <%@page contentType="text/html"%>
 2 <%@page pageEncoding="UTF-8"%>
 3 <%@ taglib  uri="http://java.sun.com/jsf/core" prefix="f" %>
 4 <%@ taglib  uri="http://java.sun.com/jsf/html" prefix="h" %>
 5 <%@ taglib uri="http://www.ericsson.com/sn/mobilefaces" prefix="m"%>
 6 <f:view>
 7    <m:doctype/>
 8    <m:html>
 9       <link href="styles.css" rel="stylesheet" type="text/css"/>
10       <title>Horarios</title>
11       <m:body>
12          <center>
```

```
13          <h:form>
14            <h:dataTable styleClass="names"
15                    headerClass="namesHeader"
16                    columnClasses="last,first" value="#{form.datos}" var="nom">
17              <h:column>
18                <h:outputText value="CMAT: "/>
19                <h:outputText value="#{nom.cmat}" style="color: Blue;
20                        font-style: italic" />
21              </h:column>
22              <h:column>
23                <h:outputText value="Materia: "/>
24                <h:outputText value="#{nom.materia}" style="color: Blue;
25                        font-style: italic"/>
26              </h:column>
27              <h:column>
28                <h:outputText value="Profesor: "/>
29                <h:outputText value="#{nom.profesor}" style="color: Blue;
30                        font-style: italic"/>
31              </h:column>
32              <h:column>
33                <h:outputText value="Grupo: "/>
34                <h:outputText value="#{nom.grupo}" style="color: Blue;
35                        font-style: italic"/>
36              </h:column>
37              <h:column>
38                <h:outputText value="Dias: "/>
39                <h:outputText value="#{nom.dias}" style="color: Blue;
40                        font-style: italic"/>
41              </h:column>
42              <h:column>
43                <h:outputText value="Horas: "/>
44                <h:outputText value="#{nom.horas}" style="color: Blue;
45                        font-style: italic"/>
46              </h:column>
47              <h:column>
48                <h:outputText value="Salon: "/>
49                <h:outputText value="#{nom.salon}" style="color: Blue;
50                        font-style: italic"/>
51              </h:column>
52              <h:column>
53                <h:outputText value="----------------------"
54                        style="color: Red;
55                        />
56              </h:column>
57            </h:dataTable>
58          </h:form>
59          <a href="computacion.jsf">Menu Semestre</a>
60          <a href="menu.jsf">Menu principal</a>
61        </center>
62      </m:body>
63    </m:html>
64 </f:view>
```

Ocupamos la etiqueta <h:data Table>, su explicación de cómo se utiliza esta en el capitulo 5, para mayor información. En todos ocupamos hoja de estilo para darle formato de salida al texto

mediante la etiqueta <link href="styles.css" rel="stylesheet" type="text/css"/>, esta se encuentra en Web Pages, su código es:

```
1 body {
 2    background: #eee;
 3 }
 4 .emphasis {
 5    font-size: 1.3em;
 6    font-style: italic
 7
 8 .enabledColors {
 9    font-size: 1.5em;
10    font-style: italic;
11    color: Blue;
12    background: Yellow;
13 }
14 .disabledColors {
15    font-size: 0.8em;
16    font-style: italic;
17    color: Yellow;
18    background: Blue;
19 }
20 .names {
21    border: thin solid black;
22 }
23 .namesHeader {
24    text-align: center;
25    font-style: italic;
26    color: Snow;
27    background: Teal;
28 }
29 .last {
30    height: 25px;
31    text-align: center;
32    background: MediumTurquoise;
33 }
34 .first {
35    text-align: left;
36    background: PowderBlue;
37 }
```

Ahora expliquemos el Modelo que debe seguir nuestra aplicación Web, este lleva la lógica del negocio, primero hagamos una paquete que se llame “base_datos”, en el creamos dos clases una llamada “Campos” y la otra “Conexión”, Campos guardar los valores arrojados por la base de datos, que son los siguientes:

- § cmat: matricula de la materia.
- § materia: nombre de la materia
- § profesor: nombre del profesor.

- grupo: grupo donde se encuentra la materia.
- días: días que se imparte la materia.
- horas: horas que se da la materia.
- salón: salón donde se imparte.

Y ésta sólo contiene métodos set y get para recoger y cambiar los valores de las variables privadas de la clase, y el constructor que recoge los valores de inicio para el estado de la variable objeto que se cree, su código es:

```
1 package base_datos;
 2 public class Campos {
 3   private String cmat;
 4   private String materia;
 5   private String profesor;
 6   private String grupo;
 7   private String dias;
 8   private String horas;
 9   private String salon;
10    public Campos(String a1,String a2,String a3,String a4,String a5,
11        String a6,String a7) {
12      this.setCmat(a1);
13      this.setMateria(a2);
14      this.setProfesor(a3);
15      this.setGrupo(a4);
16      this.setDias(a5);
17      this.setHoras(a6);
18      this.setSalon(a7);
19    }
20    public String getCmat() {
21      return cmat;
22    }
23    public void setCmat(String cmat) {
24      this.cmat = cmat;
25    }
26    public String getMateria() {
27      return materia;
28    }
29    public void setMateria(String materia) {
30      this.materia = materia;
31    }
32    public String getProfesor() {
33      return profesor;
34    }
35    public void setProfesor(String profesor) {
36      this.profesor = profesor;
37    }
38    public String getGrupo() {
39      return grupo;
40    }
41    public void setGrupo(String grupo) {
42      this.grupo = grupo;
```

```
43   }
44   public String getDias() {
45     return dias;
46   }
47   public void setDias(String dias) {
48     this.dias = dias;
49   }
50   public String getHoras() {
51     return horas;
52   }
53   public void setHoras(String horas) {
54     this.horas = horas;
55   }
56   public String getSalon() {
57     return salon;
58   }
59   public void setSalon(String salon) {
60     this.salon = salon;
61   }
62 }
```

La clase Conexión es la misma que hemos ocupado a lo largo de todos los ejemplos de esta tesis, lo único que le agregamos es al constructor que reciba dos argumentos uno es para indicar el "semestre" y el otro la "carrera" que nos sirve para ver que tabla de la base de datos consultaremos, y un método "getDatos" que arroja un arreglo de objetos de tipo de la clase Campos para rellenar la tabla del JSP "resultados", el código es el siguiente:

```
1 package base_datos;
 2 import java.sql.*;
 3 import java.io.*;
 4 import java.util.*;
 5 public class Conexion {
 6   private Connection conexion;
 7   private Statement instruccion;
 8   private Campos[] datos=null;
 9   private String tabla=null;
10   public Conexion(String semestre,String carrera)throws Exception{
11     Class.forName("com.mysql.jdbc.Driver");
12     conexion=DriverManager.getConnection("jdbc:mysql://localhost:3306/horarios",
13         "root","mike97");
14     //creando Statement para la consulta BD
15     instruccion=conexion.createStatement();
16     ResultSet resultados=instruccion.executeQuery("SELECT cmat,materia," +
17         "profesor,grupo,dias,horas,salon FROM "+carrera+" where semestre="+
18           "'"+semestre.toString()+"'");
19     int indice=1;
20     while(resultados.next()){
21       indice++;
22     }
23     resultados.first();
24     resultados.previous();
25     datos=new Campos[indice-1];
26     int i=0;
```

```
27      while(resultados.next()){
28        datos[i]=new Campos(resultados.getString(1),resultados.getString(2),
29            resultados.getString(3),resultados.getString(4),resultados.getString(5),
30            resultados.getString(6),resultados.getString(7));
31        ++i;
32      }
33      try {
34        instruccion.close();
35        conexion.close();
36      }
37      // procesar posible excepcion SQLException en operación de cierre
38      catch ( SQLException excepcionSQL ) {
39        excepcionSQL.printStackTrace();
40      }
41    }
42    public Campos[] getDatos(){
43      return datos;
44    }
45 }
```

El Bean encargado de juntar, llamar y recoger valores según las opciones escogidas por el usuario en la parte de Vista es “RegistroForm”, este lo creamos dentro de un paquete llamado “formularios”, su código es:

```
1 package formularios;
 2 import base_datos.Conexion;
 3 import base_datos.Campos;
 4 import java.util.ArrayList;
 5 import java.util.List;
 6 import javax.faces.component.UIComponent;
 7 import javax.faces.component.UIOutput;
 8 import javax.faces.model.SelectItem;
 9 import javax.faces.event.ActionEvent;
10 public class RegistroForm {
11    private Integer semestre;
12    private Conexion datos=null;
13    private String carrera;
14    public Integer getsemestre(){
15      return semestre;
16    }
17    public void setsemestre(Integer arg){
18      this.semestre=arg;
19    }
20    public List getSemestres(){
21      return reg_semestre;
22    }
23    //revicion de la opcion que se escoja
24    public String revisarDatos() throws Exception{
25      if(semestre.equals(null))
26        return "nopaso";
27      else{
28        datos=new Conexion(semestre.toString(),carrera);
29        return "paso";
30      }
31    }
32    //checar la carrera que se consultara
```

```
33   public void checarCarrera(ActionEvent evento){
34      UIComponent componente=evento.getComponent().findComponent("primero");
35      if (componente!=null)
36         carrera="primero";
37      else
38         carrera="arq";
39   }
40   //obtener los datos de la B.D
41   public Campos[] getDatos(){
42      return datos.getDatos();
43   }
44   private static List reg_semestre;
45   //llenando los item del la caja de lista
46   static{
47      reg_semestre=new ArrayList();
48      for(int i=1;i<10;i++){
49         reg_semestre.add(new SelectItem(new Integer(i)));
50      }
51   }
52 }
```

El método revisarDatos(), se encarga de ver si el usuario por lo menos escogió una opción en la caja de lista si no es así se regresará una cadena que se ocupará en faces-config.xml como navegador dinámico que lo manda a la página principal de la Vista, o la de resultados, si se escogió el semestre que se consultará, se crea la conexión de la base de datos, pasando el semestre y la carrera deseada, el método checarCarrera() obtiene el identificador del componente que se le pide, con esto sabemos qué carrera se consulta, y por último, el método getDatos(), arroja un arreglo de objetos de la clase Campos, con esto rellenamos nuestra tabla del JSP "resultados". Nuestro archivo de configuración de JSF faces-config.xml le tenemos que añadir los siguientes elementos para que realice la navegación dinámica deseada.

```
<managed-bean>
    <managed-bean-name>form</managed-bean-name>
    <managed-bean-class>formularios.RegistroForm</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
  <navigation-rule>
    <from-view-id>/computacion.jsp</from-view-id>
    <navigation-case>
      <from-outcome>paso</from-outcome>
      <to-view-id>/resultados.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>nopaso</from-outcome>
      <to-view-id>/menu.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
```

```
<navigation-rule>
  <from-view-id>/arquitectura.jsp</from-view-id>
  <navigation-case>
    <from-outcome>paso</from-outcome>
    <to-view-id>/resultados.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>nopaso</from-outcome>
    <to-view-id>/menu.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Sólo falta indicar que para nuestra página de inicio sea el JSP menu.jsp hacemos lo siguiente, en Projects señalamos al proyecto horarios y le damos clic con el botón derecho y nos vamos a Properties, y en Categories vamos a Run, y en Relative URL ponemos menu.jsf y corremos la aplicación, a continuación se muestran las imágenes ocupando el emulador Openwave.


Default Device ...
File Edit View Settings Help
Go http://localhost:8084/horarios/
Menu Principal
Escoja la Carrera
Computación
Select Options
Default Device - O...
Go http://localhost:8084/horarios/com
Semestres
Computacion Escoja el Semestre
1
2
Enviar
Edit Options
Default Device - O...
Go http://localhost:8084/horarios/com
Horarios
CMAT: 19
Materia:
ADMINISTRACION CONTA. Y COSTOS
Profesor: IBARRA ROSAS ALBERTO
Grupo: 2207
Options


Default Device ...
File Edit View Settings Help
Go http://localhost:8084/horarios/
Menu Principal
Arquitectura
Select Options
Default Device - O...
Go http://localhost:8084/horarios/arqu
Semestres
Arquitectura Escoja el Semestre
1
2
Enviar
Edit Options
Default Device - O...
Go http://localhost:8084/horarios/arqu
Horarios
CMAT: 1121
Materia:
COMPOSICION I
Profesor: ALVAREZ MARTINEZ MARIA ESTHER
Grupo: 2101
Options

Conclusiones

Hoy día, cada vez son más estrictas las cosas en cuanto a la calidad del desarrollo de las aplicaciones. Las exigencias parten desde la documentación, el manejo de estándares, la programación de aplicaciones con patrones de diseño.

Usando Java EE, podríamos decir que el programador tiene garantizada la solución de la portabilidad, pero falta hacer una interfaz de usuario sencilla, entonces basta con poder escoger un buen Framework MVC que desacople la lógica de negocios de la capa de Vista.

Utilizar patrones de diseño es un punto fundamental para lograr una buena calidad en el desarrollo de software, porque cuando se realiza el diseño de aplicaciones con patrones se logra no sólo un código fuente más legible, sino también un mantenimiento a menor costo, mayor facilidad para encontrar errores y sobre todo si se utiliza bien permitimos optimizar el rendimiento de los programas.

Aquí partimos desde explicar qué es un Framework, analizar el patrón MVC, e ir introduciéndonos a las tecnologías como son Servlets y JSPs, cada una de estas tecnologías aportan una solución parcial a los problemas que tenemos que resolver, y a su vez dan pauta para que nuevas tecnologías surjan para resolver los problemas que presentan las anteriores.

Tenemos el caso de Struts que logró adaptarse más al patrón MVC, logrando así poder estructurar cada unos de sus componentes por partes, la parte del Modelo que llevara la lógica del negocio, la Vista que se encarga de mostrar los resultados al usuario y por último, la de Control que llevara el flujo de la aplicación, con este avance se puede hacer grupos de personas que trabajen con cada capa del patrón MVC, para que al final se logre acoplar todas las partes de la aplicación Web.

Incluso se tiene la posibilidad de adaptar nuestra aplicación a diferentes tipos de lenguajes según tenga configurado el navegador el usuario, la desventaja principal que se vio con Struts que no es

fácil hacer el flujo de la aplicación o la interacción entre los diferentes componentes para pasar de un Bean a otro la información. Esta desventaja la tomó JavaServer Faces y la mejoró, también tomo las características de Struts con sus diferentes etiquetas ya diseñadas para facilitar el diseño de páginas JSP que favorecen la creación de la Vista en el patrón MVC. Incluso fue más allá ya que sus componentes pasan a un proceso de renderer, que no es otra cosa que transformar sus valores a diferentes tipos de nuevos componentes, logrando así que el programador pueda crear nuevas etiquetas y adaptarlas a nuevos dispositivos de naturaleza diferente como un dispositivo móvil.

El Framework de JSF permite que la navegación dinámica sea más sencilla, el manejo de los Beans es muy natural a como está acostumbrado uno en Java, tiene una gran variedad de etiquetas para el diseño de las páginas en JSP, además la línea de aprendizaje es mas rápida que la del Framework de Struts.

Se puede hacer notar que teniendo las librerías de Mobile JSF, se tuvo una funcionalidad extra para JSF que fue la comunicación para dispositivos móviles, esto abre un mundo de posibilidades entre la comunicación de las aplicaciones Web, ya que no necesitaremos una PC o Laptops que soporten WAP, ya que un celular que tenga un navegador con WAP 1 o WAP 2 puede tener acceso a la información y la creación de la aplicación es fácil y no tediosa.

Se puede decir que Struts tiene madurez y estabilidad que es su punto fuerte ante JSF, pero como el creador de Struts (Craig McClanahan), fue líder en la creación de las especificaciones para JSF, este adopto las funcionalidades de Struts y las mejoró, prueba de ello es su alto grado de integración e extensibilidad que se tiene, ante la utilización de las librerías de Mobile Faces.

Bibliografía

Libros

- **CAY S. HORSTAMANN,GARY CORNELL,** *"Core Java 2 Volumen 1-fundamentos", Prentice-Hall, Séptima Edición.*
- **CAY S. HORSTAMANN,GARY CORNELL,** *"Core Java 2 Volumen 2-fundamentos", Prentice-Hall, Séptima Edición.*
- **DEITEL,** *"Como Programar en Java", Prenticel-Hall, Quinta Edición.*
- **JIM KEOGH,** *"Manual de Referencia, J2EE",Mc Graw Hill.*
- **ANDRES MARTINEZ QUIJANO,** *"Programación Web en Java",M-P.*
- **DAVID GEARY, CAY HORSTMANN,** *"Core JavaServer Faces", Prentice-Hall.*
- **CRAIG LARMAN,** *"UML Y PATRONES",Prentice-Hall, Segunda Edición.*
- **ALONSO RODRÍGUEZ ZAMORA,** *"Publicaciones en Internet y tecnología XML", Alfaomega.*
- **AGUSTÍN FROUFE QUINTAS, PATRICIA JORGE CARDENES,** *"J2ME Manual de usuario y tutorial", Alfaomega.*
- **CESAR PEREZ,** *"MySQL para Windows y Linux", Alfaomega.*
- **JESÚS BOBADILLA, SANTIAGO ALONSO,** *"HTML Dinámico a través de ejemplo", Alfaomega.*

Revistas

- **User .Code #11**, *"J2EE: El furor de Java Enterprise",Ramiro González Maciel, Páginas 16-28, Un informe detallado de las tecnologías J2EE, su estado actual y hacia dónde se dirigen.*
- **User .Code #14**, *"Interfaces Web con Struts",Pablo Franceskin, Páginas 42-46, Trabajando con este Framework Open Source para JSP y Servlet.*
- **User .Code #15**, *"El Rompecabezas de J2ME",Maximiliano R. Firtman, Páginas 44-48, Análisis de todas las APIs que forman parte de Java para móviles.*

- **User .Code #26**, *"Struts",José Jaliff, Páginas 34-36, Presentación de uno de los Framework más usado.*

Referencias en Internet

- *http://java.sun.com/javaee/* Sitio oficial de Java 2 Enterprice Edition.
- *http://struts.apache.org/* Sitio oficial de Struts.
- *http://developers.sun.com/jscreator/reference/techart/2004q2/jsf/index.html* Documentación sobre JavaServer Faces.
- *http://jakarta.apache.org/* Sitio oficial de Jakarta Tomcat.
- *http://mysql.com/* Sitio oficial de MySQL.
- *http://www.ericsson.com/mobilityworld/sub/open/technologies/open_development_tips/tools/mobile_jsf_kit/* Sitio Oficial de Ericsson Mobile JSF kit.
- *http://developer.openwave.com/dl/Openwave_v70_Simulator.exe/* Sitio oficial de Openwave, un manejador móvil.
- *http://www.adictosaltrabajo.com/tutoriales/* Sitio donde se encuentra tutoriales sobre Servlets, JSPs, Struts y JSF.
- *http://www.horstmann.com/corejsf/* Sitio donde se baja los programas ejemplo del libro "Core JavaServer Faces".
- *http://www.programacion.net/java/* Sitio donde se encuentra tutoriales sobre todo referente a Java.
- *http://www.javahispano.org/* Sitio para obtener tutoriales de Java.
- *http://www.mygnet.net/articulos/j2me/97/* Articulo que explica como empezar con J2ME.
- *http://es.wikipedia.org/* Enciclopedia virtual on-line, de información general.
- *http://www.netbeans.org/* Sitio oficial de Netbeans.
- *http://www.netbeans.org/kb/articles/jAstrologer-intro.html/* Articulo sobre una introducción con JSF a través de Netbeans.
- *http://www.desarrolloweb.com/articulos/2392.php/* Articulo sobre como funciona JSF.