Diseño e implementación de un framework de presentación

Enrique Mengíbar Vázquez
Ingeniería Informática 2.º ciclo

Director: Oscar Escudero Sánchez
Universitat Oberta de Catalunya

Barcelona, enero de 2012

Agradecimientos

Quiero dar mi primer agradecimiento a mis padres, por haber confiado siempre en mí, sobre todo cuanta mas falta me hizo.

A Mila por su paciencia y comprensión conmigo durante estos últimos años, sin su apoyo no lo hubiera conseguido.

Por último a la memoria de mi buen amigo Emilio, ojalá estuvieras aquí para compartir conmigo este momento.

Resumen ejecutivo

El objetivo de este proyecto es la construcción de un framework de presentación para el desarrollo de aplicaciones Web basadas en la plataforma J2EE. El proyecto comprende el estudio de las características de los frameworks más importantes disponibles en el mercado, prestando una atención especial a su arquitectura. Finalmente se ha realizado el diseño e implementación de un framework que reúne algunas de las características estudiadas. El framework construido, denominado SOFP, proporciona una arquitectura base basada en el patrón MVC y en los patrones de diseño Service to Worker (Front Controller y Application Controller), así como en recomendaciones reconocidas de buenas prácticas de la industria. La implementación se ha realizado utilizando tecnologías estándar, abiertas y open source.

Contenido

Lista de figuras

Lista de tablas

Capítulo 1. Introducción

1.1 Justificación y contexto del presente proyecto

En la actualidad, la plataforma J2EE se encuentra sin duda fuertemente consolidada en el mercado de desarrollo de aplicaciones empresariales. Todo ello sin duda gracias a que se trata de un sistema robusto, multi-plataforma, abierto, interoperable, escalable, bien documentado y con una amplia comunidad de desarrolladores que lo apoyan. También contribuye a su éxito que esté basado en los principios de diseño de la programación orientada a objetos y en el lenguaje de programación Java, un lenguaje consistente, multi-plataforma que permite el desarrollo de aplicaciones de forma eficiente.

Una de las partes más importantes de una aplicación web compleja en el entorno J2EE, es la capa de presentación. Para este tipo de aplicaciones el patrón *Modelo-Vista-Controlador* (MVC) se ha convertido en un estándar. Este patrón describe una arquitectura de diseño de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos de forma que las modificaciones al componente de la vista pueden ser hechas con un mínimo impacto en el componente del modelo de datos. Este patrón se ha utilizado ampliamente en el desarrollo de aplicaciones web, donde el controlador se implementa mediante un componente ejecutado en un contenedor Web o servidor de aplicaciones, denominado servlet que actúa como intermediario entre la vista y el modelo. Las vistas, se construyen mediante la tecnología JSP (*Java Server Pages*), que por medio de un conjunto de etiquetas XML nos proporcionaba un interfaz sencillo para el acceso a clases Java y a objetos proporcionados por el servidor de aplicaciones.

La implementación de aplicaciones exclusivamente con el uso de estas tecnologías ha provocado que las aplicaciones tengan su lógica dispersa entre los diferentes elementos de control y la vista; por consiguiente, el código generado presentaba el problema de ser complejo y difícil de mantener.

En el año 2001 el Centro Java de Sun hizo público un catálogo de patrones conocido como *Core J2EE Patterns*. Estos patrones describen problemas típicos a los que se enfrentan los desarrolladores de aplicaciones empresariales proporcionando soluciones de probada

efectividad. En esencia, estos patrones contienen las mejores soluciones para ayudar a los desarrolladores a diseñar y construir aplicaciones para la plataforma J2EE.

La presentación de estos patrones como complemento al tradicional MVC y la experiencia acumulada en el desarrollo de aplicaciones empresariales dentro del mundo J2EE auspició el nacimiento de diversos frameworks de desarrollo dentro del mundo J2EE. Un framework o marco de desarrollo no es más que un conjunto de librerías y componentes utilizados para implementar la estructura estándar de una aplicación que junto con una documentación y metodología de uso nos permite diseñar, construir e implantar aplicaciones empresariales de forma uniforme, rápida y de mayor calidad.

El objetivo de este trabajo es el diseño e implementación de un framework para la capa de presentación J2EE que proporcione al desarrollador los beneficios ya comentados de reutilización y simplificación en el desarrollo. Asimismo se acompañará el framework de una aplicación de ejemplo que mostrará su funcionamiento.

Previamente, con la idea de identificar las características básicas a incluir y por otra parte, para conocer la situación actual, se realizará un estudio de buenas prácticas y patrones de diseño así como una evaluación de los diferentes frameworks del mercado, estudiando sus diferentes características y arquitecturas.

1.2 Objetivos generales y específicos

El objetivo general de este trabajo es el desarrollo de un framework para la capa de presentación, aplicando patrones de diseño y siguiendo las buenas prácticas y recomendaciones reconocidas por la industria. El objetivo general se ha dividido en tres objetivos específicos.

Estudio de lo los frameworks disponibles en el mercado y de sus principales características con el objeto de detectar cual debe ser la funcionalidad soportada por el framework, al que hemos denominado SOFP (*Si Otro Framework de Presentacion*), cuyo diseño y construcción es uno de los objetivos de este proyecto. En este estudio se analizarán la evolución de los diseños de la capa de presentación MVC Model 1 y Model 2, así como los distintos patrones de diseño J2EE propuestos por Sun conocidos como Core J2EE Patterns.

Una vez fijados los requisitos, se realizará el diseño e implementación del framework SOFP. Se aplicarán los patrones de diseño y recomendaciones de buenas prácticas que se estimen oportunos estudiadas en el apartado anterior.

Finalmente, con el objeto de validar el trabajo realizado se programará una aplicación simple que servirá de ejemplo para ilustrar el funcionamiento del framework.

1.3 Estudio de posibles alternativas, metodología y solución adoptada.

A continuación se describe cual ha sido la metodología seguida durante el desarrollo del presente trabajo, así como el estudio de las alternativas y la solución final adoptada.

1.3.1 Metodología seguida a lo largo del proyecto

El método de trabajo que se ha seguido para el desarrollo de este trabajo se puede encuadrar dentro del tradicional modelo en cascada descrito en las técnicas de la Ingeniería del Software. Este modelo se ha adaptado bien al método de trabajo marcado por el plan docente de la asignatura, el cual planifica el desarrollo del trabajo en la elaboración de un plan de trabajo inicial (PEC1) junto con dos prácticas de evaluación continua (PEC2 y PEC3), cuyo propósito es permitir realizar un seguimiento de la evolución del trabajo por parte del consultor de la asignatura.

El principal problema del modelo en cascada es que los errores de cada fase se detectan cuando se concluye cada fase, lo que puede suele suponer un impacto en la planificación temporal importante; en nuestro caso, entendemos que los posibles errores detectados en cada entrega no deberían suponer un impacto sobre el proyecto, ya que gracias a las entregas parciales de la evaluación continua y a las sugerencias del consultor, es posible corregir los errores cometidos de cara a la elaboración de la memoria final.

En primer lugar, enmarcado en la primera Prueba de Evaluación Continuada, se ha establecido un plan de trabajo. En este plan, se ha establecido la temporalización del proyecto (coincidiendo con las recomendaciones de la UOC) y la definición del proyecto.

Las fases tradicionales por las que debe transcurrir un desarrollo según el modelo en cascada son las siguientes:

1. Análisis de requisitos
2. Diseño del sistema
3. Diseño del programación
4. Codificación
5. Pruebas
6. Implantación

Las peculiaridades a la hora de aplicar este modelo han sido que en el análisis de requisitos no ha intervenido el usuario o cliente, sino que ha sido marcado por el estudio realizado de los frameworks disponibles. La documentación ha consistido en la elaboración de esta memoria y que la implantación y aceptación se pueden equiparar a la evaluación final de este trabajo realizada por el tribunal.

Con este método la salida de cada una de las partes será la entrada de la siguiente. Como ya hemos comentado, el problema de este método es que los errores de cada fase se detectarán al final, lo que puede suponer un impacto temporal importante; dado que los requisitos son suficientemente claros, entendemos que los posibles errores detectados en cada entrega no deberían suponer un impacto importante sobre el proyecto.

1.3.2 Estudio de alternativas

En lo que se refiere al estudio de las alternativas, en este caso forman parte del enunciado y especificaciones del presente proyecto desde el inicio. Como ya se ha comentado, parte del trabajo desarrollado en la segunda práctica de evaluación continua ha consistido en un estudio de los distintos frameworks, junto con sus características y arquitecturas. Este trabajo ha servido de base valorar las diferentes alternativas y determinar el enfoque final del proyecto en cuanto a la elección del tipo de framework a desarrollar.

1.3.3 Solución adoptada

La solución adoptada final ha consistido en el desarrollo de un framework para la capa de presentación J2EE con las siguientes características: control declarativo del flujo de navegación, facilidades para la validación de formularios en el lado del servidor, librería de etiquetas personalizada para el manejo de mensajes en la internacionalización de la aplicación y gestión de excepciones.

1.4 Planificación del proyecto

La planificación del proyecto ha venido marcada por la planificación definida en el plan docente de la asignatura, el cual planifica el desarrollo del trabajo en tres prácticas de evaluación continua (PEC), cuyo propósito es permitir realizar un seguimiento de la evolución del trabajo por parte del responsable de la asignatura.

A continuación se indican cuales han sido los hitos marcados, junto con su contenido, planificación temporal y entregable.

Plan de trabajo – PEC1 – del 23/09/2011 al 05/10/2011

Contenido

- Identificación del problema a resolver
- Definición concreta del trabajo
- Descomposición del trabajo en tareas e hitos temporales.

Entregable

- Plan de trabajo

Análisis y Diseño - PEC2 – del 06/10/2011 al 10/11/2011

Contenido

- Estudio de la plataforma J2EE
- Patrones de diseño y patrones J2EE
- El concepto de framework
- Frameworks Struts, Struts 2, Spring y Java Server Faces
- Especificación de requisitos

Entregable

- Análisis y diseño del framework

Implementación – PEC3 – del 11/11/2011 al 19/12/2011

Contenido

- Especificación del frameworks
- Funcionamiento y características del framework
- Implementación del framework
- Aplicación de demostración
- Manual de despliegue

Entregable

- Diseño e implementación del framework

Entrega final: 16 de Enero de 2011

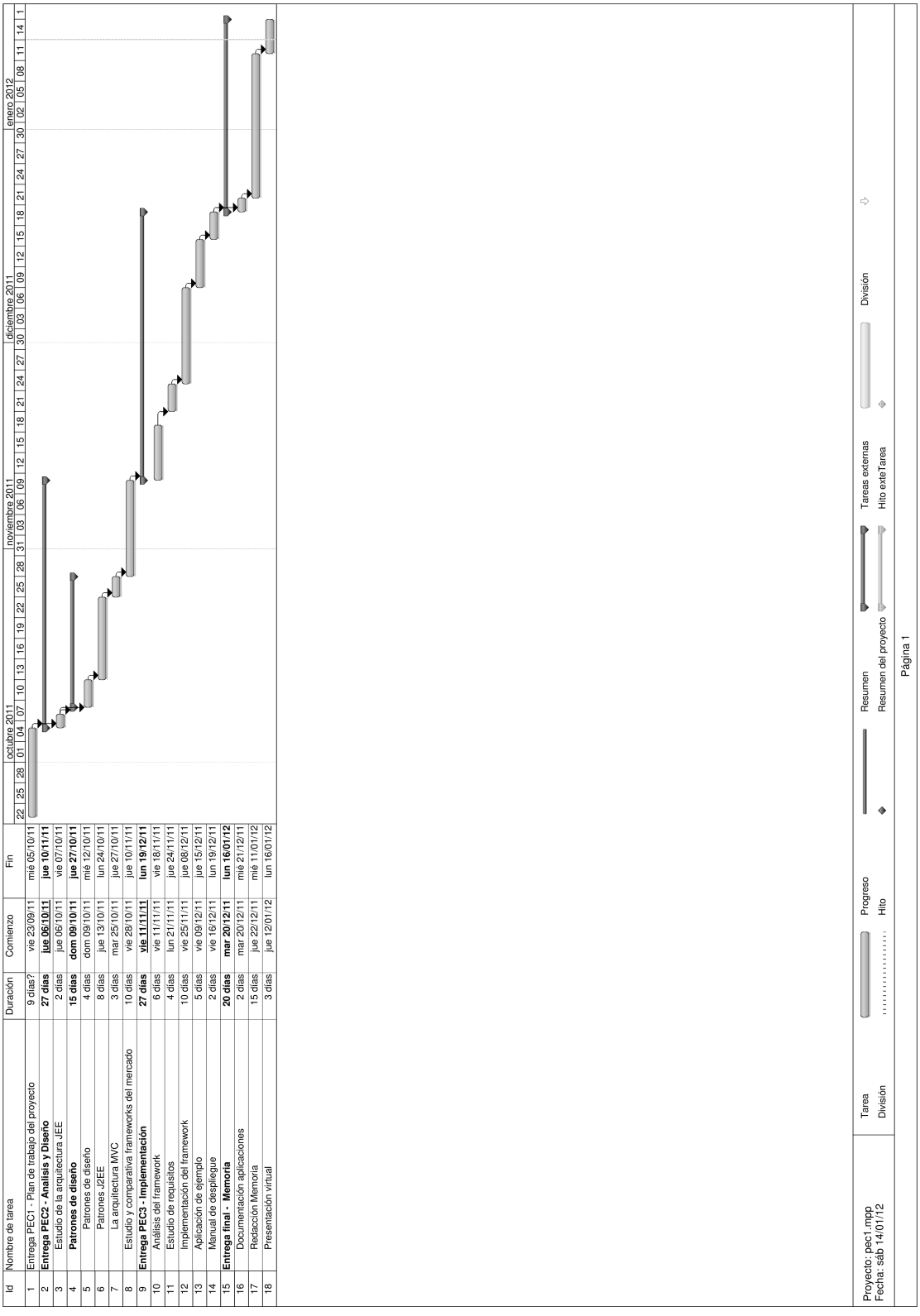
Contenido

- Redacción de la memoria
- Presentación virtual (documento de presentación)

Entregable

- Código del framework
- Aplicación de pruebas
- Memoria del proyecto
- Presentación del proyecto.

La Figura 1.1 muestra el diagrama de Gantt correspondiente a las tareas e hitos establecidos en este trabajo.



Id	Nombre de tarea	Duración	Comienzo	Fin
1	Entrega PEC1 - Plan de trabajo del proyecto	9 días?	vie 23/09/11	mié 05/10/11
2	**Entrega PEC2 - Analisis y Diseño**	**27 días**	**jue 06/10/11**	**jue 10/11/11**
3	Estudio de la arquitectura JEE	2 días	jue 06/10/11	vie 07/10/11
4	**Patrones de diseño**	**15 días**	**dom 09/10/11**	**jue 27/10/11**
5	Patrones de diseño	4 días	dom 09/10/11	mié 12/10/11
6	Patrones J2EE	8 días	jue 13/10/11	lun 24/10/11
7	La arquitectura MVC	3 días	mar 25/10/11	jue 27/10/11
8	Estudio y comparativa frameworks del mercado	10 días	vie 28/10/11	jue 10/11/11
9	**Entrega PEC3 - Implementación**	**27 días**	**vie 11/11/11**	**lun 19/12/11**
10	Análisis del framework	6 días	vie 11/11/11	vie 18/11/11
11	Estudio de requisitos	4 días	lun 21/11/11	jue 24/11/11
12	Implementación del framework	10 días	vie 25/11/11	jue 08/12/11
13	Aplicación de ejemplo	5 días	vie 09/12/11	jue 15/12/11
14	Manual de despliegue	2 días	vie 16/12/11	lun 19/12/11
15	**Entrega final - Memoria**	**20 días**	**mar 20/12/11**	**lun 16/01/12**
16	Documentación aplicaciones	2 días	mar 20/12/11	mié 21/12/11
17	Redacción Memoria	15 días	jue 22/12/11	mié 11/01/12
18	Presentación virtual	3 días	jue 12/01/12	lun 16/01/12

Figura 1.1: Diagrama de Gantt correspondiente a la planificación del PFC

1.5 Productos obtenidos

Los productos obtenidos en el presente trabajo han sido los siguientes:

- Memoria del proyecto
- Presentación virtual (documento de presentación)
- Producto
    - Framework SOFP: código fuente, librería JAR, documentación, manuales.
    - Aplicación de pruebas: código fuente, fichero de despliegue WAR.

Capítulo 2. La plataforma JEE

Java EE (Java Platform, Enterprise Edition) es una puesta al día de la tecnología conocida anteriormente como Java 2 Platform, Enterprise Edition o J2EE. La plataforma JEE es una colección de especificaciones que definen una infraestructura para desarrollar aplicaciones distribuidas multicapa.

2.1 Características

La plataforma soluciona varios problemas comunes en el desarrollo de aplicaciones empresariales, como por ejemplo la distribución, transaccionalidad, portabilidad y control de la seguridad. Su principal objetivo es simplificar el trabajo de los desarrolladores.

Algunas de las novedades introducidas por la nueva especificación JEE:

- Introduce un modelo simplificado de programación
- Uso de XML y Anotaciones
- Programación basada en POJOs (Plain Old Java Objects)
- Inyección de dependencias. El contenedor JEE automáticamente inyecta referencias a otros componentes requeridos
- Ofrece un nuevo API de persistencia (JPA)
- Provee una correspondencia objeto/relación para manejar datos relacionales en beans empresariales, componentes web y clientes

La plataforma ofrece un marco y una serie de convenciones, junto un conjunto de servicios sobre los cuales desarrollar aplicaciones multicapa. Esto permite al desarrollador centrarse en el diseño e implementación del sistema, delegando las tareas típicas y cuestiones de más bajo nivel ajenas a la propia aplicación a la infraestructura del servidor de aplicaciones JEE.

Las aplicaciones JEE tienen una arquitectura dividida en capas: una capa de cliente o de presentación que proporciona el interfaz de usuario, una o más capas intermedias que proporcionan la lógica de negocio de la aplicación y una capa final con los sistemas de

información que mantienen aplicaciones y bases de datos corporativas. Entre las características de este tipo de aplicaciones se encuentran las siguientes:

- Posibilidad de **altas productividad** en el desarrollo de las distintas tecnologías J2EE para la integración de aplicaciones corporativa e integración de sistemas existentes.
- **Mayor escalabilidad** al describir las características básicas de transacciones y desarrollando distintos tipos de componentes de aplicación J2EE con modelos flexibles de seguridad.
- **Libertad de elección de plataformas** de desarrollo y producción, lo que define los aspectos necesarios que pueden solucionar una determinada problemática.
- La utilización de **herramientas libres** que agilizan el desarrollo de software con J2EE y que permiten el funcionamiento en los distintos módulos de ejecución.

2.2 Arquitectura J2EE

La especificación de J2EE define su arquitectura basándose en los conceptos de capas, contenedores, componentes y servicios. Tal y como muestra la Figura 2.1, las aplicaciones J2EE se dividen en cuatro capas: la capa cliente, la capa web, la capa negocio y la capa datos.

Capa Cliente. Se corresponde a la parte localizada en el equipo cliente. Es la interfaz gráfica del sistema y se encarga de interactuar con el usuario. J2EE soporta diferentes tipos de clientes incluyendo clientes HTML, applets Java y aplicaciones Java.

Capa Web. Se encuentra en el servidor Web y contiene la lógica de presentación que utilizada para generar una respuesta al cliente. Recibe los datos del usuario desde la capa cliente y basado en estos genera una respuesta apropiada a la solicitud. J2EE utiliza en esta capa las componentes Servlets y JavaServer Pages para crear los datos que se enviarán al cliente.

Capa Negocio. Se encuentra en el servidor de aplicaciones y contiene el núcleo de la lógica del negocio de la aplicación. Provee las interfaces necesarias para utilizar el servicio de componentes del negocio. Las componentes del negocio interactúan con la capa de datos y son típicamente implementadas como componentes EJB.

Capa EIS. Esta capa es responsable del sistema de información de la empresa o Enterprise Information System (EIS) que incluye bases de datos. Esta capa es el punto donde las aplicaciones J2EE se integran con otros sistemas no J2EE.

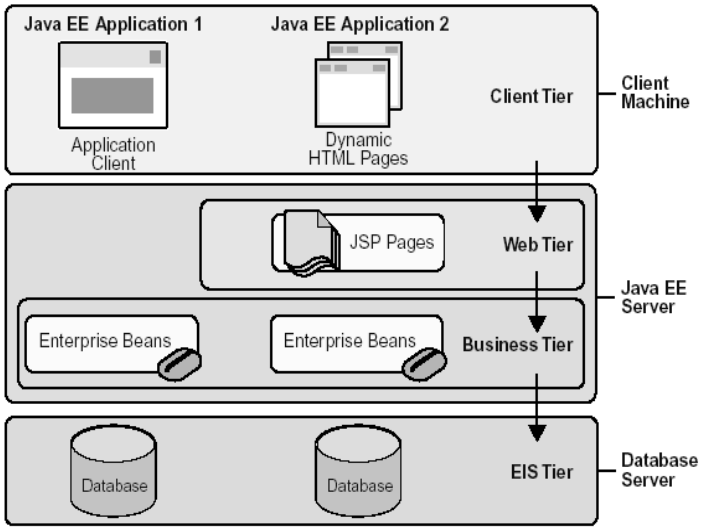


Figura 2.1: Arquitectura J2EE

2.2.1 Capa Cliente

Contiene programas nativos escritos en Java que en general poseen su propia interfaz gráfica y que se ejecutan en un proceso independiente. Son ejecutados dentro del contenedor de aplicación proporcionado por el JRE de Java.

Podemos distinguir dos tipos: los clientes Web y las aplicaciones Cliente.

Web Clients

Un cliente Web está formado por dos partes

- Páginas dinámicas que son generadas por componentes web que se ejecutan en la capa Web.
- Navegador Web encargado de renderizar las páginas recibidas del servidor.

Normalmente nos referimos a los clientes Web como clientes ligeros. Estos no necesitan acceder a bases de datos, ejecutar reglas de negocio o conectarse a aplicaciones heredadas.

Una página Web puede incluir un applet embebido. Un applet es una pequeña aplicación escrita en Java que se ejecuta dentro del la máquina virtual de Java instalada en el navegador Web. El navegador que carga y ejecuta el applet se conoce en términos genéricos como el contenedor de applets. Los applets pueden ejecutarse en una variedad de aplicaciones o

dispositivos que proporcionen soporte para el contenedor.. Son utilizados como alternativa a interfaces más limitadas basadas en HTML.

Los componentes Web (Servlets o JSP) son el API preferido para crear clientes Web, ya que no necesitan de la instalación de plugins o de políticas de seguridad en el sistema del cliente. Por otra parte, los componentes web permiten diseñar aplicaciones más modules, ya que se proporcionan medios para realizar la separación entre la programación de la aplicación y el diseño de la página Web.

Aplicacion Cliente

Una aplicación cliente se ejecuta sobre la máquina del cliente y permite al usuario manejar tareas que requieren de una interface de usuario mas rica que la que proporciona un cliente Web. Normalmente disponen de una interface de usuario creada con el API Java de Swing o AWT.

Las aplicaciones cliente pueden acceder directamente a los EJB que se ejecutan en la capa de negocio.

La Figura 2.2 muestra los distintos elementos que pueden formar parte de la capa cliente. El cliente se comunica con la capa de negocio ejecutándose en el servidor EE directamente o en el caso del cliente que se encuentra en el navegador, lo hace a través de páginas JSP o servlets que se ejecutan en la capa Web.

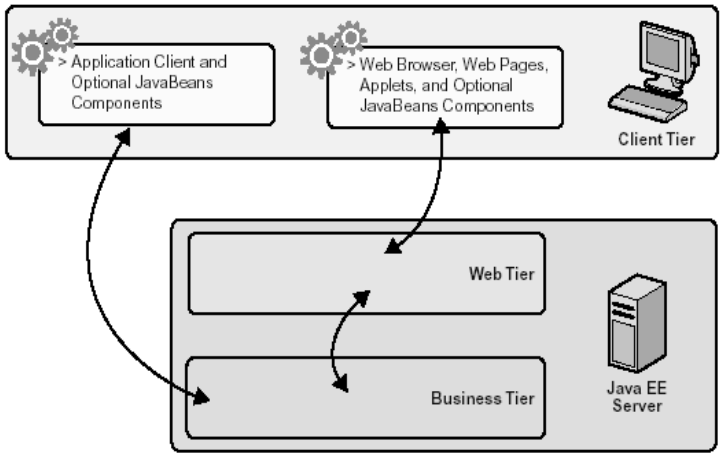


Figura 2.2: Capa cliente en arquitectura J2EE

2.2.2 Capa Web

Los Servlets y Java Server Pages, son denominados habitualmente simplemente componentes web. Se ejecutan en un servidor web y su misión es responder a solicitudes HTTP desde los clientes, también pueden generar páginas HTML, que en general corresponde a la interfaz de usuario de una aplicación, o puede generar XML u otro formato de datos que será utilizado por otras componentes de la aplicación.

Como se muestra en la Figura 2.3, la capa Web al igual que la capa cliente, puede incluir un componente Java Beans que se encargua de manejar la entrada del usuario y enviar esa entrada para que sea procesada a un Enterprise Bean que se esté ejecutando en la capa de negocio.

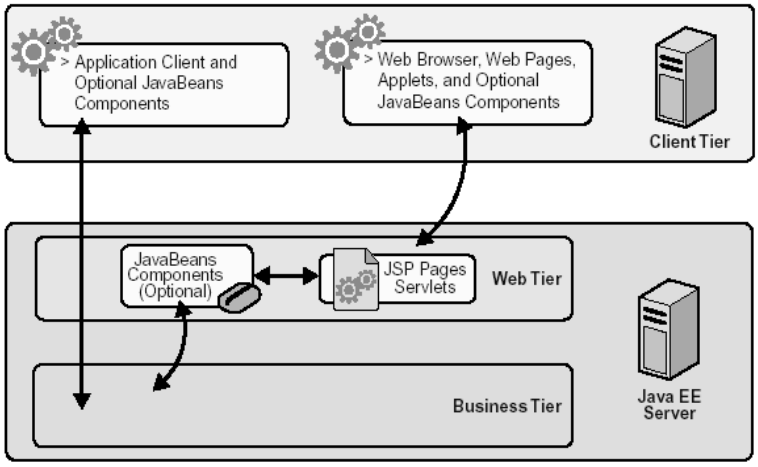


Figura 2.3: Capa web en arquitectura J2EE

2.2.3 Capa de negocio

Los Enterprise JavaBeans son componentes que contienen la lógica del negocio para una aplicación J2EE. Se ejecutan en un entorno distribuido que soporta transacciones. Encapsulan el acceso al EIS a través de la utilización de objetos que proporcionan la funcionalidad necesaria para el manejo de transacciones y persistencia.

La arquitectura de EJB define tres tipos diferentes de objetos Enterprise Beans, Beans de sesión, los Bean de entidad y los Beans dirigidos por mensajes.

Beans de sesión

Los beans de sesión son objetos no persistentes que se ejecuta en el servidor e implementan la lógica del negocio. Podemos pensar en un bean de sesión como en una extensión lógica del programa cliente que se ejecuta en el servidor y contiene información específica del cliente.

El tiempo de vida es limitado por la duración de la sesión del cliente, por lo cual no son persistentes en el tiempo. Cada bean de sesión mantiene una interacción con un cliente que se desarrolla a través de la ejecución de los distintos métodos que proporciona. Existen dos tipos de session beans que varían en la forma de modelar esta interacción: sin estado (stateless) y con estado (stateful).

Beans de entidad

Proporcionan una vista de los objeto Java a los datos del negocio. Permiten un acceso compartido de múltiples usuarios y tienen un tiempo de vida independiente de la duración de las sesiones de los clientes.

Los beans de entidad se utilizan para dar una visión y un acceso orientado a objetos sobre una base de datos relacional. Son componentes de datos que saben como utilizar una unidad de almacenamiento para mantener la información que guardan en forma persistente. Por tanto proporcionan una capa que envuelve el almacenamiento de los datos simplificando la tarea de su acceso y manipulación.

Beans dirigidos por mensajes

Modelan acciones, pero sólo se ejecutan luego de recibir un mensaje. Contienen la lógica de procesar un mensaje en forma asíncrona como puede ser recibir un mensaje con la necesidad de actualizar el stock de cierto producto e invocar el bean de sesión encargado.

La Figura 2.4 muestra como los Enterprise Bean reciben los datos de los programa clientes, los procesan (si es necesario) y los envían al Enterprise Information System (EIS) para su almacenamiento. Un Enterprise Bean también recupera datos de su almacenamiento, los procesa (si es necesario) y los envía de vuelta al programa cliente.

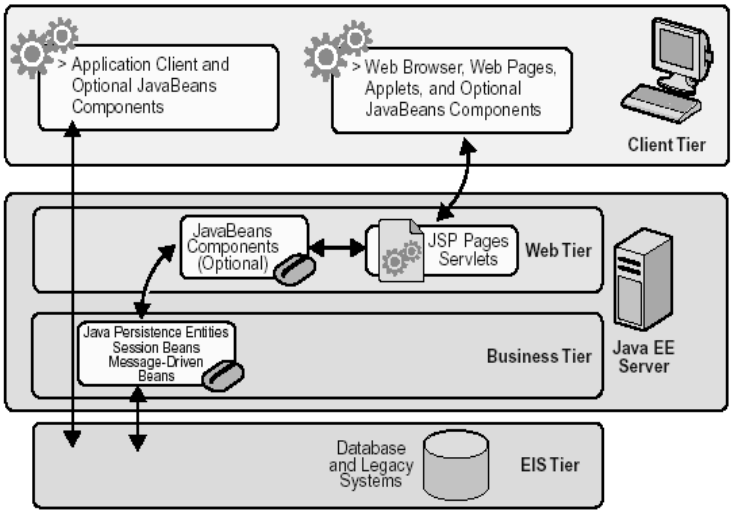


Figura 2.4: Capa de negocio en la arquitectura J2EE

La capa Enterprise Information System EIS es la encargada del manejo del software EIS que puede incluir ERP, sistemas transaccionales, bases de datos y cualquier otro sistema de información heredado.

Capítulo 3. Patrones de diseño

3.1 Patrones de diseño

El concepto de patrón de diseño que manejamos en Ingeniería del Software proviene del campo de la arquitectura. En 1977 se publica el libro *A Pattern Language: Towns/Building/Construction*, de Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King y Shlomo Angel, Oxford University Press.

En esta obra Alexander comenta que "*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 siquiera dos veces de la misma forma*". El patrón es un esquema de solución que se aplica a un tipo de problema, esta aplicación del patrón no es mecánica, sino que requiere de adaptación y matices. Por ello, los numerosos usos de un patrón no se repiten dos veces de la misma forma.

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

Los patrones documentan y explican problemas de diseño, y luego discuten una buena solución a dicho problema. Con el tiempo, los patrones comienzan a incorporarse al conocimiento y experiencia colectiva de la industria del software, lo que demuestra que el origen de los mismos radica en la práctica misma más que en la teoría.

Con la aparición del libro *Design Patterns: Elements of Reusable Object Oriented Sofware* escrito por los conocidos como Gang of Four (GoF, La banda de los cuatro en español) formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Se realizó la

recopilaron y documentaron de 23 patrones de diseño aplicados habitualmente por expertos diseñadores de software orientado a objetos.

3.1.1 Categorías y componentes

El grupo de GoF clasificó los patrones en tres 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 ocultando los detalles de cómo se crean e instancian los objetos.

Estructurales. Los patrones estructurales describen como las clases y objetos se pueden combinar para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser objetos simples u objetos compuestos.

Comportamiento. Los patrones de comportamiento nos ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de estos patrones es reducir el acoplamiento entre los objetos.

Para que una solución sea considerada un patrón debe poseer ciertos componentes fundamentales:

Nombre del patrón. Permite describir brevemente un problema de diseño junto con sus soluciones y consecuencias.

Problema. Indica cuándo aplicar el patrón. En algunas oportunidades el problema incluye una serie de condiciones que deben darse para que el patrón pueda aplicarse. Muestra la verdadera esencia del problema. Este enunciado se completa con un conjunto de fuerzas, término que se utiliza para indicar cualquier aspecto del problema que deba ser considerado a la hora de resolverlo, entre ellos:

- Requerimientos a cumplir por la solución.
- Restricciones a considerar.
- Propiedades deseables que la solución debe tener.

Son las fuerzas las que ayudan a entender el problema dado que lo exponen desde distintos puntos de vista.

Solución. No describe una solución o implementación en concreto, sino que un patrón, es similar a una plantilla que se puede aplicar en diversas situaciones diferentes. El patrón brinda una descripción abstracta de un problema de diseño y cómo lo resuelve una determinada

disposición de objetos. Que la solución sea aplicable a diversas situaciones denota el carácter recurrente de los patrones.

Consecuencias. Resultados en términos de ventajas e inconvenientes.

3.1.2 Clasificación

Según el nivel de abstracción los patrones de diseño pueden clasificarse de la siguiente forma:

- **Patrones arquitectónicos.** Centrados en la arquitectura del sistema. Definen una estructura fundamental sobre la organización del sistema. Proveen un conjunto predefinido de subsistemas, cuáles son sus responsabilidades y como se interrelacionan.

- **Patrones de diseño.** Esquemas para refinar los subsistemas o componentes de un sistema de software, o sus relaciones. Describen una estructura recurrente y común de componentes comunicantes que resuelven un problema de diseño dentro de un contexto. Ejemplo: el patrón *Singleton* asegura que exista sólo una instancia de una determinada clase.

- **Patrones de codificación o modismos (*idioms*).** Patrones que ayudan a implementar aspectos particulares del diseño en un lenguaje de programación específico. Ejemplo: en Java implementar una interface en una clase anónima.

Los patrones arquitectónicos pueden considerarse estrategias de alto nivel que abarcan componentes a gran escala, propiedades y mecanismos del sistema. Tienen implicaciones muy amplias que afectan tanto a la estructura como a la organización del sistema.

Los patrones de diseño son tácticas de medio nivel para profundizar en la estructura y comportamiento de ciertos componentes y sus relaciones. Los patrones de diseño no influencian la estructura del sistema sino que definen micro-arquitecturas para los subsistemas y componentes.

Por último, los modismos son técnicas específicas del paradigma y lenguaje de programación que complementan detalles de bajo nivel (internos o externos) de la estructura de un componente.

3.1.3 Patrones J2EE

En el año 2001 el Centro Java de Sun hizo público un catálogo de patrones conocido como *Core J2EE Patterns*. Estos patrones describen algunos problemas típicos encontrado habitualmente por los desarrolladores de aplicaciones empresariales y proveen soluciones para

estos problemas. En esencia, estos patrones contienen las mejores soluciones para ayudar a los desarrolladores a diseñar y construir aplicaciones para la plataforma J2EE.

Aunque estos patrones son representados desde un nivel lógico de abstracción, todos contienen en sus descripciones originales en la web de Sun, varias estrategias que ofrecen una gran cantidad de detalles para su implementación.

Desde que J2EE es una arquitectura por si misma que involucra otras arquitecturas, incluyendo Servlets, JavaServer Pages, Enterprise JavaBeans, y más, merece su propio conjunto de patrones específicos para diferentes tipologías de aplicaciones empresariales. Los patrones están divididos en tres capas: presentación, negocios e integración.

Capa de Presentación

Los patrones de la capa de presentación contienen toda la información relacionada con los servlets y tecnología JSP

- **Decorating Filter / Intercepting Filter.** Representa un objeto que está entre el cliente y los componentes Web y que procesa las peticiones y las respuestas.

- **Front Controller/ Front Component.** Se trata de un objeto que acepta todas las peticiones de un cliente y las redirecciona a los manejadores apropiados. El patrón Front Controller puede dividir la funcionalidad en dos objetos diferentes: el Front Controller y el Dispatcher. En ese caso, será el Front Controller quien acepte todas las peticiones de un cliente y realice la autenticación, y el Dispatcher será el encargado de direccionar las peticiones a los manejadores apropiados.

- **View Helper.** Un objeto helper que encapsula la lógica de acceso a datos en beneficio de los componentes de la presentación. Por ejemplo, los JavaBeans pueden usarse como patrón View Helper para páginas JSP.

- **Composite view.** Un objeto vista que está compuesto de otros objetos vista. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva include o el action include es un patrón Composite View.

- **Service To Worker.** Es como el patrón de diseño MVC con el Controlador actuando como Front Controller pero con una diferencia: aquí el Dispatcher (el cual es parte del Front Controller) usa View Helpers a gran escala y ayuda en el manejo de la vista.

- **Dispatcher View.** Similar al Service to Worler, pero no utiliza View Helpers y realiza muy poco trabajo en el manejo de la vista. El manejo de la vista es manejado por los mismos componentes de la Vista.

Capa de Negocio

Los patrones de la capa de Lógica de Negocios, contienen toda la información relacionada con los Enterprise Java Beans.

- **Business Delegate.** Un objeto que reside en la capa de presentación y en beneficio de los otros componentes de la capa de presentación llama a métodos remotos en los objetos de la capa de negocios.

- **Value Object/ Data Transfer Object/ Replicate Object.** Un objeto serializable para la transferencia de datos sobre lar red.

- **Session Facade/ Session Entity Facade/ Distributed Facade.** El uso de un bean de sesion como una fachada (facade) para encapsular la complejidad de las interacciones entre los objetos de negocio y participantes en un flujo de trabajo. El Session Façade maneja los objetos de negocio y proporciona un servicio de acceso uniforme a los clientes.

- **Aggregate Entity.** Un bean entidad que es construido o es agregado a otros beans de entidad.

- **Value Object Assembler.** Un objeto que reside en la capa de negocios y crea Value Objets cuando es requerido.

- **Value List Handler/ Page-by-Page Iterator/ Paged List.** Es un objeto que maneja la ejecución de consultas SQL, caché y procesamiento del resultado. Usualmente implementado como beans de sesión.

- **Service Locator.** Consiste en utilizar un objeto Service Locutor para abstraer toda la utilización JNDI y para ocultar las complejidades de la creación del contexto inicial, de búsqueda de objetos home EJB y recreación de objetos EJB. Varios clientes pueden reutilizar el objeto Service Locutor para reducir la complejidad del código, proporcionando un punto de control.

Capa de Integración

Por ultimo, los patrones de la capa de Integración contienen toda la información relacionada con el Java Message Service y la tecnología para conexión con base de datos de java JDBC

- **Data Access Object Service Activator**. Consiste en utilizar un objeto de acceso a datos para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.

- **Service Activator.** Se utiliza para recibir peticiones y mensajes asíncronos de los clientes. Cuando se recibe un mensaje, el Service Activator localiza e invoca a los métodos de los componentes de negocio necesarios para cumplir la petición de forma asíncrona.

3.2 Patrones principales J2EE para la capa de presentación

A continuación describimos con mas detalles algunos de los patrones mas importantes relacionados con los frameworks que vamos a estudiar y que son relevantes en este trabajo.

3.2.1 View Helper

Los cambios en la capa de presentación son muy frecuentes y son difíciles de desarrollar y mantener cuando la lógica de acceso a los datos de negocio y la lógica del formateo de la presentación está mezclada. Esto convierte al sistema en menos flexible, menos reutilizable, y generalmente menos adaptable a los cambios.

Mezclar la lógica de negocio y de sistema con el procesamiento de la vista reduce la modularidad y también proporciona una pobre separación de los roles entre los equipos de producción Web y de desarrollo de software.

Una vista contiene código de formateo, delegando sus responsabilidades de procesamiento en sus clases de ayuda, implementadas como JavaBeans o etiquetas personalizadas. Las clases de ayuda o helpers también almacenan el modelo de datos intermedio de la vista y sirven como adaptadores de datos de negocio.

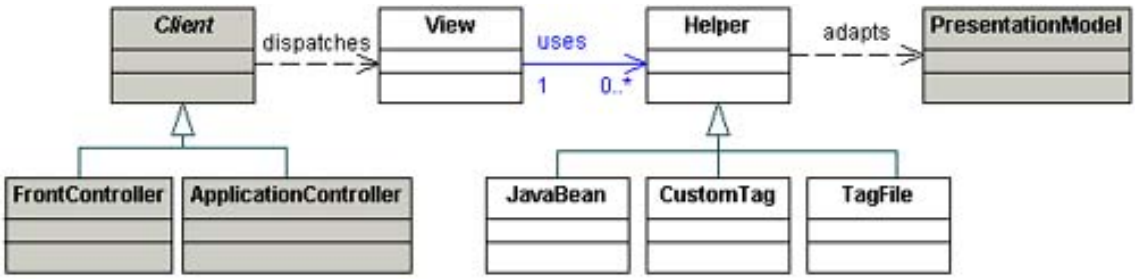


Figura 3.1: Diagrama de clases del patrón View Helper

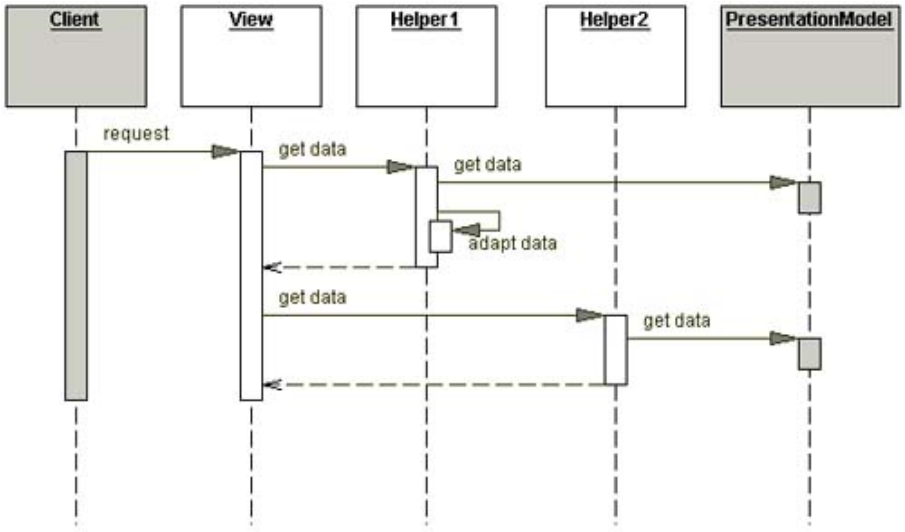


Figura 3.2: Diagrama de de secuencia del patrón View Helper

3.2.2 Composite View.

Las aplicaciones web mas complejas suelen presentar contenido de numerosas fuentes de datos, usando múltiples subvistas, que conforman una única página a visualizar., lo que suele complicar el código de la aplicación. Una solución es el uso de vistas compuestas que están formadas por múltiples subvistas atómicas, Cada componente de la plantilla puede ser incluido dinámicamente en el total, y el esquema de la página puede ser gestionado de forma independiente al contenido.

Un requisito típico de las aplicaciones web es el que su visualización tenga un estructura similar a lo largo de todo el web. Una plantilla es un componente de presentación que compone vistas separadas en una única página con un diseño específico. De esta forma se mejora la modularidad y la reutilización, mejora la flexibilidad, el mantenimiento, y la manejabilidad, tiene un impacto negativo (aunque muy pequeño) en el rendimiento.

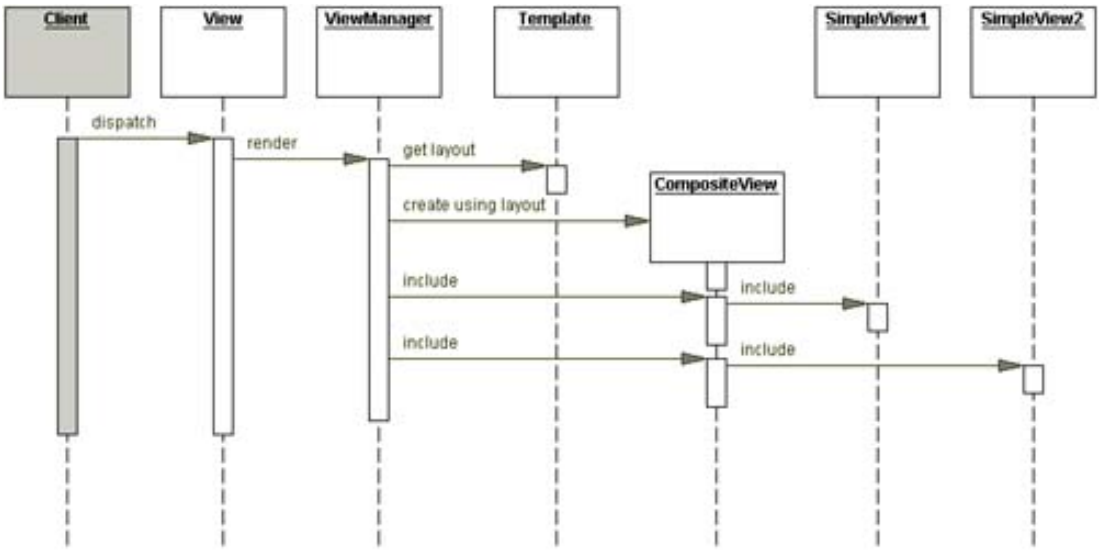


Figura 3.3: Diagrama de secuencia del patrón Composite View

3.2.3 Front Controller

La gestión de las peticiones http a nuestra aplicación web puede ser centralizada o distribuida. Tener distribuida la gestión de peticiones en nuestras páginas jsp lleva a aplicaciones de baja calidad, en las que se genera mucho código similar distribuido de forma repetida por todas nuestras páginas (vistas).

La solución es utilizar un controlador como punto inicial para la gestión de las peticiones. El controlador gestiona se encarga de gestionar estas peticiones, y realizar algunas funciones como: comprobación de restricciones de seguridad, manejo de errores, mapeos y delegación de las peticiones a otros componentes de la aplicación que se encargarán de generar la vista adecuada para el usuario.

Centralizando los puntos de decisión y control, el controlador ayuda a reducir la cantidad de código java que se encuentra embebido en nuestras páginas jsp. Lo que logramos es centralizar el propio control en el controlador y reducir la lógica de negocio en las vistas. De esta manera centralizamos control, mejoramos la manejabilidad de la seguridad, y aumentamos la reusabilidad del código.

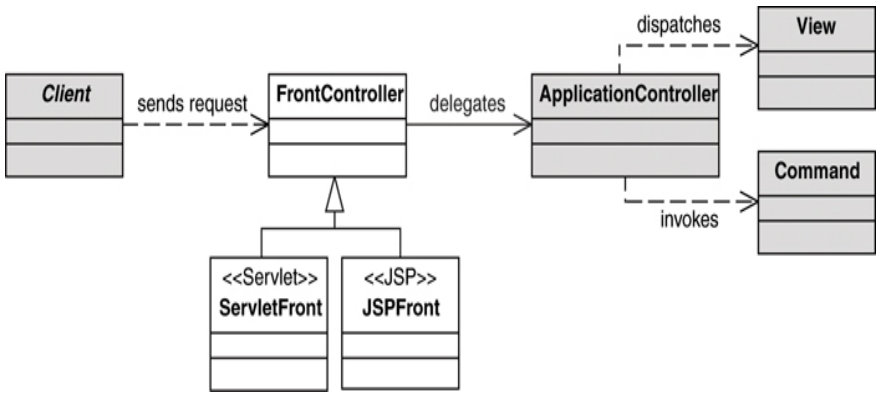


Figura 3.4: Diagrama de clases del patrón Front Controller

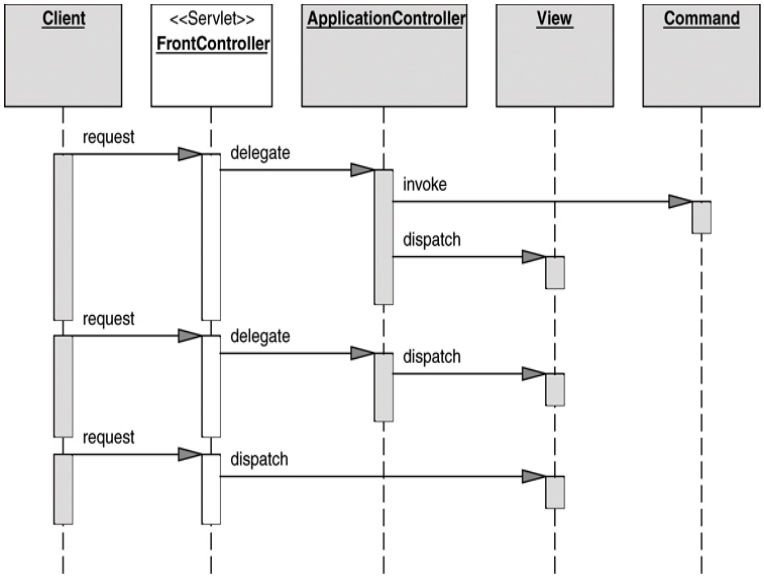


Figura 3.5: Diagrama de secuencia del patŕon Front Controller

3.2.4 Service to Worker

Está formado por la unión del patrón Front Controller y Application Controller. El patrón Front controller es el encargado del manejo de los elementos que forman parte de la petición. El papel del control se delega al patrón Application Controller, que recibirá un objeto de tipo Context Object que encapsula la petición.

El funcionamiento del patrón es el siguiente. El Front Controller recibe la petición y la delega al Application Controller, el cual se ocupará de llevar a cabo el manejo de las acciones correspondientes, resolviendo la petición entrante al comando apropiado que será invocado posteriormente. El comando internamente invocará a un servicio de la capa de negocio, el cual devolverá un modelo de presentación adecuado para la vista, que podrá ser una vista de tipo Composite View.

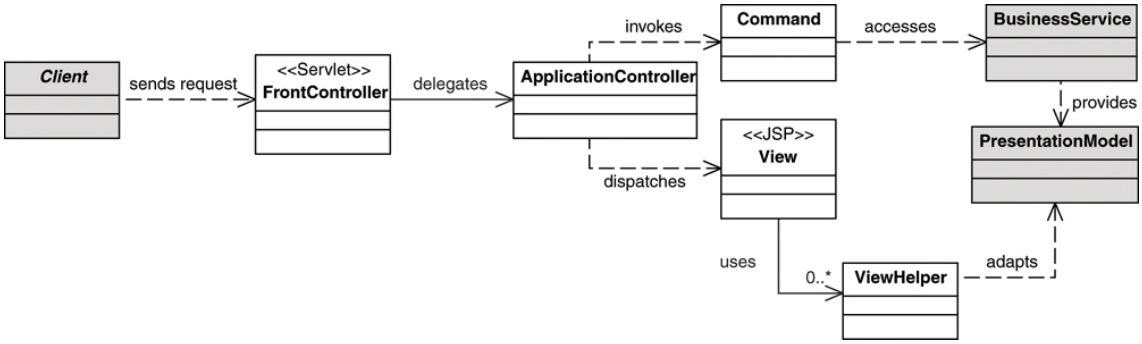


Figura 3.6: Diagrama de clases del patrón Service to Worker

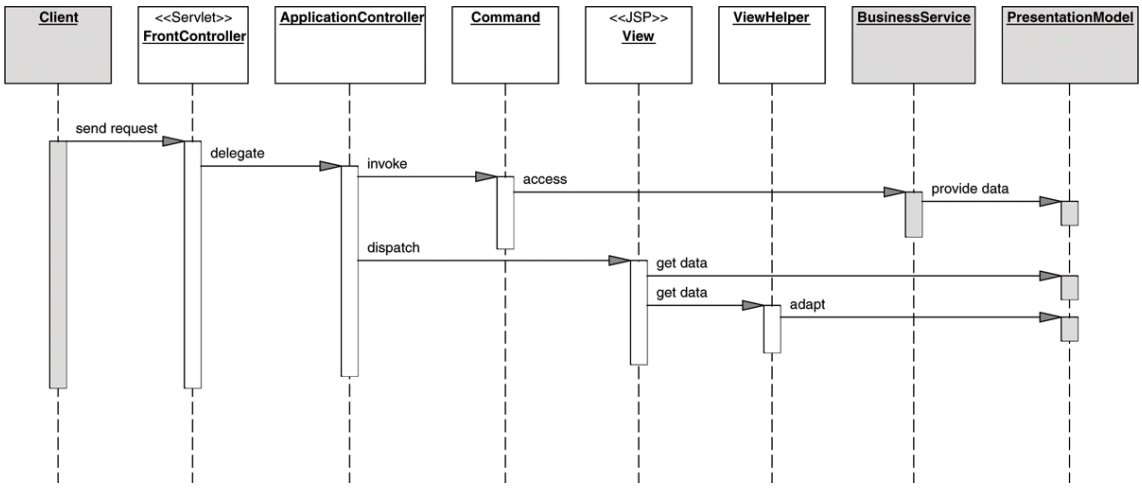


Figura 3.7: Diagrama de secuencia del patrón Service to Worker

3.2.5 Application Controller

Se encarga de resolver la acción que debe ejecutarse a partir de la información contenida en la petición y posteriormente realizar la invocación correspondiente siguiendo el patrón Command. La acción se comunica con los servicios que proporciona la capa de negocio y éste retorna parte del modelo. En este punto el Application Controller se encarga de decidir, en función del resultado, la vista a mostrar y la sirve. Por último mediante View Helpers se localizan, adaptan y transforman los datos del modelo que serán utilizados por la vista.

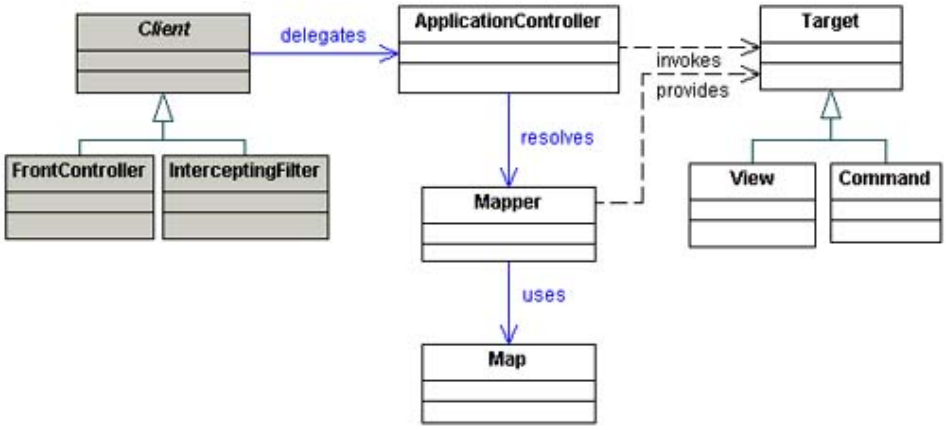


Figura 3.8: Diagrama del clases del patrón Application Controller

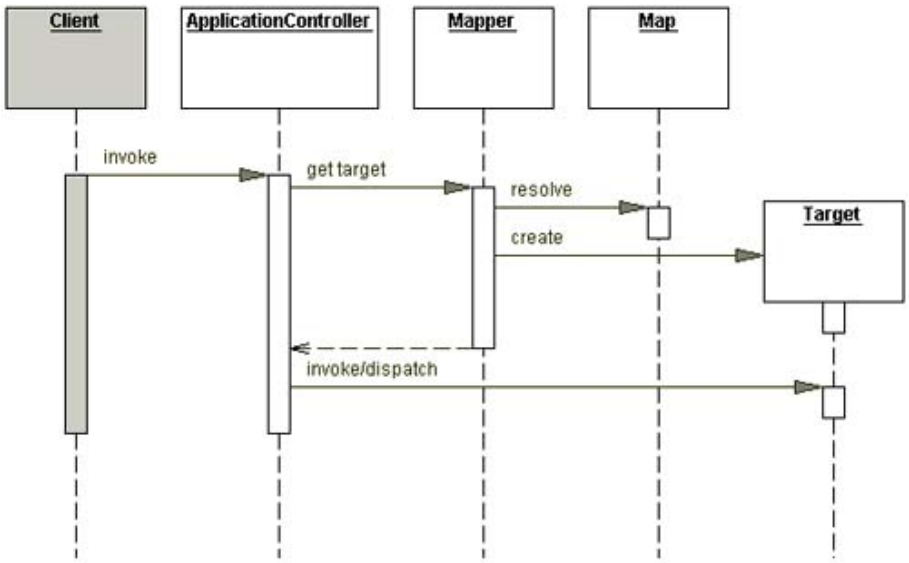


Figura 3.9: Diagrama de secuencia del patrón Application Controller

3.2.6 Dispatcher View

Los cambios de la capa de Presentación son frecuentes y son difíciles de desarrollar y mantener, debido a la combinación de la lógica de negocio para el acceso a los datos y la lógica de formateo de la presentación, esto hace que el sistema sea menos flexible, menos reutilizable, y menos adaptable a los cambios.

Combinar un dispatcher con Vistas y helpers para manejar las solicitudes de los clientes y preparar una presentación dinámica como respuesta. Un dispatcher es responsable por el manejo de vistas y la navegación, y puede ser o no encapsulado dentro de un controller, o puede ser un componente independiente que trabaje de forma sincronizada con los componentes internos.

Este patrón y el Service to Worker poseen una estructura similar, aunque cada patrón sugiere una división diferente de las labores que debe realizar cada componente. El controller y el dispatcher tienen responsabilidades limitadas, debido a que el procesamiento principal y el manejo de vistas es muy básico. Mas aún, si el control centralizado de los recursos subyacentes es considerado innecesario, entonces el controller no es necesario y el dispatcher puede ser reemplazado por una Vista.

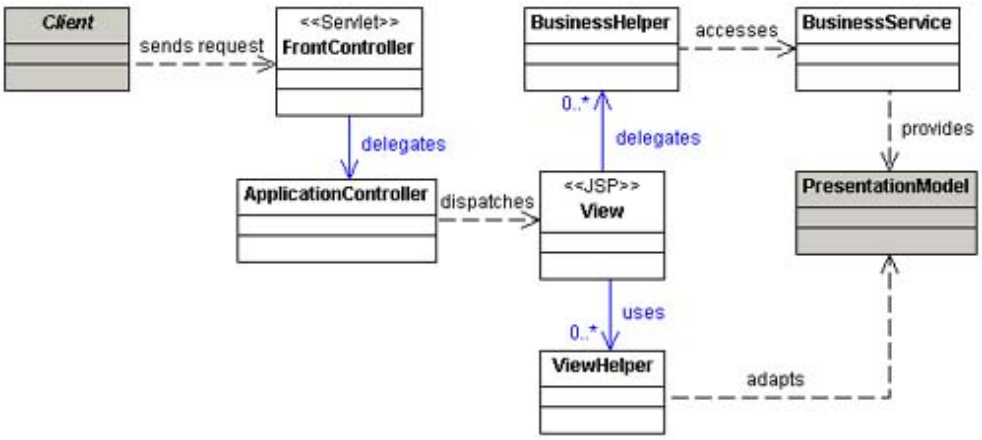


Figura 3.10: Diagrama de clases del patrón Dispatcher View

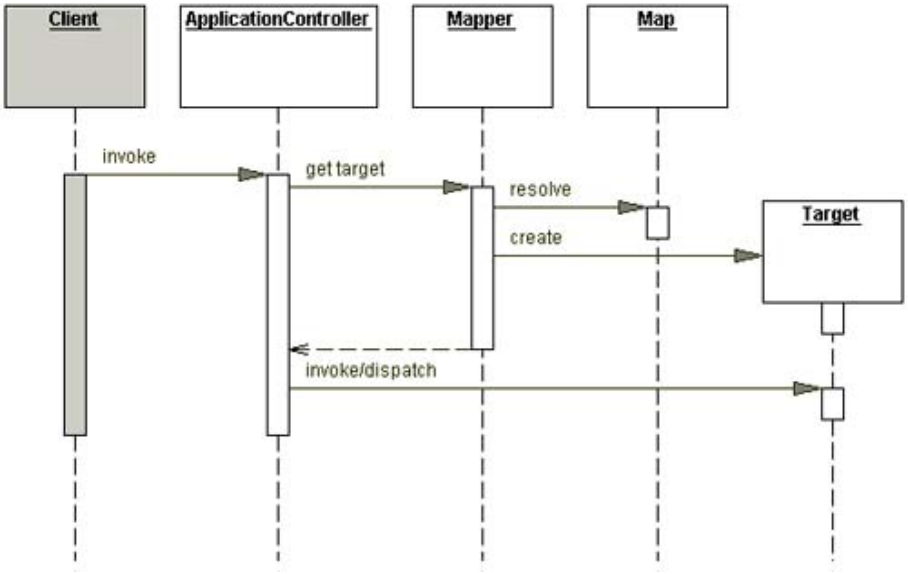


Figura 3.11: Diagrama de secuencia del patrón Dispatcher View

3.3 La arquitectura MVC

Model-View-Controller (MVC) es una arquitectura de diseño de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos de forma que las modificaciones al componente de la vista pueden ser hechas con un mínimo impacto en el componente del modelo de datos.

3.3.1 Módulos

Las aplicaciones web complejas son más difíciles de diseñar que las aplicaciones tradicionales de escritorio, el patrón MVC ofrece una solución para ayudar a disminuir dicha complejidad. Construir una aplicación web usando una arquitectura MVC implica dividir la aplicación en tres partes o módulos:

- **Modelo (Model):** Esta es la representación específica del dominio de la información sobre la cual funciona la aplicación. Contiene la lógica de negocio de la aplicación. Esta añade significado a los datos.

- **Vista (View):** Presenta el modelo en un formato adecuado para interactuar, usualmente un elemento de interfaz de usuario. Muestra al usuario la información que necesita.

- **Controlador (Controller)**: Corresponde a la parte de la aplicación centralizada en recibir peticiones del cliente, decidiendo qué función de la lógica de negocio debe

ejecutarse y delegando la responsabilidad de producir la siguiente fase del interfaz de usuario en un componente apropiado de la vista. Es decir, responde a eventos, recibe e interpreta la interacción del usuario, actuando sobre modelo y vista de manera adecuada para provocar cambios de estado en la representación interna de los datos, así como en su visualización.

Esta arquitectura ha demostrado ser muy apropiada para las aplicaciones web y especialmente adaptarse bien a las tecnologías proporcionadas por la plataforma J2EE, de manera que:

- **El modelo.** Contiene la lógica de negocio, es modelado por un conjunto de clases Java, existiendo dos claras alternativas de implementación, utilizando objetos java tradicionales o bien utilizando EJB (Enterprise Java Beans) en sistemas con unas mayores necesidades de concurrencia o distribución.

- **La vista**, proporciona de forma dinámica una serie de páginas web al cliente, siendo para él simples páginas HTML. Lo más común es utilizar páginas JSP, que mediante un conjunto etiquetas XML proporcionan un interfaz sencillo y adecuado a las clases Java y objetos proporcionados por el servidor de aplicaciones.

- **El controlador** en la plataforma J2EE se desarrolla mediante servlets, que actúan como intermediarios entre la vista y el modelo, más versátiles que los JSP para esta función al estar escritos como clases Java normales, evitando mezclar código visual (HTML, XML...) con código Java.

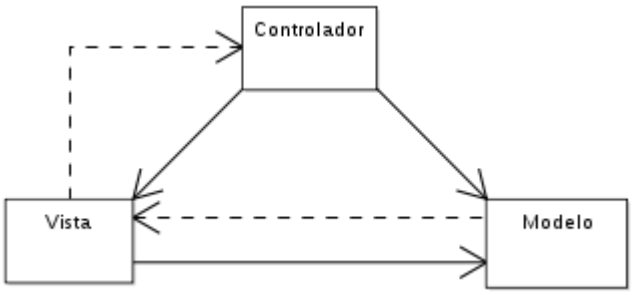


Figura 3.12: Arquitectura MVC Model 1

Con todo lo anterior, el funcionamiento de una aplicación web J2EE que utilice el patrón arquitectural MVC se puede descomponer en una serie de pasos:

1. El usuario interactúa con la interfaz de usuario, es decir, realiza una acción en su navegador, que llega al servidor mediante una petición HTTP y es recibida por un

servlet (controlador). Esa petición es interpretada y se transforma en la ejecución de código java que delegará al modelo la ejecución de una acción de éste.

2. El modelo recibe las peticiones del controlador a través de un interfaz o fachada que encapsulará y ocultará la complejidad del modelo al controlador. El resultado de esa petición será devuelto al controlador.
3. El controlador recibe del modelo el resultado, y en función de éste, selecciona la vista que será mostrada al usuario, y le proporcionará los datos recibidos del modelo y otros datos necesarios para su transformación. Una vez hecho esto el control pasa a la vista para la realización de esa transformación.
4. En la vista se realiza la transformación tras recibir los datos del controlador elaborando la respuesta HTML para generar la interfaz apropiada donde se reflejan los cambios en el modelo para que el usuario la visualice.
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

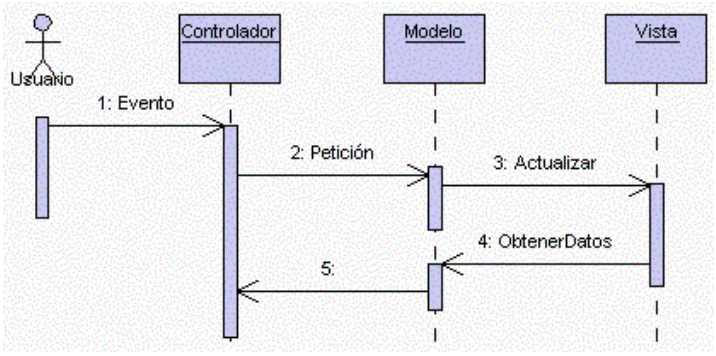


Figura 3.13: Diagrama de secuencia arquitectura MVC

3.3.2 Ventajas del modelo MVC

La principal ventaja de este modelo es la obtención de una separación total entre lógica de negocio y presentación. A esto se le pueden aplicar opciones como el multi-lenguaje, distintos diseños de presentación para diferente tipo de dispositivos, etc. Todo esto sin alterar la lógica de negocio. La separación de capas como presentación, lógica de negocio, acceso a datos es fundamental para el desarrollo de arquitecturas consistentes, reutilizables y fácilmente mantenibles, lo que al final repercute en un ahorro de tiempo en el desarrollo y mantenimiento.

3.3.3 Model 2

Model 2 es la variación de la arquitectura MVC, recomendada para las aplicaciones web desarrolladas sobre J2EE. Dicha arquitectura consiste en el desarrollo de una aplicación según el patrón de diseño Model-View-Controller descrito, pero especificando que el

controlador debe estar formado por un único servlet, que centralice el control de todas las peticiones al sistema, y que basándose en la URL de la petición HTTP y en el estado actual del sistema, derive la gestión y control de la petición a una determinada acción de entre las que se encuentren registradas en la capa controlador. La ventajas principal que ofrece este patrón es su capacidad para gestionar en un único punto la aplicación de filtros a las peticiones, las comprobaciones de seguridad, la realización de logs, etc.

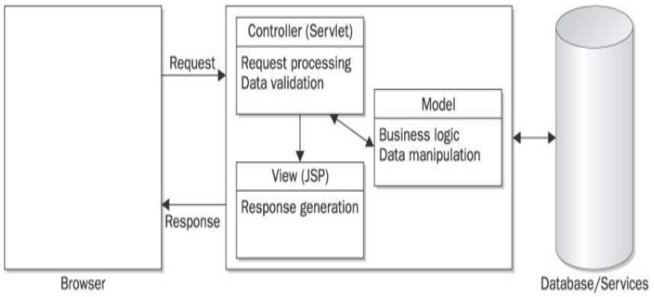


Figura 3.14: Arquitectura MVC Model 2

Capítulo 4. Frameworks

Podemos definir un framework o marco de desarrollo como un conjunto de librerías y componentes utilizados para implementar la estructura estándar de una aplicación que junto con una documentación y metodología de uso nos permite diseñar, construir e implantar aplicaciones empresariales de forma uniforme, rápida y de mayor calidad.

4.0.1 Objetivos

El objetivo del uso de un framework es promover la reutilización de código, con el fin de reducir los tiempos en los proyectos de desarrollo. Al no ser necesario la reescritura repetitiva de código para cada nueva aplicación. Existen diferentes frameworks para diferentes propósitos, algunos orientados al desarrollo de aplicaciones web, otros para desarrollar aplicaciones multiplataforma, para sistemas operativos específicos o lenguajes de programación en concreto, entre otros.

Un framework determina la arquitectura de una aplicación , el framework define la estructura general, sus particiones en clases y objetos, las responsabilidades clave, así como la colaboración entre dichas clases y objetos. Todos estos parámetros son definidos por el framework, evitando que sea el usuario quien tenga que definirlos, permitiendo que el enfoque pueda hacerse sobre la propia aplicación en lugar de hacerlo sobre su arquitectura.

El framework captura las decisiones de diseño que son comunes a su dominio de aplicación. Un framework no sólo promueve la reutilización de código sino también la reutilización de diseño. Un framework ayuda a que se desarrolle una aplicación de una manera más rápida, ya que se no desperdician tiempo y recursos en aquellos detalles de diseño que en muchas ocasiones emplean más tiempo del que tomo construir la lógica de la aplicación en sí misma.

Ya hemos mencionado que uno de los objetivos del uso de un framework de desarrollo es la reducción en los tiempos de desarrollo, ayudándonos principalmente en los siguientes puntos:

- Ofreciendo generadores de código y plantillas arquitectónicas, desarrolladas con las últimas versiones de componentes y recogiendo todas las mejores prácticas de los mismos.
- Evitando el síndrome del “papel en blanco”: por dónde empiezo y qué utilizo.
- Conocimiento adquirido reutilizable en todos los proyectos.
- Simplificando los procesos de mantenimiento a largo plazo.

4.0.2 Características comunes

A continuación enunciamos una serie de características que podemos encontrar en prácticamente todos los frameworks.

- **Abstración de URLs y sesiones.** No es necesario manipular directamente las URLs ni las sesiones, el framework ya se encarga de hacerlo.
- **Acceso a datos.** Incluyen las herramientas e interfaces necesarias para integrarse con herramientas de acceso a datos, en BBDD, XML, etc.
- **Controladores.** La mayoría de frameworks implementa una serie de controladores para gestionar eventos, como una introducción de datos mediante un formulario o el acceso a una página. Estos controladores suelen ser fácilmente adaptables a las necesidades de un proyecto concreto.
- **Autenticación y control.** Incluyen mecanismos para la identificación de usuario.

4.0.3 Principales frameworks

A continuación indicamos una breve descripción de las principales características de los principales frameworks.

- **Apache Struts.** Es un framework basado en el patrón MVC bajo la plataforma J2EE perteneciente al proyecto Jakarta de la Apache Software Libre pero que actualmente ya es un proyecto independiente con el nombre de Apache Struts. El hecho de que sea open source y que sea compatible con todas las plataformas en las que J2EE está disponible hacen que sea una herramienta muy utilizada por la comunidad de programadores a nivel mundial.

- **Apache Struts 2.** Se trata de un framework Web totalmente nuevo, no es una revisión de su predecesor. Su arquitectura es completamente nueva y está basada en el framework *Open Symphony WebWork*. Struts 2 es un framework web de segunda generación, que implementa el conocido patrón Modelo-Vista Controlador (MVC),

- **Spring Framework.** Conocido simplemente como Spring, es un framework de código abierto para la plataforma J2EE. Aunque Spring no obliga a utilizar un modelo de programación en particular, se ha popularizado dentro de la comunidad y se le ha considerado como un sustituto al modelo EJB (Enterprise Java Bean) que es el estándar propuesto por la plataforma J2EE. Por su diseño, Spring ofrece mucha libertad a los

programadores y unas soluciones bien documentadas y fáciles de utilizar para las prácticas comunes en la industria.

- **Java Server Faces.** Los principales componentes de la tecnología JavaServer Faces son: Un API y una implementación de referencia para: representar componentes de interface de usuario y manejar su estado; manejo de eventos, validación del lado del servidor y conversión de datos; definición de navegación entre páginas; soporte de internacionalización y accesibilidad; y proporcionar extensibilidad para todas estas características.

Todo esto junto con una librería de etiquetas JavaServer Pages (JSP) personalizadas para representar componentes complejos de interface de usuario dentro de una página JSP.

Este modelo de programación y la librería de etiquetas para componentes de interface de usuario facilitan de forma significativa la tarea de la construcción y mantenimiento de aplicaciones Web con interfaces de usuario en el lado del servidor.

Con un mínimo esfuerzo, podemos:

- Conectar eventos generados en el cliente a código de la aplicación en el lado del servidor.
- Mapear componentes UI a una página de datos del lado del servidor.
- Construir un UI con componentes reutilizables y extensibles.
- Grabar y restaurar el estado del UI más allá de la vida de las peticiones de servidor.

Capítulo 5. Struts

Apache Struts es un framework de software libre orientado al desarrollo de aplicaciones web basadas en J2EE, ajustándose al patrón Modelo-Vista-Controlador, que establece una separación de la capa de lógica de negocio, la capa de presentación y la capa de control. Struts se desarrollaba como parte del proyecto Jakarta de la Apache Software Foundation, pero actualmente es un proyecto independiente conocido como Apache Struts. A continuación se introduce las líneas generales el comportamiento del patrón Modelo-Vista Controlador en Struts.

La capa Modelo es la encargada de la gestión de los datos. La capa Vista tiene la función de presentar los datos del modelo al usuario y recoger la peticiones de éste para enviarlas a la capa del Controlador, que responderá a las acciones del usuario y realizarás cambios tanto en el modelo como en la vista.

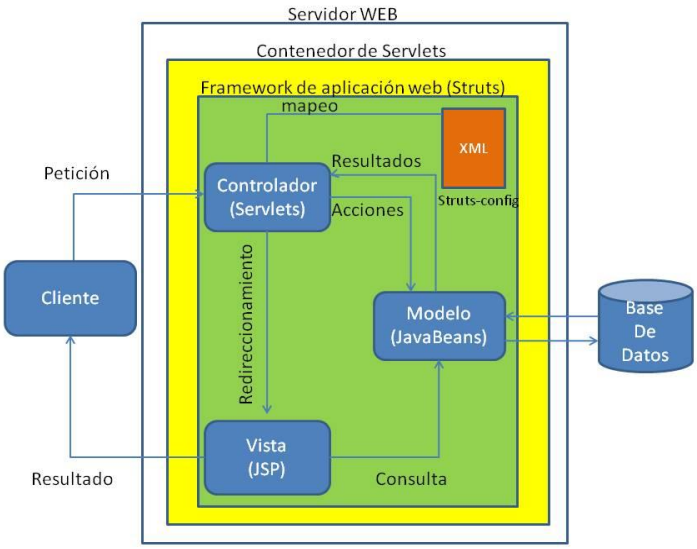


Figura 5.1: Diagrama patrón MVC en Struts

Podemos resumir su esquema de funcionamiento mostrado en la Figura 5.1 de la siguiente forma:

1. El cliente manda una petición HTTP que tiene como destino un Servlet (Controlador).
2. Al recibir la petición HTTP el Servlet, delega el proceso de los datos al Modelo.
3. El Modelo ejecuta los procesos necesarios y obtiene unos resultados.
4. El Controlador redirige el conjunto de procesos de la petición a la Vista.
5. La Vista recupera los datos generados por el Servlet y genera una respuesta

5.1 Capa Modelo

Struts no ofrece características propias para el desarrollo de la capa modelo. A primera vista puede parecer extraño, teniendo en cuenta que es un framework basado en el patrón MVC, sin embargo supone una gran ventaja a la hora de afrontar un proyecto, ya que será flexible al utilizar distintos objetos de negocio, como son Enterprise JavaBeans, Java Data Objects, o objetos de acceso a datos. También hay que destacar que desde esta capa no existe ninguna referencia al código propiamente dicho de Struts.

5.2 Capa Vista

Es la capa que interactúa directamente con el usuario, se centra en la interfaz y presentación de la información. Para el diseño de las vistas Struts si proporciona un conjunto rico de funcionalidades y características. Los principales componentes en los que se apoya la Vista en Struts, utilizando HTML/JSP son:

- **JSP.** Es el principal componente de la capa vista. Están formadas por código HTML estático y librerías de etiquetas JSP que generan código HTML dinámico, que conjuntamente son enviados al navegador para mostrar los datos al usuario. Recalcar que en Struts, dentro de las JSP no se hará ningún cómputo ni función correspondiente al modelo, solo labores de obtención de datos del usuario y presentación de los mismos.

- **Form Beans.** Proporcionan la vía de comunicación de datos entre las capas de Vista y Control. Esencialmente son contenedores de datos que almacenan los datos introducidos en los formularios HTML (Java Beans cuyos atributos se corresponden con los campos de los formularios). Son clases que toman la información recogida en los formularios y son utilizadas por la capa de Control para poner dichos datos al servicio del Modelo para su procesamiento.

Figura 5.2: Ciclo de ejecución de un Form Bean

- **JSP Tag Libraries.** Struts provee un conjunto propio de etiquetas JSP, que facilitan del desarrollo de este tipo de páginas. Básicamente, un Tag de JSP consiste en una etiqueta con formato <prefijoTagLib:nombreTag atributo=valor ... > que cuando la JSP es compilada es sustituido por una llamada a la clase TagHandler que se encarga de resolver su funcionalidad.
- **HTML.** Utilizadas para generar formularios HTML que interactúan con distintas APIs de Struts.
- **Bean.** Utilizadas para trabajar con objetos Java Beans, asi como para acceder a las propiedades de los mismos.
- **Logic.** Utilizadas para aplicar lógica condicional en la páginas JSP. Cabe destacar que la principal ventaja de este tipo de etiquetas es la de no incluir código java directamente en la página JSP mediante el uso de scriplets, lo que nos proporcionará un código más limpio y legible.
- **Resource Bundles (Paquetes de Recursos)**. Son ficheros con extensión .properties constituidos por pares etiqueta-valor que son leídos por la aplicación en tiempo de ejecución. Estos paquetes de recursos pueden ser accedidos desde páginas JSP para rellenar su contenido o desde el mismo Controlador para generar mensajes de error. Con este recurso se facilita de manera considerable la traducción de la aplicación a varios idiomas, lo que se conoce también como internacionalización.

5.2.1 Recursos

El concepto de la internacionalización (i18n) se basa en conseguir generar un aplicativo que soporte cualquier idioma mediante el uso de un fichero de properties en el que definiremos las claves y su respectivo literal en el idioma en concreto.

De este modo tendremos un fichero de recursos por cada idioma soportado por la aplicación. No hay límite en el número de idiomas que una aplicación puede soportar. Struts adapta el soporte de multiidioma de Struts basado en la variable locale del browser, mediante la cual se detecta el idioma del navegador y se utiliza el recurso (fichero de properties) adecuado.

5.2.2 ActionForms

Muchas de las JSPs de una aplicación contienen datos de entrada en un formulario HTML. En la arquitectura Struts, estos datos quedan a disposición de las Actions en forma de objetos del tipo ActionForm.

Es dentro de las ActionForms donde se produce la validación de los datos. En las del primer tipo ya se ve que el método validate() es lo que hace este trabajo. El método devuelve un objeto ActionErrors con la lista de los errores que se han encontrado. Si esta lista no es vacía, Struts vuelve a redireccionar a la página de entrada del Action donde el usuario podrá ver los errores de validación mediante los tags <html:errores> correspondientes.

5.2.3 ActionErrors

Cualquier excepción que se produzca en el modelo durante la ejecución de la aplicación, o cualquier error del usuario, por ejemplo en forma de datos de entrada incorrectos, acabar convirtiéndose en un ActionError.

Los ActionErrors se pueden crear a partir de una string con el mensaje de error, pero lo recomendable es hacerlo a partir de una clave en el fichero de recursos, lo cual permitirá que el error se presente en el idioma del usuario.

El tag <html:errors> es el utilizado para mostrar los ActionErrors a los usuarios. Éste tag puede estar en cualquier JSP de entrada de datos. Si el objeto ActionErrors está vacío no genera nada, pero si contiene algún ActionError entonces se mostrará. Además con el parámetro property se pueden seleccionar qué mensajes de la lista se quieren incluir en aquella posición concreta de la página. Así, si se quiere, se pueden mostrar mensajes de error de validación cerca de los campos de entrada donde se han producido.

5.3 Capa Controlador

Struts se centra principalmente en la capa del controlador. El diagrama de secuencia de la Figura 5.3 muestra cual es el flujo de iteración del usuario con una aplicación Struts.

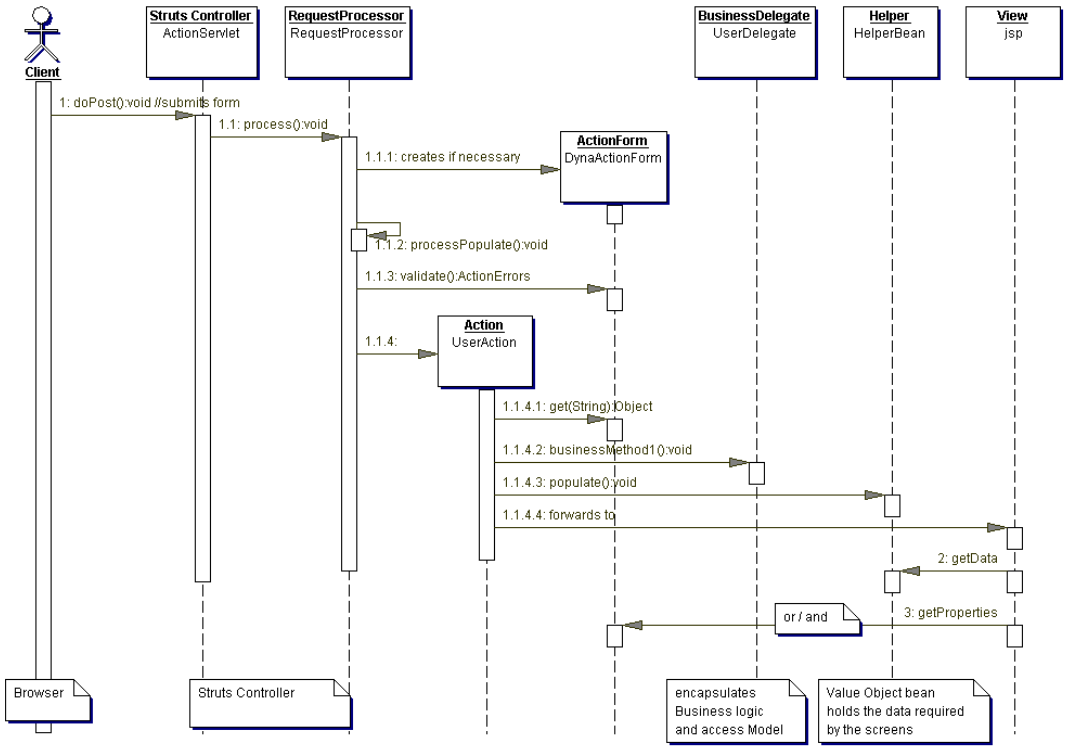


Figura 5.3: Diagrama de secuencia de Struts

Todas las peticiones que hace el usuario desde su navegador hacia la aplicación pasan por un mismo servlet, el ActionServlet, que delega la ejecución de la petición en un objeto de tipo RequestProcessor, que es el que controla el flujo de la petición.

En primer lugar instancia un objeto ActionForm. Los datos que conforman el input de la operación y que el usuario ha rellenado en un formulario HTML son recogidos por Struts y puestos dentro del ActionForm. Estos datos pueden tener diferentes niveles de visibilidad, y dependiendo de este nivel Struts los almacena en un puesto u otro: request (HttpServletRequest), session (HttpSession), o application (ServletContext).

Según la URL invocada (ActionMapping), se instancia una subclase de Action entre los configurados en el fichero struts-config.xml. El código del Action es el que hace los llamamientos al modelo, bien directamente, bien a través del patrón de diseño BusinessDelegate, que viene a ser como una interfaz restringida del modelo en las necesidades particulares de la aplicación.

Los datos de vuelta de la operación llenan un bean que puede ser el mismo ActionForm si tiene scope de sesión, o bien, un bean específico por los datos de respuesta de la petición.

El resultado final de una petición es un objeto ActionForward que el Action ha escogido según ha ido la ejecución de la lógica de negocio (“ok”, “error”,...). En el fichero de configuración, se establece la relación (mapping) entre cada final de operación y el recurso que debe recibir control en cada caso. Este recurso puede ser otro Action, una JSP, etc. El controlador le pasa la petición al objeto correspondiente, haciendo un forward o un redirect según se haya definido en el mapping.

Destacamos a continuación cuales son los componentes del controlador mas significativos.

5.3.1 ActionServlet

Es el componente que hace propiamente de controlador servlet del JSP, y que por tanto centraliza todas las peticiones que se hacen en la aplicación. Delega en un objeto RequestProcessor.

Normalmente una aplicación Struts no necesitará más que especificar el servlet en su descriptor de despliegue y asignarle las URLs que le interesen (típicamente, *.do)

La aproximación del patrón controlador Front Controller es atender las solicitudes HTTP y decidir qué hacer con ellas . Con el Front Controller hay un único objeto que recibe todas las peticiones y crea un objeto separado para procesar la solicitud. Esto permite centralizar toda la gestión de las solicitudes HTTP en un único objeto.

5.3.2 RequestProcessor

RequestProcessor está diseñado para poder modificar el comportamiento. Consta de una serie de métodos como por ejemplo, processPreProcess, processActionForm, processValidate y processForwardConfig que se invocan de manera ordenada y documentada, de manera que se puede modificar el comportamiento.

5.3.3 Ficheros de configuración

Todos los parámetros configurables de una aplicación Struts se especifican en un único fichero struts-config.xml. Mediante este fichero se pueden cambiar flujos de ejecución del programa simplemente variando parámetros XML, sin tener que entrar a modificar código java. En este fichero se especifica por completo la interacción del controlador.

5.3.4 ActionMapping

Un objeto ActionMapping no es más que la definición de un Action que se ha encontrado en el fichero de configuración transformada en un objeto. Lo utiliza el propio framework para saber que Action concreto se ha de instanciar, y después la pasa también como parámetro al propio Action por sí se necesita (normalmente sólo hace falta para buscar el forward a retornar).

5.3.5 Action

El objeto Action tiene su método principal, execute(), que es donde se escribe la lógica de la operación. Cada operación se implementa habitualmente en una subclase de Action diferente. Struts pasa 4 parámetros a execute():

La petición, en forma de objeto HttpServletRequest

- La respuesta, en forma de objeto HttpServletResponse
- El ActionMapping que ha originado la petición
- El ActionForm que contiene los datos de entrada

El proceso de la ejecución de un Action es, de manera simplificada:

- Obtener los datos de entrada (en este punto ya han sido validadas)
- Instanciar objetos del modelo y de servicios, invocar los métodos que sean necesarios para llevar a cabo la operación.
- Poner los datos de salida en el puesto que haga falta (como request attribute, o en el mismo ActionForm si ésta tiene un ámbito de sesión).
- Retornar un objeto ActionForward según haya sido el resultado de la operación

Si los objetos del modelo lanzan alguna excepción, ésta se ha de capturar, y retornar un forward adecuado, que probablemente será uno que lleve a una JSP de error, o en la misma JSP de entrada de datos, donde se mostrará un mensaje entendedor para el usuario.

5.3.6 ActionForward

Un Action siempre tiene que acabar devolviendo un objeto ActionForward. No lo ha de crear explícitamente, sino que estará configurado en el struts-config.xml, y por tanto disponible en el objeto ActionMapping que se le pasa con execute().

5.4 Conclusiones Struts

Algunas de las ventajas que Struts brinda son: que existe un variado número de trabajos y proyectos ya hechos lo que ofrecen un mayor número de ejemplos para poder tomar un punto de partida y de referencia; otra ventaja es que brinda librerías de tags para HTML bastante útiles.

5.5 Patrones en Struts

Siguiendo la clasificación y los patrones descritos en el catálogo Core J2EE Patterns, la arquitectura de Struts implementa varios patrones importantes, incluyendo Service to Worker, Front Controller, Singleton, Dispatcher, View Helper, Value Object, Composite View y Synchronizer Token, entre otros. La Tabla 5.1 muestras que patrones son implementados por cada componentes en Struts.

Patrón	Componente Strut
Service to Worker	ActionServlet, Action
Command, Command and Controller, Front Controller, Singleton, Service Locator	ActionServlet, Action
Dispatcher, Navigator	ActionMapping, ActionServlet, Action, ActionForward
View Helper, Session Facade, Singleton	Action
Transfer Object, Value Object Assembler	ActionForm, ActionErrors, ActionMesssages
View Helper	ActionForm, ContextHelper, tag extensions
Composite View, Value Object, Assembler	Template taglib, Tiles taglib

Tabla 5.1: Patrones utilizados en los componentes de Struts

Capítulo 6. Struts 2

Apache Struts 2 es un framework Web totalmente diferente a Struts, no se trata de una revisión de su predecesor, su arquitectura es completamente nueva y está basada en el framework WebWork.

Struts 2 que implementa el diseño de Modela-Vista Controlador (MVC), ha sido construido siguiendo las mejores y más probadas prácticas; estas afirmaciones también se pueden aplicar al framework Struts 1, pero pasado los años la comunidad advirtió las limitaciones de la primera versión del framework.

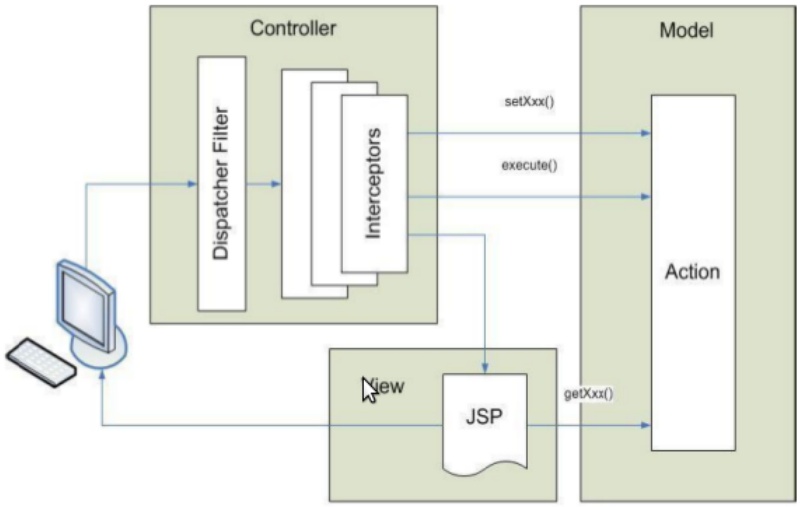


Figura 6.1: Arquitectura Struts2 MVC

Struts2 sigue las mejores prácticas y patrones de diseño actuales. Este también fue el objetivo de su padre Struts el cual introdujo el patrón MVC dentro de los Framework de las

aplicaciones web. Aprovechando toda esta experiencia en el mercado, se introducen varias características nuevas que hacen del Framework más limpio y flexible.

Estas nuevas características incluyen los interceptores de WebWork2, permitiendo la interceptación de la petición antes de que se ejecute la lógica de la acción, por otro lado aprovecha la aparición de Java 5 para incluir configuraciones mediante anotaciones, reduciendo al máximo la configuración mediante ficheros XML. Por último, también hay que destacar la introducción de un poderoso lenguaje de expresiones llamado OGNL (Object-Graph Navigation Language) que junto a la Value Stack, permitirá a Struts2 acceder a los datos de una manera más rápida y eficaz que otros Frameworks..

Algunas de sus características principales, son las siguientes:

- Todas las clases del Framework están basadas en interfaces y el núcleo principal es independiente del protocolo HTTP.
- Basado en el patrón MVC bajo la plataforma J2EE.
- Alta configuración y extensibilidad.
- Permite el uso de Plugins de componentes e integración con otros Frameworks.
- Cualquier clase puede llegar a ser usada como una “Action class” (POJO)
- Las acciones de Struts2 son fácilmente integrables con el Framework Spring. (OF)
- Integración de AJAX, esto hace la aplicación más dinámica.
- Proporciona un framework de validación flexible que permite desacoplar las reglas de validación del código de las acciones.
- Conversión de tipos automática que mapea de forma transparente los valores HTTP a las acciones mediante setters, solucionando y simplificando uno de los mayores esfuerzos que se necesitan al crear aplicaciones web.
- Define un lenguaje OGNL compatible con JSTL que expone las propiedades de múltiples objetos como si fueran un único JavaBean.
- Ficheros de configuración modulares que permiten descomponer la configuración en varios archivos para mejorar la gestión de los mismos en proyectos grandes

6.1 Arquitectura

Struts2 es un framework MVC basado en el modelo MVC model 2 pero, a diferencia de éste último, las acciones pueden toman el rol del modelo. Esto significa que las vistas tienen la capacidad de obtener datos directamente de la acción sin necesidad de tener un objeto intermedio con datos del modelo disponible para traspasar a la vista. Por este motivo se le ha llamado a este modelo pull-MVC. Struts2 está compuesto de cinco componentes principales: acciones, interceptores, pila de valores / OGNL, tipos Result y las vistas.

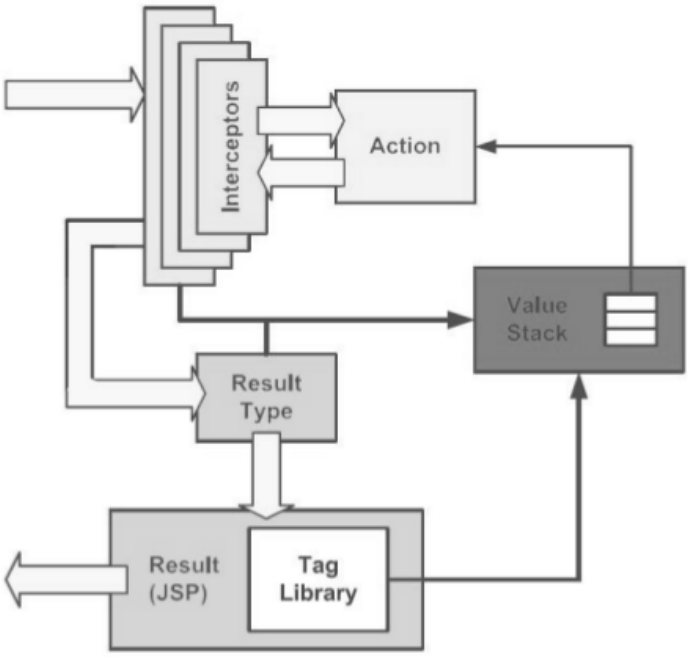


Figura 6.2: Relación entre los diferentes elementos del nucleo de Struts2

6.2 Actions

Las Actions son el núcleo del Framework Struts2. Cada dirección URL tiene mapeada una acción específica la cual provee lógica de proceso para la petición del usuario. Las Actions serán las encargadas de ejecutar la lógica necesaria para manejar una petición determinada. A diferencia de la versión anterior de Struts, los Actions no está obligados a implementar o heredar de una interfaz o clase ya definida. Pueden ser POJO´s.

6.2.1 Interceptors

Proveen lógica de pre-procesamiento antes de que la acción sea llamada Interactúan con la Action, proveyendo información como puede ser la inyección de dependencias controlado por Spring y poner a punto los parámetros de la petición en la Action. Proveen lógica de pro-procesamiento después de que la acción haya sido llamada. Pueden modificar el resultado que esté siendo retornado Recogen excepciones y por lo tanto pueden cambiar el tipo de resultado que se tenga que retornar.

6.2.2 Pila de valores (value stack)

La pila de valores (value stack) y OGNL (Object Graph Navigational Language) están muy relacionadas. El Value Stack es exactamente lo que significa, una pila de valores, que en este caso, es una pila de objetos, aunque existen unas pequeñas diferencias que la diferencian de una pila normal. La primera diferencia es que el contenido de la pila está formado por cuatro niveles.

Otra diferencia es como se usa la pila. En una pila tradicional, para poder llegar a un elemento habría que ir introduciendo o extrayendo hasta llegar al que deseáramos (push & pop). Con la pila Value Stack se realiza mediante búsquedas o evaluando una expresión OGNL. Al igual que en otros lenguajes de expresión, como puede ser el JSTL, OGNL nos proporciona un mecanismo para poder navegar sobre los diferentes objetos de la pila, utilizando una notación de punto, expresiones y llamadas a métodos de los objetos.

6.2.3 Result Type

El Result Type es el tipo de información que se retornará al usuario. La mayoría de estos tipos de resultados ya están pre-configurados en Struts2 o se proveen mediante Plugins. Pero en el caso de que nos encontremos en una situación en que ninguno de los Results Types disponibles no se adapte a nuestro diseño, siempre podemos crear nuestro Result Type personalizado.

6.2.4 Librería de etiquetas

Las librerías de etiquetas se usan en los .jsp para obtener o añadir información a las Actions, se puede decir que proveen una intersección entre las acciones y las vistas, favoreciendo la mantenibilidad y manteniendo la lógica encapsulada, reduciendo la tentación de cortar y pegar código.

Las principales diferencias entre las librerías de Tags de Struts2 y el resto como pueden ser las de JSTL son las siguientes:

- Mejor integración con el Framework Struts2.
- Mejor uso de la Value Stack.
- Mejor uso del OGNL para evaluar expresiones.

Capítulo 7. Spring

Spring es un framework ligero basado en la técnica Inversión de Control (IoC) y una implementación de desarrollo según el paradigma de Orientación a Aspectos (AOP). No obliga a usar un modelo de programación concreto, se ha popularizado en la comunidad de programadores en Java al tratarse de una alternativa y sustituto del modelo de Enterprise JavaBeans.

Uno de los aspectos mas importantes de Spring, es que cuenta con una arquitectura modularizada, y cada uno de sus módulos puede utilizarse de forma independiente. Cada módulo está enfocado en una tarea concreta. Algunos de los módulos están diseñados para permitir la integración con otras herramientas o incluso con otros frameworks. La filosofía es tratar de *no reinventar la rueda*, la cual quiere decir en este caso que si ya hay disponible en el mercado una herramienta que realice una tarea de forma eficiente en un ámbito se debe conjuntar con Spring para llevar a cabo un mejor desempeño y poder obtener así un mejor resultado.

Otro de los principales enfoques de Spring es que simplifica el desarrollo de aplicaciones J2EE, al intentar evitar el uso de EJB. La programación con EJB muchas veces se ha criticado por considerarse demasiado complicada.

7.1 Arquitectura de Spring

Spring es un framework modular que cuenta con una arquitectura divida en siete capas o módulos, como se muestra en la Figura 7.1. Esto nos permite utilizar solamente los módulos que nos interesen y combinarlos con libertad.

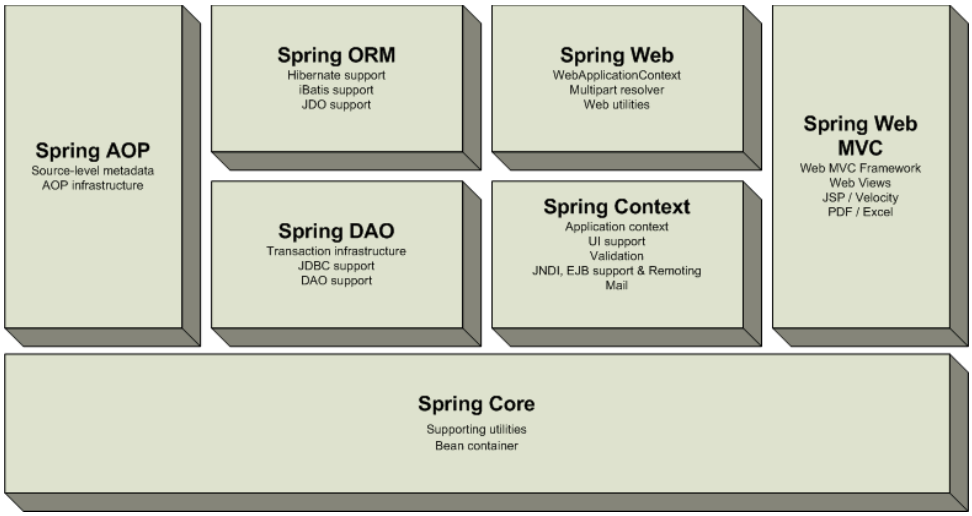


Figura 7.1: Arquitectura modular de Spring

7.1.1 Sprint Core

Este módulo proporciona la funcionalidad esencial del framework, está compuesta por el BeanFactory, el cual utiliza el patrón de Inversión de Control (Inversion of Control) y configura los objetos a través de Inyección de Dependencia (Dependency Injection).

Bean Factory

Es uno de los componentes principales del núcleo de Spring. Es una implementación del patrón Factory, pero a diferencia de las demás implementaciones de este patrón, que muchas veces sólo producen un tipo de objeto, BeanFactory es de propósito general, ya que puede crear muchos tipos diferentes de Beans. Los Beans se pueden llamar por su identificador y ellos mismos se encargan de manejar las relaciones entre objetos. Se soportan objetos de dos tipos diferentes:

- **Singleton.** Existe únicamente una instancia compartida de un objeto con un nombre particular, que se puede llamar o devolver cada vez que se necesite. Este modo está basado en el patrón de diseño que lleva el mismo nombre.

- **Prototype.** También conocido como non-singleton. En este método cada vez que se realiza una llamada se crea un nuevo objeto independiente. BeanFactory es el contenedor real que instancia, configura y administra los beans. Estos beans normalmente colaboran unos con otros, y los por tanto tienen dependencias entre ellos.

Inversion of Control

Una de las funcionalidades más importantes de Spring es el uso del patrón de Inversión de Control (IoC, Inversion of Control) utilizado por el BeanFactory. Esta parte es la encargada de mantener la separación del código de la aplicación que se está desarrollando, los aspectos de configuración y las especificaciones de dependencia del framework. Todo esto se realiza mediante la configuración de los objetos a través de Inyección de Dependencia o Dependency Injection, que se explicará más adelante.

Una forma sencilla de explicar el concepto de IoC es el conocido Principio de Hollywood: "*No nos llames, nosotros te llamaremos*". Traduciendo este principio a nuestro contexto, en lugar de que sea el código de la aplicación el que llame a una clase de una librería, un framework que utiliza IoC llama al código. Por esta razón se llama Inversión, ya que invierte la acción de llamada a alguna librería externa.

Inyección de dependencias

Con este principio, en lugar de que el código de la aplicación utilice el API del framework para resolver dependencias como parámetros de configuración u objetos colaborativos, las clases de la aplicación exponen o muestran sus dependencias a través de métodos o constructores que el framework puede llamar con el valor apropiado en tiempo de ejecución, basado en la configuración.

El principio básico es que los beans definen sus dependencias (es decir, los otros objetos con los que trabajan) sólo a través de argumentos para métodos constructores, argumentos para una factoría o propiedades que se inicializan una vez el objeto se ha creado o devuelto a través de una factoría. Por lo tanto, el trabajo del contenedor es el de inyectar estas dependencias cuando se crea el bean. Esto es fundamentalmente lo inverso (de aquí el nombre) del bean instanciando o localizando sus dependencias por si mismo usando clases o constructores directamente. Se ve claramente que el uso de IoC proporciona código mucho más limpio y además se logra un alto grado de desacoplamiento entre los beans porque éstos no buscan sus dependencias directamente.

La inversión de control / Inyección de dependencias tiene dos variantes principales:

- Inyección de dependencias basada en métodos set (setter-based): se realiza llamando a los métodos set de los beans después de invocar a un constructor sin argumentos o a un método sin argumentos de una factoría estática para instanciar al bean. Spring generalmente recomienda el uso de este tipo de inyección de dependencias porque un constructor con muchos argumentos puede volverse poco manejable, especialmente cuando algunas propiedades son opcionales.

- Inyección de dependencias basada en constructores (constructor-based): se realiza invocando a un constructor con un número de argumentos, cada uno representando a un colaborador o una propiedad. Aunque Spring generalmente recomienda la otra

aproximación para la inyección de dependencias, también se soporta completamente la inyección basada en constructores para permitir el uso de beans ya predefinidas que puedan proporcionar constructores con argumentos pero métodos set. Además, esta aproximación se puede usar como mecanismo para asegurar que beans sencillos no se construyan en un estado inválido.

7.1.2 Spring Context

Mientras que el paquete beans proporciona la funcionalidad básica para administrar y manipular beans, normalmente de un modo programático, Spring context añade el ApplicationContext que mejora la funcionalidad del BeanFactory.

En sí, Spring Context es un archivo de configuración que proporciona información contextual al framework general. Además proporciona otros servicios como JNDI, EJB, e-mail y validación.

Application Context

La base para el paquete context es la interfaz ApplicationContext. Derivando de la interfaz de BeanFactory, Spring Context proporciona toda la funcionalidad de ese interfaz y además añade información de la aplicación que se puede utilizar por todos los componentes, proporcionando además lo siguiente:

- Localización y reconocimiento automático de las definiciones de los Beans
- Carga de múltiples contextos jerarquizados, permitiendo que cada uno se enfoque en una determinada capa.
- Herencia de contextos
- Jerarquía de mensajes y soporte para internacionalización (i18n)
- Acceso a recursos
- Propagación de eventos, para permitir que los objetos de la aplicación publicar y opcionalmente registrarse para ser notificados de los eventos.

7.1.3 Spring AOP

La Programación Orientada a Aspectos, más conocida como AOP por su nombre en inglés Aspect Oriented Programming, es un modelo de programación que aborda un problema específico: capturar las partes de un sistema que los modelos de programación habituales obligan a que estén repartidos a lo largo de distintos módulos del sistema. Estos fragmentos que afectan a distintos módulos son llamados aspectos y los problemas que solucionan, problemas cruzados (crosscutting concerns).

Usando un lenguaje que soporte AOP, podemos capturar estas dependencias en módulos individuales, obteniendo un sistema independiente de ellos y podemos utilizarlos o no

sin tocar el código del sistema básico, preservando la integridad de las operaciones básicas. En Spring es capaz de utilizarse en el manejo de

- Persistencia
- Manejo de transacciones
- Seguridad
- Logging
- Debugging

El contenedor IoC de Spring no depende de la implementación AOP, por tanto el uso de AOP no es obligado. Spring AOP soporta las siguientes funcionalidades:

- Intercepción: se puede insertar comportamiento personalizado antes o después de invocar a un método en cualquier clase o interfaz.
- Introducción: especificando que un advice (acción tomada en un punto particular durante la ejecución de un programa) debe causar que un objeto implemente interfaces adicionales.
- Pointcuts dinámicos y estáticos: para especificar los puntos en la ejecución del programa donde debe de haber intercepción.

7.1.4 Spring ORM

Spring no dispone de un módulo ORM (Object-Relational Mapping) propio. El módulo ORM soporta los frameworks ORM más populares del mercado, como son Hibernate, iBATIS y Apache OJB.

7.1.5 Spring DAO

El patrón DAO (Data Access Object) es uno de los patrones más importantes y usados en aplicaciones J2EE, y la arquitectura de acceso a los datos de Spring provee un buen soporte para este patrón.

Existen dos opciones para llevar a cabo el acceso, conexión y manejo de bases de datos: utilizar alguna herramienta ORM o utilizar el template de JDBC (Java Database Connectivity) que brinda Spring. La elección de una de estas dos herramientas es totalmente libre y en lo que se debe basar el desarrollador para elegir es en la complejidad de la aplicación.

7.1.6 Spring Web MVC

Spring incluye un framework para el desarrollo web basado en el paradigma Modelo-Vista-Controlador (MVC). Aunque es similar en algunas características a otros frameworks populares, Spring Web MVC posee algunas ventajas que le diferencias de otros frameworks. Spring MVC proporciona una implementación lista para usar del típico flujo propio de las aplicaciones web. Es además, muy flexible, permitiendo elegir para la implementación de la

vista de una variedad de tecnologías. También nos permite una integración total con otras capas implementadas en Spring que den soporte a la lógica de negocio de la aplicación, todo esto gracias a las características de inyección de dependencias ya comentadas.

El Web MVC de Spring presenta algunas similitudes con otros frameworks web, pero con algunas características especiales:

- Spring hace una clara división entre controladores, modelos de JavaBeans y views.
- El MVC de Spring está basado en interfaces y es bastante flexible.
- Provee interceptores (interceptors) al igual que controladores.
- Spring no obliga a utilizar JSP como única tecnología para la Vista, es capaz de utilizar otras.
- Los Controladores se configuran de la misma manera que los demás objetos en Spring, a través de IoC.

Para intentar comprender cada parte de la arquitectura del Web MVC de Spring incluimos la Figura 7.2 que representa un diagrama de flujo con el ciclo de vida de una petición request junto con un diagrama de secuencia:

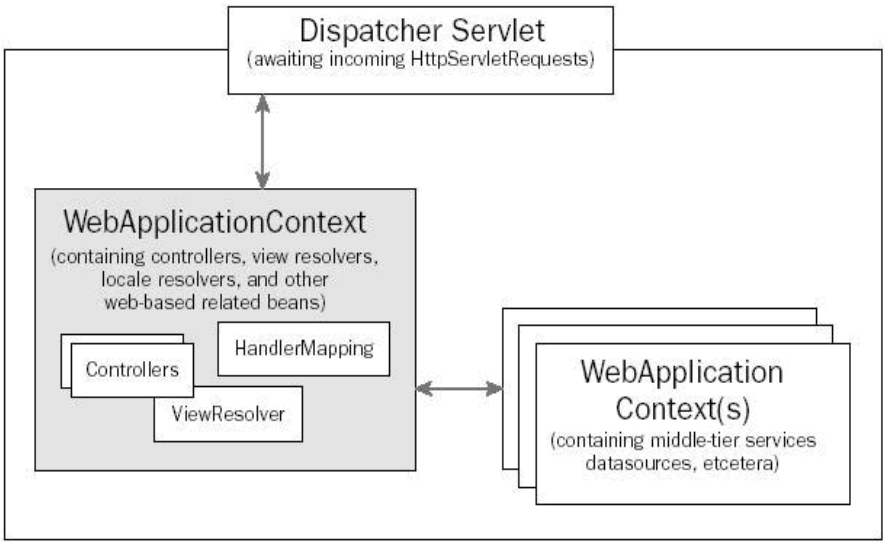


Figura 7.2: Diagrama de flujo de una petición request en Spring

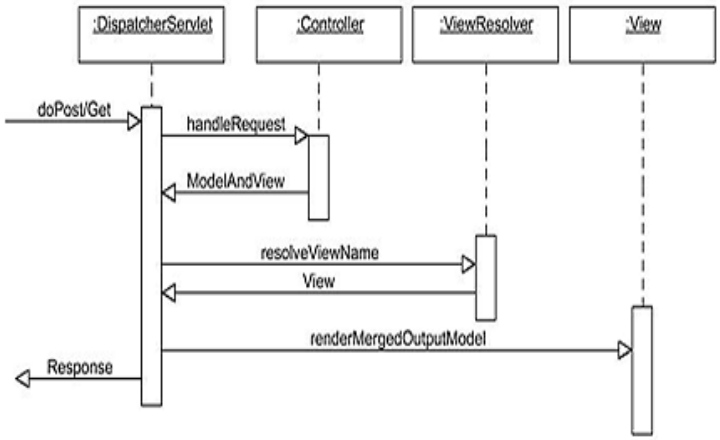


Figura 7.3: Diagrama de secuencia de una petición Request

1. El navegador envía una petición que es recibida por el DispatcherServlet.
2. Se debe escoger que Controller manejará la petición, para esto el HandlerMapping mapea los diferentes patrones de URL hacia los controladores correspondientes, y se retorna al DispatcherServlet el Controller elegido.
3. El Controller elegido recibe la petición y ejecuta la tarea.
4. El Controller regresa un ModelAndView al DispatcherServlet.
5. Si el ModelAndView contiene el nombre lógico de un Vista se utilizará un ViewResolver para buscar ese objeto View que representará a la petición modificada.
6. Finalmente el DispatcherServlet entrega la petición al View.

Spring cuenta con una gran cantidad de controladores, de los cuales podemos elegir el que mas nos interese dependiendo de la tarea. Entre los más populares se encuentra: Controller y AbstractController para tareas sencillas; el SimpleFormController que ayuda a controlar formularios y MultiActionController que ayuda a tener varios métodos dentro un solo controlador a través del cual se pueden mapear las diferentes peticiones a cada uno de los métodos correspondientes.

Dispatcher Servlet

Tanto la configuración del DispatcherServlet como la del servlet central, se tiene que hacer como cualquier servlet normal de una aplicación web, en el archivo de configuración web.xml (Deployment Descriptor).

Handler Mappings

Existen diversas maneras en las que el DispatcherServlet puede determinar que controlador es el encargado de procesar un request, y a que bean del Application Context se lo puede asignar. Esta tarea la lleva a cabo el Handler Mapping o el manejador de mapeo, existen 2 tipos principales:

- BeanNameUrlHandlerMapping: mapea el URL hacia el controlador en base al nombre del bean del controlador.
- SimpleUrlHandlerMapping: mapea el URL hacia el controlador basando en una colección de propiedades declarada en el applicationContext. En este Handler Mapping se declara una lista de propiedades en las cuales se indicará cada una de las URLs como propiedad, con la URL como atributo clave y tomando el valor el nombre del bean que será responsable de procesar la petición o request.

View Resolvers

En Spring MVC una vista o View es un bean que transforma los resultados para hacerlos visibles al usuario. Spring MVC cuenta con cuatro View Resolvers diferentes:

- InternalResourceViewResolver: Resuelve los nombres lógicos en un archivo tipo View que es convertido utilizando una plantilla de archivos como JSP, JSTL o Velocity
- BeanNameViewResolver: Resuelve los nombres lógicos de las vistas en beans de tipo View en el applicationContext del DispatcherServlet
- ResourceBundleViewResolver: Resuelve los nombres lógicos de las vistas en objetos de tipo View contenidos en un ResourceBundle o un archivo con extensión .properties.
- XMLViewResolver: Resuelve los nombres los lógicos de las vistas que se encuentran en un archivo XML separado.

Controladores

Spring dispone de varios controladores cada uno especializado para una tarea en particular. Para poder crear un controlador basta con implementar la interfaz del Controlador deseado y sobrescribir los métodos que sean necesarios para procesar la petición. Esta situación ayuda a poder modularizar la aplicación ya que con la combinación de diversos controladores que sean enfocados a una tarea en particular puede concentrarse en cada parte de aplicación aislada y contribuir a una mejor estructuración.

Existe una variedad de controladores, los cuales poseen una jerarquía. Spring brinda libertad al desarrollador de escoger que tipo de controlador desea implementar, no lo limita como en algunos otros frameworks.

La siguiente Figura 7.4 muestra un diagrama de clases con una lista parcial de los principales controladores.

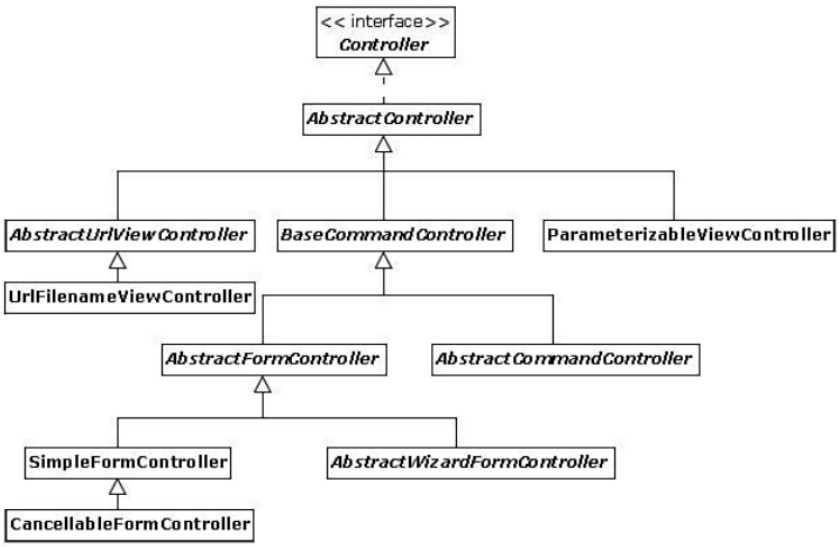


Figura 7.4: Diagrama de clase de los principales controladores en Spring

7.1.7 Procesamiento de formularios

Spring ofrece muchas facilidades para el trabajo con formularios a través de tres controladores diferentes. Se proporcionan medios para facilitar el procesamiento, cumplimentado de los formularios, almacenamiento de datos, así como el manejo de errores de validación y confirmaciones. Este proceso es totalmente transparente para el usuario.

Sprint presenta una buena integración con JSTL. Java Standard Tag Library (JSTL) es un conjunto de etiquetas JSP que ayuda en algunas situaciones comunes de la capa de presentación. JSTL proporciona etiquetas para establecer, obtener y mostrar variables del entorno, iterar en colecciones, crear condiciones, dar formato a fechas y números, entre otras cosas.

Capítulo 8. Java Server Faces

JSF es un framework para aplicaciones Web en Java de Sun Microsystems. Está orientado a la interfaz gráfica de usuario (GUI), facilitando el desarrollo de éstas, y que sin embargo, realiza una separación entre comportamiento y presentación, además de proporcionar su propio servlet como controlador, implementando así los principios del patrón de diseño Model-View-Controller (MVC), lo que da como resultado un desarrollo más simple y una aplicación mejor estructurada.

El enfoque mencionado anteriormente no es nada nuevo. Lo que hace a JSF tan atractivo, entre muchas otras cosas más, es que brinda un modelo basado en componentes y dirigido por eventos para el desarrollo de aplicaciones Web, que es similar al modelo usado en aplicaciones GUI standalone durante años, como es el caso de Swing, el framework estándar para interfaces gráficas de Java.

Otra característica muy importante de JavaServer Faces es que, a pesar de que HTML es su lenguaje de salida predeterminado, no está limitado a éste ni a ningún otro, pues tiene la capacidad de utilizar diferentes renderers para los componentes GUI y obtener así diferentes salidas para enviar al cliente. Así mismo, JSF es suficientemente flexible para soportar diversas tecnologías de presentación, destacando entre éstas JSP, ya que es una tecnología soportada, requerida y especificada para toda implementación de JavaServer Faces.

8.1 Características

JSF incluye las siguientes características principales:

- Un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entradas, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad.
- Un conjunto de componentes por defecto para la interfaz de usuario.
- Dos bibliotecas de etiquetas personalizadas para JavaServer Pages que permiten expresar una interfaz JavaServer Faces dentro de una página JSP.
- Un modelo de eventos en el lado del servidor.

- Administración de estados.
- Beans administrados.
- JSF es una especificación desarrollada por la Java Community Process.

Las principales implementaciones de JSF son:

- **JSF Reference Implementation** de Sun Microsystems.
- **MyFaces** proyecto de Apache Software Foundation.
- **ICEfaces** Contiene diversos componentes para interfaces de usuarios más enriquecidas, tales como editores de texto enriquecidos, reproductores de multimedia, entre otros.
- **jQuery4jsf** Contiene diversos componentes sobre la base de uno de los más populares framework javascript jQuery.

8.1.1 Descripción

Los principales componentes de la tecnología JavaServer Faces son:

- Un API y una implementación de referencia para: representar componentes UI y manejar su estado; manejo de eventos, validación del lado del servidor y conversión de datos; definir la navegación entre páginas; soportar internacionalización y accesibilidad; y proporcionar extensibilidad para todas estas características.

- Una librería de etiquetas JavaServer Pages (JSP) personalizadas para dibujar componentes UI dentro de una página JSP.

Este modelo de programación bien definido y la librería de etiquetas para componentes UI facilitan de forma significativa la tarea de la construcción y mantenimiento de aplicaciones Web con UIs del lado del servidor. Con un mínimo esfuerzo, podemos:

- Conectar eventos generados en el cliente a código de la aplicación en el lado del servidor.
- Mapear componentes UI a una página de datos del lado del servidor.
- Construir un UI con componentes reutilizables y extensibles.
- Grabar y restaurar el estado del UI más allá de la vida de las peticiones de servidor.

Como se puede apreciar en la Figura 8.1, el interfaz de usuario que creamos con la tecnología JavaServer Faces (representado por myUI en el gráfico) se ejecuta en el servidor y se renderiza en el cliente.

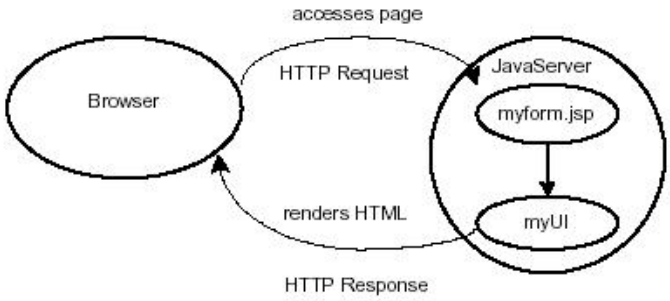


Figura 8.1: Flujo de una petición Request en JSF

La página JSP, myform.jsp, dibuja los componentes del interfaz de usuario con etiquetas personalizadas definidas por la tecnología JavaServer Faces. El UI de la aplicación Web (representado por myUI en la imagen) maneja los objetos referenciados por la página JSP:

- Los objetos componentes que mapean las etiquetas sobre la página JSP.
- Los oyentes de eventos, validadores, y los conversores que están registrados en los componentes.
- Los objetos del modelo que encapsulan los datos y las funcionalidades de los componentes específicos de la aplicación

8.1.2 Terminología Básica

- **Componente UI.** Se trata de un objeto con estado, mantenido por el servidor, que proporciona funcionalidad específica para interactuar con un usuario final. Los Componente UI son JavaBeans con propiedades, métodos y eventos. Están organizados en una vista (View), que es un árbol de componentes normalmente mostrados como una página.
- **Renderer.** Es el responsable de mostrar un componente UI y traducir la entrada del usuario en valores de componentes. Los Renderers pueden ser diseñados para trabajar con uno o más Componente UI, y un componente UI puede ser asociado con varios Renderer diferentes.
- **Validador.** Es el responsable de asegurar que el valor introducido por un usuario es correcto. Podemos asociar uno o más validadores a un componente UI.
- **Backing Beans.** JavaBeans especializados que recogen valores de los componentes UI e implementan métodos listener de eventos. También pueden almacenar referencias a componentes UI.
- **Converter.** Convierte un valor de un componente a y desde una cadena para mostrarlo.Los converter se asocian a un componente UI.

- **Events y Listeners.** JSF usa el modelo event/listener de JavaBeans (también usado por Swing). Los Componente UIs (y otros objetos) generan eventos y los listeners pueden ser registrados para manejar dichos eventos.
- **Mensajes.** Información que se muestra al usuario. Cualquier parte de la aplicación (backing beans, validadores, converters, etc...) pueden generar información o mensajes de error que pueden ser mostrados de vuelta al usuario.
- **Navegación.** Representa la capacidad de movernos de una página a la siguiente. JSF tiene un sistema de navegación avanzado que está integrado con escuchadores de eventos especializados.

8.1.3 Pasos del Proceso de Desarrollo

Desarrollar una sencilla aplicación JavaServer Faces requiere la realización de estos pasos:

- Desarrollar los objetos del modelo, los que contendrán los datos.
- Añadir las declaraciones del bean controlado al fichero de configuración de la aplicación.
- Crear las páginas utilizando las etiquetas de componentes UI y las etiquetas core.
- Definir la navegación entre las páginas.

Estas tareas se pueden realizar simultáneamente o en cualquier orden. Sin embargo, la gente que realice las tareas necesitará comunicarse durante el proceso de desarrollo. Por ejemplo, el autor de las páginas necesita saber los nombres de los objetos del modelo para poder acceder a ellos desde la página.

8.2 Ciclo de vida

Otro aspecto muy importante dentro de JavaServer Faces es su ciclo de vida, el cual es similar al de una página JSP: el cliente hace una petición HTTP y el servidor responde con la página en HTML. Sin embargo, debido a las características que ofrece JSF, el ciclo de vida incluye algunos pasos más.

El proceso de una petición estándar incluye seis fases, como se muestra en la Figura 3.1.1, representadas por los rectángulos blancos:

En la Figura 8.2 los rectángulos etiquetados con la leyenda Process Event representan la ejecución de cualquier evento producido durante el ciclo de vida. El funcionamiento de cada etapa se describe brevemente a continuación.

1. **Reconstitute Component Tree:** es la primera etapa que se lleva a cabo, e inicia cuando se hace una petición. Su objetivo es la creación de un árbol con todos los componentes de la página en cuestión.

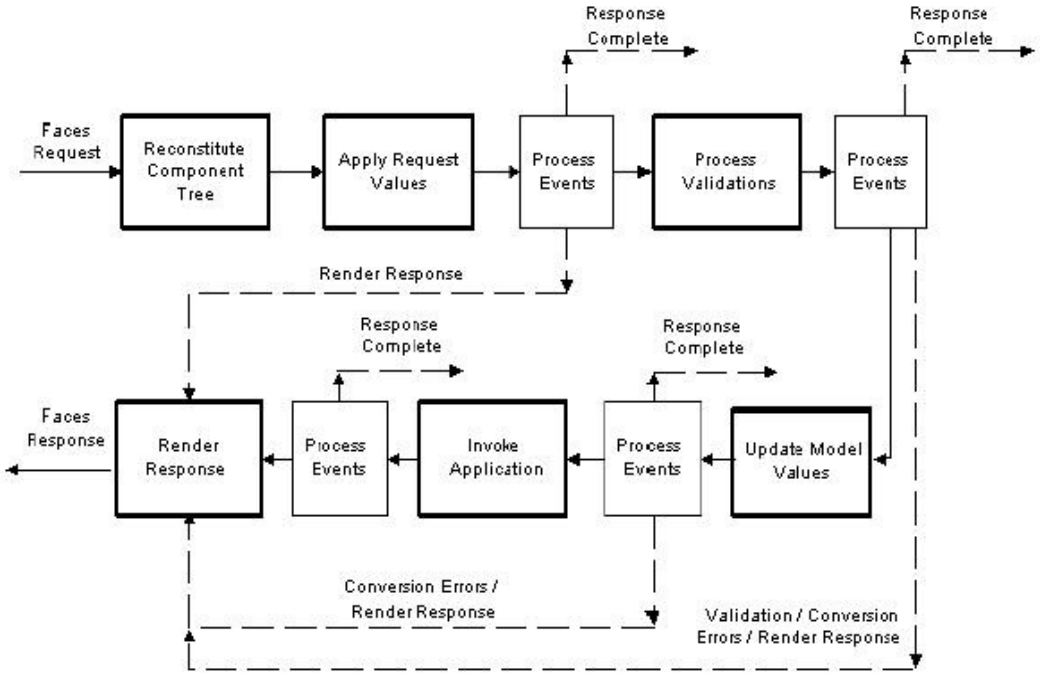


Figura 8.2: Ciclo de vida JSF

2. **Apply Request Values: c**ada uno de los componentes del árbol creado en la fase anterior obtiene el valor que le corresponde de la petición realizada y lo almacena.
3. **Process Validations**: después de almacenar los valores de cada componente, estos son validados según las reglas que se hayan declarado.
4. **Update Model Values:** durante esta fase los valores locales de los componentes son utilizados para actualizar los beans que están ligados a dichos componentes. Esta fase se alcanzará únicamente si todas las validaciones de la etapa anterior fueron exitosas.
5. **Invoke Application:** se ejecuta la acción u operación correspondiente al evento inicial que dio comienzo a todo el proceso.
6. **Render Response:** la respuesta se renderiza y se regresa al cliente. Dependiendo del éxito o fracaso de las tareas en cada una de las fases del ciclo de vida, el flujo normal descrito puede cambiar hacia caminos alternos según sea el caso.

8.3 Navegación

La navegación se refiere a la forma en que se desplegarán las diferentes páginas de la aplicación a través de los eventos producidos por el usuario. La navegación se determina por medio de reglas declaradas en el archivo faces-config-xml, que serán evaluadas con la respuesta obtenida del evento generado en la página actual para determinar la siguiente vista a desplegar. Es decir, cuando el usuario hace clic en un botón, y éste origina un submit de la forma, la aplicación debe analizar los datos mandados y determinar qué página desplegar a continuación. Para determinar la vista a mostrar, la aplicación toma la respuesta generada por el

atributo action del componente que originó el evento, y en base a ésta y a las reglas establecidas, selecciona la página adecuada.

Existen dos tipos diferenciados de navegación: navegación estática y dinámica.

Navegación estática . Considere el caso en el que un usuario rellena un formulario de una página web. El usuario puede escribir en campos del texto, puede hacer clic sobre enlaces, pulsar botones o seleccionar elementos de una lista, entre otras muchas cosas. Todas estas acciones ocurren dentro del navegador del cliente. Cuando, por ejemplo, el usuario pulsa un botón, envía los datos del formulario y éstos son gestionados por el servidor. Al mismo tiempo, el servidor JSF analiza la entrada del usuario y debe decidir a que página ir para dar la respuesta.

En una aplicación web simple, la navegación es estática. Es decir, pulsar sobre un botón suele redirigir al navegador a una misma página para dar la respuesta.

Navegación dinámica. En la mayoría de aplicaciones web, la navegación no es estática. El flujo de la página no depende de qué botón se pulsa, sino que también depende de los datos que el cliente introduce en un formulario. Por ejemplo, una página de entrada al sistema puede tener dos resultados: El éxito o el fracaso. Y en función del resultado, el usuario tendrá que recibir una página diferente.

8.4 Conversión y Validación

Antes de que los datos ingresados por el usuario sean almacenados en los beans correspondientes del modelo, estos deben pasar por dos procesos necesarios de JSF: conversión y validación, siempre en este orden.

La conversión es el proceso a través del cual se transforman los datos ingresados por el usuario en la forma Web en los tipos de datos que mapean en sus correspondientes variables de su bean. Es decir, los datos que proporciona el usuario, manejados como cadenas de texto, se transforman en int, Date o según sea el tipo de dato que se ha definido para cada uno de ellos en el bean especificado.

La validación es el proceso mediante el cual JSF analiza los datos del usuario de manera semántica, de acuerdo a ciertas reglas especificadas para cada uno de estos, como pueden ser la longitud mínima y máxima para un campo de la forma.

Ambos procesos tienen como objetivo filtrar los datos del usuario, de tal manera que los valores que se guardan en los beans sean coherentes y aceptados por el proceso llevado en la lógica aplicativa.

8.5 Manejo de Eventos

Los eventos juegan un papel muy importante dentro de JavaServer Faces, como se ha mencionado. Los componentes GUI de JSF responden a las acciones del usuario disparando eventos, que serán manejados por código de la aplicación, llamados listeners, que han sido registrados para ser notificados de la ocurrencia del evento.

Java Server Faces soporta tres tipos de eventos para los componentes GUI:

- **Eventos de cambio de valor.** Son disparados con los componentes de entrada cuando el valor del componente cambia y el formulario es tramitado.
- **Eventos de acción.** Son disparados por componentes de comando, cuando el botón o el enlace es activado.
- **Eventos de fase.** Son disparados por el ciclo de vida de JSF.

La idea central del manejo de eventos es registrar los listeners adecuados para cada componente GUI, de tal manera que se realicen las operaciones apropiadas y deseadas.

8.6 Estructura de una aplicación JSF

Se necesitan dos cosas para correr aplicaciones con JavaServer Faces: un contenedor Web para Java y una implementación de la especificación de JSF. Los elementos propios de JavaServer Faces que conforman una aplicación son típicamente los siguientes:

- Archivos JSP, que constituyen la interfaz gráfica de la aplicación y que contienen las diversas funcionalidades de JSF, como los tags que representan los componentes GUI.
- Archivos XML, específicamente el archivo faces-comfig.xml que almacena las diferentes configuraciones y elementos a utilizar en la aplicación. Un ejemplo de este archivo se muestra en la Figura 3.1.4.
- Archivos Java, típicamente desempeñando el rol de beans.
- Archivos de paquetes de mensajes (message bundles).

8.6.1 JavaBean

De acuerdo a la especificación de JavaBeans, estos se definen como componentes de software reusables que pueden ser manipulados en una herramienta de desarrollo. De una manera más simple, un JavaBean se puede ver como una clase Java tradicional o un POJO (Plain Old Java Object) que sigue ciertas especificaciones de programación.

Las características más importantes de un bean son las propiedades que ofrece. Una propiedad es cualquier atributo de un bean que tiene un nombre, un tipo una serie de métodos para obtener (leer) y/o establecer (escribir) el valor de la propiedad.

En JavaServer Faces se usan beans cuando se necesita conectar clases Java con páginas Web; en otras palabras, se utilizan para los datos que necesitan ser accedidos desde una página. Los beans son el medio para conectar la interfaz gráfica con la lógica aplicativa.

En JSF los beans son utilizados en las páginas que representan la interfaz gráfica, típicamente JSP, específicamente en los componentes. Para poder hacer uso de estos beans, se deben declarar en el archivo faces-config.xml.

8.6.2 Paquetes de Mensajes

Mejor conocidos como message bundles, estos paquetes agrupan todos los mensajes a desplegar en una página Web, es decir, la parte estática de éstas, con el objetivo de facilitar la internacionalización. Los mensajes de la aplicación deberán ser guardados en un archivo con el formato “properties”

8.7 JBoss Seam

JBoss Seam es una plataforma de integración de tecnologías que tiene como objetivo facilitar el desarrollo de aplicaciones JEE (sobre todo aplicaciones web). Es un proyecto Open Source con una comunidad abierta y aunque cuenta con el respaldo de JBoss (que es una división de Red Hat), su funcionalidad no está ligada al servidor de aplicaciones de este. El desarrollo de Seam, utilizando JSF y EJB que son las tecnologías con más soporte, es muy ágil ya que reduce el nivel de configuración necesario para la integración y aprovecha al máximo las ventajas de cada una de las tecnologías haciendo al proyecto mas estable, legible, predecible y mantenible.

8.7.1 Características

Tenemos como su característica más importante a la habilidad de integrar tecnologías de diferentes ámbitos a través de la inyección de dependencias. A eso se le agrega el tema de los contextos, tenemos más contextos lo que no quiere decir que ocupemos más memoria, si no que todo lo contrario vamos a tener más opciones para elegir donde guardar nuestras instancias y éstas van a permanecer en memoria el tiempo indicado.

Seam ofrece un generador automático de aplicaciones, Seam-gen, que permite generar el esqueleto de un CRUD funcional que permita altas, bajas cambio y modificaciones a partir de una base de datos existente. El desarrollo WYSIWYG es facilitado a través del uso de las JBoss Tools, que es un conjunto de plug-ins diseñados para el entorno integrado de desarrollo Eclipse. Seam puede ser integrado con las bibliotecas de componentes JSF JBoss RichFaces o con ICEsoft ICEFaces. Ambas bibliotecas poseen soporte para AJAX.

Principales características de Seam

- Integra JSF con EJB 3.0
- Integra AJAX

- Programación declarativa utilizando meta-información)
- Soporta bijeccion e inyección de dependencias
- Los annotations reemplaza a los ficheros XML

8.7.2 Arquitectura de una aplicación Seam

El aspecto más relevante de Seam es la forma en la que integra el uso de varias tecnologías ya existentes para la creación de aplicaciones web en Java para facilitar la implementación del patrón MVC de una forma que resulta más intuitiva para desarrollador y más rápida de programar, pero sin perder la potencia y las características que provee JEE.

8.7.3 Integración de EJB3 y JSF

EJB3 es la última versión de EJB (Enterprise Java Beans). EJB es una especificación de un framework que define unos componentes software (los Enterprise Java Beans), que debidamente diseñados y configurados por el desarrollador, y alojados en un servidor de aplicaciones JEE, conforman la capa del modelo en el patrón MVC (a la capa del modelo también se le llama lógica del negocio bussiness logic en el entorno de la gestión empresarial).

EJB3 provee una gran cantidad de funcionalidad al desarrollador a la hora de implementar el modelo, funcionalidad que de una u otra forma éste tendría que implementar por sí mismo. Entre otras se encuentran las siguientes características:

- Proceso de Transacciones. El servidor JEE se encarga de que las modificaciones que se realicen al modelo que estén encerradas dentro de una transacción se realizarán todas o ninguna.
- Integración con los servicios de persistencia que ofrece Java Persistence API (JPA). JPA integra el mapeo ORM como un la forma de persistencia para los 'beans de entidad' de EJB. Su equivalente en Ruby on Rails es el ActiveRecord. En realidad, la parte de persistencia de EJB3 es la integración en el API de EJB de los conceptos de ORM implementados con gran éxito por Hibernate .
- Control de concurrencia.
- Pool de recursos y clustering, que asegura la escalabilidad del producto.
- Seguridad.

8.7.4 Los Componentes y Contextos Seam

Seam aprovecha las capacidades de extensión de JSF y utiliza EJB3 para producir la integración de la siguiente forma:

- El desarrollador puede crear componentes del modelo (los componentes Seam) que van a vivir en el servidor como componentes Seam, pero que van a poder ser accedidos directamente desde la capa de presentación, como si fueran JSF Managed Beans. Seam se encargará de controlar su ciclo de vida, dependiendo del contexto al que pertenezcan.

- El método para crear dichos componentes es simplemente crear un POJO con la funcionalidad deseada, o un Enterprise Java Bean, y añadirle las anotaciones java necesarias para que el servidor sepa qué tipo de componente quiere y sus otras características. Los Spring Beans también se pueden convertir en componentes Seam.
- Cada componente se enlaza a uno de los contextos disponibles en el framework. En este sentido, el concepto de contexto de Seam es similar al concepto de ámbito de aplicación, pero además, Seam se encargará de crear o destruir los componentes, dependiendo de la necesidad que de éste se tenga, y de cómo haya sido configurado. Existen varios contextos en Seam:
    - Stateless context
    - Event (i.e., request) context
    - Page context
    - Conversation context
    - Session contextg
    - Business process context
    - Application context
- También existe una librería de modelos de uso de Seam, que ayuda al programador a realizar pantallas CRUD con muy poco esfuerzo. Esta librería es la usada por la utilidad de generación de pantallas CRUD llamad seam-gen, incluida en la distribución.

8.7.5 La Bi-inyeccion (Bijection)

La noción de Inversión del Control de Dependencia existe en JSF y EJB y otros frameworks, como Spring. Consiste, a grandes rasgos, en delegar en el framework el conocimiento de como instanciar un determinado componente software, evitando la dependencia del consumidor del servicio prestado por el componente del método empleado para la instanciación del mismo.

Con manejo de contextos de Seam, los componentes se declaran para pertenecer a un determinado contexto. Cuando un componente es inyectado en otro, es el Seam el que maneja el ciclo de vida del componente, permitiendo a una clase cliente obtener instancias de objetos cuyo ciclo de vida es mas largo que el suyo.

Seam también introduce el *outjection*, que significa que una clase cliente puede instanciar y modificar objetos, que tras esa modificación, estarán disponibles para otros componentes, siempre dependiendo del entorno del componente *outjected*.

8.7.6 Ventajas e inconvenientes

El uso de Seam es recomendable por las siguientes ventajas:

- **Facilita la integración con JSF** JSF, en su especificación 1.x, está fuertemente ligado a XML, Seam reemplaza la configuración XML con unas cuantas anotaciones,

reduciendo el código y haciendo que el programador sea más productivo. Seam extiende JSF con funcionalidades extra como operaciones multiventanas y administración del espacio de trabajo, validación basada en el modelo, flujo de páginas basado en jBPM, internacionalización y fragmentación del cacheo de página.

- **Mejora el modelo de Persistencia** Seam fue diseñado e implementado por las mismas personas que crearon Hibernate, las mismas personas que acuñaron el termino de contexto de persistencia. El modelo de conversación de Seam resuelve los problemas de programación referentes a la persistencia causados por las arquitectura de aplicaciones no orientadas a estados. Tanto el caso de que se utilice Hibernate o JPA, Seam hace sencillo y natural el uso de contextos extendidos de persistencia, y ayuda a evitar replicaciones innecesarias de estados cuando se use un contexto de persistencia extendida en un entorno en grupo.

- **Uso extendido de las anotaciones** Seam es el primer modelo de programación que permite usar anotaciones Java, desde la capa de persistencia hasta el interfaz de usuario. Ya no resulta necesario entrar en conflictos con el uso de XML, ya que las tareas habituales de programación se simplifican mediante el uso de las anotaciones.

- **Integra EJB3 de forma sencilla** Seam ofrece un soporte para la integración de EJB3 con JSF. LA relación entre ambos frameworks es mas sencilla mediante el uso de Seam y sus características de biyección que dan soporte a las necesidades de ambos.

Capítulo 9. Análisis del framework SOFP

Una vez estudiadas las particularidades de la plataforma JEE, los patrones mas importantes de su catálogo y las características principales de los frameworks actuales, vamos a definir cuales serán los requisitos y estructura del framework SOFP, objetivo de este proyecto.

9.1 Funcionalidad

Tal como se ha visto en el estudio de los diferentes frameworks, existe un conjunto de funcionalidad y características comunes importantes que nuestro framework debe incorporar, entre las que se encuentran:

- Control declarativo. Se debe permitir construir el flujo de navegación de forma declarativa, esto es, debe permitir declarar la sucesión de vistas que debe seguir la aplicación, definir las acciones a ejecutar y relacionarlos con objetos de validación mediante reglas de forma externa al código.

- Validación de datos. Debe proporcionar mecanismos para facilitar el proceso de validación y conversión de tipos de los datos de las peticiones, así como informar de los errores de validación y conversión al usuario automáticamente.

- Colecciones de etiquetas especializados. Se debe proporcionar librerías de etiquetas que faciliten el desarrollo de las vistas y evite el uso de código en las mismas.

- Gestión de excepciones. Se deben proporcionar mecanismos que permitan de forma declarativa tratar las diferentes excepciones que pueden lanzan las acciones al ejecutar la lógica de negocio.

- Internacionalización. Se dispondrá de los mecanismos que permitan extraer los mensajes de la aplicación desde las vistas a un fichero de configuración, con el objeto de poder manejar de forma simple la adaptación de los mensajes de la aplicación al idioma del usuario.

9.2 Arquitectura

Nuestro framework estará basado en la arquitectura MVC ya descrita en el apartado 3.3 La arquitectura MVC del Capítulo 3.

En el estudio que hemos realizado de los distintos frameworks del mercado hemos estudiado dos aproximaciones diferentes, las implementadas por Struts 1, Struts 2 y Spring, que siguen un modelo basado en acciones, y la solución de JSF que se basa en un modelo basado en componentes. Para nuestro framework utilizaremos la arquitectura MVC Model 2 basada en acciones,

A continuación vamos a detallar un nuevo patrón denominado Service to Worker, que no es mas que la agrupación de algunos patrones ya conocidos. Este patrón define un marco global donde se separan las distintas responsabilidades a la hora de manejar el envío de una petición y junto con la arquitectura MVC Model 2, descrita anteriormente, conformará la arquitectura del framework SOFP.

9.2.1 Patrón Service to Worker

Tal y como se muestra en el diagrama de clases de la Figura 9.1 el patrón Service to Worker se organiza alrededor de la utilización de otros patrones ya conocidos. El control centralizado, el manejo de las peticiones y la creación de las vistas son realizados por un Front Controller, un Application Controller y un View Helper respectivamente.

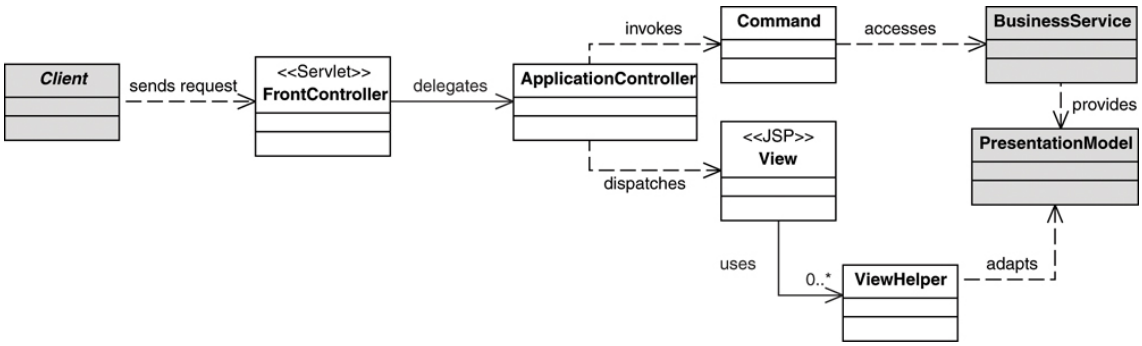


Figura 9.1: Diagrama de clases del patrón Service to worker

El Front controller se encarga del manejo de los elementos que forman parte de la petición. El papel del control se delega al patrón Application Controller, que recibirá un objeto de tipo Context Object que encapsula la petición. El Application Controller actúa como un Command Handler, encargándose del mapeo de los nombres lógicos a sus correspondientes comandos, para posteriormente invocar al comando correspondiente.

El funcionamiento del patrón es el siguiente. El Front Controller recibe la petición y la delega al Application Controller, el cual se ocupará de llevar a cabo el manejo de las acciones

correspondientes, resolviendo la petición entrante al comando apropiado para invocarlo luego. El comando internamente invocará a un servicio de la capa de negocio, el cual devolverá un modelo de presentación adecuado para la vista, que podrá ser una vista de tipo Composite View.

Finalmente, el sistema es el encargado de controlar el flujo de ejecución y el acceso a los datos de negocio, a partir de los cuales se crea el contenido que la aplicación presentará finalmente al usuario.

El diagrama de secuencia de la Figura 9.2 representa la interacción de los elementos y el intercambio de mensajes que éstos realizan.

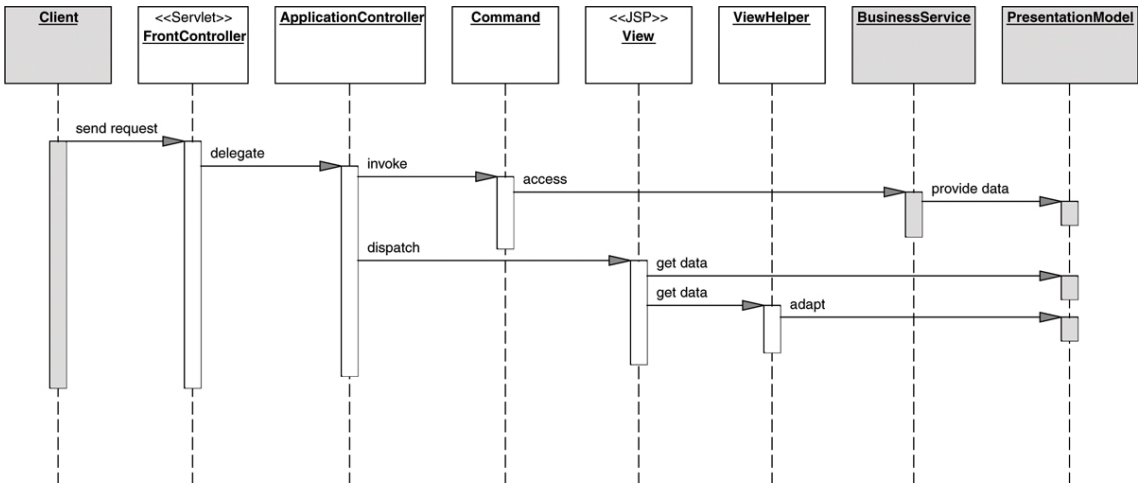


Figura 9.2: Diagrama de secuencia del patrón Service to worker

- **Front Controller.** Es el encargado de manejar la petición, delegando al Application Controller el manejo de las acciones y las vistas.

- **Application Controller.** Es el responsable del manejo de las acciones y vistas. Su misión es elegir la acción apropiada y su vista correspondiente para satisfacer la petición. Para los casos mas simples, este comportamiento puede ser incluido en el propio Front Controller.

- **View.** La vista es la encargada de representar y mostrar la información al cliente. El modelo adecuado para la presentación es adaptado por los View Helpers. La vista normalmente será un Composite View.

- **View Helper.** Un helper es el responsable de ayudar o apoyar a una vista o a un controlador a llevar a cabo un proceso específico. En nuestro caso, un ViewHelper recupera y adapta los datos para ayudar a que sean representados por la vista.

- **Presentation Model.** El presentation Model mantiene los datos que se han recuperado del servicio de negocio y son usados para la generación de la vista.

Todos estos patrones se han descrito con mas detalle en el apartado 3.2 Patrones principales J2EE para la capa de presentación del Capítulo 3. dedicado a los patrones de la capa de presentación.

Algunas de las ventajas del uso del patrón Service to Worker y de la arquitectura MVC Model 2 son las siguientes

- Por medio del uso del patrón Service to Worker se centraliza el control y se logra que el sistema sea mas modular, reusable y mantenible. Centralizar el control y el manejo de las peticiones mejora la modularidad del sistema y su reusabilidad. El código común encargado del procesamiento puede reutilizarse, reduciendo el tipo de duplicación que suele presentarse cuando la lógica de procesamiento se incluye dentro de las propias vistas. Menos duplicación de código también significa mejorar la mantenibilidad, ya que los cambios se hacen en una única localización.

- Se mejora la separación de roles. Se centraliza el control y la lógica del manejo de las peticiones, manteniéndolo separado del código de creación de las vistas permitiendo una separación mas claras de los roles en los equipos. Los desarrolladores se pueden dedicar a mantener la lógica de programación, mientras que los diseñadores pueden hacerlo sobre las vistas.

En general, el uso de la arquitectura planteada para el frameworks ofrece las siguientes ventajas

- La aplicación desarrollada e beneficia de la separación de la presentación y la lógica de control.
- Se posibilita que puedan existir distintas representaciones de los mismos datos (vistas diferentes).
- Se simplificar el desarrollo.
- Se permite una separación de roles en el equipo de trabajo, permitiendo que los diseñadores gráficos trabajen sobre las vistas y los programadores en las acciones.
- El uso de del patrón de MVC Model 2 y el patrón Service to Worker nos permite disponer de un punto de control único centralizado.

Capítulo 10. Arquitectura y diseño del framework SOFP

10.1 Estrategia final

El uso del patrón Front Controller nos indica que el sistema debe disponer de un punto de acceso centralizado único para el manejo de las peticiones. Esto nos permitirá una mayor integración en la recuperación de contenidos, manejo de la seguridad, manejo de las reglas de negocio, control de errores y estrategias de selección de contenidos.

De las muchas estrategias que podemos seguir a la hora de implementar el controlador nos hemos inclinado por utilizar la estrategia denominada Servlet Front Strategy, donde el controlador será implementado como un Servlet. Mediante la configuración adecuada del fichero descriptor de implementación web.xml podremos redirigir fácilmente todas las peticiones hacia el servlet controlador

El patrón Front Controller suele trabajar junto con un componente de tipo Dispatcher que es el responsable del manejo de las vistas y la navegación. Su responsabilidad es elegir cual será la siguiente vista y dirigir el control al recurso. En nuestro caso el papel del Dispatcher lo realizará otro componente basado en el patrón Application Controller. Las tareas que se realizarán serán la resolución de la petición entrante al comando apropiado, la posterior invocación del comando y finalmente la entrega del control a la vista apropiada.

10.2 Requerimientos

Para el desarrollo del framework SOFP se ha utilizado el lenguaje de programación Java, haciendo uso de componentes compatibles con las especificaciones J2EE. Es necesario una máquina virtual de Java versión 1.6 o superior.

El producto generado será una librería JAR de nombre frameworkSOFP-1.0.jar que debe incluirse dentro de cada proyecto Web que haga uso del framework y ejecutado en un contenedor de aplicaciones compatible con las especificaciones J2EE. El desarrollo se ha hecho sobre un servidor Tomcat v1.6

10.3 Estructura y jerarquía de paquetes

El código fuente se ha organizado en diferentes paquetes Java. Un Paquete en Java es un espacio de nombres y contenedor de clases que permite agrupar las distintas partes de un programa cuya funcionalidad tienen elementos comunes, el uso de paquetes nos permite el agrupamiento de clases, la reutilización de código y garantiza que no haya duplicidad de clases.

El Diagrama 10.1 muestra la estructura de paquetes. En la Tabla 10.1 se describe la funcionalidad de cada paquete dentro del framework SOFP.

Nombre del paquete	Descripción
org.sofp	Clases principales
org.sofp.ambito	Clases de ayuda para manejar el ámbito
org.sofp.comandos	Clase abstracta de definición de comandos
org.sofp.config	Clases para el manejo de la configuración
org.sofp.excepciones	Clases para el manejo de errores y excepciones
org.sofp.forms	Interface de definición de los FormBeans
org.sofp.taglib	Clases de definición de la librería de etiquetas

Tabla 10.1: Jerarquia de paquetes Java

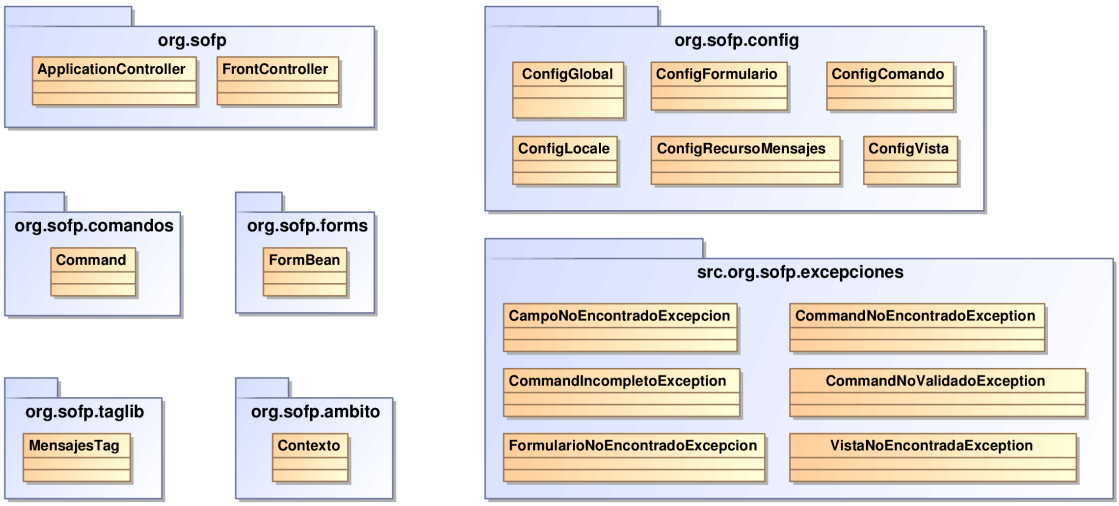


Figura 10.1: Diagrama de paquetes

10.4 Dependencias

El diagrama de dependencias de la Figura 10.2 representa las dependencias externas del proyecto. Se han utilizado algunas de las librerías del proyecto Apache Common, un proyecto de la fundación Apache que ofrece distintos tipos de componentes Java reutilizables. Para la construcción de la librería de etiquetas ha sido necesario utilizar la librería JavaServer Pages Standard Tag Library (JSTL) que encapsula la funcionalidad principal de las librerías de etiquetas. Todo esto junto con las librerías proporcionadas por Tomcat, el contenedor de Servlets del proyecto Jakarta de la fundación Apache.

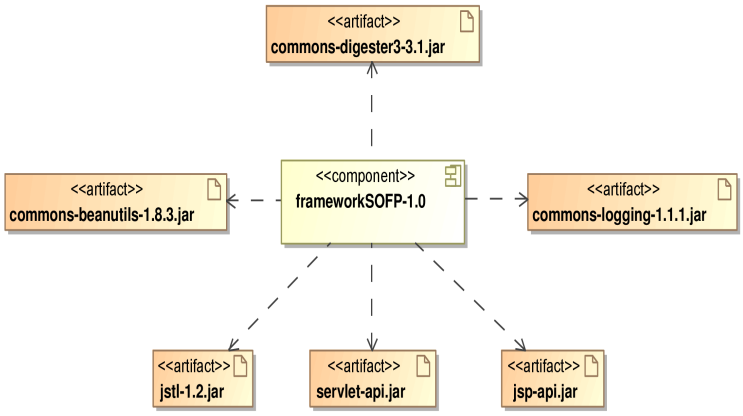


Figura 10.2: Diagrama de dependencias del proyecto

10.5 Diccionario de clases y métodos

Se ha generado el diccionario de clases y métodos mediante la herramienta Javadoc, la documentación se encuentra dentro de la carpeta del proyecto. Para más información consultar el anexo A que contiene las instrucciones de compilación y despliegue de la librería.

10.6 Características del diseño

Describimos ahora los detalles de diseño principales del framework

10.6.1 Conceptos

A continuación presentamos los conceptos de Comando y Form Bean, a los que se hará referencia en este capítulo.

Comandos

Hemos denominado comando a cada una de las acciones que se ejecutan en el servidor asociadas al procesamiento o submit de un formulario HTML. El comando será el encargado de

realizar el procesamiento de los datos necesario en el servidor, ya sea invocando a otros componentes de la capa de negocio o íntegramente.

Los comandos son reconocidos por el framework por un identificador que forma parte de la URL de la petición. La aplicación Web está parametrizada en el fichero descriptor web.xml para que todas las URL que se dirigen a un comando sean capturadas por el servlet que hace el papel de Front Controller.

Una vez tomado el control por el Front Controller, se analizará la URL de petición para obtener el identificador del comando. Con esta información se localizará en la configuración cual es la clase correspondiente al comando asociado a ese identificador, será la clase que actuará como ejecutora de la lógica y que hemos denominado Command. La clase correspondiente al comando será instanciada dinámicamente en tiempo de ejecución e invocada por el Application Controller.

Tal y como muestra el diagrama de clases de la Figura 10.3 todos los comandos deben implementar la interface Command del paquete org.sofp.commands, esta interface define los métodos ejecutar y validar, que implementarán todas las clases comand. El comando recibe como parámetros el propio contexto de Servlet junto con un objeto Form Bean que representa una clase simple que facilita el acceso a los datos del formulario HTML. Previamente a la ejecución del comando se realiza una validación mediante la invocación al método validar por parte del Application Controller..

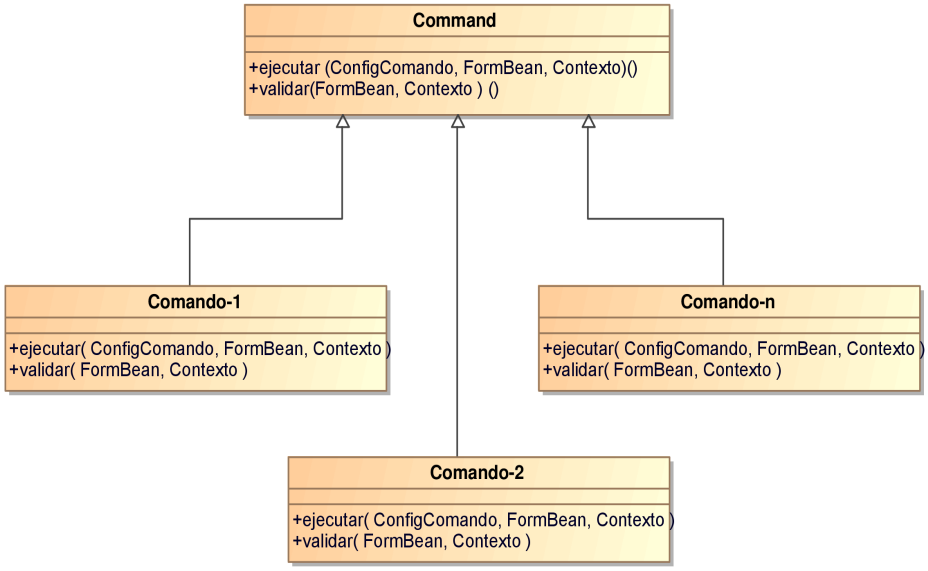


Figura 10.3: Diagrama de clase de la clase Command

Form Beans

Los Fom Beans son clases que facilitan el acceso a los campos de entrada del formulario HTML a los objetos Comando. Estas clases contienen tantos atributos como campos estén presente en el formulario, junto con sus correspondientes métodos get y set.

Cuando un usuario completa un formulario y lo envía, el Application Controller a partir de la configuración localiza cual es el Form Bean asociado al formulario y dinámicamente lo instancia y automáticamente asigna a cada campo su valor correspondiente. El mapeo entre los nombres de los atributos y los campos del formulario HTML es automático, con la restricción de que los nombres de los campos y los atributos deben coincidir. Los Bean son pasados como parámetro a los objetos comandos en su ejecución.

Como puede verse en el diagrama de clases de la Figura 10.4 cada clase FormBean extienden de la clase abstracta FormBean del paquete org.sofp.forms. Esta clase abstracta ya cuenta con la implementación del método que realiza el poblado del FormBean a partir de los datos del formulario.

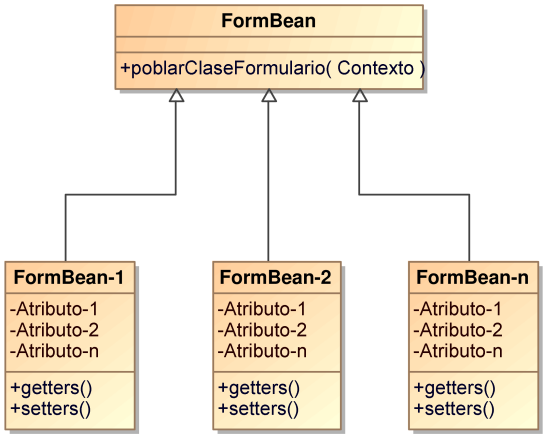


Figura 10.4: Diagrama de clases de la clase FormBean

10.6.2 Configuración

La configuración del framework SOFP se realiza de forma declarativa en el fichero soft-config.xml. Es el propio servlet que actúa como Front Controller el encargado de la lectura de la configuración. La lectura se realiza durante la instanciación del propio servlet, en la ejecución del método init. La instanciación se produce la primera vez que el servlet se carga en memoria por parte del servidor de aplicaciones.

La Tabla 10.2 muestra un ejemplo de fichero de configuración sofp-config.xml.

La información de configuración debe estar disponible durante todo el ciclo de vida del framework, por lo que para optimizar el rendimiento, una vez leída y parseada se almacena en una variable de aplicación para que sea accesible por todas las sesiones de la aplicación.

La Figura 10.5 muestra el diagrama de clases correspondiente a las clases responsables de la definición, lectura y manejo de la configuración.

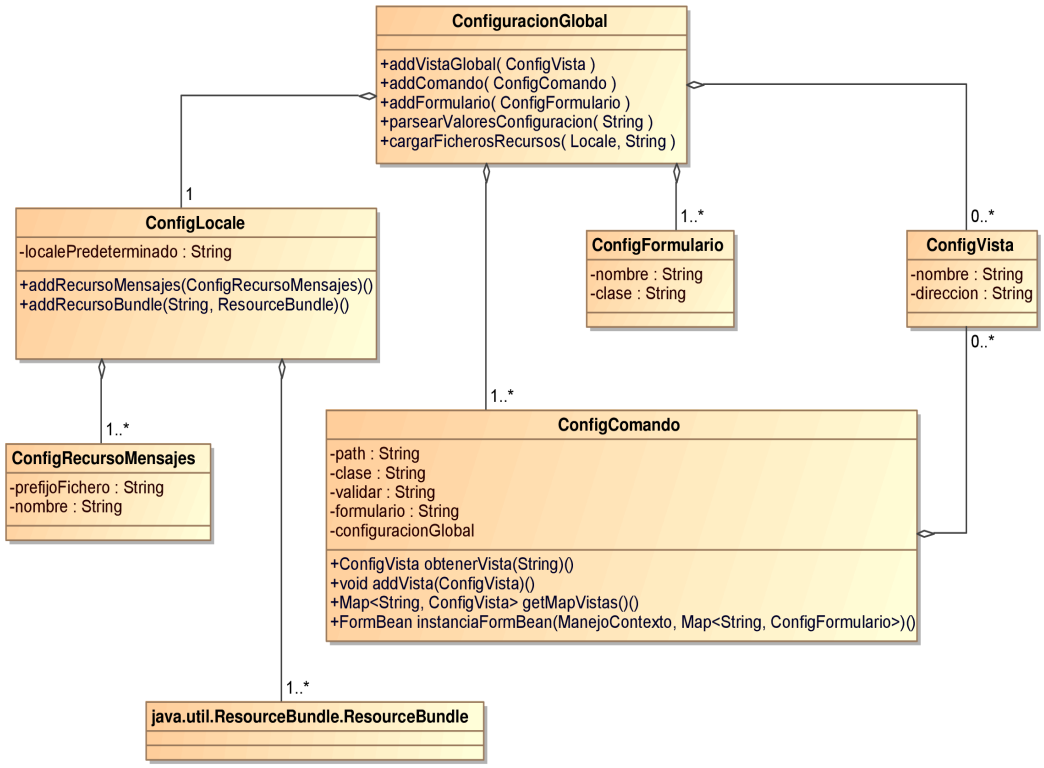


Figura 10.5: Diagrama de clases de las clases relacionadas con la configuración

```
<?xml  version="1.0"  encoding="UTF-8"?>
<configuracion>

    <vistasGlobales>
        <vista  nombre="help"  direccion="/help.jsp"  />
        <vista  nombre="errorDesconocido"  direccion="/sofpError.jsp"  />
    </vistasGlobales>

    <recursos>
        <recurso-mensajes  prefijoFichero="fichero"  nombre="errores"  />
    </recursos>

  <formularios>
     <formulario  nombre="login"  clase="org.sofp.test.formbeans.LoginFormbean"  />
     <formulario  nombre="detalleNoticia"
                  clase="org.sofp.test.formbeans.IdNoticiaFormbean" />
     <formulario  nombre="BorrarComentarioForm"
                  clase="org.sofp.test.formbeans.IdComentarioFormbean"  />
     <formulario  nombre="NoticiaForm"
                  clase="org.sofp.test.formbeans.NoticiaFormbean"  />
  </formularios>

  <comandos>
       <comando  path="/NoticiasMostrar"
                 clase="org.sofp.test.comandos.NoticiasMostrarCmd"
                 formulario="VacioForm">
           <vista  nombre="ok"  direccion="/NoticiasMostrar.jsp"  />
       </comando>

       <comando  path="/NoticiaBorrar"
                 clase="org.sofp.test.comandos.NoticiasBorrarCmd"
                 formulario="detalleNoticia">
          <vista  nombre="ok"  direccion="/NoticiasBorrarOk.jsp"  />
          <vista  nombre="error"  direccion="/NoticiasBorrarError.jsp"  />
       </comando>

       <comando  path="/NoticiaAnadir"
                 clase="org.sofp.test.comandos.NoticiasAnadirCmd"
                 formulario="NoticiaForm">
          <vista  nombre="ok"  direccion="/NoticiasAnadirOk.jsp"  />
       </comando>

       <comando  path="/ComentariosMostrar"
                 clase="org.sofp.test.comandos.ComentariosMostrarCmd"
                 formulario="detalleNoticia">
          <vista  nombre="ok"  direccion="/ComentariosMostrar.jsp"  />
       </comando>

       <comando  path="/ComentarioBorrar"
                 clase="org.sofp.test.comandos.ComentarioBorrarCmd"
                 formulario="BorrarComentarioForm">
          <vista  nombre="ok"  direccion="/ComentarioBorrarOk.jsp"  />
          <vista  nombre="error"  direccion="/ComentarioBorrarError.jsp"  />
       </comando>

       <comando  path="/Bienvenida"
                 clase="org.sofp.test.comandos.BienvenidaCmd"
                 formulario="login">
           <vista  nombre="ok"  direccion="/Bienvenida.jsp"  />
       </comando>
     </comandos>
</configuracion>
```

Tabla 10.2: Ejemplo de fichero de configuración sofp-config.xml

Parseo del XML de configuración

En lo que se refiere al parseo del fichero de configuración. En lugar de utilizar los tradicionales parsers tipo SAX o DOM como por ejemplo XERCES del Apache XML Project, hemos seleccionado el parser conocido como Digester, perteneciente al conjunto de librerías de componentes de Apache Commons.

Digester nos permite procesar los documentos XML en una forma que podemos denominar *dirigida por eventos*. Con este método de trabajo, durante la lectura del XML por parte del parser, cada vez que un determinado patrón o elemento XML se reconoce, los objetos Java necesarios se crean y los métodos apropiados se invocan de forma automática durante el procesamiento del fichero. Digester es similar al parseo SAX (*Simple API for XML Parsing*), aunque su uso es mas simple, ya que nos permite definir las reglas de parseo de forma declarativa en un fichero de configuración, en lugar de hacerlo programáticamente.

El bloque de código de la Tabla 10.3 corresponde al fichero de configuración del procesado de la configuración del *framework SOFP.*.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE digester-rules PUBLIC "-//Apache Commons //DTD digester-rules XML V1.0//EN"
"http://commons.apache.org/digester/dtds/digester-rules-3.0.dtd">

<digester-rules>
    <pattern value="configuracion">
        <pattern value="vistasGlobales">
            <pattern value="vista">
                <object-create-rule classname="org.sofp.config.ConfigVista" />
                <set-properties-rule />
                <set-next-rule methodname="addVistaGlobal" />
            </pattern>
        </pattern>

        <pattern value="formularios">
            <pattern value="formulario">
                <object-create-rule classname="org.sofp.config.ConfigFormulario" />
                <set-properties-rule />
                <set-next-rule methodname="addForm" />
            </pattern>
        </pattern>

        <pattern value="comandos">
            <pattern value="comando">
                <object-create-rule classname="org.sofp.config.ConfigComando" />
                <set-properties-rule />
                <pattern value="vista">
                    <object-create-rule classname="org.sofp.config.ConfigVista" />
                    <set-properties-rule />
                    <set-next-rule methodname="addVista" />
                </pattern>
                <set-next-rule methodname="addComando" />
            </pattern>
        </pattern>
    </pattern>
</digester-rules>
```

Tabla 10.3: Fichero de configuración del parser Digester

Descripción de las Clases

A continuación realizamos una breve descripción de las clases encargadas de manejar la configuración. Todas las clases pertenecen al paquete org.sofp.config y se corresponden a las mostradas en el diagrama de clases de la Figura 10.5.

ConfigFormulario . Define un formulario definido por el usuario. Contiene el nombre lógico del formulario y la clase del su correspondiente FormBean.

ConfigVista. Una vista define uno o varios elementos que se corresponden con el concepto de vista en el patrón MVC. Cada ítem tiene un nombre lógico de vista y una dirección. Cada vez que se ejecuta un comando, éste realiza el procesamiento que le corresponde y en función del resultado o de la lógica de su proceso retornará el nombre de la vista que mostrará el resultado.

Es posible definir vistas no asociadas a ningún comando en particular. Este tipo de vistas, denominadas globales, se utilizarán para vistas que son reutilizadas y no dependen de ningún comando en concreto.

ConfigComando. Contiene la información asociada a un comando. Cada comando se compone de los siguientes elementos:

- **Atributo path.** Es una cadena identificadora que debe ser única. Forma parte del path de petición que recibe el front controller. En nuestro caso, las peticiones vendrán formadas por el path comand/path. Donde comando es el patrón (command/*) que hemos definido en el fichero descriptor de implementación web.xml para realizar el mapeo de las peticiones a nuestro Servlet Controller.

- **Atributo clase.** Es el nombre de la clase comando que hemos asociamos al comando. Esta clase es la encargada de realizar el procesamiento correspondiente. El procesamiento se realiza en el método ejecutar de cada comando, que será invocado por el Application Controller. Recordemos que éste es el encargado de centralizar y modularizar el manejo de los comandos y las vistas.

- **Atributo validar.** Es un campo booleano que indica si es necesario ejecutar alguna validación de los datos de entrada. La validación se incluye en el lado del servidor y dentro de cada objeto comando. La validación se implementa en el propio objeto command dentro del método validar..

- **Atributo formulario.** Cada command normalmente utiliza un formulario HTML para recoger valores. En este elemento se indica el nombre lógico del formulario. Un mismo formulario puede ser compartido entre distintos Commands.

- **Atributo mapVistas.** Corresponde a una mapa de las vistas. Está implementado como una colección de vistas indexada por el nombre de la vista. Contiene todas las posibles vistas que tiene asociado cada comando.

ConfigLocale. Se utiliza para dotar de características de internacionalización a la aplicación. Es posible definir en el fichero de configuración varios grupos de ficheros de mensajes diferentes, cada grupo se utilizará en distintos ámbitos, mensajes de error, mensajes de ayuda, mensajes de aviso, etc. Para cada grupo se definirá un identificador del grupo y un prefijo para el fichero de mensajes. El sistema localizará todos los ficheros de mensajes en cada uno de los idiomas que estén disponibles por su prefijo correspondiente.

Posteriormente se darán mas detalles sobre el funcionamiento de la internacionalización en el framework SOFP.

El prefijo se completa con un sufijo que representa el valor de un locale que describirá un idioma y un país. Este sufijo será la abreviatura del idioma seguida de un carácter '_' y luego el código del país en mayúsculas. El código es_ES se corresponde a España con idioma Español, ca_ES para idioma Catalán en España y en_US para idioma inglés en EEUU.

Cada grupo de mensajes se define en una clase ConfigRecursoMensajes, que contiene el prefijo del fichero y el nombre del grupo de recursos.

La clase ConfigLocale también está relacionada con una colección de objetos ResourceBundle que pertenecen al paquete java.util.ResourceBundle del sistema y son los que proporcionan los métodos nativos Java para trabajar con este tipo de ficheros de recursos de soporte a la internalización.

ConfiguracionGlobal. Contiene la configuración global. Incluye el mapa de comandos, el mapa de formularios y el mapa de vistas globales. También contiene métodos para añadir elementos a estos mapas. Esta clase es instanciada desde el método init del Servlet que actúa como Front Controller mediante el parseo del fichero de configuración sofp-config.xml. Como ya se ha comentado anteriormente, el parseo se realiza con el parser Digester, perteneciente al conjunto de librerías de componentes de Apache Commons. Se trata de un parseo *dirigido por eventos*, por lo que son los métodos de esta clase los que permiten añadir a los mapas correspondientes nuevos comandos, formularios y vistas globales, siendo invocados por Digester de forma automática durante el parseo.

10.6.3 Validación de datos en el servidor

Todos los comandos implementan la clase comand del paquete org.sofp.command. Existen dos métodos que deben implementar obligatoriamente. Uno de ellos es el método ejecutar, del que ya hemos hablado anteriormente. Existe otro método de nombre validar, que nos permite realizar una validación simple en el lado del servidor. Aquí el programador podrá añadir las reglas que desee para validar los datos de entrada. En caso de que el método detecte un valor incorrecto en algún campo del formbean, lanzará una excepción

CommandNoValidadoException que se propagará hasta el Application Controller y será mostrara al usuario.

Este no es el método más eficiente de realizar validaciones. Es mucho más ágil realizar la validación en el lado del cliente utilizando un lenguaje de script en el lado del cliente, como por ejemplo Javascript. De esta forma evitamos tener que hacer el submit y enviar todos los datos al servidor y posteriormente tener que esperar la respuesta de error de éste. La mayoría de los frameworks estudiados permiten este tipo de validación en el servidor, en algunos casos, como en Struts es posible definir reglas mediantes expresiones y definirlas declarativamente, por lo que en la mayoría de los casos la definición de las reglas de validación no implica escritura de código, como ocurre en el framework SOFP.

10.6.4 Gestión de errores

Se han creado varios tipos de excepciones para controlar diferentes errores de la aplicación. Todas las excepciones pertenecen al paquete org.sofp.excepciones. Algunas de ellos referidas a inconsistencias en el fichero de configuración o en la instanciación del Formbean.

Cuando se produce un error, se propaga hacia arriba en la pila de llamadas, hasta que el Application Controller lo captura. En ese momento se define una variable de ámbito request con el mensaje de error que se desea mostrar y se redirige a una vista genérica cuya misión es mostrar el correspondiente mensaje de error al usuario.

El diagrama de clases de las excepciones se muestra en el Diagrama de Clases de la Figura 10.6. Su propósito se explican a continuación.

CampoNoEncontradoExcepcion. Se lanza durante la instanciación del Formbean en tiempo de ejecución. Es posible que durante esta fase, se intente localizar en el formulario Web el valor correspondiente a un atributo del Formbean asociado a ese formulario y no se localice. Normalmente este tipo de situaciones se corresponde a algún tipo de error humano al definir el fichero de configuración.

CommandIncompletoExcepxion. Se lanza si se produjo algún problema a la hora de localizar la definición de un comando en el fichero de configuración. Si alguno de los atributos del comando, ya sea la clase que se corresponde con el comando o el Formbean que se asocia con el formulario es inconsistente con la configuración de la aplicación, se lanza esta excepción.

CommandNoEncontradoExcepción. Esta excepción se lanza cuando se solicita una url al servidor, esta es capturada por el servlet que actúa como Front Controller y a la hora de obtener el nombre de su comando correspondiente, este no se encuentra definido en la configuración de la aplicación.

CommandNoValidadoExcepcion. Se produce cuando un comando no supera la fase de validación previa a su ejecución. El código de validación construirá el mensaje de error con un mensaje apropiado que se devolverá al usuario.

FormularioNoEncontradoExcepcion. Se lanza cuando en la definición de la configuración existe una referencia a un formulario inexistente.

VistaNoEncontradaExcepcion. Se lanza cuando tras la ejecución de un comando, este retorna el identificador de una vista que no se ha definido ni como vista local ni como vista global en el fichero de configuración.

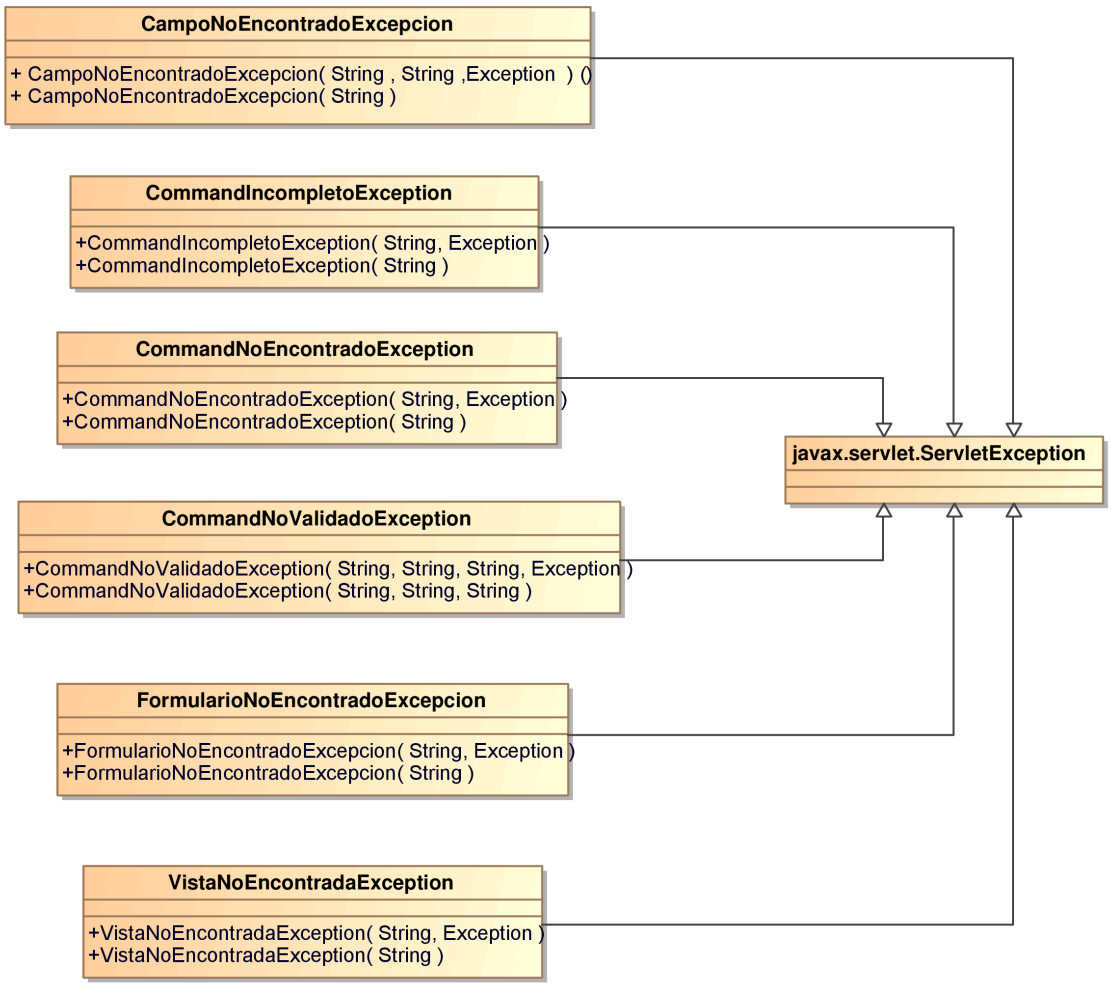


Figura 10.6: Diagrama de clases de las distintas Excepciones

10.6.5 Internacionalización

Es posible definir en el fichero de configuración varios grupos de ficheros de mensajes diferentes, cada grupo se utilizará en distintos ámbitos, mensajes de error, mensajes de ayuda, mensajes de aviso, etc. Para cada grupo se definirá un identificador del grupo y un prefijo para el fichero de mensajes. El sistema localizará todos los ficheros de mensajes en cada uno de los idiomas que estén disponibles por su prefijo correspondiente.

En el nombre del archivo de recursos, el prefijo se completa con un sufijo que representa el valor de un ámbito locale que describirá un idioma y un país. Este sufijo será la abreviatura del idioma seguida de un carácter '_' seguido del código del país en mayúsculas. El código es_ES se corresponde a España con idioma Español, ca_ES para idioma Catalán en España y en_US para idioma inglés en EEUU.

Cada grupo de mensajes se define en una clase ConfigRecursoMensajes, que contiene el prefijo del fichero y el nombre del grupo de recursos. Los detalles de dicha clase pueden verse en la diagrama de la Figura 10.5.

La clase ConfigLocale también está relacionada con una colección de objetos ResourceBundle que pertenecen al paquete java.util.ResourceBundle del sistema y son los que proporcionan los métodos nativos Java para trabajar con este tipo de ficheros de recursos que dan soporte a la internacionalización.

Para manejar los mensajes que formen parte de la internacionalización en las vistas se ha incorporado al framework una librería de etiquetas. De esta forma mediante una etiqueta específica en la que indicamos cual es el grupo de recursos y el identificador asociado al mensaje, podremos mostrar en la vista el mensaje en el idioma apropiado al locale que corresponda.

La aplicación puede definir mediante el fichero de configuración un locale predeterminado para el framework. En caso de existir una configuración en el fichero global de propiedades se tomará dicha configuración como valor predeterminado, en caso de que no se haya indicado ninguno se tomará la del navegador, que se obtendrá durante el procesado de la configuración por medio del objeto request del contexto del servlet.

Para la definición de la biblioteca de etiquetas se ha creado la clase MensajesTag que extiende la clase TagSupport del paquete javax.servlet.jsp.tagext. También ha sido necesario crear un fichero de descripción de la librería de etiquetas que se denomina mensajesTag.tld. El diagrama de clases se representa en el diagrama de clases de la figura 10.7.

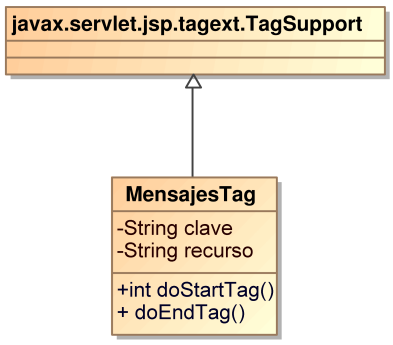


Figura 10.7: Diagrama de clases de la librería de etiquetas MensajesTag

Como ejemplo de uso, en el fragmento de código de la Tabla 10.4 vemos como sería una vista que muestra un mensaje de error en función de la configuración actual del sistema.

```
<%@ taglib uri="mensajesTag.tld" prefix="m" %>

<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<title>Error</title>
</head>
<body>

<m:mensaje clave="errorFormato" recurso="errores"/>

</body>
</html>
```

Tabla 10.4: Vista JSP mostrando un mensaje de error

10.7 Procesamiento y manejo de las peticiones

A continuación mostramos cual es el proceso que se sigue en el procesamiento de las peticiones y cuales son los elementos que intervienen.

10.7.1 Recepción por el Front Controller

Como se ha comentado anteriormente, SOFP emplea el patrón Front Controller. Mediante el uso de este patrón disponemos de un punto de acceso centralizado único para el manejo de las peticiones. Esto nos permitirá centralizar la recuperación de contenidos, manejo de la seguridad, manejo de las reglas de negocio, control de errores y diferentes estrategias de

selección de contenidos. La Figura 10.8 muestra el diagrama de clases correspondiente al Front Controller, Application Controler y resto de clases relacionadas.

Para que la clase Front Controller actúe será necesario realizar una configuración apropiada en el fichero descriptor de implementación web.xml situado en el directorio WEB-INF/ de cada aplicación que se desarrollo bajo este framework. El fragmento de código de la Tabla 10.5 muestra un fichero de configuración de ejemplo. En este fichero describimos la asociación entre el patrón /command/* con nuestro servlet Front Controller, de forma que cualquier petición que contenga este prefijo en su path será redirigida al servlet.

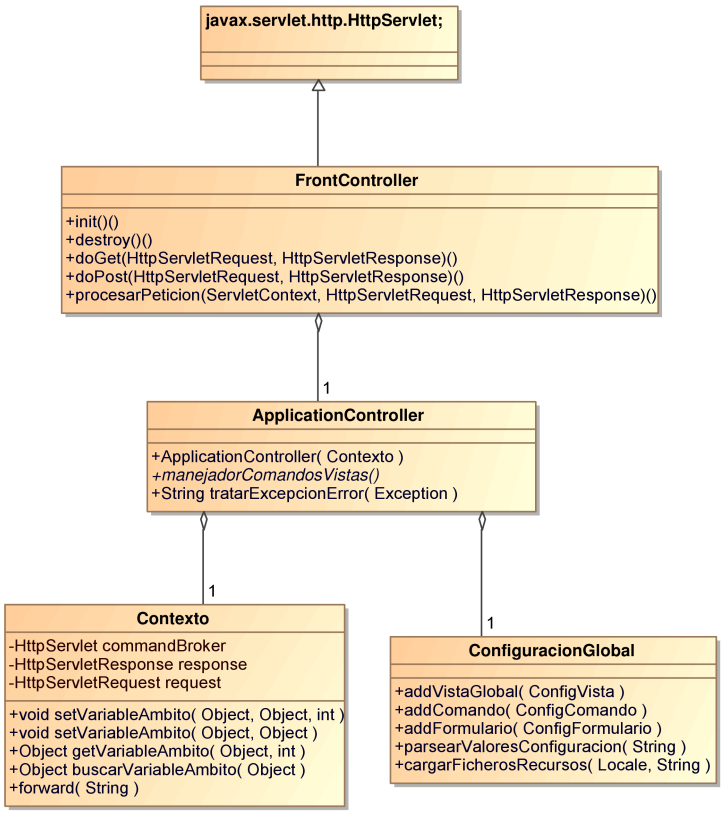


Figura 10.8: Diagrama de clases de Front Controller y Application Controller

```
<?xml version="1.0" encoding="UTF-8"?>
  <display-name>PEC3</display-name>
  <welcome-file-list>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <description></description>
    <display-name>FrontController</display-name>
    <servlet-name>FrontController</servlet-name>
    <servlet-class>org.sofp.FrontController</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>FrontController</servlet-name>
    <url-pattern>/command/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Tabla 10.5: Fichero descriptor web.xml

10.7.2 Manejo de la petición por el Application Controller

El patrón Application Controller trabaja junto con el patrón Front Controller tal y como se ha indicado en capítulos anteriores en la descripción del patrón conocido como Service to Worker. Las tareas que son responsabilidad del Application Controller son las siguientes:

- La resolución de la petición entrante al comando apropiado,
- La posterior invocación del comando, primero a la validación y luego a la ejecución.
- Finalmente, en función de la vista retornada por el el comando, se realiza la entrega del control a la vista apropiada.

El diagrama de secuencia de la Figura 10.9 representa el intercambio de mensajes entre el Front Controller, el Application Controler y resto de clases involucradas en la instanciación dinámica de los Comandos y Formbeans.

El framework SOFP implementa el Application Controller en la clase ApplicationController del paquete org.sofp. El diagrama de clases se corresponde con la Figura 10.8.

Mas detalladamente, las tareas que realiza el Application Controller son las que se indican a continuación.

- Primero se obtiene el nombre del comando a ejecutar a partir del path de la petición que se ha pasado al Front Controller. En nuestro caso, como ya se ha indicado, todos los comandos comienzan su path con el patrón command/*.

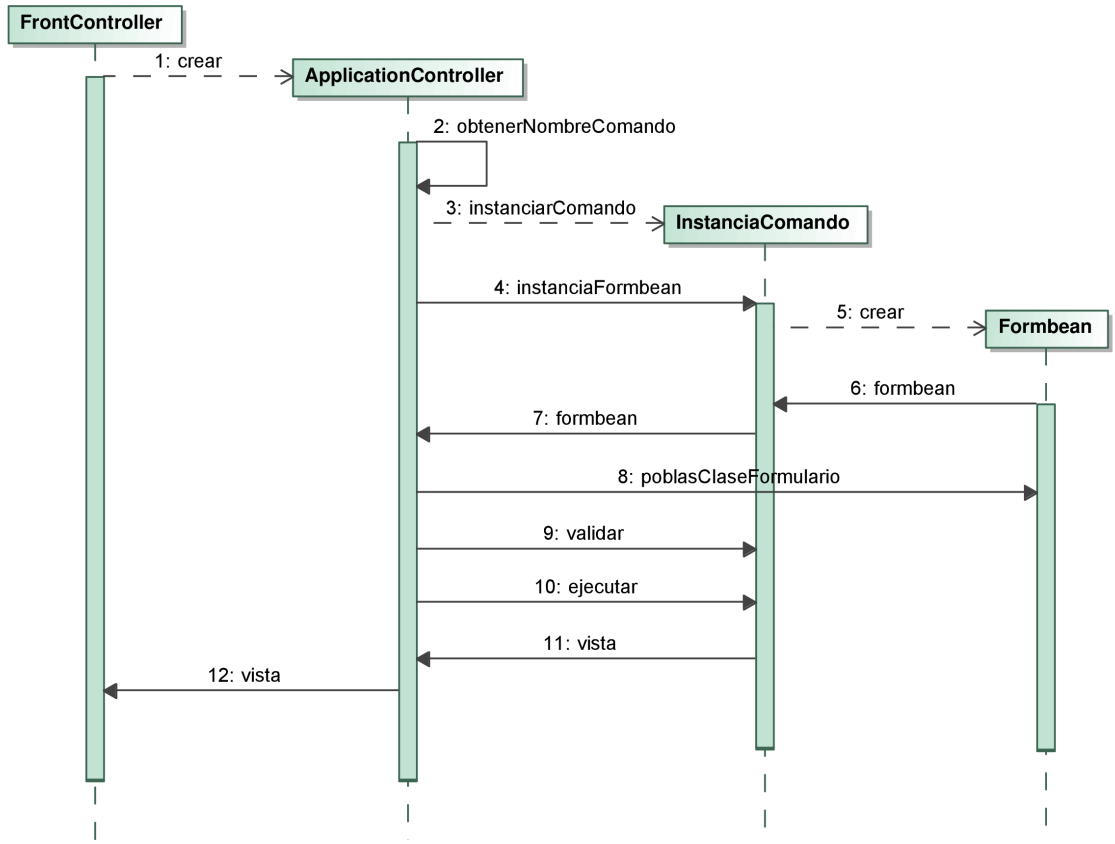


Figura 10.9: Diagrama de secuencia del funcionamiento del patrón Service to Worker

- Posteriormente, una vez disponible el nombre del comando, se localiza en la configuración y se obtiene cual es el nombre de la clase encargada de la ejecución del comando. Todas las clases de tipo comando implementan la interface Command del paquete org.sofp.

- Para terminar, a partir del nombre de la clase comando asociada a nuestro comando, de forma dinámica se instancia la clase a partir del nombre. En caso de que el comando no se localice en la configuración, se muestra un error en el navegador.

A modo de ejemplo, para clarificar lo explicado anteriormente, si se solicita al servidor de aplicaciones una url que contiene el siguiente path.

```
/command/borrado?id=3
```

- Primero se asignará la cadena borrado como identificador del comando. Posteriormente se buscará en la configuración cual es la clase del comando que implementa esta acción.

- Una vez localizada la clase e instanciada dinámicamente, tendremos que recuperar los valores del formulario HTML y adaptarlos a un formato mas manejable, de esto se encargará el FormBean asociado al formulario que a su vez está asociado al comando y cuyas relaciones se han definido en el fichero de configuración correspondiente. Para ellos volveremos a mirar en la configuración cual es el nombre del formulario asociado a nuestro comando y luego localizar la definición de ese formulario nuevamente en la configuración. Posteriormente crearemos dinámicamente en tiempo de ejecución una instancia de la clase FormBean correspondiente

- Ahora nos encargaremos de poblar la clase FormBean, por lo que también recorreremos dinámicamente sus atributos y los iremos asignado a los elementos del formulario de entrada HTML. Una vez terminado el proceso tendremos un objeto Form Bean con los valores de los campos del formulario. Este objeto será recibido por el comando como parámetro.

- A continuación invocaremos al método execute del comando, pasando como argumentos la configuración, el FormBean con los datos de entrada y los objetos request y response del servlet.

- Para terminar, el comando ejecutará sus reglas de negocio y una vez concluida su ejecución tendrá que retornar al Application Controller el nombre de la vista a la que se debe realizar la redirección, normalmente para mostrar los datos obtenidos por el comando.

  Para ello se usará el método obtenerVista de la configuración, que a partir del identificador de la vista nos retornará un objeto vista, ya sea de las vistas asociadas al comando o una de las vistas globales definidas en la configuración y que pueden ser compartidas por diversos comandos.

10.7.3 Contexto e intercambio de datos entre comandos y vistas

El intercambio de datos entre el comando y la vista que utiliza el framework SOFP es el nativo que nos ofrece la tecnología JSP. Éste se realiza añadiendo desde el comando los objetos deseados al ámbito apropiado (sesión, aplicación, request), esto nos permite luego el acceso a los datos desde la página JSP deforma muy cómoda mediante el uso de JavaBeans y de las etiquetas apropiadas.

Para trabajar de forma más cómoda con el contexto se ha creado la clase de nombre Contexto, cuyo detalle puede verse en la Figura 10.8. Esta clase nos ofrece una capa de abstracción sobre los objetos que manejan el contexto en los servlets, como son HttpServlet, HttpServletRequest y HttpServletResponse. Permitiéndonos acceder más fácil a la definición y recuperación de variables de contexto.

En una aplicación Web se han definidos varios ámbitos en los que se definen la visibilidad de los objetos y su duración. Hay cuatro tipos de ámbitos: page, request, session y application.

En una aplicación Web se han definidos varios ámbitos en los que se definen la visibilidad de los objetos y su duración.

Ámbito de la solicitud (Request)

Este ámbito se crea cada vez que un cliente emite una solicitud HTTP, en ese momento el servidor crea un objeto que implementa la interfaz *javax.servlet.http.HttpServletRequest*.

Este objeto contiene una colección de pares de atributos clave / valor que se puede utilizar para almacenar objetos, cuya vida coincidirá con la vida de la solicitud. La clave de cada par es una cadena, y el valor puede ser cualquier tipo de objeto.

Una vez que el servidor atiende una solicitud y la respuesta es devuelta al cliente, la solicitud y sus atributos ya no estarán disponibles y pasarán a ser eliminados por el recolector de basura de la máquina virtual de Java.

En el framework SOFP este es el ámbito recomendado para el intercambio simple de información entre el comando y la vista Esto hace que sea mucho más fácil de acceder a los datos JavaBeans sin tener que preocuparse mas tarde por hacer un mantenimiento de limpieza de los objetos. Los objetos almacenados en el ámbito *request* son visible sólo a los recursos que tienen acceso a esa petición. Una vez que la respuesta ha sido devuelta al cliente, la visibilidad desaparece y los objetos también. Los Objetos que se almacenan en el ámbito *request* no son visibles para otra solicitud del cliente.

Ámbito de sesión (Session)

El siguiente nivel de visibilidad es el ámbito session. El contenedor web crea un objeto que implementa la interfaz javax.servlet.http.HttpSession para identificar a un mismo usuario a través de múltiples solicitudes de páginas diferentes.

La sesión del usuario se mantendrá por un período de tiempo relacionado con la frecuencia con que el usuario hace peticiones. El periodo de inactividad es configurable a través del descriptor de aplicación.

La sesión también permite definir una colección de objetos que se almacenan en base a un esquema de pares clave / valor como en el ámbito request. La única diferencia entre éste y el proporcionado por la solicitud es la duración de los objetos. Los objetos de sesión existen a través de múltiples de solicitudes del cliente. Los objetos almacenados en una sesión de usuario no son visibles para los usuarios con una sesión diferente.

Ámbito de aplicación (Application)

Los objetos almacenados en el ámbito de aplicación tienen el mayor nivel de visibilidad y duración. Estos objetos son visibles a todos los clientes y existen hasta que sean 'eliminados explícitamente' o hasta que la aplicación termina.

El servlet crea un contenedor de objetos, con interfaz javax.servlet.ServletContext para cada aplicación Web que está instalado en el contenedor.

Podemos almacenar JavaBeans visibles para todos los usuarios. Normalmente, los objetos son almacenados en este ámbito durante la aplicación de inicio y permanecer allí hasta que la aplicación termine.

Ámbito de página JSP

El último ámbito, el ámbito página, tiene que ver exclusivamente con páginas JSP. Los objetos con ámbito página se almacenan en la instancia de javax.servlet.jsp.PageContext asociada a cada página y son accesibles sólo dentro de la página JSP en el que fueron creados. Una vez que la respuesta se envía al cliente o la página remite a otros recursos, los objetos ya no estarán disponibles.

Recuperación de los JavaBeans desde las vistas JSP

La recuperación desde las vistas JSP de los objetos creados por el servlet se lleva a cabo mediante el uso de las etiquetas <jsp:usebean...> y <jsp:getProperty...> como se muestra en el fragmento JSP correspondiente a la Tabla 10.6.

```
<!--  declaramos el bean que se creará en el Servlet en la ejecución del comando -->
<jsp:useBean id="datosFomulario" type="test.org.sofp.Modelo.FormBeanEjemplo"
scope="request" />

<html>
<head>
<title>Test</title>
</head>
<body>

<!--  recuperamos los atributos del Bean devuelto por el servlet -->
Los datos pasados desde el command a la vista son<br>
nombre:  <jsp:getProperty name="datosFomulario" property="nombre" /></br>
atributo:<jsp:getProperty name="datosFomulario" property="atributo" /><br>

</body>
</html>
```

Tabla 10.6: Ejemplo de vista JSP con acceso a datos de un Bean

10.8 Mejoras

Algunas mejoras que permitirían mejorar la funcionalidad del framework SOFP y que están presente en algunos de los frameworks estudiados son las siguientes:

Validación en el lado del cliente mediante librerias Javascript. La validación en el lado del cliente consigue aplicaciones más ágiles. En muchos casos las validaciones son simples y se refieren únicamente al formato de los datos. Si se pueden detectar datos mal formados antes de realizar el *submit* al servidor, la aplicación ganará en fluidez.

Esta característica podría ser implementada incorporando una definición del formato de los datos y generando el código Javascript apropiado.

Nuevas librerias de tags JSP. Aunque mediante el uso de los *JavaBeans* es en las páginas JSP es posible acceder a los datos de un *Bean* sin tener que escribir código Java, en muchas ocasiones no es posible evitar la existencia de código. Para ello disponemos de la posibilidad de crear nuevas librerías de etiquetas Javascript. Estas librerías nos permiten definir nuevas acciones propietarias que nos permitan evitar mucho código repetitivo dentro de nuestra capa de presentación JSP. Como conclusión, una librería de TAG especialmente diseñada para nuestro *framework* nos permitiría aumentar su versatilidad por medio de la reutilización de código.

Seguridad. El uso del patrón *Application Controller* nos proporciona la posibilidad de abstraernos del mecanismo de petición-manejo propio de la tecnología Web. Esta mejora de la modularidad nos permite incorporar fácilmente a nuestros comandos mecanismos como el manejo de errores, la autenticación y el control de acceso. Una función interesante sería precisamente la de la implementación del control de acceso a nivel de comandos. Pudiendo definirse de forma declarativa, encargándose el propio *Application Controller* de realizar las redirecciones y cambios flujos necesarios en la ejecución de determinados comandos o accesos a vistas para solicitar si fuera necesario algún tipo de credencial al usuario.

Formularios dinámicos. Añadir la posibilidad de crear formularios dinámicos de forma declarativa. De esta forma no sería necesario crear una clase *FormBean* por cada formulario asociado a cada *command*.

Javascript. Aunque podemos incorporar Javascript en las vistas JSP. No proporcionamos ninguna facilidad para ellos. Sería interesante contar con alguna biblioteca de código Javascript que se integrara con el *framework*.

Módulo AJAX. Otra funcionalidad interesante sería proporcionar etiquetas que por medio de la tecnología conocida como AJAX nos permitan realizar peticiones asíncronas al servidor de aplicaciones. Existen muchas librerías de código abierto que podrían integrarse con nuestro *framework* mediante librerías de etiquetas.

Anexo A. Instrucciones de compilación y despliegue

A.1 Herramientas utilizadas

Para la compilación y despliegue de la librería y de la aplicación de ejemplo se han preparados sendos script para la herramienta Ant. Ant es una herramienta Open-Source utilizada para la automatización de las tareas de compilación y empaquetado de aplicaciones Java, Ant es equivalente a la herramienta make utilizada en el lenguaje C. La herramienta puede descargarse de la url http://ant.apache.org

A.2 Librería

A continuación incluimos las instrucciones para construir la librería del framework, el producto final será un fichero JAR

A.2.1 Generación de la librería

La estructura de directorios del proyecto es la siguiente

```
frameworkSOFP
├── build
├── build.xml
├── dist
│   └── lib
│       └── frameworkSOFP-1.0.jar
├── javadoc
├── src
│   └── org
│       └── sofp
│           ├── ApplicationController.java
│           ├── FrontController.java
│           ├── ambito
│           │   └── Contexto.java
│           ├── comandos
```

```
|           |   └── Command.java
|           ├── config
|           |   ├── ConfigComando.java
|           ├── forms
|           |   └── FormBean.java
|           └── taglib
|               └── MensajesTag.java
└── WebContent
    └── WEB-INF
        ├── lib
        |   ├── commons-beanutils-1.8.3.jar
        |   ├── commons-digester3-3.1.jar
        |   ├── commons-logging-1.1.1.jar
        |   └── jstl-1.2.jar
        └── web.xml
|           |   ├── ConfigFormulario.java
|           |   ├── ConfigLocale.java
|           |   ├── ConfigRecursoMensajes.java
|           |   ├── ConfiguracionGlobal.java
|           |   └── ConfigVista.java
|           ├── Constantes.java
|           ├── excepciones
|           |   ├── CampoNoEncontradoExcepcion.java
|           |   ├── CommandIncompletoException.java
|           |   ├── CommandNoEncontradoException.java
|           |   ├── CommandNoValidadoException.java
|           |   ├── FormularioNoEncontradoExcepcion.java
|           |   └── VistaNoEncontradaException.java
|           ├── forms
|           |   └── FormBean.java
|           └── taglib
|               └── MensajesTag.java
└── WebContent
    └── WEB-INF
        ├── lib
        |   ├── commons-beanutils-1.8.3.jar
        |   ├── commons-digester3-3.1.jar
        |   ├── commons-logging-1.1.1.jar
        |   └── jstl-1.2.jar
        └── web.xml
```

En el directorio raíz se encuentra el fichero build.xml para la herramienta ant, por lo que situados en una sesión de comandos en el directorio donde se encuentra el fichero build.xml, simplemente introduciendo desde el prompt del sistema la orden ant se iniciará la compilación de la librería y el empaquetado.

En el directorio WebContent/WEB-INF/lib se encuentra las librerías externas que se han utilizado:

- **Apache Common.** Librerías que pertenecen al parser Digester del proyecto apache. Las librerías se pueden descargar en su versión mas reciente de las siguientes url:

  http://commons.apache.org/digester/download_digester.cgi
  http://mvnrepository.com/artifact/commons-beanutils/commons-beanutils/1.8.3
  http://mvnrepository.com/artifact/org.apache.commons/commons-digester3/3.2
  http://mvnrepository.com/artifact/commons-logging/commons-logging/1.1.1

- **JSTL:** Extensión de la tecnología JSP que nos permiten haciendo uso de etiquetas incorporar funcionalidades ya desarrolladas en nuestras páginas.

http://mvnrepository.com/artifact/jstl/jstl/1.2

Aunque no es necesario distribuir las librerías, al tratarse de ficheros de poco tamaño y para simplificar el proceso de generación de los empaquetados se han mantenido en el directorio lib por comodidad para la construcción de la librería, ya que se trata de ficheros pequeños, pero sería mas correcto no distribuir estos ficheros junto con la librería.

Para terminar, es importante comentar que el fichero build.xml que se ha proporcionado contiene dos variables que apuntan a rutas donde se encuentran las librerías necesarias para la compilación del producto. A las citadas anteriormente hay que añadir las librerías de servlets propias que se distribuyen con el contenedor de servlets, Tomcat en nuestro caso. El fragmento al que nos referimos del fichero build.xml es el siguiente:

```
<property name="web.dir" location="WebContent/WEB-INF/lib/"/>
<property name="appserver.lib" location="/usr/share/tomcat6/lib/"/>
```

La variable web.dir apunta al directorio interno del proyecto que ya hemos comentado y si se mantienen en esa hubicación las librerias externas que distribuimos, no es necesario modificalo. Si se quieren ubicar las librerías en otra ubicación o ya existe algún tipo de repositorio local, se podría modificar para apuntar a un directorio diferente.

La variable appserver.lib apunta a las librerías de Tomcat, el contenedor de servlets. En el sistema que se ha utilizado para desarrollar este proyecto, un Linux Ubuntu 10.04LTS con Tomcat 6, la ruta es la que se indicada, pero puede ser que en otros sistemas la ruta sea diferente. En sistemas con Microsoft Windows la sintaxis es particular, ya que utiliza también la barra slash para separar directorios. En un sistema con Windows la ruta sería similar a la siguiente:

```
"C:/Archivos de programa/Apache Software Foundation/Tomcat 6.0/lib"
```

En caso de que se trate de una distribución linux o un equipo con Microsoft Windows, la ruta tendrá que ser editada y reemplazada por la correcta.

El fichero de configuración de Ant tiene varios targets diferentes, es posible eliminar una compilación mediante ant clean, para posteriormente volver a compilar con el comando ant sin argumentos.

Una vez terminada la compilación y el empaquetado, el fichero jar lo podremos localizar en:

```
dist/lib/frameworkSOFP-1.0.jar
```

Este es el fichero que tendremos que distribuir a los usuarios que deseen desarrollar con el framework SOFP.

Destacar que la documentación Javadoc también se genera durante el procesado con ant con cada compilado, la documentación final se puede encontrar en la carpeta Javadoc.

A.3 Aplicación de ejemplo

A continuación vamos a describir las instrucciones de despliegue de la aplicación de ejemplo, así como su funcionamiento.

A.3.1 Generación del fichero de despliegue

Para la generación del fichero de despliegue WAR se ha preparado un fichero de configuración ant que realiza el proceso de forma automática.

El procedimiento es similar, situados en el directorio raíz, donde se ubica el fichero build.xml, lanzar desde la línea de comandos la herramienta ant.

La estructura de directorios es la siguiente:

```
frameworkSOFP-Ejemplo/
├── build
│   └── classes
│       ├── basededatos.properties
│       ├── digesterReglas.xml
│       ├── org
│       │   └── sofp
│       │       └── test
│       │           ├── basededatos
│       │           ├── comandos
│       │           └── formbeans
│       └── sofp-config.xml
├── build.xml
├── deploy
│   └── frameworkSOFP-DemoPEC3.war
├── src
│   ├── basededatos.properties
│   ├── digesterReglas.xml
│   ├── go.java
│   ├── org
│   │   └── sofp
│   │       └── test
│   │           ├── basededatos
│   │           │   ├── Basededatos.java
│   │           │   ├── Comentario.java
│   │           │   ├── Constantes.java
│   │           │   └── Noticia.java
│   │           ├── comandos
│   │           │   ├── BienvenidaCmd.java
│   │           │   ├── ComentarioAnadirCmd.java
│   │           │   ├── ComentarioBorrarCmd.java
│   │           │   ├── ComentariosMostrarCmd.java
│   │           │   ├── NoticiaAnadirCmd.java
│   │           │   ├── NoticiaBorrarCmd.java
│   │           │   └── NoticiasMostrarCmd.java
│   │           └── formbeans
```

```
│   │               ├── ComentarioFormbean.java
│   │               ├── IdComentarioFormbean.java
│   │               ├── IdNoticiaFormbean.java
│   │               ├── LoginFormbean.java
│   │               ├── NoticiaFormbean.java
│   │               └── VacioFormbean.java
│   └── sofp-config.xml
│
└── WebContent
    ├── Bienvenida.jsp
    ├── Cabecera.jsp
    ├── ComentarioAnadidoOk.jsp
    ├── ComentarioBorrarError.jsp
    ├── ComentarioBorrarOk.jsp
    ├── ComentariosMostrar.jsp
    ├── hojadeestilos.css
    ├── index.jsp
    ├── mensajesTag.tld
    ├── META-INF
    │   └── MANIFEST.MF
    ├── NoticiasAnadirOk.jsp
    ├── NoticiasBorrarError.jsp
    ├── NoticiasBorrarOk.jsp
    ├── NoticiasMostrar.jsp
    ├── sofpError.jsp
    └── WEB-INF
        ├── lib
        │   ├── commons-beanutils-1.8.3.jar
        │   ├── commons-digester3-3.1.jar
        │   ├── commons-logging-1.1.1.jar
        │   ├── frameworkSOFP-1.0.jar
        │   └── jstl-1.2.jar
        └── web.xml
```

Nuevamente, se ha vuelto a duplicar las librerías en la carpeta WEB-INF/lib, esta vez incluyendo junto a las Common Lib de Apache y a JSTL la librería del propio framework.

El fichero *web.xml*, ya está configurado correctamente y se encuentra ubicado en el lugar correcto. Recordemos que las aplicaciones que utilicen el framework, deben incluirs un fragmento de código que realiza el mapeo del patrón definido para las url y que lanzará la ejecución de nuestro servlet. La ubicación del fichero es la siguiente:

```
WebContent /WEB-INF /web.xml
```

Otro fichero importante es el fichero de definición de reglas (digesterReglas.xml) para el parseo de Digester. Este fichero debe acompañarse en la distribución del framework y es siempre el mismo. No ocurre lo mismo con el fichero de configuración (sofp-config.xml), que será propio de cada aplicación, recordemos que es en este fichero donde se definen declarativamente los comandos, vistas, formularios y todos los elementos del framework SOFP. La ubicación de los ficheros es la siguiente:

```
frameworkSOFP-Ejemplo/
├── build
    └── classes
        ├── basededatos.properties
        └──digesterReglas.xml
```

A.3.2 Despliegue de la aplicación de ejemplo

Una vez ejecutada la herramienta Ant, esta generará el fichero de despliegue WAR definitivo en la ruta

```
├── deploy
│   └── frameworkSOFP-DemoPEC3.war
```

Para desplegarlo en Tomcat copiaremos el fichero war en la carpetade despliegue de Tomcat. Si se trata como ya he comentado de un equipo con Linux basado en Debian, la ruta de despliegue de Tomcat 6 es la siguiente:

```
/var/lib/tomcat6/webapps
```

En una máquina con Microsoft Windows, la ruta sería similar a esta

```
c:\Archivos de programa\apache-tomcat-6.0.26\webapps
```

Una vez copiado el archivo WAR al Webapps, el servidor lo detectará automáticamente, lo descomprimirá y realizará su despliegue. Una vez concluido el despliegue, podremos probar el funcionamiento de la aplicación de ejemplo desde la url:

```
http://localhost:8080/frameworkSOFP-DemoPEC3/
```

A.3.3 Funcionamiento de la aplicación

El funcionamiento y la lógica de la aplicación es muy sencillo. La aplicación simula un agregador social de noticias y contenidos, donde los usuarios pueden publicar noticias que consideren interesantes. Por cada noticia publicada se incluye un titular, el enlace a la noticia y un pequeño resumen.

Posteriormente los usuarios podrán agregar comentarios a cada noticia, dando su opinión y debatiendo sobre ella.

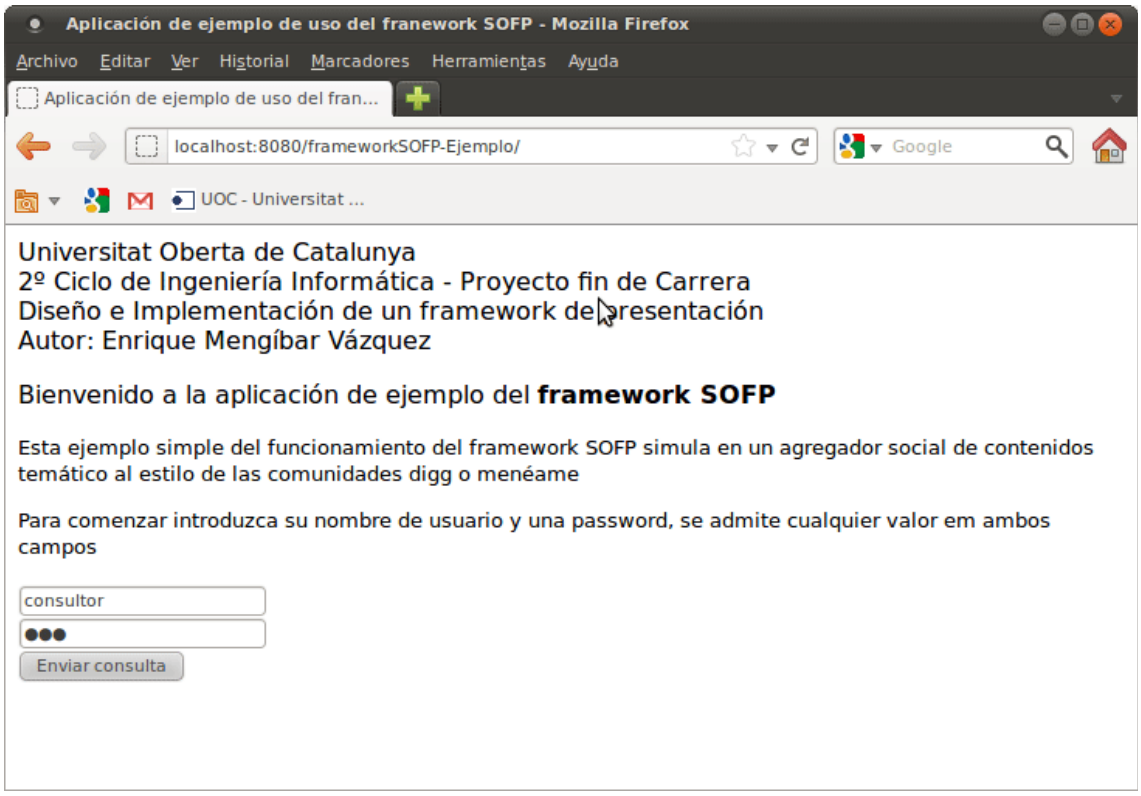
La aplicación de ejemplo es muy simple y no utiliza ningún tipo de base de datos, pero simula una mediante colecciones de objetos. Existe una clase principal de nombre basededatos que contiene una colección de noticias y cada noticia a su vez, una colección de comentarios. Para mayor comodidad, la clase basededatos se inicializa leyendo un fichero de propiedades de nombre basededatos.properties que contiene algunos registros de prueba para poder trabajar mas cómodamente.

Para probar las características del framework se han introducido las siguientes características en la aplicación.

- No es necesario registrarse, cualquier nombre de usuario y password se acepta.

- Un usuario no puede borrar una noticia si no ha sido creada por él.
- Un usuario no puede borrar un comentario si no ha sido creado por él.
- Se realiza una validación de los campos a la hora de dar de alta una noticia, no se permiten indicar campos vacíos y las dirección de correo electrónico debe estar bien formada.
- Si se borra una noticia se borran todos sus comentarios.
- Se mantiene una variable de sesión con el nombre de usuario que se utiliza para el alta de noticias, el dato no se vuelve a pedir al usuario.
- El nombre de usuario almacenado en la citada variable de sesión se muestra en la cabecera de cada página.
- Cada vez que se realiza una operación en el comando, se define una variable de ámbito request para retornar un valor a la vista e ilustrar el funcionamiento del intercambio de datos comando-vista.

A continuación mostramos algunas capturas de pantalla de la aplicación de ejemplo


Aplicación de ejemplo de uso del franework SOFP - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
Aplicación de ejemplo de uso del fran...
localhost:8080/frameworkSOFP-Ejemplo/
Google
UOC - Universitat ...
Universitat Oberta de Catalunya
2º Ciclo de Ingeniería Informática - Proyecto fin de Carrera
Diseño e Implementación de un framework de presentación
Autor: Enrique Mengíbar Vázquez
Bienvenido a la aplicación de ejemplo del framework SOFP
Esta ejemplo simple del funcionamiento del framework SOFP simula en un agregador social de contenidos temático al estilo de las comunidades digg o menéame
Para comenzar introduzca su nombre de usuario y una password, se admite cualquier valor em ambos campos
consultor
Enviar consulta

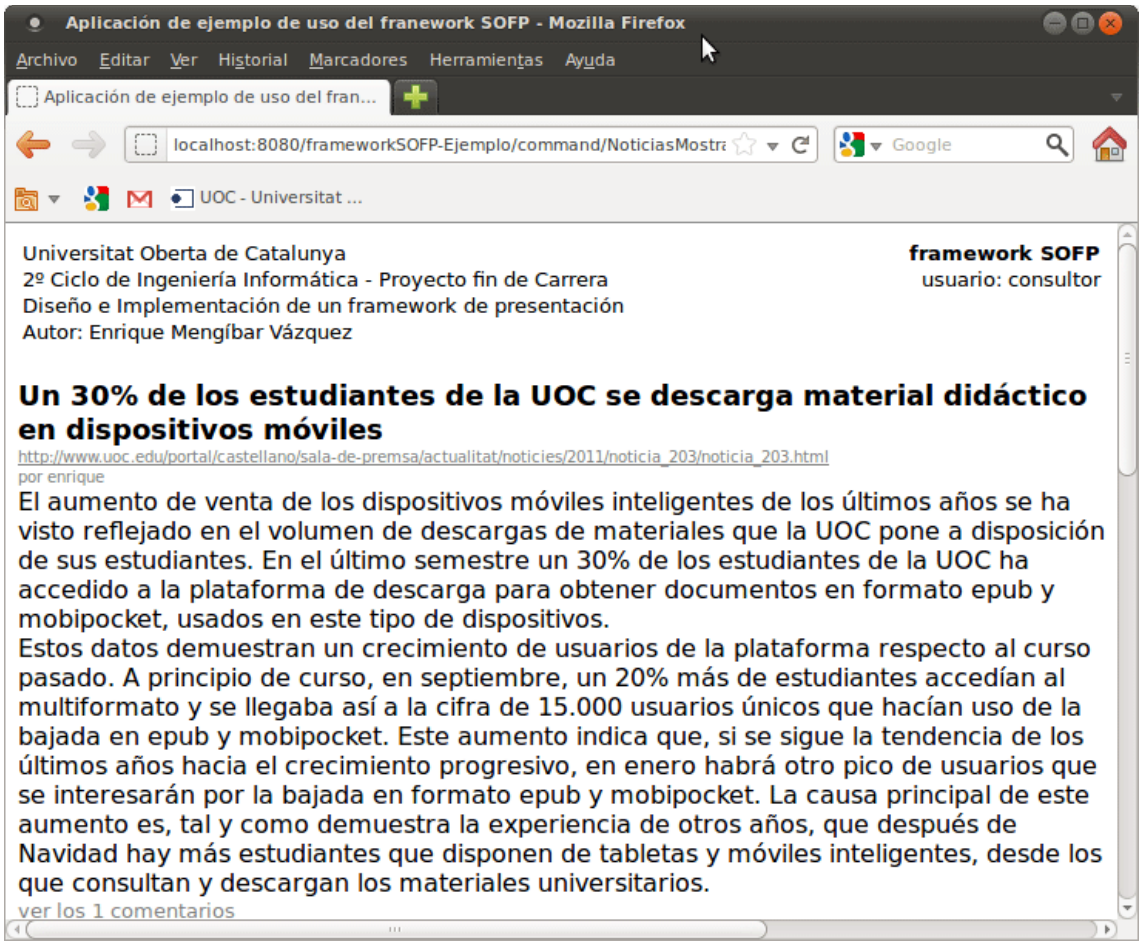
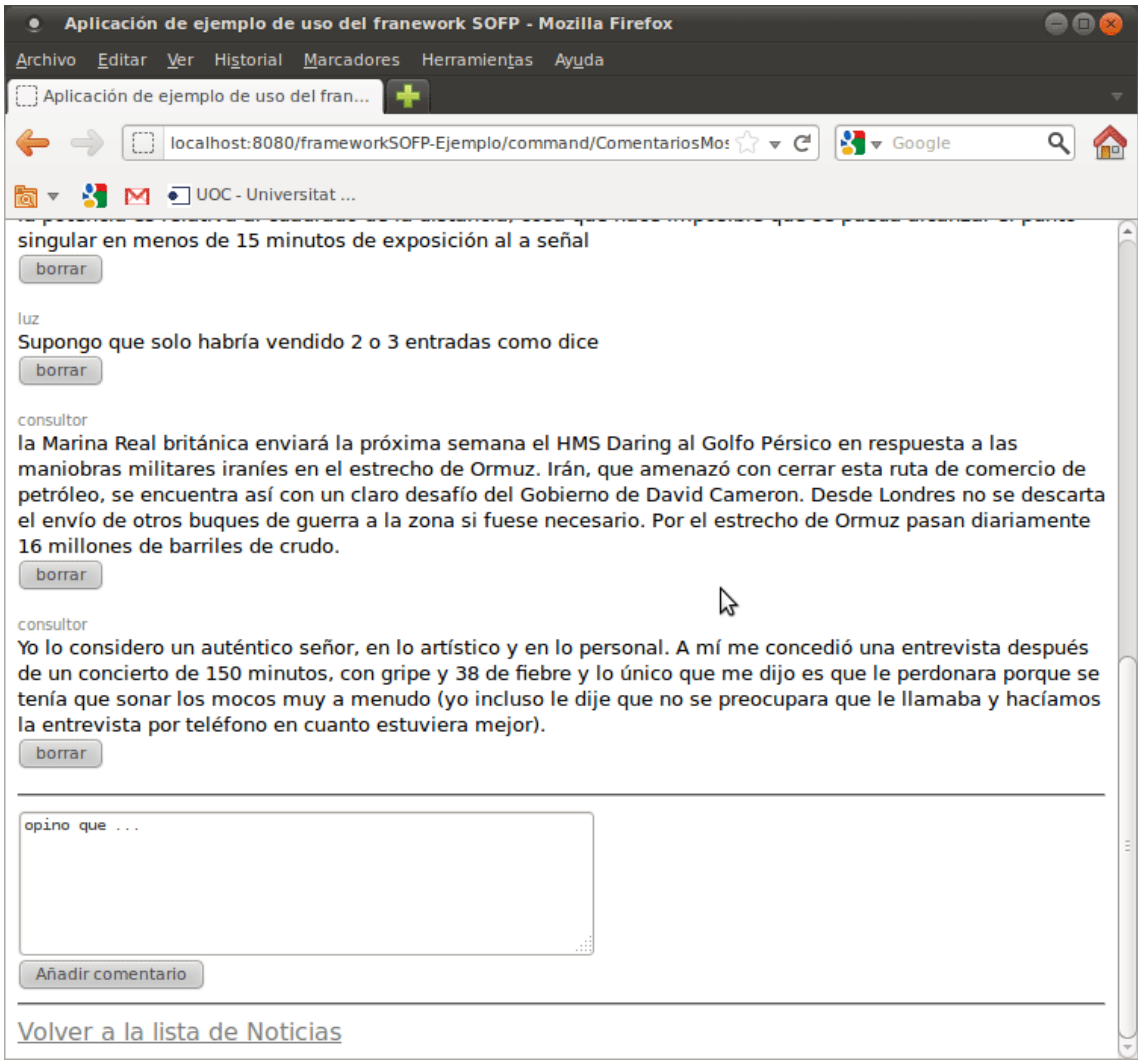


Figura 10.10: Vista de Noticias, muestra al usuario las noticias publicadas, junto con un enlace, resumen, titular y autor


Aplicación de ejemplo de uso del franework SOFP - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
Aplicación de ejemplo de uso del fran...
localhost:8080/frameworkSOFP-Ejemplo/command/ComentariosMos
Google
UOC - Universitat ...
singular en menos de 15 minutos de exposición al a señal
borrar
luz
Supongo que solo habría vendido 2 o 3 entradas como dice
borrar
consultor
la Marina Real británica enviará la próxima semana el HMS Daring al Golfo Pérsico en respuesta a las maniobras militares iraníes en el estrecho de Ormuz. Irán, que amenazó con cerrar esta ruta de comercio de petróleo, se encuentra así con un claro desafío del Gobierno de David Cameron. Desde Londres no se descarta el envío de otros buques de guerra a la zona si fuese necesario. Por el estrecho de Ormuz pasan diariamente 16 millones de barriles de crudo.
borrar
consultor
Yo lo considero un auténtico señor, en lo artístico y en lo personal. A mí me concedió una entrevista después de un concierto de 150 minutos, con gripe y 38 de fiebre y lo único que me dijo es que le perdonara porque se tenía que sonar los mocos muy a menudo (yo incluso le dije que no se preocupara que le llamaba y hacíamos la entrevista por teléfono en cuanto estuviera mejor).
borrar
opino que ...
Añadir comentario
Volver a la lista de Noticias

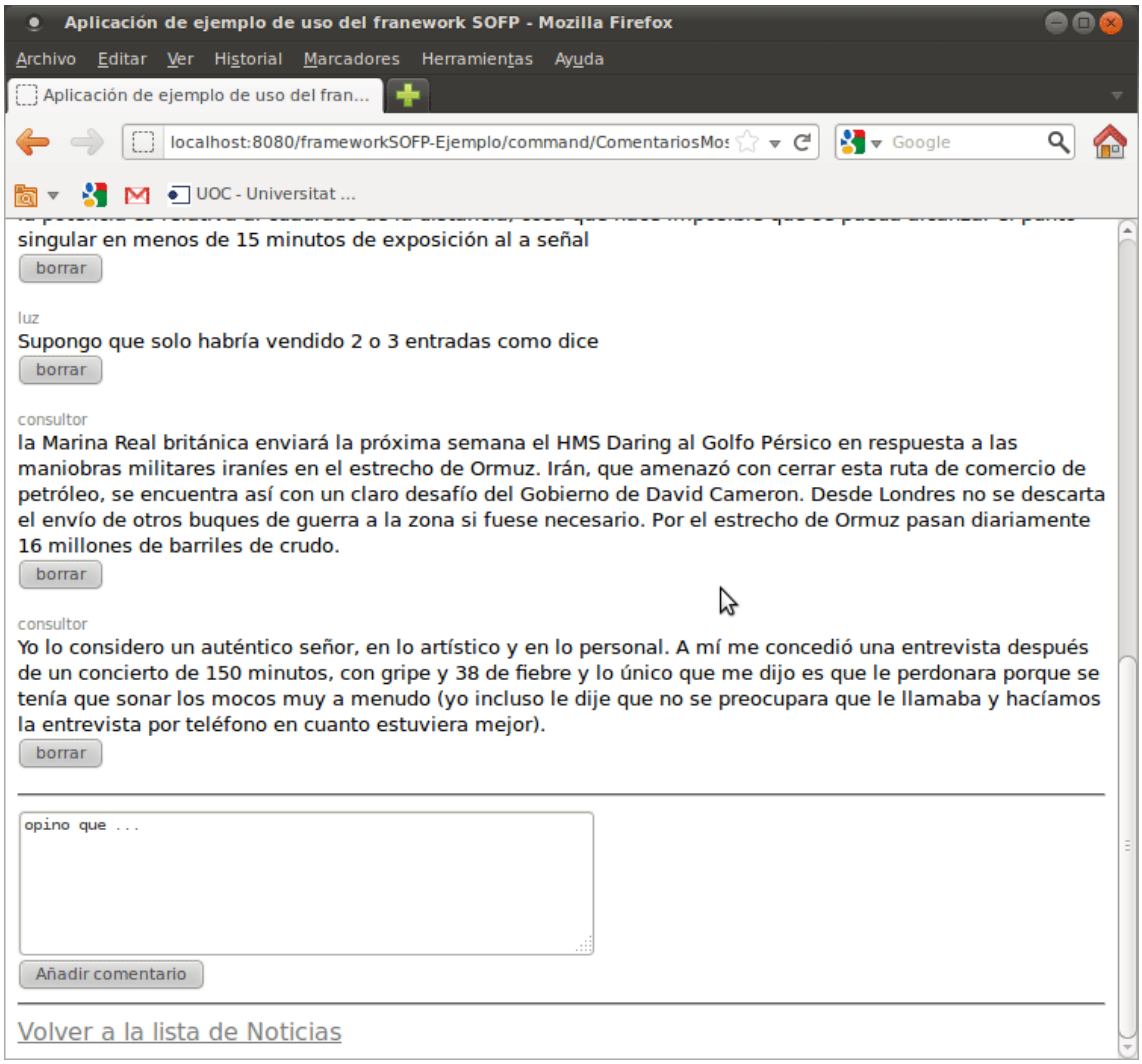


Figura 10.11: Vista que lista los comentarios asociados a una noticia junto con el formulario de alta

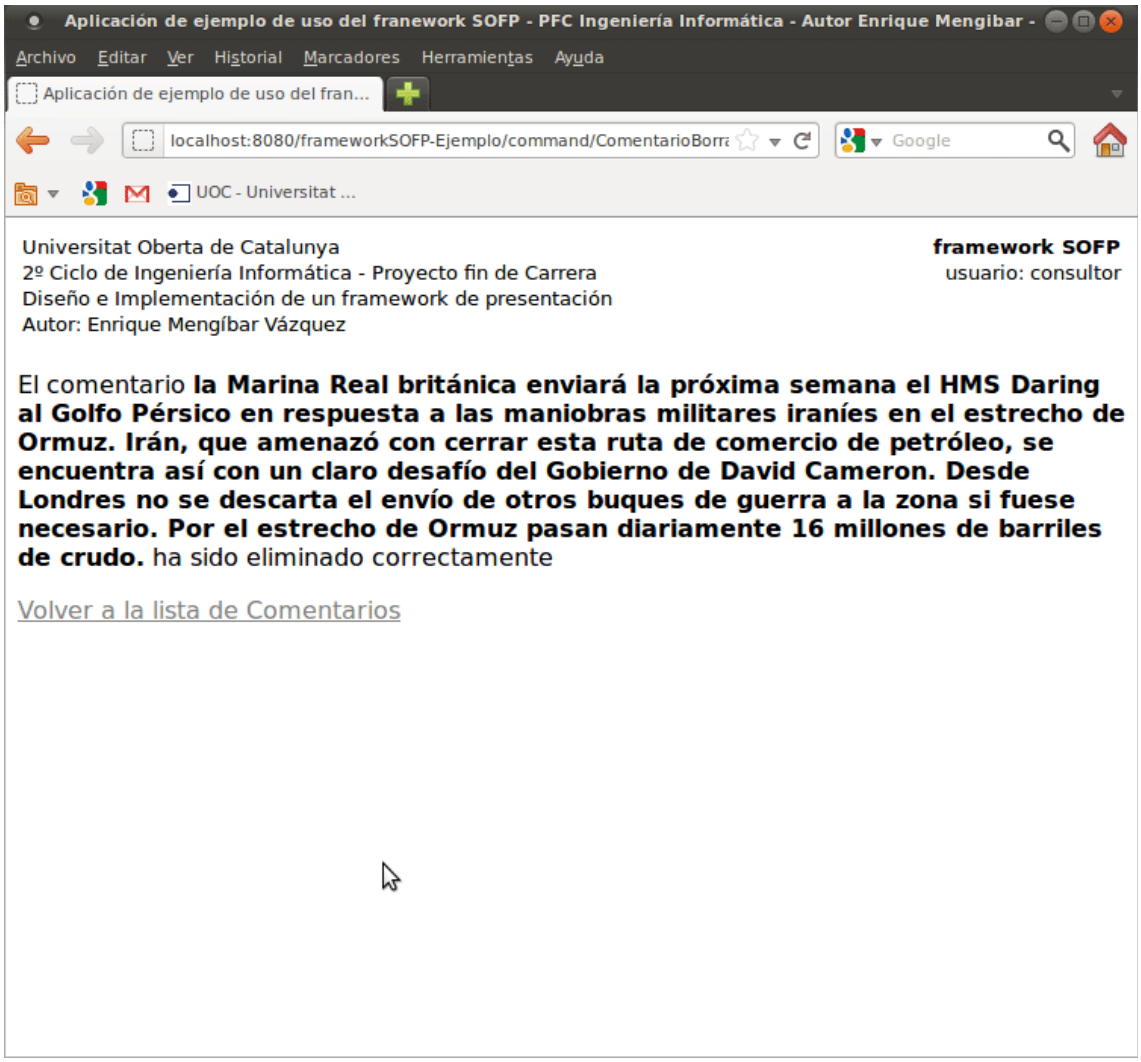


Figura 10.12: Vista informativa de que un comentario ha podido ser eliminado

Figura 10.13: Vista informativa de error que avisa que no se pudo eliminar un comentario por no pertenecer al usuario

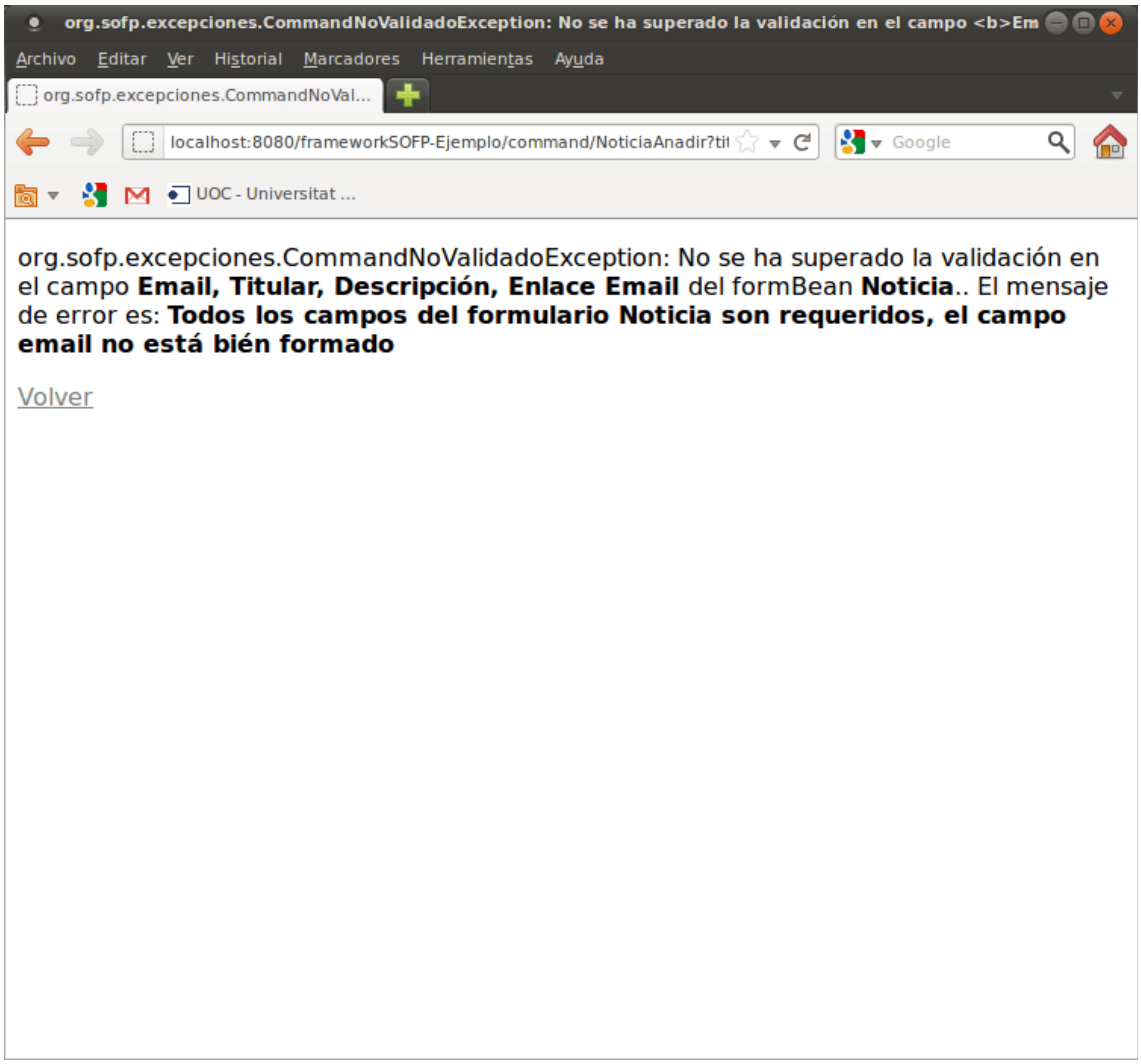


Figura 10.14: Vista de error que muestra un aviso al usuario indicando que los campos de la noticia no han superado la validación

Anexo B. *Glosario*

Acoplamiento: medida del grado en el que un objeto o componente depende de otro.

Ajax (Asynchronous JavaScript And XML): técnica de desarrollo web para crear aplicaciones interactivas que se ejecutan en el navegador manteniendo una comunicación asíncrona con el servidor en segundo plano. (Se realizan cambios en las páginas web sin recargarlas).

AOP (Aspect-Oriented Programming): paradigma de programación orientada a aspectos que pretende conseguir una modularización apropiada y una mejor separación de conceptos, que disminuya o elimine las dependencias inherentes entre diferentes módulos.

Apache Commons: proyecto de la Apache Software Foundation. Su propósito es proporcionar software Java de fuente abierta reutilizable.

Apache Software Foundation (Fundación de software Apache): Fundación sin ánimo de lucro creada para apoyar los proyectos Apache, incluyendo el popular servidor HTTP Apache. La Apache Software Foundation es una comunidad descentralizada de programadores que trabajan en proyectos de código abierto. Los proyectos Apache se caracterizan por ser software libre.

API (Application Programming Interface): conjunto de especificaciones de comunicación entre componentes de software que proporciona abstracción entre los niveles inferiores y superiores de una aplicación.

Applet: tipo especial de aplicación Java que se puede ejecutar directamente en un navegador Web. A un applet se le imponen diversas restricciones de seguridad. Por ejemplo, un applet no se puede ejecutar operaciones de entrada/salida en un sistema de usuario.

Apache Ant: herramienta usada en las tareas de desarrollo para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (build). Es, por tanto, un software para procesos de automatización de compilación, similar a Make

pero desarrollado en lenguaje Java, así que es más apropiado para la construcción de proyectos Java. No depender de las órdenes del shell de cada sistema operativo, sino que se basa en archivos de configuración XML y clases Java para la realización de las distintas tareas, siendo idónea como solución multi-plataforma.

Bean: componente de software reutilizable. Disponen de una interfaz pública para instanciar los mismos y unos métodos para acceder a los atributos e insertar valores en los atributos.

Capa de datos: en el modelo J2EE es la encargada de almacenar de forma persistente los datos de la aplicación al SGBD sobreviviendo a la ejecución de la aplicación.

Capa de negocio: en el modelo J2EE es la capa más importante pues encapsula toda la lógica de negocio de la aplicación. Una de las funciones más importantes es la de separar la presentación del acceso a datos y por tanto no estar afectada por estas capas.

Capa de persistencia: ubicación dónde se guardan los datos persistentes, es decir que sobreviven la ejecución del programa o las sesiones de usuario. El formato de almacenamiento no volátil puede ser un sistema de BD relacional.

Capa de presentación: también llamada capa web en el modelo J2EE. Su principal función es separar la interfaz de la lógica de negocio y facilitar el acceso de las peticiones de los diferentes clientes en la lógica de negocio.

Capa EIS (Enterprise Information System): integra la aplicación J2EE con otros sistemas de información. Provee datos almacenados o bien servicios de información de otros sistemas.

Core J2EE Patterns: Catálogo completo de patrones de diseño y arquitectónicos específicos para los problemas comunes en J2EE. También se identifican malas prácticas que deben evitarse. Está organizado en patrones de la capa de presentación, patrones en la capa de negocio y patrones de la capa de integración .

CSS (Cascading style sheets): lenguaje formal utilizado para definir el estilo de un documento html o xml.

DAO (Data Access Object): patrón de diseño utilizado para la capa de persistencia realizando la abstracción y encapsulación de los métodos de acceso a datos.

Descriptor de despliegue: es un componente en aplicaciones J2EE que describe como una aplicación web debe desplegarse. Dirige una herramienta de desarrollo de módulos o de toda la aplicación con opciones para un contenedor específico y describe requerimientos concretos de configuración que el desplegador ha de resolver. En J2EE es el fichero web.xml que debe almacenarse en el directorio WEB-INF bajo la raíz de la aplicación.

Design Pattern (patrón de diseño): plantilla que se utiliza como diseño para una solución de un determinado problema durante el desarrollo.

Document Object Model – DOM : interfaz independiente de la plataforma y del lenguaje que permite a los programas y guiones acceder dinámicamente y actualizar el contenido, estructura y estilo de los documentos. DOM proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar de como se combinan esos documentos y una interfaz estándar para su acceso y manipulación.

EJB (Enterprise Java Bean): API para J2EE implementada por Sun Microsystems que ofrece un modelo de componentes del lado del servidor para desarrollar la lógica de negocio en aplicaciones distribuidas solventando la problemática de gestión de la concurrencia, transacciones, seguridad.

Esquemas XML: componentes del mundo XML muy extendidos que permiten definir cuando un documento XML es válido.

Espacio de nombre. colección de nombres, identificados por una referencia para identificar unívocamente términos y permitir que dichos términos se usen sin ambigüedades entre aplicaciones, propiciando la posibilidad de semánticas compartidas.

Etiqueta: hace referencia a los elementos HTML y en muchos casos sólo al inicio y final del elemento.

Framework: un framework es una estructura conceptual usada para solucionar o conducir un problema complejo. En software, generalmente consiste en un diseño reutilizable o un subsistema que proporciona ayudas o implementaciones a ciertos casos de uso comunes en los programas.

Hibernate: es una solución implementada por mapeo de objetos relacionales (ORM) para aplicaciones Java, sobre una base de datos relacional. Sus propósitos básicos son los de liberar el programador de una serie de tareas propias de la persistencia de datos relacionales y dotar a las aplicaciones de portabilidad entre SGBD diferentes.

HTML (Hyper Text Markup Language): lenguaje de marcas utilizado para el desarrollo de páginas web.

iBatis (MyBatis): Framework de código abierto y desarrollado por Apache Software Foundation para la capa de persistencia.

Internacionalización: Técnicas y convenios para un diseño Web sin barreras para los diferentes idiomas, sistemas de escritura, códigos de caracteres y otras convenciones locales que existen. El carácter global de la Web requiere un sistema en el que exista facilidad a la hora

de crear y procesar información para una audiencia variada, permitiendo así publicar material e intercambiar datos en cualquier idioma. La Internacionalización también es conocida como i18n, que es la abreviatura de Internacionalización, ya que entre la primera «i» y la última «n» existen 18 letras (tanto es inglés como en español).

Inyección de Dependencias: Patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto.

IOC (Inversion of Control): principio de la programación orientada a objetos por el que se reduce el acoplamiento inherente en los programas informáticos. El control se pasa de la aplicación al framework.

JavaBeans: son un modelo de componentes creado por Sun Microsystems para el desarrollo de aplicaciones en Java. La especificación de JavaBeans de Sun Microsystems los define como componentes de software reutilizables que se puedan manipular visualmente en una herramienta de desarrollo.

Java: lenguaje de programación orientado a objetos, basado en C++, cada día más extendido, especialmente a través de Internet. Pretende ser un lenguaje totalmente portable entre distintos sistemas operativos y dispositivos, gracias a que no se compila a código máquina,sino a un lenguaje intermedio que luego es interpretado por la máquina virtual Java, que sí es específica de cada plataforma. Es uno de los lenguajes de programación más utilizados, y se utiliza tanto para aplicaciones web como para aplicaciones de escritorio.

JavaServer Faces (JSF): framework de aplicación web basado en lenguaje Java. Simplifica el desarrollo de interfaces de usuario para aplicaciones J2EE. En su tecnología para mostrar la web utiliza las páginas JSP.

JBoss: servidor de aplicaciones J2EE de código abierto implementado en Java puro. Al estar basado en Java, JBoss puede ser utilizado en cualquier sistema operativo para el que esté disponible Java

JEE: Java Platform, Enterprise Edition o Java EE (conocido como Java 2 Platform Enterprise Edition o J2EE hasta la versión 1.4), es una plataforma de programación para desarrollar y ejecutar software escrito con el lenguaje Java con una arquitectura distribuida con niveles, basada en componentes de software, todo ello ejecutándose en un servidor de aplicaciones.

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

JSP (Java Server Pages): documentos de texto que se ejecutan como servlets pero que permiten una aproximación más natural a la creación de contenido estático. También llamadas vistas pues las JSP implementan las vistas del modelo MVC en la capa de presentación.

JSTL: extensión de la tecnología JSP que nos permiten haciendo uso de etiquetas incorporar funcionalidades ya desarrolladas en nuestras páginas.

Locale: atributo del sistema operativo que define ciertos comportamientos relacionados con el idioma. La configuración regional define la página de códigos, o combinaciones de bits, que se utiliza para almacenar datos de caracteres y el orden en el que éstos se clasifican. También define elementos específicos del idioma, como el formato utilizado para fechas y horas y el carácter utilizado para separar los decimales en los números.

Lógica de negocio: rutinas que realizan entradas de datos, consultas a los datos, generación de informes y más específicamente todo el procesamiento que se realiza detrás de la aplicación visible para el usuario.

MVC: acrónimo para Modelo-Vista-Controlador, patrón de arquitectura en ingeniería de software. En una aplicación compleja que presenta una gran cantidad de datos al usuario, es deseable separar los datos (Modelo) y los problemas de la interfaz de usuario (Vista), de tal manera que los cambios en la interfaz no afecten el manejo de los datos, y que los datos puedan ser organizadas sin cambiar la interfaz de usuario. MVC soluciona este problema desacoplando el acceso a los datos y la lógica de negocio de la presentación de los datos e interacción del usuario, mediante un componente intermediario: el Controlador.

OOP (Object Oriented Programming): paradigma de programación que define los programas en términos de clases de objetos y en ellos encapsula estado (datos), comportamiento (métodos y procedimentos) e identidad.

Open Source: código abierto, con este término se conoce el software que se distribuido y desarrollado libremente por la comunidad.

RM (Object-Relational Mapping): técnica de programación para convertir datos del tipo utilizado en el lenguaje de programación orientado a objetos y los datos usados en una base de datos relacional. Se utiliza un framework para realizar estas tareas.

Patrón de diseño: solución general a un problema común y recurrente en el diseño de software. Un patrón de diseño es una descripción o plantilla para resolver un problema que se puede utilizar en muchas situaciones diferentes.

POJO (Plain Old Java Object): clase definida en Java que no es de ningún tipo especial (EJBs), ni implementa una interfaz específica.

Servlet: clase Java ejecutada por un servidor de aplicaciones y que responde a invocaciones HTTP, sirviendo páginas dinámicas. Un objeto Servlet es capaz de recibir una invocación y generar una respuesta en función de los datos de la invocación, del estado del propio sistema y los datos a que pueda acceder.

SOFP: el nombre con el que se ha denominado al framework fruto de este trabajo, corresponde a las siglas *Si Otro Framework de Paginación*.

Spring: framework de aplicación que soporta muchas tecnologías de generación de vistas como las JSP y se integra perfectamente con Hibernate. Spring es de código abierto para la plataforma Java, aplica los principios de Inversión de Control utilizando la técnica de Inyección de dependencias.

Struts: framework de aplicación web de código abierto para desarrollar aplicaciones J2EE. Utiliza y extiende Java Servlet API para permite a los desarrolladores adoptar el patrón MVC (modelo-vista-controlador). La principal ventaja de Struts es que separa el Modelo (la lógica de negocio que interactúa con la BD) de la Vista (páginas HTML mostradas al cliente) y el Controlador (Instancia que pasa información entre Vista y Modelo).

Swing: biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas.

Tag: etiqueta o marca de un lenguaje basado en XML como por ejemplo HTM. Las etiquetas permiten asociar atributos y propiedades que configuran el elemento. Por lo que respecta a las librerías de tags entendemos como un conjunto de etiquetas reutilizables y encapsuladas para ser utilizadas en diferentes aplicaciones en el desarrollo de las vistas de la capa de presentación.

Thin Client (Cliente ligero): equipo con unos requerimientos de hardware mínimos que permite conectar a internet y realizar peticiones web, puede ser un ordenador personal (PC), portátil o dispositivo de bajo rendimiento.

TLD (Tag Library Descriptor): descriptores de librerías de tags, archivos de configuración XML que definen las características de los tags o acciones utilizados en la librería como son sus atributos, el tipo de acciones.

Tomcat: Tomcat (Jakarta Tomcat): contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Sun Microsystems. Se le considera un servidor de aplicaciones.

UML: lenguaje unificado de modelado que proporciona notación

URL (Uniform Resource Locator): secuencia de caracteres usada siguiendo un formato estándar para nombrar recursos, documentos e imágenes en internet. Se utiliza para poder localizarlos.

VO (Value Object, también llamado DTO o Data Transfer Object): son simplemente beans contenedores de datos sin ninguna lógica. Se usan para enviar información entre diferentes capas de la aplicación.

Webwork: framework para el desarrollo de aplicaciones web en Java desarrollado por OpenSymphony. Actualmente se ha fusionado con el actual framework Strut2. Se creo con la intención específica de mejorar la productividad del desarrollo y la simplicidad de codificación

Xerces: colección de bibliotecas para el parseo, validación, serialización y manipulación de documentos XML de la Apache Software Foundation.

XML (eXtensible Markup Language): lenguaje de marcas extensible, es un metalenguaje extensible, de etiquetas, desarrollado por el World Wide Web Consortium (W3C). Permite la compatibilidad entre sistemas, permitiendo compartir información de una manera segura, fiable y fácil.

XSLT : lenguaje XML para la realización de transformaciones en documentos XML. Permite la transformación de la estructura de un documento XML en otro documento XML con estructura diferente.

hola

Bibliografía

Alur, Deepak; Crupi, John; Malks, Dan [2003], *Core J2EE Patterns: Best Practices and Design Strategies (2º edición),* Prentice Hall.

Apache Software Foundation, *Apache Struts.* [en línea] http://struts.apache.org [fecha de consulta: 05/10/2011].

Apache Software Foundation. *Struts2 Framework*. [en línea] http://struts.apache.org/2.x [fecha de consulta: 28/10/2011]

Basham, Bryan; Bates, Bert; Sierra, Kathy [2004], *Head First Servlets & JSP: Passing the Sun Certified Web Component Developer Exam. Sebastopol,* O'Reilly.

Booch, G; Jacobson, I; Rumbaugh, J [2000], *El lenguaje unificado de modelado. Manual de referencia,* Addison - Wesley.

Booch, Grady; Rumbaugh, James; Jacobson, Ivar [1999], *El lenguaje unificado de modelado,* Addison Wesley Iberoamericana.

Eckel, Bruce [2002], *Thinking in Java*. Prentice-Hall.

Fowler, Martin [2003], *Patterns of Enterprise Application Architecture,* Addison - Wesley.

Fowler, Martin; Scott, Kendall [2000], *UML Gota a Gota* , Prentice-Hall

Freeman,Eric;Bates, Bert; Sierra Kathy [2004], *Head First Design Patterns ,* O'Reilly

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides John [1995], Design Patterns: Elements of Reusable Object-Oriented Software, Addison - Wesley.

Hall, Marty, *Core Servlets and JavaServer Pages* [2003], Prentice Hall & Sun Microsystems

Johnson, Mark; Singh, Inderjeet; Stearns, Beth [2002], *Designing Enterprise Applications with the J2EE Platform,* Addison - Wesley.

Johnson, Rod [2002], Expert One-on-One J2EE Design and Development , Wrox

Keogh, Jim [2003], *J2EE. Manual de referencia*, McGraw-Hill.

Oracle. *JavaServer Faces Technology - Documentation*. [en línea]. http://www.oracle.com/technetwork/java/javaee/documentation/index-137726.html [fecha de consulta: 1/11/2011].

Spring Source Community. Spring Framework. [en línea]. http://static.springsource.org/ [fecha de consulta: 14/10/2011].

Wikipedia. *"Comparison of web Application Frameworks"*. Wikipedia, the free encyclopedia. [en linea], [fecha de consulta: 28/10/2011]. http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks