Diseño e implementación de un marco de trabajo (*framework*) de presentación para aplicaciones J2EE

Miguel Angel Montejano Maldonado
Ingeniero Técnico de Informática de Sistemas

Josep Maria Camps Riba

14/01/2008

Agradecimientos y dedicatoria

Mil gracias Heidy por tu comprensión y paciencia. Sin ti a mi lado no lo habría podido hacer, estoy seguro.

Resumen

El proyecto de fin de carrera propuesto consiste en el análisis, diseño e implementación de un conjunto de componentes que simplifican la construcción de la capa de presentación para aplicaciones web, sirviendo de base para el desarrollo de otros productos.

La primera parte del trabajo realizado consistió en el estudio y comparación de *frameworks* de presentación existentes. Una vez analizados, se definieron las características y el alcance esperado en el proyecto de fin de carrera.

Teniendo presente en todo momento el patrón de diseño Modelo-Vista-Controlador, el marco de trabajo propuesto desarrolla componentes que nos permiten:

- Disponer de un entorno de presentación muy potente y con elementos altamente reutilizables a partir de *tiles*.
- Incluir menús horizontales y verticales de forma sencilla utilizando ficheros XML.
- Disponer de un conjunto de páginas que nos ofrezcan una presentación homogénea de acuerdo a los distintos casos de uso que se pretendan abordar con éstas.

El *framework* está formado por una aplicación web, donde los componentes principales es una estructura de *tiles* que nos permiten extender unas páginas *jsp's* básicas que cubran los siguientes escenarios:

- Mantenimiento directo de datos de una entidad (altas, bajas, modificaciones y borrados).
- Consulta de datos de una entidad.
- Mantenimiento de una relación Maestro-Detalle entre dos entidades.

La comunicación entre las páginas *jsp's* y el modelo de negocio se realiza haciendo uso de *Struts*. La presentación de las posibles operaciones se basan en menús que pueden ser configurados utilizando ficheros XML y presentados utilizando XSLT.

Índice de contenido

1 Memoria del proyecto

1.1 Introducción

1.1.1 Justificación y contexto del *framework*

En la actualidad, el desarrollo de aplicaciones web eficientes, robustas, extensibles, portables y basadas en componentes reutilizables es dificil y se ha convertido en esencial comprender correctamente las técnicas que han solucionado de forma efectiva problemas comunes durante el desarrollo del software.

Los patrones de diseño y *frameworks* ponen a nuestra disposición una serie de técnicas ya contrastadas que nos facilitan la toma de decisiones de los diseños de los sistemas o nos proporcionan un conjunto de componentes para resolver de forma eficaz desarrollos requeridos por las aplicaciones.

Un patrón de diseño muy extendido en las aplicaciones Web es el conocido como Modelo-Vista-Controlador (MVC), el cual nos permite definir los sistemas a partir de tres capas con responsabilidades funcionales diferentes:

- Modelo: Datos y reglas de negocio
- Vista: La presentación
- Controlador. Control de flujo.

Cada una de éstas capas está muy poco acoplada dotándo al sistema de mucha flexibilidad y un bajo impacto ante los cambios de cualquiera de ellas.

Un *framework* de aplicación es una colección de componentes que colaboran entre sí para producir una arquitectura reutiliable en un conjunto de aplicaciones. El concepto de *framework* se diferencia de una librería de clases tradicional en que los *frameworks* conforman el esqueleto de un particular dominio de las aplicaciones (presentación, persistencia, seguridad, integraciones...) y son una aplicación "semi-completada".

En el mercado existen y están apareciendo diferentes *frameworks* para ayudar y simplificar la creación de la capa de presentación de las aplicaciones MVC.

El presente trabajo consiste en el análisis y desarrollo de un *framework* propio que agilice el desarrollo de la capa de presentación de una aplicación J2EE.

Durante la elaboración del trabajo, será necesario estudiar en profundidad las

técnologías y *frameworks* existentes, para determinar la micro-arquitectura del nuevo *framework*.

La idea principal del proyecto consiste en crear una serie de componentes basados en estándares J2EE que nos permitan disponer de una aplicación base que sirva como punto de partida para la construcción de aplicaciones más sofisticadas y complejas y nos resuelva una serie de tratamientos habituales en todos los sistemas, como estandarización de todo el interfaz del usuario, gestión y presentación de errores, comportamiento homogeneo del sistema y seguridad.

La aplicación base estará compuesta por los siguientes componentes principales:

- **Menú.** Componente mediante el cual podremos definir los menús de la aplicación de forma declarativa, por ejemplo mediante un xml. La flexibilidad de los menús estaría limitada a un alcance determinado en la fase de análisis y diseño, donde habría que fijar los niveles soportados, su ubicación en el aplicativo, etc.
- **Páginas.** Conjunto de componentes mediante los cuales podamos de forma sencilla disponer de varios típos de páginas con características, funcionalidades y usos distintos. Su alcance también quedaría determinado en la fase de análisis y diseño, planteandose inicialmente las siguientes posibilidades:
  - *Formatos tabulares*. Páginas que representen información a modo de multiregistro. En principio sería información de consulta, no de entrada.
  - *Formato formulario*. Página que presente la información a modo de un único registro, editable si se desea por el usuario.
  - *Formato maestro-detalle*. Página dividida en dos bloques. El primero de ellos presentaría un registro de información en modo consulta (maestro), y a partir del cual obtendríamos una información relacionada multiregistro en modo consulta (detalle).
  - *Formato consulta*. Páginas que permitan realizar búsquedas de información sobre elementos del negocio concretos.
- **Operaciones.** El componente sería encargado de dotar a las páginas que se considere necesaria de operaciones comunes, como paginación, inserción y borrado de un registro, ayuda del sistema, etc. Este componente puede condicionar que el modelo cumpla una serie de interfaces para que aquellas operaciones que requieran interación con el modelo sean posibles.
- **Core.** Tratamiento de errores, librerías de etiquetas, ficheros de configuración, etc.

El *framework* resultante, nos permitiría construir una aplicación nueva a partir de las siguientes tareas:

- **Construcción del modelo de negocio**.

- **Diseño de las páginas.**
- **Diseño de interfaces**
- **Definición de los menús**.
- **Creación de las páginas**.

El *framework* que implementemos haría uso de las siguientes tecnologías y herramientas:

- JSP Spec 2.0
- Servlet Spec 2.4
- Struts 2.0, posiblemente extendiendo su funcionalidad.
- CSS2
- Tiles
- Tag Librarys.
- Javascript.
- XHTML 4.0.1
- Eclipse 3.2
- Tomcat 5.5
- J2EE 5.0

1.1.2 Objetivos

Podemos clasificar los objetivos del proyecto en dos grandes grupos: Objetivos Generales y Objetivos Específicicos.

Los objetivos generales del proyecto son:

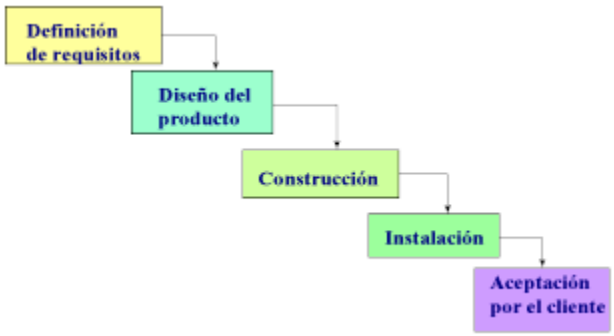
a) Conocer en profundidad el funcionamiento de los *frameworks* de presentación más importantes y sus características.
b) Profundizar en el empleo y uso de patrones de diseño.
c) Aumentar mis conocimientos sobre las tecnologías disponibles para el desarrollo de aplicaciones web.
d) Analizar y diseñar un *framework* que simplifique el desarrollo de la capa de presentación de aplicaciones web, basado en estándares J2EE.

Los objetivos específicos más importantes son:

a) Implementación de un conjunto de componentes que agilicen el desarrollo de nuevos sistemas basados en éstos.
b) Conseguir un *framework* orientado a definiciones declarativas, para evitar desarrollos adicionales para su uso.
c) Obtención de una base que permita un facil desarrollo de una capa de presentación.
d) Arquitectura y diseños flexibles y basados en estándares que puedan ser facilmente adaptables y extendibles a las aplicaciones que lo usen.

1.1.3 Enfoque y método seguido

El planteamiento seguido para el análisis e implementación del framework, ha seguido las primeras fases del ciclo de vida en cascada tradicional en la construcción del software:



Como primera fase del proyecto, se estableció un plan del proyecto que nos permitiera una correcta gestión del mismo. El plan del proyecto incluyó:

i. Alcance de alto nivel.
ii. Objetivos globales y específicos que se esperaban obtener.
iii. Actividades a realizar
iv. Cronograma
v. Tecnología requerida para la implementación

En una segunda fase, se realizó la especificación de los requisitos del framework (capítulo 1.3), una vez analizado y estudiado otros frameworks de presentación existentes en el mercado (capítulo 1.2).

En la tercera fase del proyecto, una vez concluida la especificación, se realizó el diseño del marco de trabajo (capítulo 1.4) teniendo presente como input de la fase el resultado de los requisitos acordados. En paralelo, pudieron comenzar ciertas actividades de implementación.

Como mecanismo de control, se han ido realizando entregables parciales del proyecto, en concreto 3 pruebas de evaluación continua.

1.1.4 Planificación del proyecto

Para poder gestionar el proyecto, se elaboró el plan de trabajo mostrado en la siguiente tabla de actividades:

Actividad	*Descripción*	*F. Inicio*	*F.Fin*
Elaboración Plan de Trabajo	El presente documento. Contiene Descrpción del proyecto, objetivos y cronograma.	18/09	28/09
Preparación del entorno	Preparar las herramientas de diagramas, IDE de desarrollo y librerías.	30/09	2/10
Estudio de tecnologías	Actividad general durante la cual se estudiarían frameworks existentes y distintas teconologías y estándares.	30/09	29/10
Estudio de tecnologías	Actividad general durante la cual se estudiarían frameworks existentes y distintas teconologías y estándares.	30/09	29/10
Frameworks	Análisis de diferentes frameworks de presentación y elaboración de un estudio comparativo.	30/09	29/10
XHTML, CSS y Javascript	Estudio de xhtml, css y javascript para determinar su aplicabilidad en el proyecto.	30/09	14/10
JSP's, Servlets y Tiles	Estudio de jsp's, servlets y tiles para determinar su aplicabilidad en el proyecto.	15/10	29/10
Análisis del framework	Actividad general donde se determinen las funcionalidades del framework.	30/09	29/10
Alcance y limitaciones	Actividad consistente en determinar y documentar las características definitivas del framework y exclusiones explícitas del desarrollo.	30/09	09/10
Funcionamiento del framework.	Actividad consistente en determinar y documentar las funcionalidades del framework .	10/10	24/10
Actividades diseño e Implementación	Una vez conocido el alcance y las funcionalidades esperadas, se replanificarían de forma más precisa las actividades de diseño e implementación requeridas.	25/10	29/10
Descripción app test	Elaboración de una aplicación de prueba para ir testeando los componentes implementados.	25/10	29/10
Diseño e implementación	Actividad general en la que se realizarían los diseños e implementaciones de los componentes requeridos.	02/11/09	05/01
Preparación capa Modelo	Implementación de una capa de persistencia a partir de la app test, utilizada como base para el desarrollo del framework.	02/11	06/11
Componente Core	Diseño e implementación de un componente que gestione el flujo entre la vista y el modelo, el tratamiento de errores, lectura de ficheros de configuración, etiquetas utilizadas por las jsp's, etc.	07/11	21/11
Página Base	Diseño e implementación de un componente main que a partir de tiles componga la plantilla básica de todas las	22/11	26/11

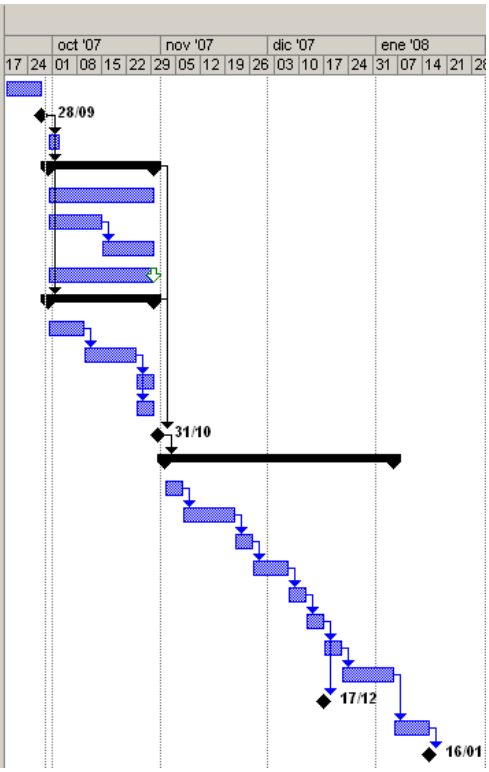
Actividad	Descripción	F. Inicio	F.Fin
	páginas. Dividiría la página en distintas regiones (cabecera, pie de página, menú lateral, menú superior, etc).		
Menú	Elaboración del metadato para los menús de la app y su integración en la página base.	27/11	06/12
Página tabular	Elaboración del metadato para la presentación de las páginas tabulares y su integración en la página base.	07/12	11/12
Página formulario	Elaboración del metadato para la presentación de las páginas formulario y su integración en la página base.	12/12	16/12
Página búsqueda	Elaboración del metadato para la presentación de las páginas de búsqueda y su integración en la página base.	17/12	21/12
Página maestro-detalle	Elaboración del metadato para la presentación de las páginas maestro-detalle y su integración en la página base.	22/12	05/12
Documentación PFC	Elaboración del documento final.	06/01	15/01

El proyecto se puede dividir en 4 fases, con los siguientes entregables, dando lugar además a los hitos del proyecto:

Fase	Descripción	Entregable	Fecha
PEC 1	Plan de trabajo	Documento Plan de trabajo (mmontejano_platreball.pdf)	28/09
PEC 2	Estudio tecnologías y análisis del framework	Documento Estudio Framworks de presentación. Documento análisis framework.	31/10
PEC 3	Diseño parcial del sistema e implementación.	Documento de diseño de Core, página base, menús, páginas tabulares y formularios. Código fuente.	17/10
Entrega final	Completar implementación y documento final.	Documento de diseño de páginas de búsqueda y maestros detalle. Código fuente. Trabajo PFC.	15/01

La planificación prevista, da como resultado el siguiente cronograma de trabajo:

Nombre de tarea	Duración	Comienzo	Fin
Elaboración del plan de trabajo	10 díast	mar 18/09/07	vie 28/09/07
PEC 1	2 días	vie 28/09/07	sáb 29/09/07
Preparación del entorno de trabajo	3 días	dom 30/09/07	mar 02/10/07
Estudio de tecnologías existentes	**30 días**	**dom 30/09/07**	**lun 29/10/07**
Frameworks	30 días	dom 30/09/07	lun 29/10/07
XHTML, CSS y Javascripts	15 días	dom 30/09/07	dom 14/10/07
JSP's, Servlets y Tiles	15 días	lun 15/10/07	lun 29/10/07
Elaborar estudio comparativo	30 días	dom 30/09/07	lun 29/10/07
Análisis del framework	**30 días**	**dom 30/09/07**	**lun 29/10/07**
Detallar alcance y limitaciones	10 días	dom 30/09/07	mar 09/10/07
Definir funcionamiento del framework	15 días	mié 10/10/07	mié 24/10/07
Identificación actividades diseño e implementación	5 días	jue 25/10/07	lun 29/10/07
Descripción app test	5 días	jue 25/10/07	lun 29/10/07
PEC 2	2 días	mié 31/10/07	jue 01/11/07
Diseño e implementación	**65 días**	**vie 02/11/07**	**sáb 05/01/08**
Preparación capa Modelo	5 días	vie 02/11/07	mar 06/11/07
Componentes Core	15 días	mié 07/11/07	mié 21/11/07
Página base	5 días	jue 22/11/07	lun 26/11/07
Componente Menu	10 días	mar 27/11/07	jue 06/12/07
Componente Página Tabular	5 días	vie 07/12/07	mar 11/12/07
Componente Página Formulario	5 días	mié 12/12/07	dom 16/12/07
Componente Página Búsqueda	5 días	lun 17/12/07	vie 21/12/07
Componente Página Maestro-Detalle	15 días	sáb 22/12/07	sáb 05/01/08
PEC 3	1 día?	lun 17/12/07	lun 17/12/07
Documentación del PFC	10 días	dom 06/01/08	mar 15/01/08
Entrega final	5 días	mié 16/01/08	dom 20/01/08



1.1.5 Productos obtenidos

Como resultado del proyecto desarrollado, se han generado los siguientes entregables hasta la fecha:

a) Estudio de *frameworks* en el mercado.
b) Especificación de requisitos del proyecto.
c) Diseño y modelado del *framework* de presentación.
d) Aplicación Web formada por los siguientes componentes:
    i. Tiles (plantillas, fichero de definiciones y jsp's)
    ii. Definición de menús y transformaciones (XML, XSD y XSLT)
    iii. Tag Library (tagsPFC.tld)
    iv. edu.uoc.pfc.*.java
    v. Struts (struts-config.xml y struts.jar)
    vi. Componentes web (jsp's, css e imagenes)

1.1.6 Descripción del resto de capítulos

En los próximos capítulos, se presentará el detalle del trabajo realizado en profundidad.

En el capítulo 1.2 se profundiza en diversas soluciones sobre la capa de presentación existentes en el mercado, describiendolas en detalle y comparándolas entre sí.

El capítulo 1.3 recoge la especificación formal de requisitos en que se basó el trabajo de fin de carrera. Se podrá apreciar que he tratado de seguir en la medida de lo posible las recomendaciones IEEE 830 tanto en la estructura del capítulo como en las prácticas establecidas para una correcta identificación y descripción de los requisitos.

El capítulo 1.4 describe el análisis técnico y consideraciones de diseño tomadas para la implementación de la solución. Durante el diseño se han considerado las interfaces de usuario y el modelado UML de los componentes que conforma el *framework*. Las consideraciones de diseño no consideradas en estos elementos, se han descrito aparte.

Por último, el capítulo 1.5, describe el actual estado de implementación del *framework*.

1.2 Estudio comparativo de soluciones existentes

El siguiente capítulo consiste en un análisis de los *frameworks* de presentación más significativos que existen en la actualidad referentes a la capa de presentación y controladores para implementaciones del MVC en en tecnología J2EE.

El cambio tecnológico hoy en día es muy rápido.En los últimos años han venido apareciendo nuevas soluciones conceptuales y tecnológicas que simplifican los desarrollos software, permitiendonos implementar sistemas más mantenibles, fiables, robustas, extensibles, portables y baratos. Es por tanto esencial comprender correctamente éstas técnicas que han solucionado de forma efectiva problemas comunes con anterioridad.

Los patrones de diseño y *frameworks* ponen a nuestra disposición una serie de técnicas ya contrastadas que nos facilitan la toma de decisiones de los diseños de los sistemas o nos proporcionan un conjunto de componentes para resolver de forma eficaz desarrollos requeridos por las aplicaciones.

Un patrón de diseño muy extendido en las aplicaciones Web es el conocido como Modelo-Vista-Controlador (MVC), el cual nos permite definir los sistemas a partir de tres capas con responsabilidades funcionales diferentes:

- Modelo: Datos y reglas de negocio
- Vista: La presentación
- Controlador. Control de flujo.

Cada una de éstas capas está muy poco acoplada dotándo al sistema de mucha flexibilidad y un bajo impacto ante los cambios de cualquiera de ellas.

Las distintas relaciones entre las capas pueden apreciarse en el gráfico 1.

Un *framework* de aplicación es una colección de componentes que colaboran entre sí para producir una arquitectura reutiliable en un conjunto de aplicaciones. El concepto de *framework* se diferencia de una librería de clases tradicional en que los *frameworks* conforman el esqueleto de un particular dominio de las aplicaciones (presentación, persistencia, seguridad, integraciones...) y son una aplicación "semi-completada".

En el mercado existen y están apareciendo diferentes *frameworks* para ayudar y simplificar la creación de la capa de presentación de las aplicaciones MVC.

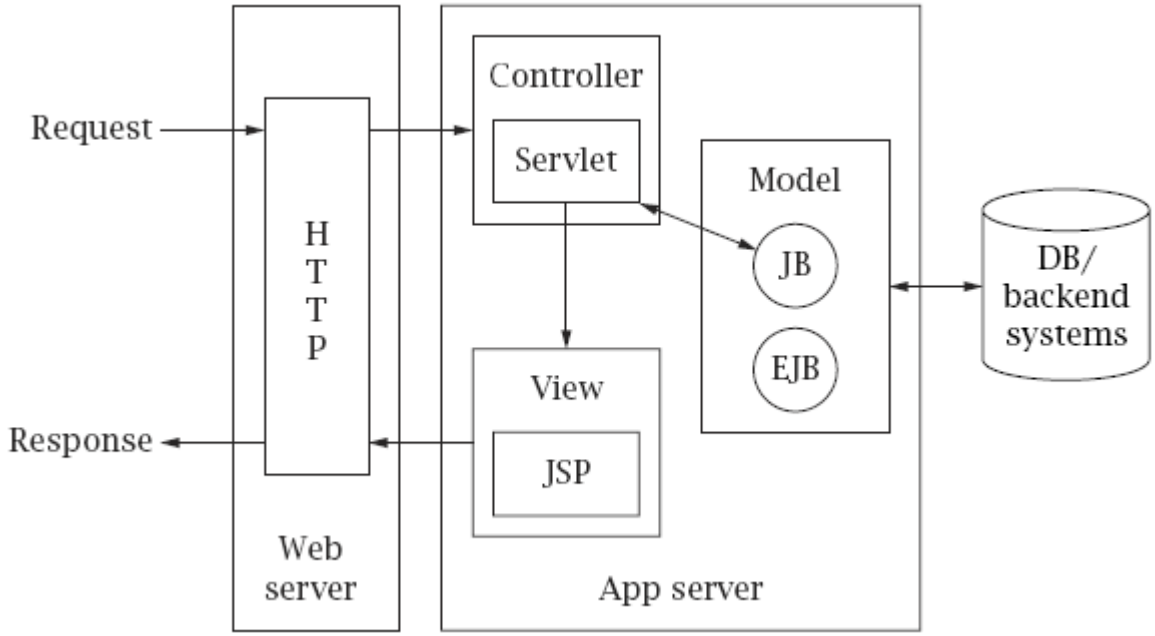


Grafico 1: Esquema Modelo-Vista-Controlador y relaciones entre las capas

A continuación, analizaremos varias tecnologías y soluciones existentes que aportan soluciones para la presentación y control de flujo en arquitecturas MVC. Tras su análisis, realizaremos una comparación e identificaremos posibles alternativas de implementación, las cuales conformarían nuestro *framework* propio de presentación.

1.2.1 Tecnologías y frameworks de presentación

Cuando se pretende realizar un nuevo sistema, uno de las primeras decisiones que hay que tomar es determinar qué arquitectura se espera implementar y bajo qué tecnología.

Como se ha comentado anteriormente, una arquitectura muy frecuente en entornos web es utilizar el patrón de diseño MVC. Para su implementación, existen una gran cantidad de opciones tecnológicas, entre otras
:

- .NET: Monorail, DotNetNuke.
- Python: Django, Gluon, Pylons, TurboGears.

- Perl: Catalyst, Interchange, Jifty, Mason.
- ColdFusion: Fusebox, Mach-II, ColdSpring.
- PHP: CackPHP, Canvas, FUSE, PHPOpenbiz.
- Flex: ARP, Cairngorn.

Dentro del ámbito J2EE, en el cual centraremos la comparación, analizaremos en detalle las características de los siguientes frameworks:

i. Apache Struts
ii. Apache Cocoon
iii. Apache Velocity
iv. FreeMarker
v. JavaServer Faces (JSF)
vi. Tapestry
vii. StrutsCX

Omitiremos del estudio tecnologías que por sí mismas no constituyan un marco de trabajo sino un conjunto de estándares, como JSP's, Tiles, XSLT, XML, etc. No obstante, se entiende que son la base de un sistema J2EE y son los componentes que en último término constituyen los sistemas a los que se refiere el estudio.

1.2.1.1 ***Apache Struts***

Fue diseñado por Craig R. McClanahan. En Mayo de 2002 donó la primera versión a ASF (*Apache Software Foundation's*), y en la actualidad ya conforma un proyecto independiente conocido como *Apache Struts*.

Es probablemente uno de los *frameworks* java más conocido y utilizados. Consiste en una serie de componentes que facilitan la implementación del patrón de diseño MVC, reduciendo tiempos de desarrollo. Tiene carácter "software libre" y es compatible con todas las plataformas en que Java Enterprise está disponible.

Struts está compuesto por un conjunto de componentes que simplifican la presentación y el flujo de comunicación entre la vista y el modelo.

Aunque es frecuente implementar *Struts* junto con la tecnología JSP, no necesariamente tiene que ser así. La característica más importante de *Struts* es permitir la definición de un único controlador (*ActionServlet*) que evalua las peticiones de un usuario mediante un archivo configurable (struts-config.xml).

La configuración declarativa de la interrelación entre las acciones y páginas de un sistema, nos permite no tener que codificar muchas líneas de código y centralizar la gestión y comportamiento del sistema de forma muy sencilla.

Por otro lado, *Struts* incluye un grupo de librerías que simplifican las validaciones de forma en las entradas de datos, los formateos de salidas, el multilenguaje, la gestión de errores, etc.

El controlador *Struts* en la versión 1.3 del framework, consiste en un conjunto de 41 clases y 17 clases de utilidad. A continuación realizaremos una breve explicación de las más importantes y mostraremos un diagrama de clases donde apreciaremos sus relaciones:

- **ActionServlet**: Es una subclase de javax.servlet.http.HttpServlet, es decir, un servlet. Alcanza las peticiones realizadas al *framework* (habitualmente urls con terminación .do).
- **RequestProcessor**: Esta clase tiene muchas responsabilidades, entre ellas obtener los recursos necesarios para las peticiones usando las configuraciones del fichero struts-config y pasar el control a la subclase *Action*.
- **Action**: Es una clase abstracta que alcanza las peticiones específicas del cliente. Siempre es necesario crear la subclase para las necesidades particulares de la petición.
- **ModuleConfig**: Contiene información sobre cada módulo de la aplicación. Parte de esta configuración es representada en las clases siguientes.
- **ActionConfig** y **ActionMapping**: Estas clases representan el mapeo entre peticiones y acciones.
- **FormBeanConfig** y **ActionFormBean**: Estas clases representan los formularios.
- **ForwardConfig** y **ActionForward**: Son las clases que representan los destinos directos a los que se puede redireccionar o enviar un petición.
- **MessageResourceConfig**: Representa el fichero de recursos asociado con un módulo de una aplicación *Struts*.
- **DataSourceConfig**: Representa los elementos datasource en el fichero struts-config. Los datasources son la implementación de las interfaces javax.sql.DataSource que nos proveerán de conexión a bases de datos y pooling.
- **ExceptionConfig**:Representa los elementos exception en el fichero struts-config. Cada elemento excepción define como *Struts* actuará ante un particular tipo de excepción.
- **PluginConfig**: Representa los elementos plug-in en el fichero struts-config.

La relación entre las clases del controlador del framework, quedan representadas por en el gráfico 2:

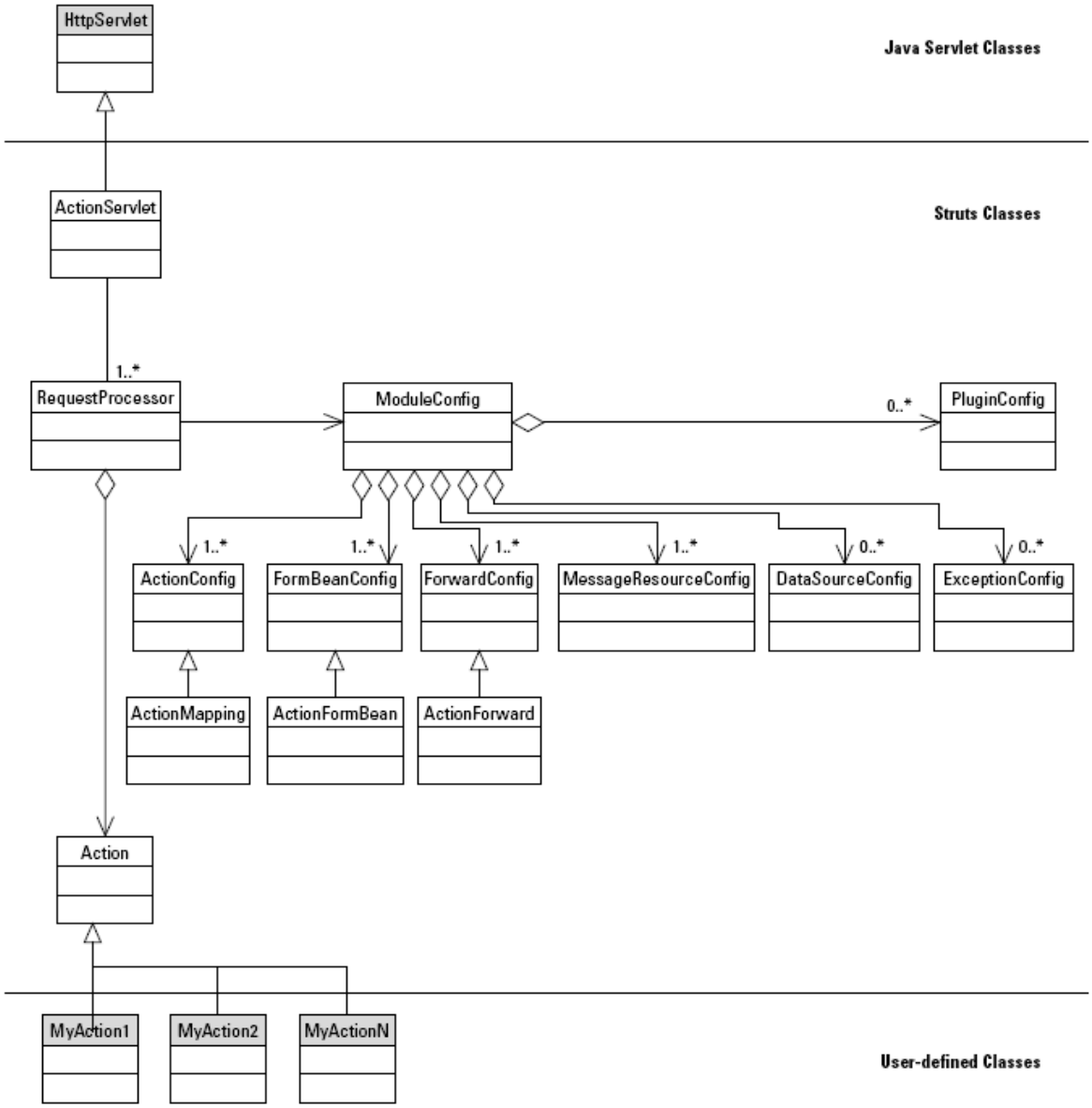


Gráfico 2: Diagrama de clases del framework Struts

1.2.1.2 ***Apache Cocoon***

Apache Cocoon es un *framework* que simplifica la creación de aplicaciones web basadas en componentes. Su objetivo simplificar la construcción de aplicaciones donde los conceptos estén claramente separados (datos y vista) y se implementen

vía componentes.

Cada componente debería especializarse en una operación particular, de tal forma que mediante *Apache Cocoon* sea posible interrelacionarlos sin requerir programación adicional y permitiendo su evolución por separado evitando las dependencias entre ellos y reduciendo el riesgo de conflictos en los cambios.

En el *framework Cocoon*, el término bloque es la unidad de modularización del sistema.

Apache Cocoon contiene *pipelines*, las cuales crean datos XML a partir de una fuente y los transforman a varias teconologías de presentación utilizando XSL.

Sus características más importantes son:

- Está basado en el *framework Spring* y en el concepto arquitectura basada en componentes.
- Para implementar la arquitectura basada en componentes, utiliza componentes diseñados para operaciones particulares que disponen de unas interfaces cláramente definidas para su utilización. Normalmente éstos componentes disponen de transformaciones, serializaciones y generadores que permiten su uso a modo de aplicaciones "Lego", sin necesidad prácticamente de programar.
- Implementa un control de flujo avanzado para tratar las peticiones y respuestas y separar claramente los componentes de datos de los componentes de vista.
- Es código libre.
- Dispone de un importante basado en comunidades.

Entre los usos más destacados nos encontramos con la capacidad para transformar datos orígenes a varios formatos de salida (pdf, html, xml, etc).

1.2.1.3 ***Apache Velocity***

Velocity es un proyecto *Jakarta* que nos provee de un lenguaje simple de scripting para crear páginas. No es necesario introducir código java en éstas páginas.

El núcleo de *Velocity* lo conforma un motor de plantillas que están basadas en Java. Permiten a los diseñadores hacer referencia a métodos definidos dentro del código, permitiendoles trabajar en paralelo con los programadores Java.

Separando claramente las responsabilidades de las personas que participan en el

proyecto, el objetivo del framework consiste en permitir centrar los esfuerzos de los diseñadores en la implementación de la vista y de los programadores en el desarrollo del código.

El mecanismo basado en plantillas permite la reutilización de componentes de forma sencilla y se podría presentar como una alternativa viable al uso de JSP's como solución para la vista en los sistemas que siguan la arquitectura MVC. La tecnología JSP no es incompatible con el framework velocity, pudiendonos encontrar sistemas que utilicen ambas soluciones.

1.2.1.4 ***FreeMarker***

FreeMarker genera texto de salida desde muchos tipos de fuente (HTML, PDF...), basandose en plantillas. Las plantillas *FreeMarker* son esencialmente diseños de páginas que no contienen lógica de aplicación, tan solo información de diseño. Provee una clara separación entre los asuntos concernientes a los diseñadores y los programadores de aplicación. Las plantillas se basan en un potente lenguaje que incluye sentencias de control de flujo, expresiones complejas (*substring*, etc), *macros*, transformaciones en bloques, etc.

Es por tanto una opción tecnológica de la capa vista alternativa a, por ejemplo, las JSP's. Según su própia documentación en la página del proyecto, no es como tal un *framework* web, sino que debe entenderse como un componente que actuará junto a un *framework* para proporcionarnos una presentación ideal basada en plantillas.

Nuevamente nos encontramos con un proyecto de código libre.

1.2.1.5 ***JavaServer Faces (JSF)***

La teconología *JavaServer* Faces simplifica la construcción de interfaces en aplicaciones Java. Los desarrolladores pueden rápidamente construir aplicaciones web incorporando componentes de interfaz reutilizables en una página y conectando éstos componentes a los datos orígenes. Estos componentes soportan eventos generados por el cliente los cuales pueden ser gestionados desde el lado del servidor.

La tecnología está diseñada para ser flexible, por lo que establece los conceptos sin limitar a los desarrolladores a utilizar un lenguaje concreto. Los componentes de interfaz de usuario incluidos con la tecnología JSF encapsulan la funcionalidad del

componente, no la presentación, permitiendo a éstos ser presentados en distintos dispositivos y permitiendo al programador a hacer uso de tags específicos de acuerdo al dispositivo esperado. Un cliente típico son las páginas HTML, por lo que se provee de una librería de etiquetas para su uso.

La tecnología incide en la importancia de la separación de los datos de su presentación y en la división de los roles de desarrollo.

1.2.1.6 ***Tapestry***

Tapestry es un *framework* de código libre del grupo apache que permite crear aplicaciones en java dinámicas, robustas y escalables. *Tapestry* complementa el api estándar *Java Servlet*.

El *framework* asume las responsabilidades de la construcción y puesta a servicio del cliente de URL, persistencia del estado de las sesiones sobre el cliente o el servidor, validación de datos de entrada, internalización y gestión de excepciones.
Las aplicaciones *Tapestry* consisten en crear plantillas HTML usando texto plano y combinandolo opcionalmente con pequeños pedazos de código java usando descriptores de ficheros XML. La aplicación debe ser creada en términos de objetos y los métodos y las propiedades de los objetos para poder permitir el tratamiento de los mismos mediante el *framework*.

1.2.1.7 ***StrutsCX***

El *framework* StrutsCX remplaza JSP con XSLT. StrutsCX devuelve como salida XML bien formado que puede ser transformado por multitud de lenguajes de marcas usando hojas de estilo.

1.2.2 Resumen comparativo

Entre las distintas tecnologías presentadas, podemos apreciar importantes similitudes entre ellas. Destacan su proximidad con el patrón MVC, proporcionando soluciones en la línea de separar claramente las reponsabilidades de las capas que lo constituyen.

Hemos podido comprobar, que los *frameworks Struts*, *Cocoon* y *Tapestry* se centran

fundamentalmente en definir una solución sólida, flexible y escalable para el controlador . Los *frameworks Velocity*, *FreeMarker*, *StrutsCX* e incluso la tecnología JSF están orientadas a ofrecer soluciones para la presentación de las aplicaciones, como alternativas (aunque no excluyentes) a la tecnología JSP.

1.2.2.1 ***Struts, Coocon y Tapestry***

Struts probablemente sea el *framework* más utilizado para la implementación de los controladores en MVC en tecnología J2EE. Existe mucha documentación tanto en literatura como en internet y su amplia difusión ha favorecido el soporte disponible (foros, ejemplos, tutoriales, etc).

Coocon destaca por su potencia para facilitar salidas en distintos formatos haciendo uso de estándares (XSL), ahorrando muchos esfuerzos de programación cuando un sistema tenga como requisito la necesidad de poder tratar con formatos de presentación muy diferentes (pdf's, html's, etc). El tratamiento que se realiza en el controlador para determinar los formatos de salida quedan encapsulados y por tanto transparentes para el desarrollador.

Tapestry se diferencia de los *frameworks* anteriores en la implementación que realiza del patrón MVC, basada en componentes. Mientras que la filosofía de *Struts* y *Cocoon* se basa en invocar acciones que calculen los datos y los ponga a disposición de la vista para que ésta los presente (conocido como arquitectura MVC "push"), *Tapestry* conecta la vista con varios controladores para obtener los datos que requiere (MVC "pull"). Cada componente de vista puede por tanto estar conectado con distintos controladores.

1.2.2.2 ***Velocity, FreeMarker, JSF y StrutsCX***

Aunque se ha decidido analizar JSF como una opción más de presentación, en sí misma no constituye un *framework*, sino un estándar tecnológico. Sin embargo, sus objetivos están estréchamente relacionados con los *frameworks* de presentación, es decir, disponer de mecanismos que simplifiquen el desarrollo de sistemas software a partir de componentes reutilizables y forzando diseños adecuados obligando a separar los datos de las presentaciones en aplicaciones web.

Velocity y *FreeMarker* tienen sin dudas más aspectos en común que diferencias. El lenguaje de *FreeMarker* es más potente por lo que quizá es necesario recurrir a solucines de terceros en menos ocasiones que con el uso de *velocity*. Sin embargo,

ésta potencia puede conducir a componentes de presentación menos simples que en *Velocity*, lo cual en algunos casos puede ser un inconveniente.

Ambas soluciones se basan en el uso de plantillas que permiten separar la lógica de la aplicación de la presentación y obtener componentes reutilizables en otras partes del sistema.

StrutsCX se presenta como una alternativa al uso de jsp's con struts. Es un complemento al framework *Strut* y le dota de la posibilidad de retornar muchos tipos de documento gracias a hojas de estilo.

1.2.3 Conclusiones del Estudio Comparativo

La importancia que ha adquirido el patrón de diseño MVC en las aplicaciones web es evidente tras analizar las soluciones existentes en el mercado. Es dificil determinar donde termina la solución de un *framework* y donde comienza la de otro. En muchos casos nos encontramos con un gran abanico de opciones posibles para la implementación de un sistema con éstas características.

Destaca la relevancia del lenguaje XML no solo como mecanismo de representación de datos sino como tecnología entorno a la cual las trasformaciones XSL nos permiten desacoplar las presentación de la información. Esta división tan clara entre datos y presentación nos permite, como evidencian los *frameworks* analizados, la clara división en las responsabilidades de los distintos roles de trabajo que participan en la elaboración del software.

1.3 Especificación de requisitos

El siguiente capítulo es una Especificación de Requisitos Software (ERS) para el *framework* de presentación que se implementará y conforma el análisis previo a la implementación del mismo.

El documento pretende establecer un alcance conceptual del sistema y demarcar las funcionalidades que se esperan obtener tras su implementación, sin profundizar exhaustívamente en las consideraciones técnicas, las cuales quedarán pendientes de las decisiones de diseño que se tomen durante el capítulo 1.4.

La estructura del capítulo se inspira en las directrices dadas por el estándar "*IEEE Recommended Practice for Software Requirement Specification ANSI/IEEE 830 1998*". Sin embargo, el capítulo 3 recomendado por el estándar (Requisitos específicos) he preferido sustituirlo por una descripción más detallada dentro del capitulo 1.3.2 (Funcionalidades del sistema).

1.3.1 Propósito, Ámbito y Visión General

1.3.1.1 ***Propósito***

El objeto del presente analisis es definir las funcionalidades y restricciones del *framework*. Debe servir de base para la elaboración del diseño y el sistema final deberá satisfacer las condiciones aquí descritas.

El análisis ha estado sujeto está sujeto a revisiones por parte del consultor y mías, presentando en ésta memoria la especificación en su última versión.

1.3.1.2 ***Ámbito del sistema***

El *framework* de presentación se desarrolla únicamente con fines académicos para profundizar en las tecnologías J2EE y en los patrones de diseño, aumentando de esta forma mis conocimientos sobre éste area de la ingeniería del software.

1.3.1.3 Visión General de la especificación de los requisitos

La ERS consta de tres secciones. Esta sección es la introdución y proporciona una visión general. En la sección 1.3.2 se detalla las características del *framework* que se implementará, con el fin de conocer las principales funciones que debe realizar, los datos asociados y los factores, restricciones, supuestos y dependencias que afectan al mismo, entrando al mayor detalle que es posible en éste momento. Los aspectos técnicos a considerar en el diseño de la solución y la situación actual del desarrollo se verá con detalle en capítulos posteriores.

1.3.2 Descripción general

En esta sección se presenta una descripción a alto nivel del sistema. Se presentarán las principales área de negocio a las cuales el sistema debe dar soporte, las funciones que el sistema debe realizar, la información utilizada, las restricciones y otros factores que afecten al desarrollo del mismo.

1.3.2.1 Perspectiva del producto

Desde la primera versión, el sistema debe servir de base para el desarrollo de otros productos, simplificando la creación de los mismos al disponer de una serie de componentes ya implementadas que soluciones problemas concretos.

Se espera que los componentes se basen en definiciones declarativas para evitar programación adicional y que se basen en estándares para permitir su facil escalabilidad y adecuación futura a nuevas tecnologías.

1.3.2.2 Funciones del framework de presentación

En términos generales, el framework estará compuesto por los siguientes componentes:

- Tiles
- Menús.

- Páginas.
- Operaciones.
- Core

Además de los componentes, es necesario añadir una aplicación que implemente a modo de ejemplo el correcto funcionamiento y uso de los mismos.

La idea es que podamos realizar una nueva aplicación siguiendo los pasos que se indican a continuación:

a) **Definición del modelo de negocio**. Esta actividad sería independiente de nuestro *framework* pero fudamental, ya que conformaría la capa modelo, es decir, la obtención de datos y manipulación de los mismos, así como la implementación de las reglas de negocio del sistema.
b) Diseño de las páginas. Durante ésta actividad habría que decidir qué opciones de menú se necesitan, el diseño de la pantalla a la que conduce cada opción de menú y su funcionalidad (si es una pantalla de búsqueda, si es una pantalla de presentación de información en modo tabular, etc).
c) **Diseño de interfaces** utilizadas en la comunicación entre las páginas y el modelo. Se completaría la capa modelo con un conjunto de interfaces adecuadas a los requerimientos de las pantallas.
d) **Definición de los menús**. A partir de ficheros de configuración, se dotaría a la aplicación de las distintas opciones posibles, relacionandolos a su vez con las páginas a las que conducen[1]
e) **Creación de las páginas**. Se espera que las páginas puedan heredar características y comportamiento de unas páginas especiales que podríamos considerar como plantillas, de tal forma que si se espera crear una página de consulta, podamos disponer de una página base que determine y estandarize el resto de páginas de este tipo en la aplicación. El desarrollo de la nueva página sería específico del negocio, centrándose en la información a buscar, la información a presentar, etc. Una vez elaborada la nueva página, sería necesario integrarla adecuadamente en el framework.

A continuación se describe con mayor detalle éstos componentes y como serán soportados en el *framework*.

1 La opción de menú se relaciona con la página mediante una acción *Struts*.

1.3.2.2.1 Secciones de la vista

El *framework* de presentación propuesto, dispondrá de una serie de componentes basados en *tiles* que nos permitirán obtener un sistema software homogéneo y fácilmente mantenible y extensible.

El sistema presentará en general una distribución de elementos similares a los presentados en el gráfico 3.

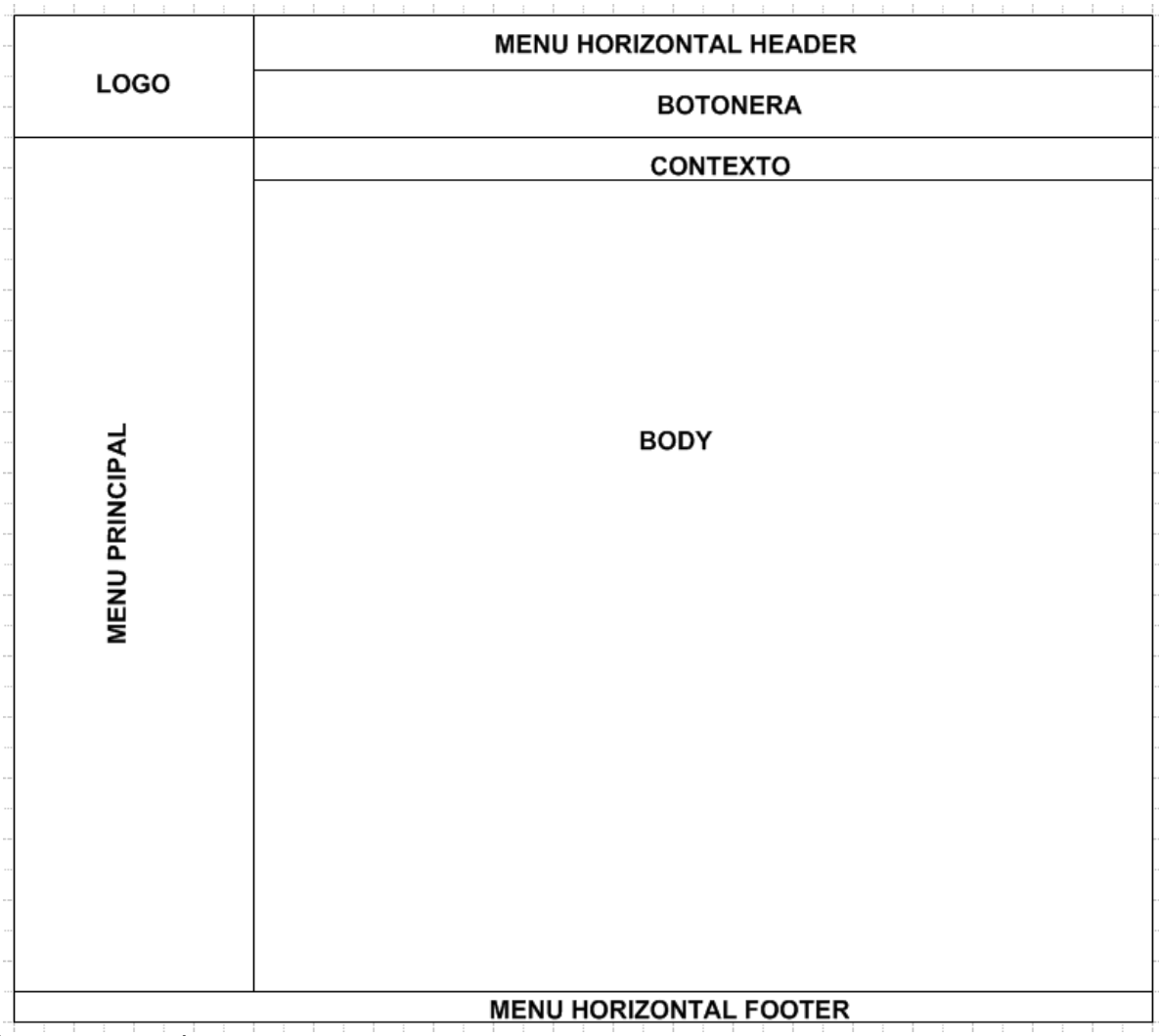


Gráfico 3: Distribución del contenido a partir de Tiles

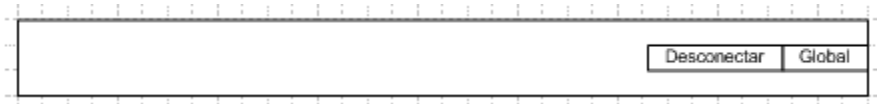
Cada región representada por un rectángulo debe ser autosuficiente y disponer de una funcionalidad particular.

Un uso ideal del *framework* consistiría en concentrar la funcionalidad específica del sistema que lo use en las regiones Contexto + Body, sin necesidad de modificar las regiones Logo, Menús ni botonera.

Intentaremos gestionar cada elemento que compone la pantalla de la forma más desacoplada posible, de tal forma que el programador pueda centrarse en la mejor implementación posible de lo que constituirá el cuerpo de las páginas y el diseñador podrá centrar sus esfuerzos en configurar cada uno de las regiones para obtener la mejor presentación posible.

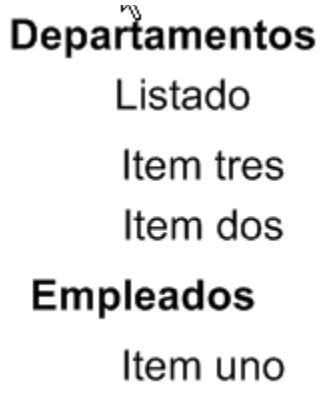
Las responsabilidades de cada area serían las siguientes:

- **Logo**: Presentación de una imagen configurable desde un fichero de recursos.
- **Menú header**: Presentación de opciones de menú horizontales, por ejemplo podrían ser:



- **Menu footer**: Presentación de opciones de menú horizontales, por ejemplo:

- **Menu principal**: Presentación de opciones de menú verticales, hasta dos niveles de indexación:



- **Botonera**: Herramienta que aplicará para determinados cuerpos de página (no siempre). Presentaría las operaciones definidas en el frameworks sobre el modelo de datos, de forma similar a la siguiente:

- **Contexto**: Región que nos permitiría establecer un título sobre el cuerpo y una información de referencia para situarnos como usuarios.
- **Cuerpo**: Area donde presentaríamos las páginas que conformen el negocio en cuestión (relación de datos, formularios, etc).

El objetivo de la implementación de estos componentes es poder conocer la tecnología *Tiles* y entender su alcance y aplicabilidad.

1.3.2.2.2 Menús

El *framework* implementará un mecanismo que permita la definición declarativa de los menús anteriores. Desde mi punto de vista, la forma más natural de conseguirlo es haciendo uso de documentos XML.

Se espera que el usuario del *framework* pueda variar las opciones de menú sin programar, especificando los menús de forma declarativa.

El elemento principal del documento XML (menu) dispone de dos atributos opcionales:

- Presentación. Puede tomar los valores horizontal o vertical. Utilizaremos el menú horizontal para los menús de cabeceras y pie de página definidos en el punto 2.2.1. El menú vertical lo reservamos para el menú principal. Cabe destacar que debido a la presentación y finalidad de éstos menús, cuando el menú sea horizontal tan solo se considerarán los items del primer nivel.
- Estilo. Identifica una clase definida en la hoja de estilo para ser utilizada.

Los menús se presentarían de acuerdo al orden establecido en el propio fichero de XML.

Se espera el siguiente comportamiento sobre los componentes Menú:

- Cuando el sistema deba utilizar un menú, el framework validará que el documento está bien formado y es válido respecto al esquema anterior.
- Si el menú es correcto, aplicará una hoja de estilo que transformará el

documento XML en código HTML.

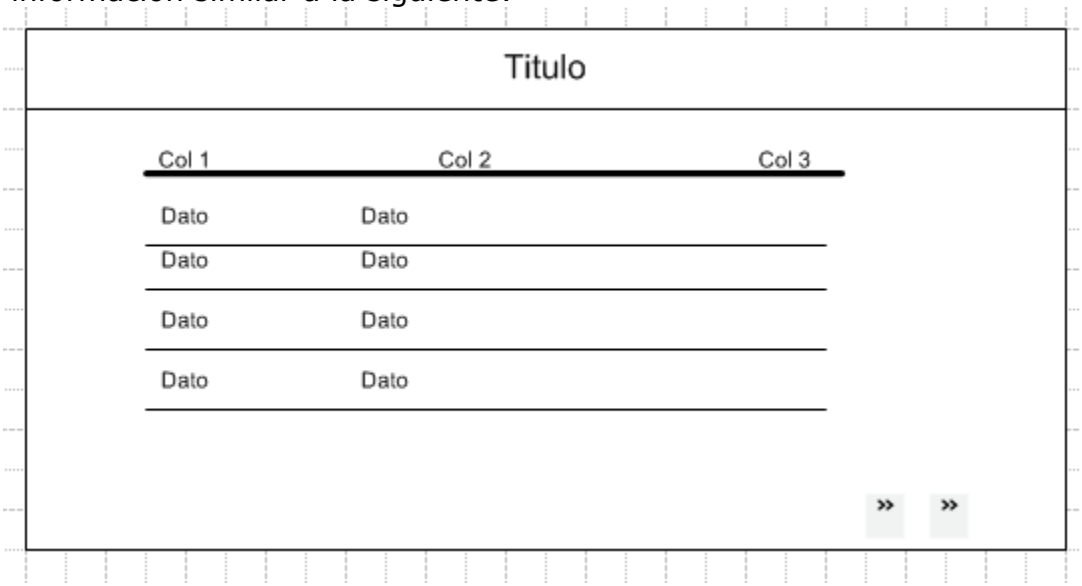
- Éste código HTML será utilizado por una página jsp para conformar el menú en cuestión una vez aplicados los estilos necesarios.

El objetivo de la implementación de estos componentes es profundizar sobre la tecnología XML y XSLT y entender su alcance y aplicabilidad.

1.3.2.2.3 **Páginas**

El *framework* incluirá una serie de componentes mediante los cuales podamos de forma sencilla disponer de varios tipos de páginas que nos proporcionarán funcionalidades distintas. Cada tipo de página dispondría de su *tile* específico para simplificar su creación:

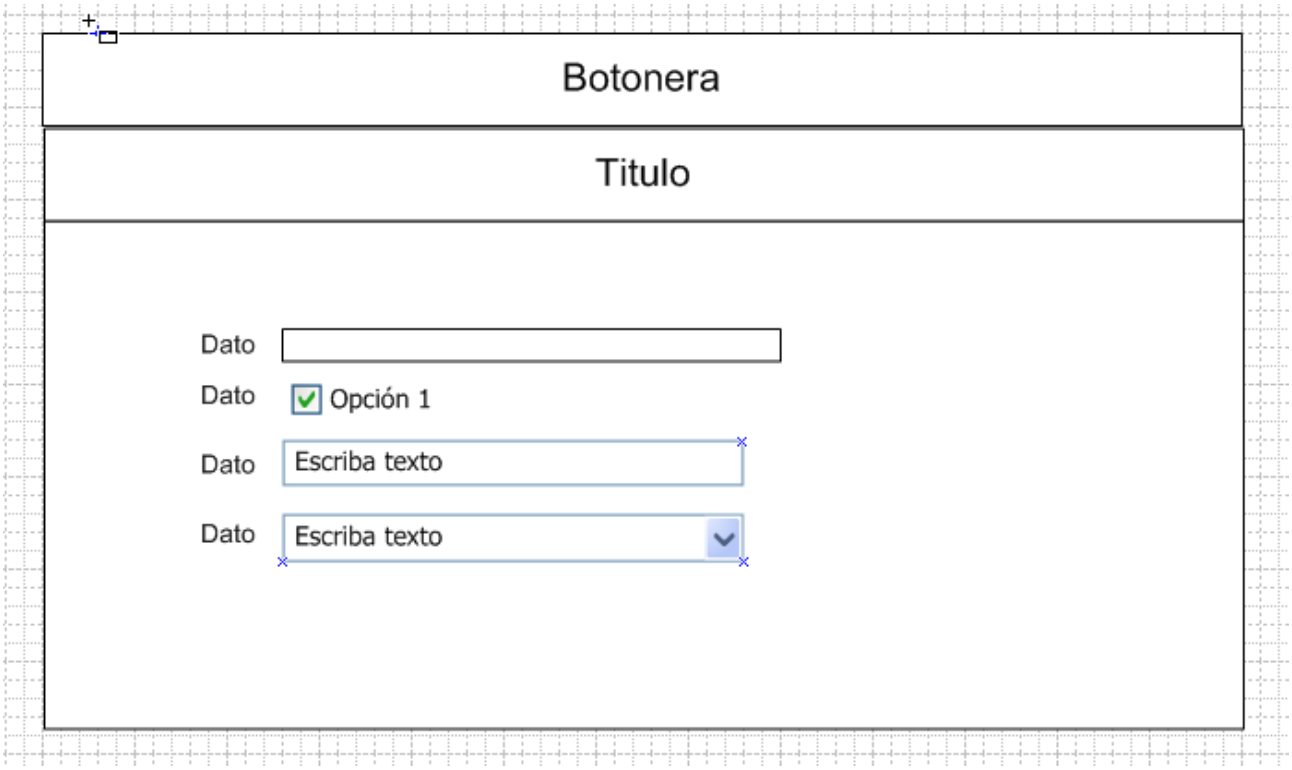
- **Formatos tabulares**. Representaría información obtenida del modelo en formato tabular y sería solo de salida. Correspondería por ejemplo a la relación de empleados indicada en el punto de menú, y se espera obtener información similar a la siguiente:



- **Formato Consulta:** La funcionalidad de búsqueda se forma mediante dos tiles distintos. Por un lado, un primer tile que permite establecer un filtro. El resultado de la consulta sería el formato tabular con la información obtenida.

- **Formato formulario**. Serían páginas que representarían la información a de un registro concreto editable por el usuario. Para éste tipo de páginas es

donde toma más importancia la botonera, ya que parte de las acciones que podrá realizar el usuario serían mediante las opciones de la barra de herramientas (confirmar los datos modificados, deshacer, etc).



Se espera que éstas páginas estén dotadas de soporte idiomático y de una gestión de validaciones a partir de las soluciones de Struts.

El objetivo de estos componentes es familiarizarme con el framework de Struts.

1.3.2.2.4 **Operaciones**

El componente sería encargado de dotar a las páginas que se considere necesaria de operaciones comunes, como paginación, inserción y borrado de un registro, ayuda del sistema, etc. Este componente puede condicionar que el modelo cumpla una serie de interfaces para que aquellas operaciones que requieran interación con el modelo sean posibles.

1.3.2.2.5 Core

El componente core estará formado por una librería de etiquetas que nos permitan definir de forma sencilla en las páginas JSP's las funcionalidades, encapsulando su implementación y delegandola al Core.

Entre otras etiquetas, contaríamos con:

- **IncluirMenu**: La JSP obtendría un Menú definido en el punto 2.2.2.
- **IncluirTabular**: La JSP incluiría un mantenimiento de tipo tabular para una entidad de negocio definida previamente.
- **IncluirFormulario**: La JSP incluiría un mantenimiento de tipo formulario para una entidad definida previamente.
- **IncluirBusqueda**: La JSP incluiría un mecanismo de búsqueda para una entidad determinada.

Es probable que requiera realizar un mapeo entre las entidades de negocio que utilice y su presentación en la página JSP. Si es necesario crearía unos ficheros de configuración en XML que nos permitan definir como debe presentarse los datos en la página, donde se considere el texto que debe aparecer, la longitud del campo, el tipo de elemento HTML que lo representa (radio buton, check, etc), validaciones, etc.

A partir de los componentes resultantes espero poder aprender y practicar el alcance y uso de las librerías de etiquetas.

1.3.2.3 Características de los usuarios

Para poder hacer uso del framework es necesario tener conocimientos previos sobre las siguientes tecnologías:

- XML, HTML y CSS
- JSP's
- Tag Librarys
- Tiles
- Struts
- Java

Conviene además estar familiarizado con las estructuras y procesos habituales de las aplicaciones Web: descriptores de aplicaciones, despliegues, etc.

1.3.2.4 ***Restricciones***

El *framework* no realiza ninguna consideración específica entorno a la seguridad, la cual deberá ser proveída externamente.
Aunque en la aplicación de ejemplo que se acompaña se ha tratado de cuidar el diseño estético, el framework no se centra excesivamente en él. Queda abierta la posibilidad de su mejora al utilizar elementos estándares J2EE como hojas de estilo, presentaciones de menús basadas en capas, no existencia de javascript, etc.

1.3.2.5 ***Suposiciones y dependencias***

El framework de presentación que se pretende elaborar requiere de la tecnología indicada en la planificación inicial, es decir, depende de:

- Java 1.5
- Struts 1.3
- JSTL, JSP y Tiles

El sistema debería poder ser utilizado con cualquier servidor de aplicaciones que conforme con J2EE con las consideraciones de versión de librerías indicadas.
Ha sido probado bajo Internet Explorer 7.0 y Firefox 2.0, aunque es probable que no presente ningun problema con otros navegadores modernos.
Se asume que el framework es ampliable y podría disponer de nuevas funcionalidades y características en el futuro.

1.4 Modelado y diseño

Este capítulo establece el diseño y decisiones técnicas establecidas para la implementación del *framework* de persistencia. El diseño representa los componentes que conformarán el producto final y describe la relación entre ellos, tratando de considerar y cumplir en todo momento el alcance definido en el capítulo anterior.

El diseño planteado para la implementación del *framework* se compone de una serie de diagramas donde se modeliza la solución al problema que esperamos resolver. Se ha intentado en la medida de lo posible representar dichas soluciones como UML, aunque por claridad cada componente descrito más adelante se acompaña de un análisis técnico detallado, que describen tanto las estructuras utilizadas como los procesos requeridos.

La sección termina presentando el diagrama de componentes final que constituirá el *framework*, reflejando la relación entre los elementos y confiando en que podrá servir de base para determinar los entregables (jars o wars) del mismo.

1.4.1 Componentes de presentación

La aplicación de pruebas que esperamos implementar para testear nuestro *framework*, constaría de las siguientes pantallas:

i. **Página de inicio:** Sería una página html muy simple, cuya principal función consistiría en presentar un link de acceso a la aplicación.y serviría de punto de acceso al sistema. No haría uso del framework, salvo por el uso de la hoja de estilos.
ii. **Página de login:** La página nos proporcionaría la funcionalidad de autenticación al sistema y se basaría en una serie de *tiles* para componer la página que finalmente ve el usuario.
iii. **Página de login exitoso**. Sería la página inicial del sistema, desde donde tendríamos acceso a las opciones principales ya que nos mostraría el menú principal.
iv. **Página de acciones**. Finalmente, a través de las opciones de menú, llegaríamos a una nueva página donde el contenido principal de la pantalla nos ofrecería la posibilidad de disponer de una sección de botonera para realizar operaciones sobre los datos contenidos en otra sección.

Las distintas páginas que requerimos se implementarían como jsp (*Java Server Page*), aún la página de inicio que podría haberse optado por implementar utilizando directamente HTML. La razón principal es poder hacer uso de los *tag library* disponibles en J2EE.

Mediante el uso de *tiles* de Struts, podemos dividir cada página en regiones y tratar a cada región como un componente independiente. De esta manera, cada región (o sección) quedará implementado mediante un *jsp* específico y evitaremos tener que repetir código en las pantallas que compartan o hagan uso de éstas regiones. Por otro lado, este diseño para abordar la implementación de la capa web nos permitirá un fácil mantenimiento del sistema, ya que podremos realizar cambios sobre las regiones de forma bastante sencilla (modificando el *tile* que lo implementa).

Los distintos *tiles* que compondrán el framework son los siguientes:

Nombre	Path	Descripción
logo	logo.jsp	Presenta el logotipo de la aplicación
tituloApp	tituloApp.jsp	Sección destinada a la presentación de un título de primer nivel
login	login.jsp	Nos proporcionará un pequeño formulario que nos permita autenticarnos en el sistema
menuHeader	menuHeader.jsp	Presentación del menú de cabecera de la aplicación, el cual se encuentra ubicado en la parte superior derecha de las páginas.
menuPpl	menuPpl.jsp	Menú principal de la aplicación, ubicado a la izquierda de la sección central.
contexto	contexto.jsp	Región utilizada en la presentación de los mantenimientos para indicar posibles errores en las operaciones, presentación de información sobre la pantalla que se está visualizando, etc.
botonera	botonera.jsp	Barra de herramienta con las opciones disponibles.
contenido	contenido.jsp	Región principal de la aplicación ubicada en el centro de las páginas donde se presentarán los datos y podremos interactuar con ellos.
menuFooter	menuFooter.jsp	Presentación del menú al pie de las páginas.

La forma de combinar los diferentes *tiles* es mediante el uso de plantillas. Una plantilla es a su vez otra jsp que considera cualquier aspectos relacionados con la presentación de la información (alineaciones, colores, fuentes, etc). De esta forma conseguiremos una clara diferenciación entre los elementos responsables de éstos aspectos de presentación y las funcionalidades que aporten las regiones.

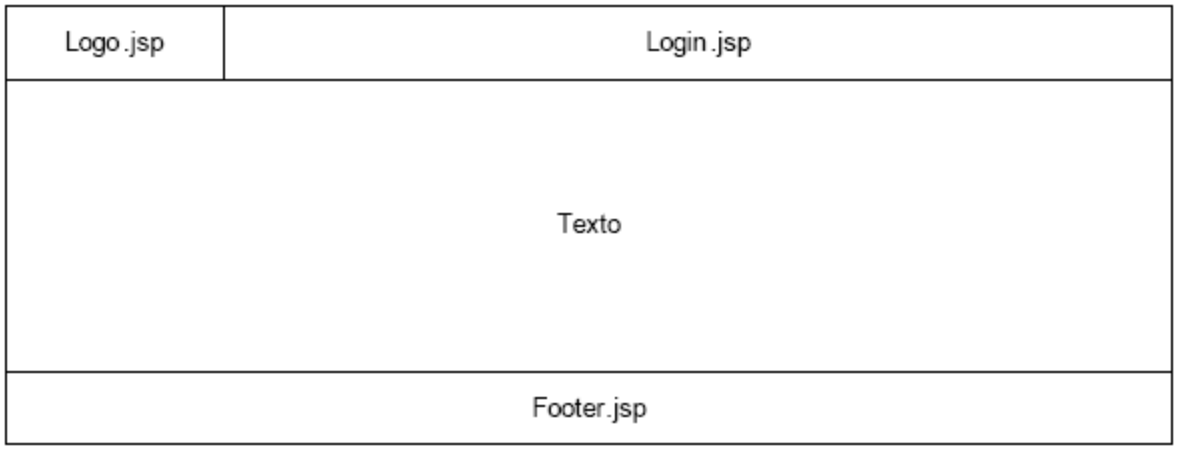
Una importante característica de los tiles es que nos permiten establecer relaciones jerarquicas entre ellos, dotandonos de un mecanismo "similar" a la herencia en java. Este aspecto es muy importante porque como se desarrollará posteriormente durante el diseño lo necesitaremos para poder "extender" el framework y dotar al sistema de las funcionalidades propias del negocio.

Partiendo de las diferentes pantallas que queremos obtener, requeriríamos una serie de plantillas que describiremos en los siguientes puntos:

i. Plantilla de autenticación
ii. Plantilla de inicio
iii. Plantilla de gestión

1.4.1.1 ***Plantilla para la autenticación***

La plantilla de autenticación presenta la estructura siguiente:



La estructura estaría formada por los tiles logo, login y footer. Presentaría un texto central que se correspondería con indicaciones sobre las siguientes acciones que puede realizar el usuario, o una imagen significativa.

En este punto de la aplicación, el *tile* footer ya debería resolver el menú horizontal, colaborando con otros componentes del framework.

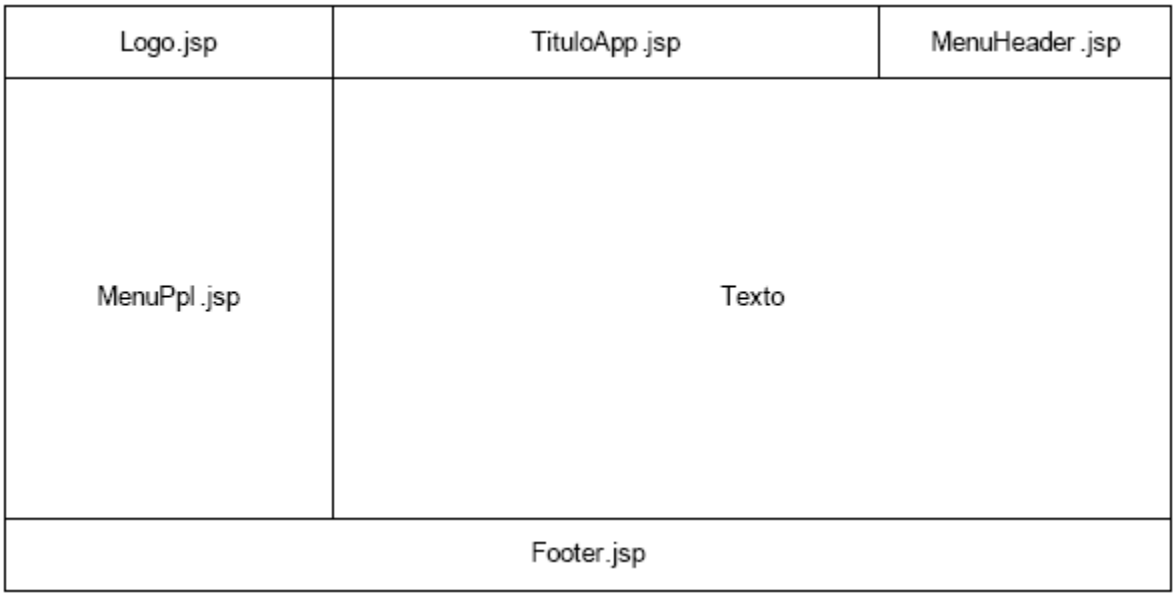
Como resultado del diseño, deberemos incluir la definición que se presenta a continuación en la configuración de los tiles:

```
<definition id="pfc.PlantillaInicioNoLogin" name="pfc.PlantillaInicioNoLogin"
  path="/jsp/layouts/PlantillaInicioNoLogin.jsp">
  <put name="titulo"  value="Indique un título para la pagina" />
  <put name="logo" value="/jsp/tiles/logo.jsp" />
  <put name="login"   value="/jsp/tiles/login.jsp" />
  <put name="menuFooter" value="/jsp/tiles/menuFooter.jsp" />
</definition>
```

El componente que servirá de plantilla para la implementación de la definición se llama PlantillaInicioNoLogin.jsp, y la jsp que lo usa se llama InicioNoLogin.jsp.

1.4.1.2 ***Plantilla de inicio***

Tras la autenticación del usuario, accedemos a la plantilla de inicio y presenta la siguiente forma:



Hace uso de los *tiles* logo, tituloApp, menuHeader, menuPpl y footer.

Al igual que en la plantilla anterior, se incluye un texto fijo dentro de la plantilla que no queda sujeto a ningún tile específico, tan solo sería información adicional.

La definición tile que conforma la nueva plantilla se ha llamado pfc.PlantillaInicio y su definición es la siguiente:

```
<definition id="pfc.PlantillaInicio" name="pfc.PlantillaInicio"
  path="/jsp/layouts/PlantillaInicio.jsp">
  <put name="titulo"  value="Indique un título para la pagina" />
  <put name="logo" value="/jsp/tiles/logo.jsp" />
  <put name="tituloApp"   value="/jsp/tiles/tituloApp.jsp" />
  <put name="menuHeader"   value="/jsp/tiles/menuHeader.jsp" />
  <put name="menuFooter" value="/jsp/tiles/menuFooter.jsp" />
</definition>
```

El componente que servirá de plantilla para la implementación de la definición se llama PlantillaInicio.jsp, y la jsp que lo usa se llama Inicio.jsp.

1.4.1.3 ***Plantilla de Gestión***

Las distintas gestiones que podamos implementar en la aplicación, pueden dar lugar a diferentes estructuras visuales. La estructura más frecuente sería la presentada a continuación:

<table>
<tr><td>Logo.jsp</td><td>TituloApp.jsp</td><td>MenuHeader.jsp</td></tr>
<tr><td rowspan="3">MenuPpl.jsp</td><td colspan="2">Botonera.jsp</td></tr>
<tr><td colspan="2">Contexto.jsp</td></tr>
<tr><td colspan="2">Body.jsp</td></tr>
<tr><td colspan="3">Footer.jsp</td></tr>
</table>

La plantilla será la más sofisticada posible, e incluiría los tiles anteriores más el tile botonera, contexto y body. La estructura es la misma que la plantilla anterior, añadiendo nuevos tiles. Parte de la potencia de los tiles, es poder reaprovechar diseños ya establecidos para extenderlos y dar lugar a nuevas plantillas.

Para poder practicar con ésta característica, la plantilla de gestión extenderá de la anterior:

```
<definition name="pfc.PlantillaGestion" extends="pfc.PlantillaGestion ">
  <put name="contexto" value="/jsp/tiles/contexto.jsp" />
  <put name="botonera" value="/jsp/tiles/botonera.jsp" />
  <put name="body" value="/jsp/tiles/body.jsp" />
</definition>
```

Durante el desarrollo, implementaríamos varios *tiles* "botonera", ya que cada dependiendo de la operación de gestión podría variar. Por ejemplo, cuando nos encontremos ante una pantalla de consulta, las operaciones que podríamos realizar sobre los datos será la ejecución de la búsqueda o la cancelación, mientras que cuando estemos ante una página de mantenimiento de datos, podremos realizar las operaciones de alta, baja, edición, etc.

El componente que implementa la definición de tiles se llama PlantillaGestion.jsp, y es utilizado por las páginas Consulta.jsp, Tabular.jsp y Formulario.jsp.

1.4.1.4 ***Esquema de componentes***

A modo de resumen, se representan gráficamente las relaciones entre los tiles, sus definiciones, las plantillas y las páginas que los utilizan en el gráfico 4.

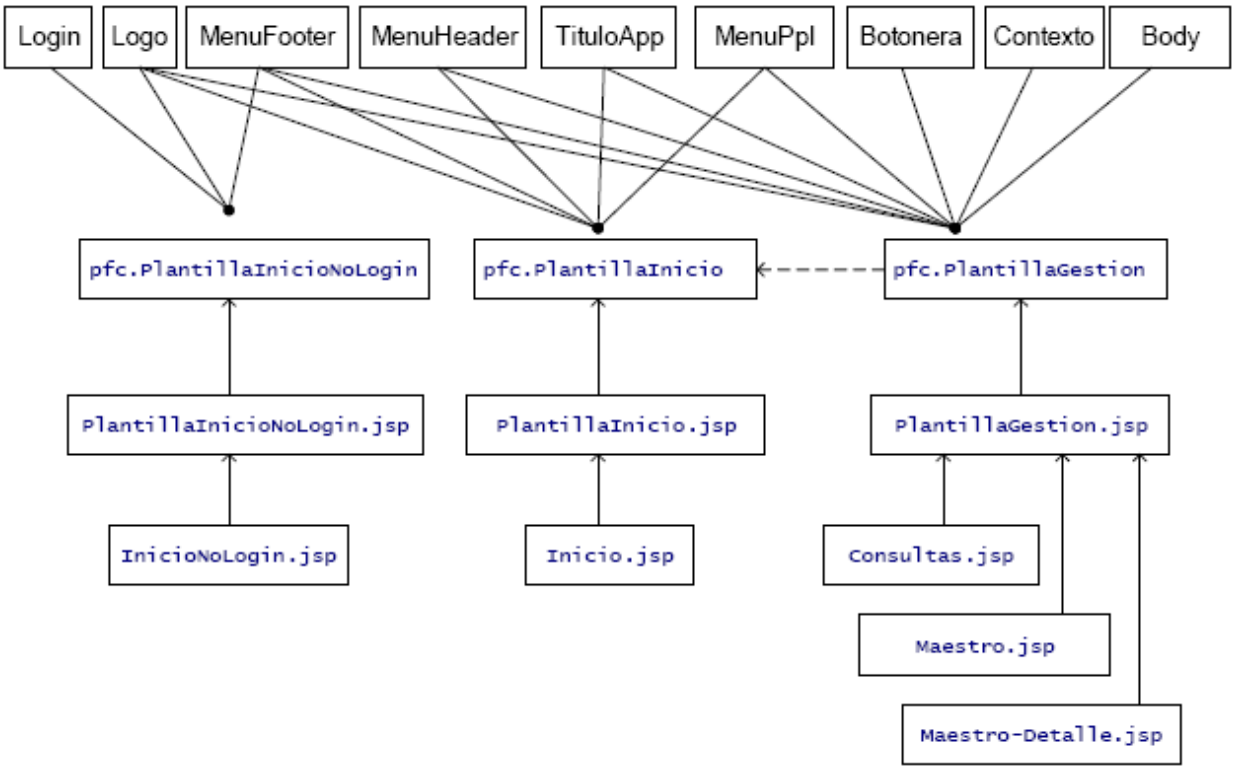


Gráfico 4: Componentes visuales y sus relaciones

1.4.2 Componentes Menú

Para poder gestionar de forma sencilla las opciones de menú, se ha planteado una solución basada en documentos XML que nos permitan describir las características del menú de manera declarativa. Los documentos XML que componen los menús son transformados a código HTML utilizando un documento XSLT.

Como componentes adicionales disponemos de un esquema XML que nos permitiría validar que el documento XML del menú es correcto, un fichero de propiedades que nos indicará que menú debemos utilizar en cada caso y una clase de utilidad que nos permita llevar a cabo la transformación de XML a HTML.

En la implementación de los componentes que se está llevando a cabo, trabajamos con tres menús diferentes:

i. **Menú de cabecera** (menuHeader.xml). Es un menú horizontal que se presentará en la parte superior de las páginas. En el ejemplo, presenta 2 links (header1 y header2). Algunos usos comunes en estos tipos de menú son la presentación de opciones de logout, administración, preferencias del sistema, etc.
ii. **Menu principal**. (menuBody.xml). Es un menú vertical que puede soportar hasta tres niveles de indexación. En nuestro ejemplo, presenta las páginas que queremos gestionar desde nuestro sistema.
iii. **Menu pie de página** (menuFooter). Es un menú horizontal, normalmente de un único nivel. Al igual que el menú de cabecera, para nuestro ejemplo presenta solamente dos links.

Durante las siguientes secciones se describirá a nivel técnico los elementos que hacen posible la presentación de ésos menús.

1.4.2.1 Estructura del menú

El esquema xml's que describen la estructura de los documentos xml de los menús se presenta en el gráfico 5.

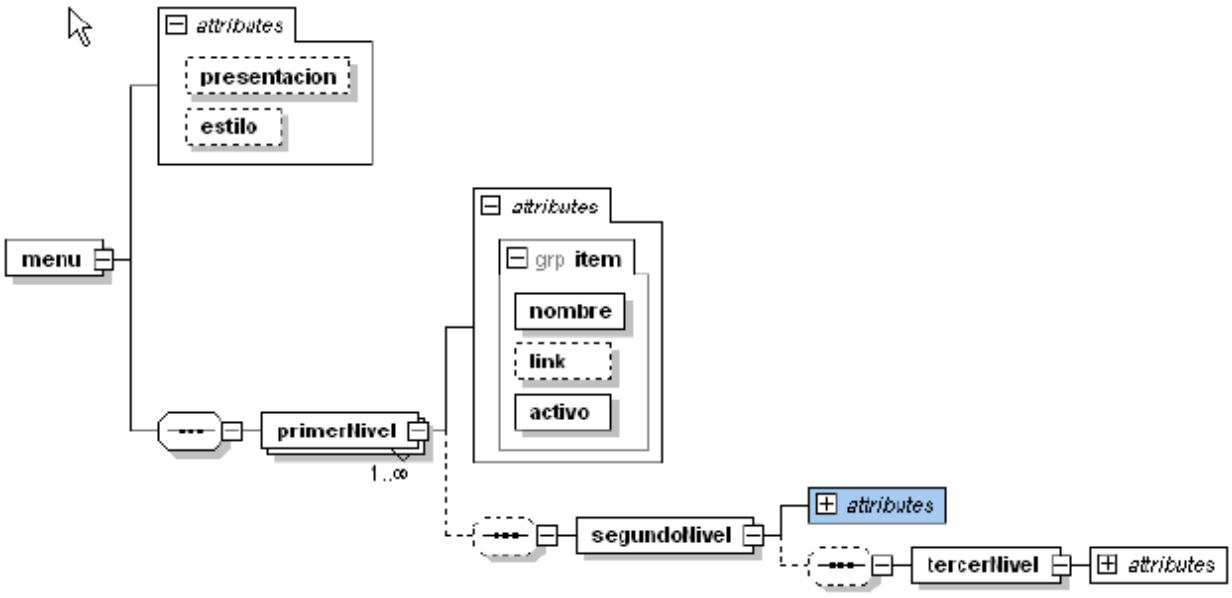


Gráfico 5: Esquema propuesto para la definición de menús

El elemento principal del documento XML, llamado menu, dispone de dos atributos opcionales:

i. **Presentación**. Puede tomar los valores horizontal o vertical. Utilizaremos el menú horizontal para los menús de cabeceras y pie de página. El menú vertical lo reservamos para el menú principal. Cabe destacar que debido a la presentación y finalidad de éstos menús, cuando el menú sea horizontal tan solo se considerarán los items del primer nivel.
ii. **Estilo**. Identifica una clase definida en la hoja de estilo para ser utilizada.

El menú se compone de uno o más items de primer nivel. A su vez, el item de primer nivel se puede componer de uno o más items de segundo nivel, aunque no es necesario. Todos los items de menú comparten un conjunto de atributos:

i. **Nombre**: Este atributo será el que se presente al usuario como nombre del menú. Es obligatorio indicar uno.
ii. **Link**: Es la URL que invocaremos cuando se pulse sobre la opción de menú.
iii. **Activo**: Permite desactivar o activar el punto de menú, lo que significa que se presentará o no al usuario.

Los menús se presentarían de acuerdo al orden establecido en el propio fichero de XML.

*1.4.2.2 **Transformación***

La forma más natural para transformar el documento XML en contenido html para que pueda ser presentado por un navegador es utilizar las hojas de estilo XSLT.

En la implementación realizada hasta el momento (gráfico 6), se incluye una primera aproximación de la hoja de estilos definitiva del framework (/conf/xslt/menu.xsl). La hoja de estilos aún no está considerando los estilos en los que se presentarán las opciones de menú ni si están o no activados.

Lo que ya está completo es la presentación tabulada de los distintos niveles del menú, sustituyendo tanto el link como el nombre como que debe aparecer en su lugar. También diferencia cuando un menú es horizontal y vertical, permitiendonos avanzar en el análisis del resto de los componentes.

El código fuente actual puede consultarse en la aplicación Test.

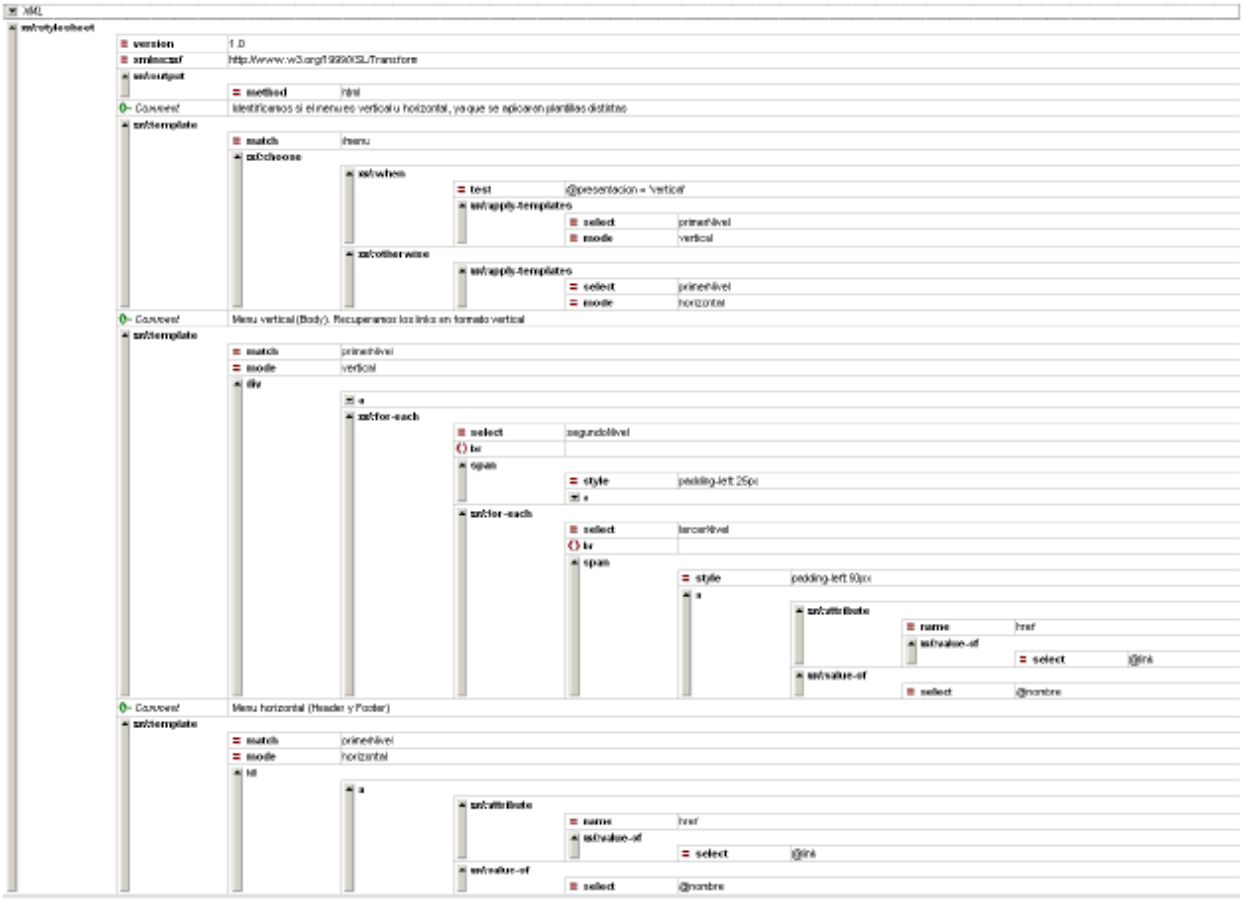


Gráfico 6: Transformación menu.xsl

Para conseguir transformar el documento XML en HTML, debemos indicar en cada menú que utilice la hoja de estilos desarrollada para que el navegador pueda aplicar la transformación.

En los documentos xml de los menús, hemos incluido por tanto la siguiente directiva:

```
<?xml-stylesheet type="text/xsl" href="../xslt/menu.xsl"?>
```

1.4.2.3 Validaciones del framework

El framework implementará la lógica de control necesaria para validar que la estructura del documento XML es válida respecto al esquema definido. La validación será responsabilidad de la clase java edu.uoc.pfc.util.ProcesaXML.

1.4.2.4 Integración con los componentes de presentación

Tal y como se describió anteriormente, los menús se deben comportar como el resto de los tiles.

En este punto es necesario precisar un aspecto técnico importante que afecta al diseño de la solución.

Las plantillas en que basaremos los componentes de presentación descritas en puntos anteriores, son páginas jsp's. Su contenido fundamental son tiles y éstos en su mayoría otros jsp's. El resto de tiles pueden ser simplemente cadenas que se sustituyen por textos (el título de la ventana, por ejemplo), o tiles que contienen menús, los cuales hemos visto que se definen mediante documentos XML.

Cuando un navegador moderno realiza la presentación de un documento XML con una hoja de estilos asociada, el mismo navegador es capaz de realizar la transformación. Sin embargo, en nuestro framework, nos encontramos con la necesidad de presentar una página jsp que contiene contenido dinámico y contenido XML transformado.

Tras analizar distintas opciones, la solución final consistirá en trasladar la responsabilidad de la transformación la clase java edu.uoc.pfc.util.ProcesaXML.

En el estado actual de implementación del framework, es el mismo tile menuFooter.jsp quien resuelve la lógica necesaria para la combinación del documento XML con los estilos XSLT, aunque se espera próximamente encapsular la lógica en la clase ProcesaXML.

1.4.2.5 Ejemplo de menús

Para que un menú sea válido, debe cumplir con el esquema descrito en el punto 1.4.2.1. Algunos menús correctos serían:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="../xslt/menu.xsl"?>
<menu presentacion="vertical" estilo="MenuBody">
  <primerNivel link="pagina1.jsp" nombre="pagina1" activo="true">
    <segundoNivel link="pagina1_1.jsp" nombre="pagina1_1" activo="true"/>
    <segundoNivel link="pagina1_2.jsp" nombre="pagina1_2" activo="true"/>
    <segundoNivel link="pagina1_3.jsp" nombre="pagina1_3" activo="true"/>
    <segundoNivel link="pagina1_4.jsp" nombre="pagina1_4" activo="true">
      <tercerNivel link="pagina1_4_1.jsp" nombre="pagina1_4_1" activo="true"/>
    </segundoNivel>
  </primerNivel>
  <primerNivel link="pagina2.jsp" nombre="pagina2" activo="true">
    <segundoNivel link="pagina2_1.jsp" nombre="pagina2_1" activo="true"/>
  </primerNivel>
</menu>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="../xslt/menu.xsl"?>
<menu presentacion="horizontal" estilo="UnEstiloCSS">
  <primerNivel link="footer1.jsp" nombre="footer1" activo="true"/>
  <primerNivel link="footer2.jsp" nombre="footer2" activo="true"/>
</menu>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="../xslt/menu.xsl"?>
<menu presentacion="horizontal" estilo="UnEstiloCSS">
  <primerNivel link="header1.jsp" nombre="header1" activo="true"/>
  <primerNivel link="header2.jsp" nombre="header2" activo="true"/>
</menu>
```

1.4.3 Páginas y operaciones

Las páginas de los datos es seguramente el componente más abierto que quedará en el framework propuesto. Debido a que las gestiones en los sistemas suelen ser muy específicas y es dificil plantear una solución general lo suficientemente abierta como para cubrir éstos requerimientos, la solución propuesta por el framework consiste en considerar ciertas situaciones frecuentes que se dan este tipo de sistemas:

i. Presentación de páginas de consultas.
ii. Presentación de datos en formato tabular.
iii. Presentación de datos en formato formulario.

Como consideraciones técnicas generales cabe indicar que las páginas resultantes para presentar y manejar la información estará dotada de soporte idiomático y de un mecanismo de validación de los datos basado en Struts.

De cara a la implementación de las pantallas, se considerarán las siguientes actividades:

i. Creación de una página jsp que se base en las plantillas creadas anteriormente y desarrolle el *tile* "body" presentando la información particular de la que trate la página.
ii. Conexión con un punto de menú, para q pueda ser accedida desde el sistema.
iii. Si es necesario, personalización de las posibles operaciones en función del tipo de pantalla que estemos tratando. La personalización no siempre sería necesaria, tan solo cuando esperemos un comportamiento distinto sobre las acciones ya implementadas por defecto.

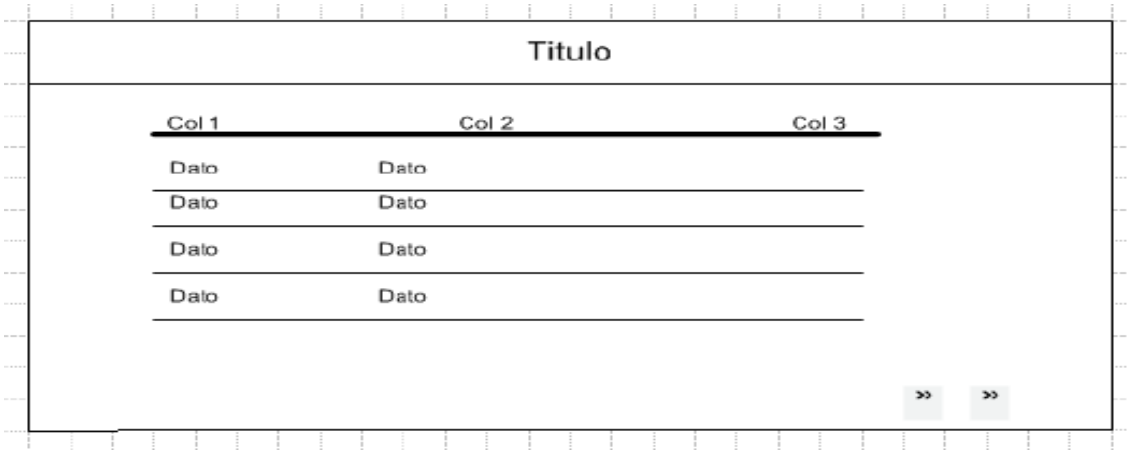
1.4.3.1 ***Utilizando las plantillas***

Las pantallas de gestión extenderán generalmente la plantilla llamada PlantillaGestion. A partir de la plantilla, deberíamos contar con parte de la estructura final de la página, en concreto con el logo, el título de la aplicación y los distintos menús.

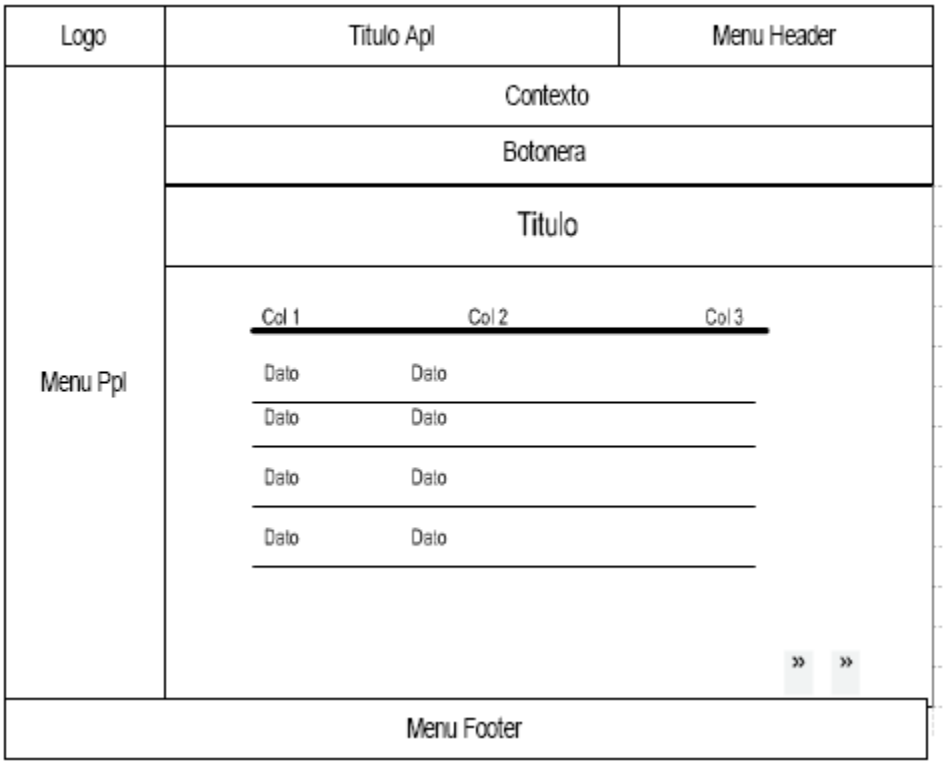
El *tile* de contexto no debería variar significativamente entre las distintas pantallas que se implementen. El *tile* nos permitirá disponer de un espacio para la presentación de los textos de las validaciones de Struts y para presentar información contextual como la página a la que hemos accedido.

De la misma manera, las opciones de la botonera deberían corresponderse al tipo de página que estamos desarrollando.

Queda por tanto implementar una página jsp por cada gestión que queramos implementar. Una página por ejemplo en modo tabular, podría ser la siguiente:



La página en cuestión debería sustituir el *tile* "body" disponible desde la plantilla, de tal forma que podamos obtener algo similar a:



1.4.3.2 ***Integración de la pantalla de gestión, el menú y el modelo***

Una vez disponemos de la página de gestión implementada, es necesario integrarla con el menú y el modelo.

En el punto 1.4.2.1. se ha desarrollado las características del componente menú y se ha explicado como podría ser invocada mediante el atributo link. En nuestro ejemplo, los links de los menús invocan a acciones struts. Esto nos permite disponer de un punto de acceso al modelo controlado, las clases Action de Struts. A través de esta serie de clases, implementaríamos la lógica de control de flujo requerida por el sistema, además de llevar a cabo una serie de validaciones sobre los datos. Éstas validaciones se corresponderían con validaciones simples sobre los formularios (obligatoriedad, tamaños de los campos, tipos de datos, etc). Las reglas de negocio de la pantalla, sería conveniente implementarlas en la capa del modelo.

Por último, plantearemos una solución genérica que nos proporcione una solución para resolver las operaciones que se habilitarán en la botonera. El gráfico 7

representa la estructura de clases que nos ofrecen éstas funcionalidades.

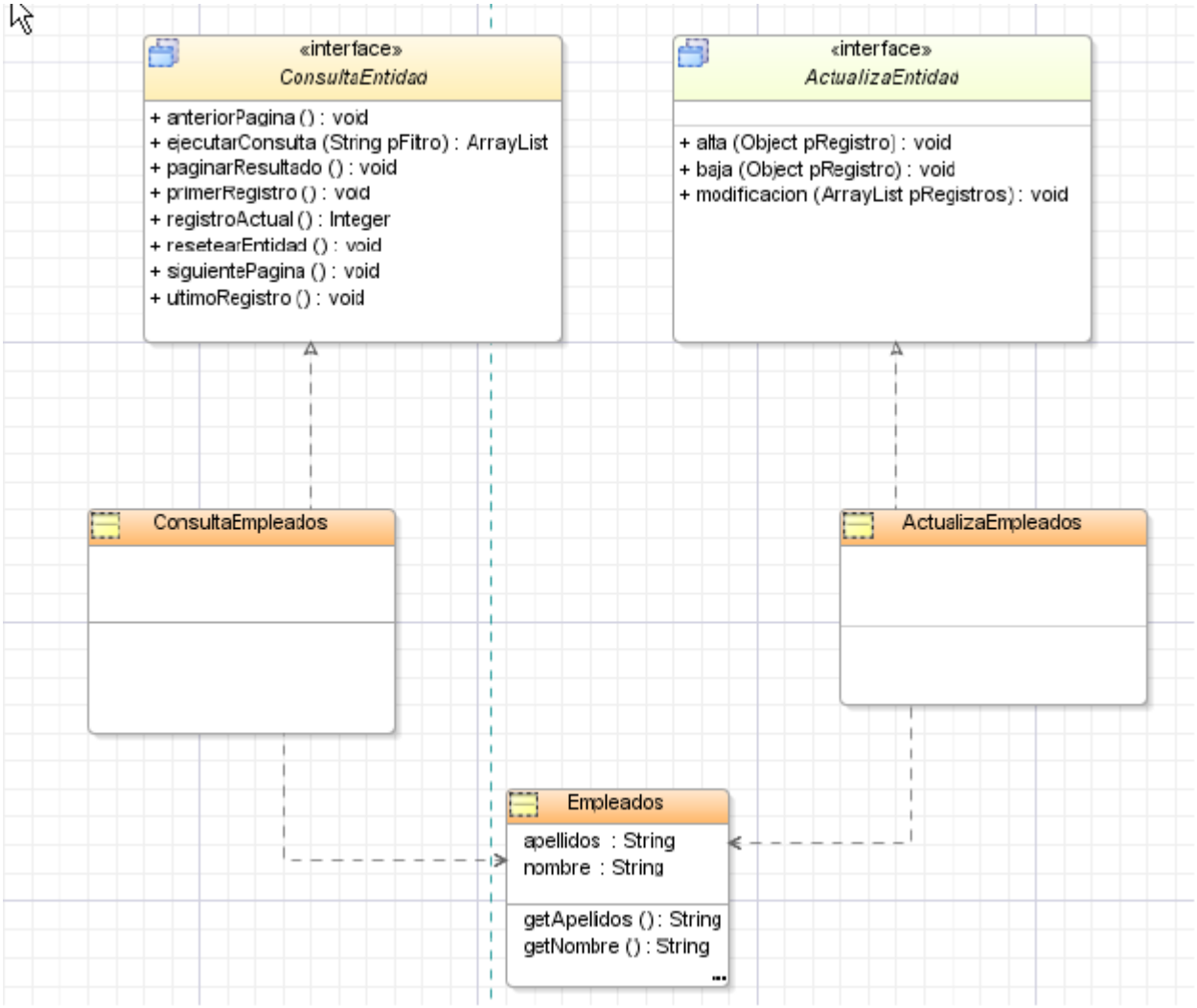


Gráfico 7: Operaciones sobre el modelo

Las operaciones que se implementarán en la botonera, deberán ser soportadas por cada entidad que necesitemos getionar. Para lograrlo, se plantean dos interfaces que deberán cumplir y desarrollarse en cada entidad del modelo que interactue con la vista:

i. Interfaz ConsultaEntidad

La interfaz incluye las siguientes operaciones:

- EjecutarConsulta. El método ejecutaría la consulta sobre la entidad en cuestión y obtendríamos un conjunto de registros como resultado.
- PaginarResultado. Nos permitiría paginar un resultado. El método nos permitiría acceder a los registros obtenidos a través de rangos predefinidos. Básicamente utilizaríamos un atributo para indicar en qué página de resultado estamos, más los métodos siguientePagina y anteriorPagina para desplazarnos entre ellas.
- SiguientePagina y anteriorPagina. Nos desplazaríamos entre un resultado paginado.
- RegistroActual. Indice que nos permite identificar un registro actual (*current record*). Debe sincronizarse con la capa de la vista.
- SiguienteRegistro y anteriorRegistro. Nos desplazaríamos en una posición respecto al registro actual hacia delante o hacia detrás.
- UltimoRegistro y primerRegistro. Nos permite acceder al primer registro del resultado o al último.

ii. Interfaz ActualizaEntidad

La interfaz incluye las siguientes operaciones:

- Alta. El método sería el responsable de dar de alta un nuevo registro en la entidad.
- Baja. El método sería el responsable de dar de baja un registro para la entidad.
- Modificacion. El método cambiaría los valores de los atributos de la entidad.

Las entidades del modelo que implementen estas interfaces, dispondrían al menos de los métodos indicados. A partir de éstos métodos, podríamos realizar las invocaciones genéricas desde la botonera, y el framework nos proporcionaría un comportamiento homogéneo para la realización de éstas operaciones.

1.4.4 **Core**

El componente Core estará formado por:

i. Librería de etiquetas. Nos permitan definir de forma sencilla en las páginas JSP's las funcionalidades, encapsulando su implementación y delegandola al Core.
ii. Clases de utilidades.
iii. Fichero de configuración

1.4.4.1 Ficheros de configuración

Como parte del componente Core, se ha incluido un fichero de propiedades en donde podremos configurar ciertos parámetros del sistema. En concreto, las propiedades actualmente configurables se refieren a los paths dentro de la aplicación web del logo que se utilizará y de los menús.

Para poder tratar adecuadamente el fichero, se ha creado una clase llamada edu.uoc.pfc.util.Propiedades. La clase realiza la lectura del fichero de propiedades y nos permite de forma sencilla recuperar los valores leidos mediante el método getPropiedad.

1.4.4.2 Tag Librarys

Dentro del paquete edu.uoc.pfc.tablibs implementaremos las clases descritas a continuación, que darán lugar a la librería de etiquetas tagsPfc.jar:

i. IncluirMenu.java: La clase implementará la lógica necesaria para obtener los menús desde las jsp's.
ii. IncluirTabular: La clase permitirá presentar una estructura tabular en una JSP para una entidad de negocio definida previamente.
iii. IncluirFormulario: La clase permitirá presentar una estructura formulario en una JSP para una entidad definida previamente.
iv. IncluirBusqueda: La clase permitiría presentar una estructura de búsqueda mediante campos definidos previamente.

1.4.4.3 Utilidades

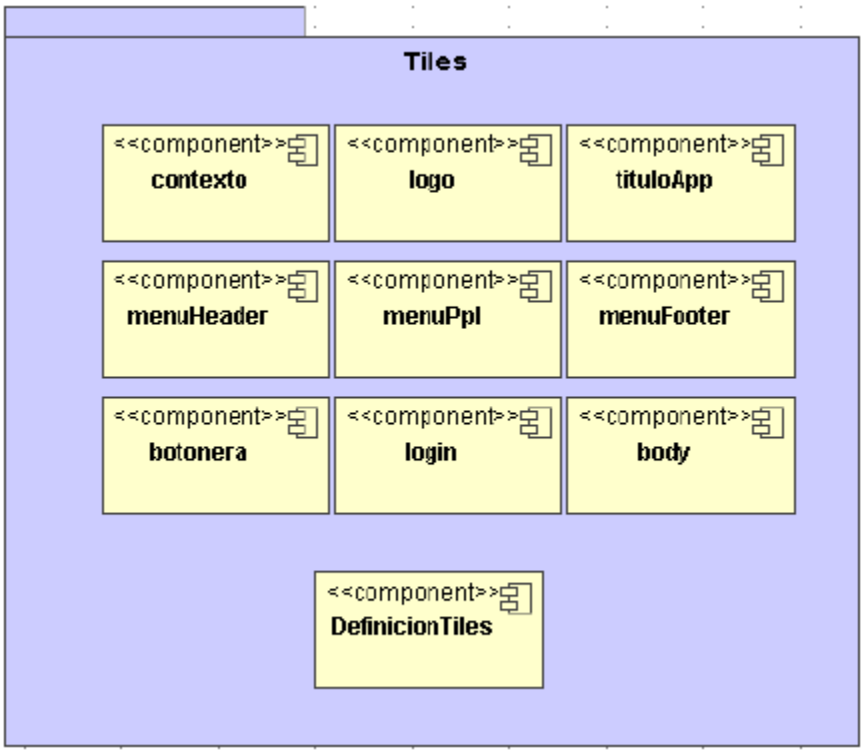
Dentro del paquete edu.uoc.pfc.util implementaremos las clases descritas a continuación, que darán lugar a la librería de utilidades utils.jar:

i. ProcesaXML.java: La clase sería la responsable de realizar las validaciones sobre la estructura XML de los menús y aplicar las reglas de transformación necesarias.
ii. Botonera.java: La clase sería la responsable de gestionar las operaciones posibles en las páginas y establecer la relación con el modelo.

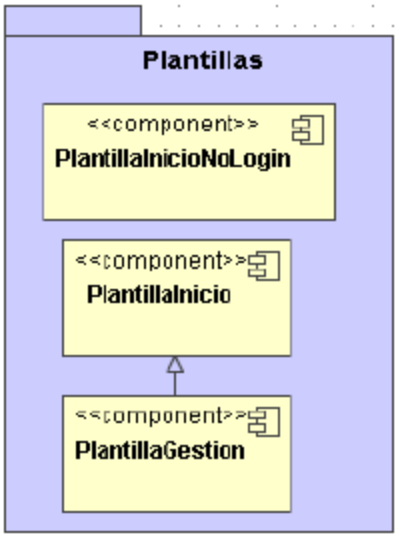
1.4.5 Diagrama de componentes

Debido al tamaño que ocupa el diagrama completo, se presentarán los diferentes componentes de forma independiente y al final se presentará un diagrama con las relaciones entre ellos. En cualquier caso el diagrama de componentes completo está a disposición del consultor y formará parte de la documentación final del proyecto.

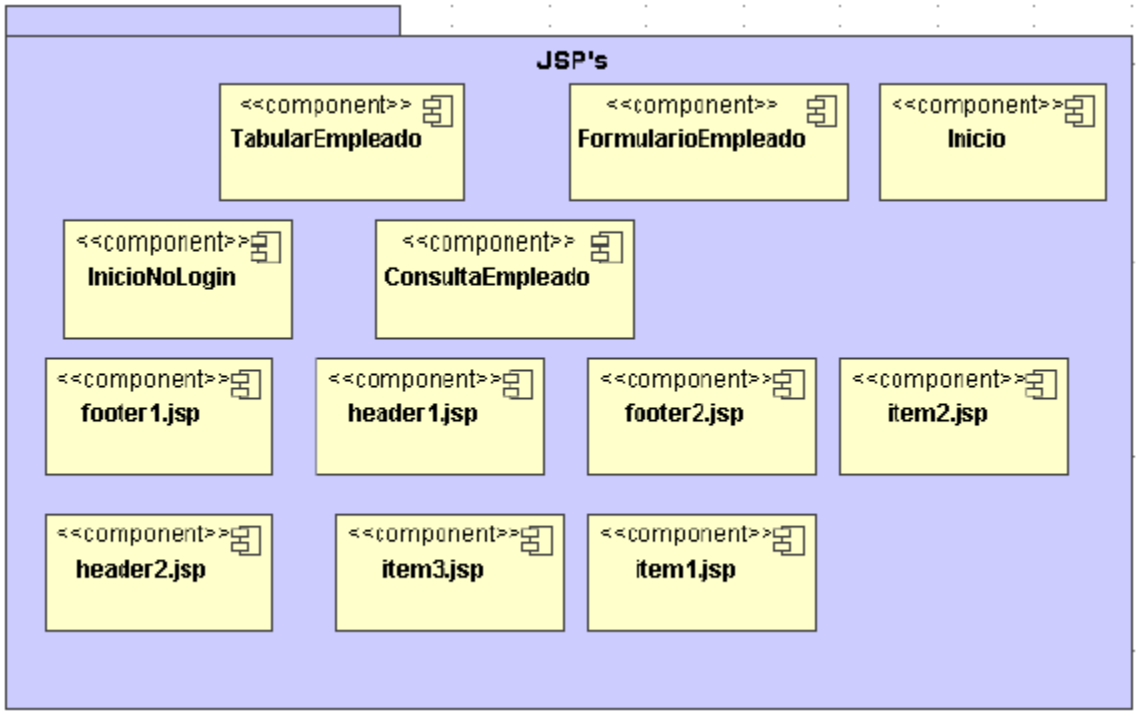
1.4.5.1 Tiles



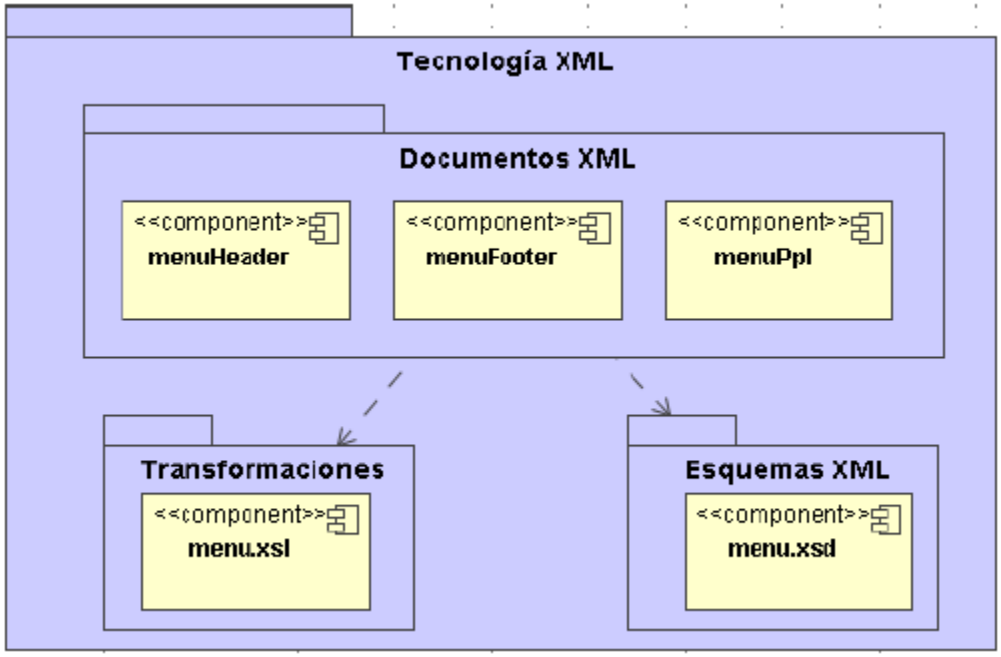
1.4.5.2 *Plantillas*



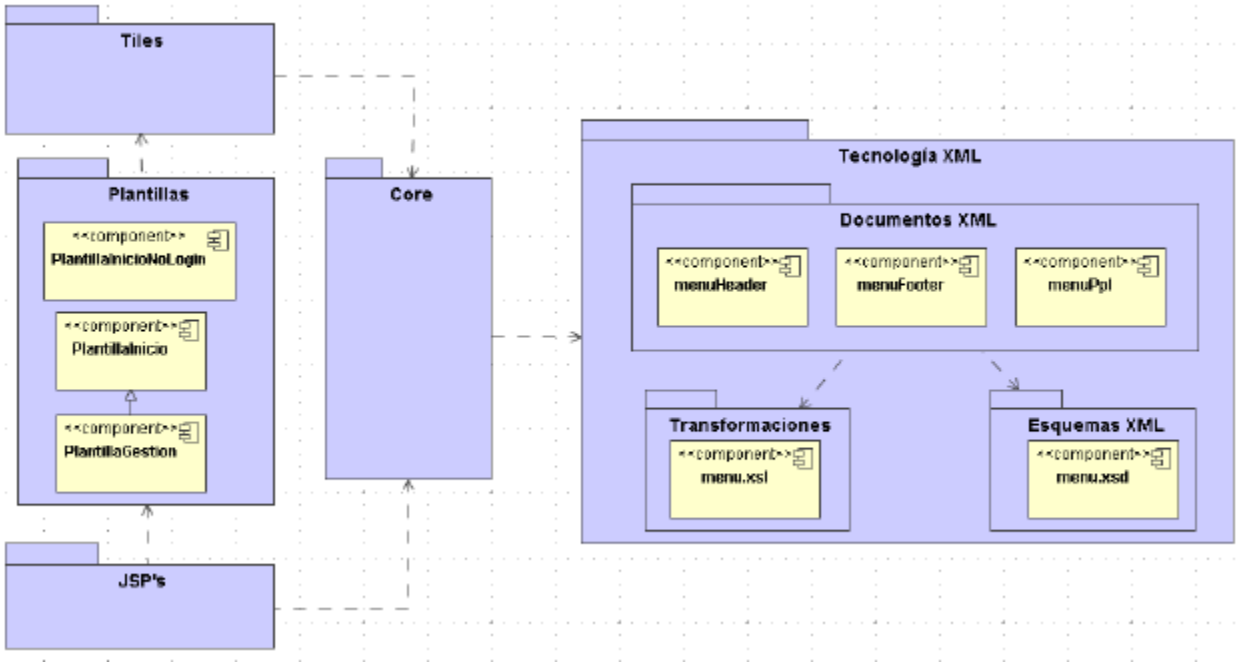
1.4.5.3 *Jsp's*



1.4.5.4 *Tecnología XML*



1.4.5.5 *Dependencias entre componentes*



1.5 Conclusiones

El proyecto de fin de carrera me ha brindado la posibilidad de plantear un proyecto en las primeras fases del ciclo de vida: Levantamiento de requisitos y el análisis y diseño. La construcción del producto, aunque se ha comenzado, no ha llegado a completarse por falta de tiempo.

Durante todo el curso, he podido profundizar no solo en los aspectos técnicos que conlleva la ejecución de un proyecto en términos de ingeniería del software, sino desde un punto de vista de la gestión del proyecto.

Así mismo, hemos podido constatar el importante papel que juegan durante el desarrollo de nuevas aplicaciones los patrones de diseño, simplificando enormemente la toma de decisiones al conocer de antemano soluciones de diseño óptimas ante problemas que sin duda se repiten frecuentemente.

Gracias a éste trabajo, he aumentado mi conocimientos sobre los frameworks existentes y las tecnologías y técnicas actuales utilizadas como soluciones de presentación en aplicaciones J2EE.

2 Glosario

Nombre	Descripción
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.
MVC	El MVC (Modelo-Vista-Controlador) es un patrón de diseño arquitectonico en el que el software se divide en diferentes capas con responsabilidades claramente definidas, obteniendo un muy bajo acoplamiento entre ellas. Las capas son la Vista (responsable de la presentación), el modelo (responsable de las reglas de negocio) y el controlador (responsable de la comunicación entre las dos capas anteriores).
J2EE	Conjunto de estándares que permiten la construcción de sistemas distribuidos basados en Java.
JSP	JavaServer Pages. Tecnología java que permite la generación de contenido web dinámico.
JSTL	Extensión de la tecnología JSP que nos permiten haciendo uso de etiquetas incorporar funcionalidades ya desarrolladas en nuestras páginas.
Tiles	Mecanismo de plantillas para la construcción de sistemas con componentes reutilizables basados en jsp's.
XML	Lenguaje de marcas que permiten la definición estructural de datos y una alta capacidad de abstracción de su representación.
XSLT	Lenguaje que nos permite aplicar transformaciones sobre documentos XML.
Esquemas XML	Componentes del mundo XML muy extendidos que permiten definir cuando un documento XML es válido.

3 Bibliografía

Para el desarrollo del proyecto, he recurrido a los siguientes recursos:

- Jakarta Struts for Dummies (2004), Wiley&Sons.
- Pro Jakarta Struts, Second Edition (2004) John Carnell y Rob Harrop.
- Design Patterns Java Workbook (2002), Steven John Metsker
- JSP Examples and Best Practices, de Andrew Patzer.
- Web Development with JavaServer Pages, de Duane K Fields, Mark A. Kolf y Shawn Byern.
- Xml Imprescindible, de Rusty Harold y Scott Means.
- http://www.w3.org/XML/
- Apache Struts Project
- http://struts.apache.org/1.x/struts-tiles/
- http://java.sun.com/products/jsp/jstl/
- http://java.sun.com/products/jsp/
- http://www.javaworld.com/javaworld/jw-09-2004/jw-0913-struts.html
- http://www.elrincondelprogramador.com/default.asp?pag=articulos/leer.asp&id=30
- http://www.cafeconleche.org/books/bible2/chapters/ch17.html
- http://www.javaworld.com/jw-03-2000/jw-0331-ssj-jspxml.html?page=4
- http://java.sun.com/products/jsp/tags/11/syntaxref11.fm14.html
- http://www.devx.com/assets/download/4510.pdf
- http://webdesign.about.com/od/htmltags/a/aa011000a.htm
- Apache Cocoon
- Velocity
- FreeMarker
- JavaServer Faces
- Apache Tapestry
- StrutsCx
- Comparacion Velocity versus Freemarker
- Comparación Frameworks Aplicaciones Web

4 Anexos

4.1 Aplicación de pruebas

Se incluye la aplicación utilizada para testear del framework en su estado actual. No ha sido posible construir el framework totalmente como estaba previsto. La implementación final adjunta a la memoria consta de los siguientes componentes:

i. Modelo.jar. Contiene un bean utilizado para la obtención de datos del modelo.
ii. edu.uoc.pfc.*. Contiene las clases Action y Form requeridas por Struts.
iii. edu.uoc.pfc.utils que continee las clases Propiedades, ProcesaXML y Tests.
iv. Ficheros de recursos: RecursosAplicacion.properties y Paths.properties.
v. Tiles (/jsp/layouts/*, /jsp/tiles/*, /WEB-INF/conf/tiles/definitionTiles.xml).
vi. Fichero descriptor de etiquetas (tagsPFC.tld).
vii. Ficheros XML y XSLT que nos permiten realizar la transformación para obtener los menús del sistema.
viii. Index.jsp.
ix. Fichero de configuración (struts-config.xml).
x. Definición del tag library.

Durante las pruebas, se creó un artefacto llamado mmontejano_pfc.war. Incluye el código fuente de los componentes Java implementados, bajo el directorio /src. Los desarrollos que han quedado pendientes son:

i. Implementación de la tag library de Core.
ii. Implementación de la botonera.
iii. Implementación de una página de consulta, tabular y formulario.
iv. Implementación de las clases requeridas por la entidad para soportar las operaciones de la botonera.
v. Integración de los componentes JSP's, Menús, Botonera y operaciones.
vi. Completar las implementaciones experimentadas hasta el momento (XML/XSLT, Tiles, Presentación menús mediante bloques DIVS y Autenticación)

Sería ideal poder empaquetar los componentes obtenidos en jars y poder referenciarlos directamente desde la aplicación que los utilice, para simplificar su uso e inclusión en los sistemas. De esta forma la forma entregable del framework serían una serie de jars que podrían quitarse o ponerse fácilmente en nuevos desarrollos. El problema es principalmente como resolver la referencia a componentes web (css, etc).

En el producto incluido junto con la memoria se han omitido varias librerías para disminuir el tamaño final del entregable. A continuación se indican qué librerías requiere el programa y donde pueden ser obtenidas:

i. Struts 1.3. Las librerías necesarias por la aplicación pueden ser obtenidas desde el siguiente enlace: http://ftp.udc.es/apache-dist/struts/library/struts-1.3.8-lib.zip. La instalación es simplemente copiar los ficheros jar en el directorio WEB-INF/lib.
ii. Servlet 2.4. Las librerías requeridas se encuentran disponibles en el servidor de aplicaciones Tomcat 5. No requiere ninguna instalación específica, ya que el servidor de aplicaciones hace uso del componente servlet-api.jar disponible en $TOMCAT_HOME/common/lib.