PFC

11 de enero

2016

Proyecto Final de Carrera – 2015 Semestre 2 – Oscar Escudero Sanchez

Adrián Chavero Ramos

Agradecimientos

A mi Farida por aguantarme fines de semana enteros en pijama y bata,

A mis padres y hermana por animarme a terminar la carrera,

A mis amigos por quitar mi estrés a base de cerveza,

Y a mi sobrino Liam por hacerme olvidar todo por un momento con su sonrisa.

1 Introducción PFC

1.1 Descripción

A grandes rasgos el PFC se basa en hacer el análisis, diseño e implementación de un framework sobre la tecnología JEE y un prototipo de aplicación que utilice este framework.

Un framework, traducido literalmente como marco de trabajo, es un conjunto de tecnologías que sirve para desarrollar y unir los diferentes componentes de un proyecto. Esta infraestructura digital está compuesta por programas, bibliotecas, lenguajes interpretados y herramientas.

Este framework se respaldará en la plataforma JEE. JEE es una plataforma para desarrollar software programado en Java. Este software se ejecutará en un servidor de aplicaciones que cumple el estándar JEE y que maneja transacciones, seguridad, escalabilidad y concurrencia. JEE cumple las especificaciones de API: JSP, Servlet, EJBs, RMI, JDBC, ServiciosWeb, y más.

En el proyecto que iremos desarrollando durante el semestre, prestaremos especial atención en utilizar patrones de diseño. Como se puede leer en los manuales de ingeniera de software, los patrones de diseño son formas de solucionar problemas recurrentes y que cuya solución es demostrada que es efectiva y reutilizable. La plataforma JEE nos encauza a utilizar la programación por capas y todo un catálogo de patrones.

El framework que crearemos tendrá como objetivo facilitar al programador la creación de formularios y listados, esto es la capa de presentación. Considerando el modelo de programación MVC (Modelo Vista Controlador), el framework elaborado en este PFC se centrará solo en las capas vista y controlador. El modelo de datos se dejará de lado. Desde la perspectiva del framework solo habrá JavaBeans con datos, se obviará como se manipulan estos datos.

Crearemos también un programa/prototipo que haga uso del framework, demostrando su funcionalidad. Será una pequeña aplicación web que gestione los miembros de una organización. Habrá listados, formularios de alta y modificación e informes.

1.2 Objetivos generales y específicos

El proyecto de final de carrera es un trabajo de síntesis de los conocimientos adquiridos durante toda la carrera.

Podemos destacar las siguientes asignaturas más relacionadas con este trabajo:

- Interacción Personas Ordenador

- Metodología de Gestión de Proyectos Informáticos
- Procesos de Ingeniería de Programas
- Ingeniera de Programas y Componentes de Sistemas Distribuidos
- Arquitectura de Sistemas Distribuidos
- Bases de Datos

Este trabajo nos brinda la oportunidad de consolidar todos los conocimientos teóricos de la ingeniería aplicándolos a la práctica.

Durante el PFC tendremos que entender toda la plataforma JEE, y conocer en profundidad ciertas partes.

Habrá también una parte importante de investigación. Antes de hacer un framework es indispensable conocer todos los frameworks existentes para no reinventar la rueda, partir sobre seguro o bien decidir realmente innovar.

El PFC nos pone como ingenieros informáticos en la mayoría de roles posibles. Seremos a lo largo de este semestre: gestores, diseñadores, arquitectos, programadores y probadores de software.

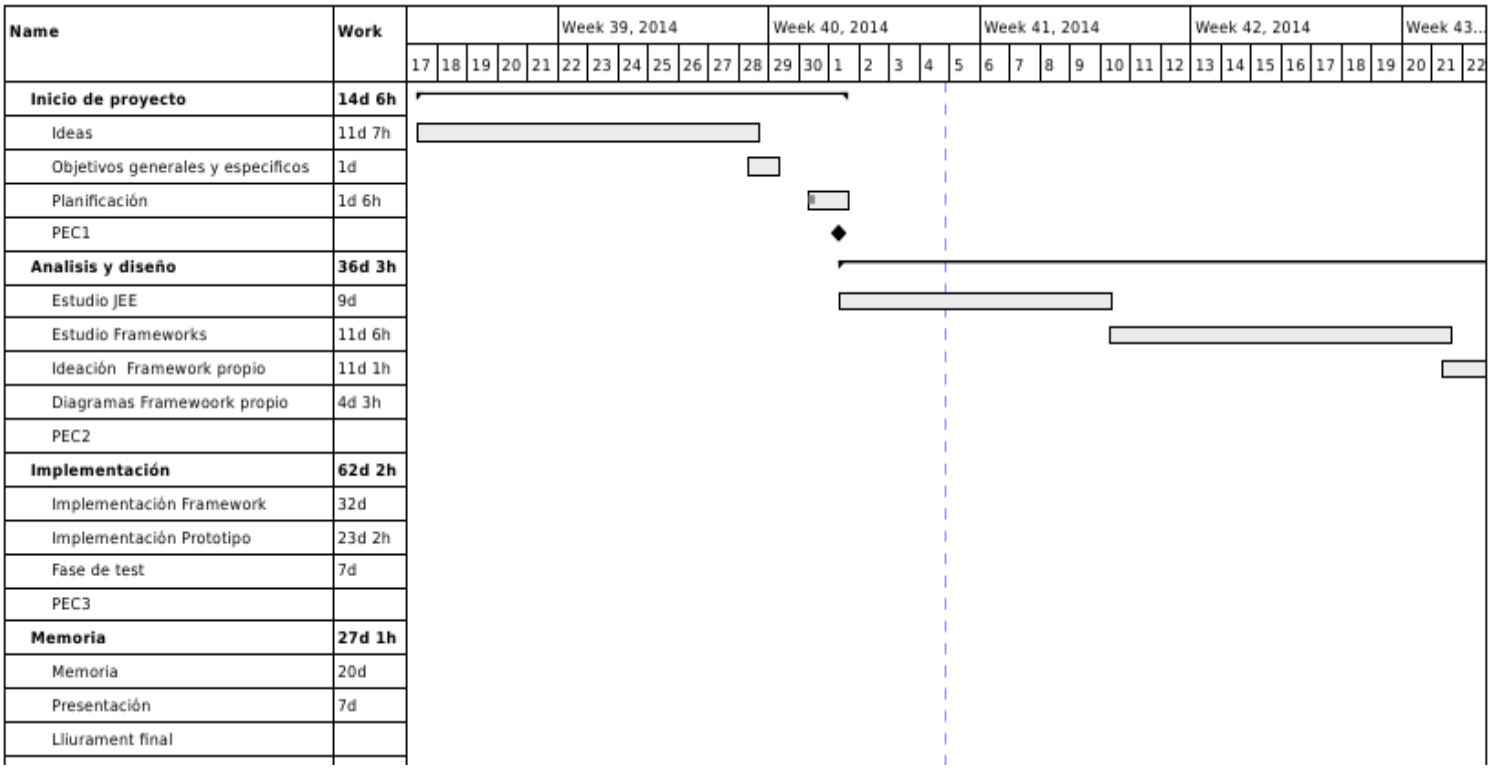
1.3 Planificación

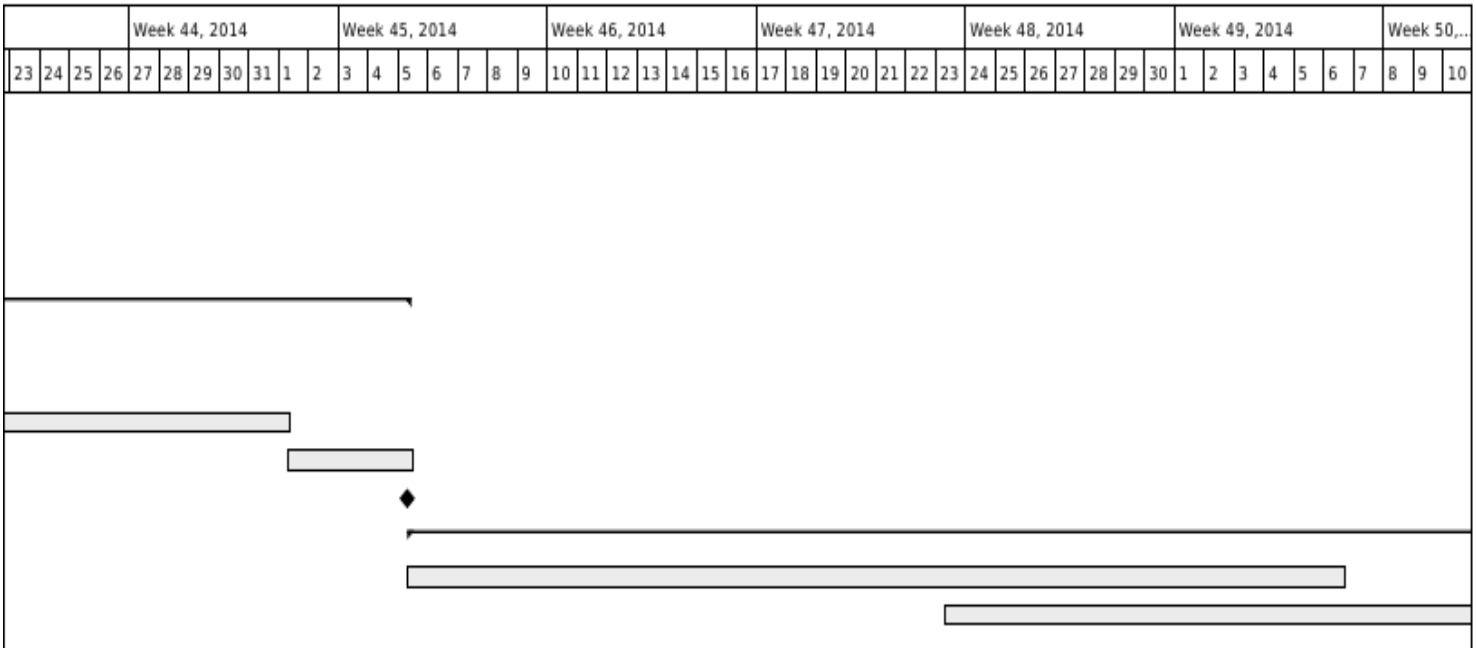
Al inicio de un proyecto hay dos cosas muy importantes: la planificación y la definición de requisitos de usuario. Los requisitos de usuario los terminaremos de definir con el consultor en el momento de la evaluación.

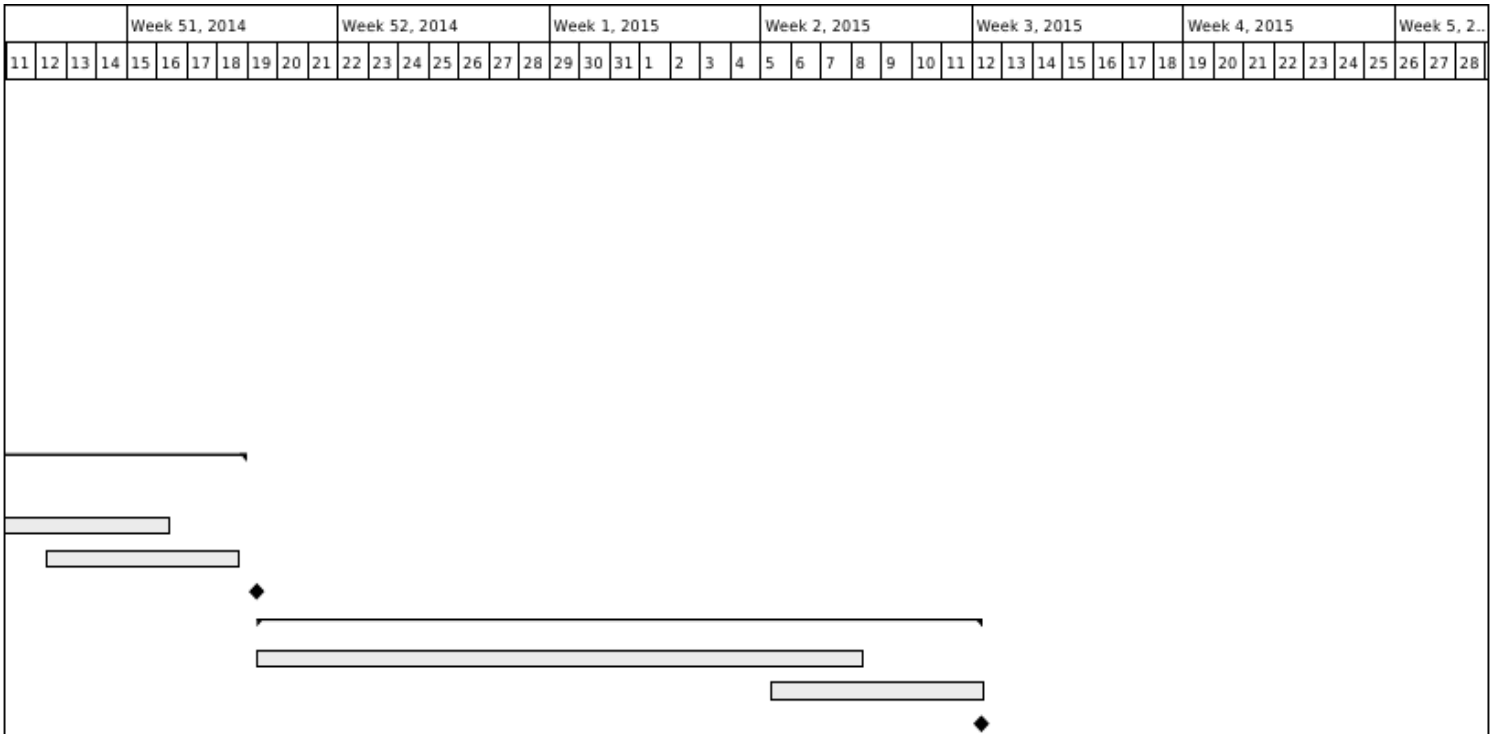
La planificación queda definida por las metas de las PEC. Cuyas fechas son:

- 30/09/2015 PEC 1
- 04/11/2015 PEC 2
- 18/12/2015 PEC 3
- 11/01/2015 Entrega final

WBS	Name	Start	Finish	Work	Duration	Slack	Cost	Assigned to	% Complete
1	**Inicio de proyecto**	**Sep 17**	**Oct 1**	**14d 6h**	**14d 6h**	**102d 2h**	**0**		0
1.1	Ideas	Sep 17	Sep 28	11d 7h	11d 7h	105d 1h	0		0
1.2	Objetivos generales y especificos	Sep 28	Sep 29	1d	1d	105d	0		0
1.3	Planificación	Sep 30	Oct 1	1d 6h	1d 6h	102d 2h	0		16
1.4	PEC1	Oct 1	Oct 1	N/A	N/A	103d	0		0
2	**Analisis y diseño**	**Oct 1**	**Nov 5**	**36d 3h**	**35d 3h**	**67d 5h**	**0**		0
2.1	Estudio JEE	Oct 1	Oct 10	9d	9d	93d 7h	0		0
2.2	Estudio Frameworks	Oct 10	Oct 21	11d 6h	11d 6h	82d 2h	0		0
2.3	Ideación Framework propio	Oct 21	Nov 1	11d 1h	11d 1h	71d 7h	0		0
2.4	Diagramas Framewoork propio	Nov 1	Nov 5	4d 3h	4d 3h	67d 5h	0		0
2.5	PEC2	Nov 5	Nov 5	N/A	N/A	68d	0		0
3	**Implementación**	**Nov 5**	**Dec 19**	**62d 2h**	**44d**	**24d**	**0**		0
3.1	Implementación Framework	Nov 5	Dec 6	32d	32d	36d	0		0
3.2	Implementación Prototipo	Nov 23	Dec 16	23d 2h	23d 2h	26d 5h	0		0
3.3	Fase de test	Dec 12	Dec 18	7d	7d	24d	0		0
3.4	PEC3	Dec 19	Dec 19	N/A	N/A	24d	0		0
4	**Memoria**	**Dec 19**	**Jan 12**	**27d 1h**	**24d**		**0**		0
4.1	Memoria	Dec 19	Jan 8	20d	20d	4d	0		0
4.2	Presentación	Jan 5	Jan 12	7d	7d		0		0
4.3	Lliurament final	Jan 12	Jan 12	N/A	N/A	30min	0		0




Week 44, 2014
Week 45, 2014
Week 46, 2014
Week 47, 2014
Week 48, 2014
Week 49, 2014
Week 50,...
23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10


Week 51, 2014
Week 52, 2014
Week 1, 2015
Week 2, 2015
Week 3, 2015
Week 4, 2015
Week 5, 2..
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

2 Investigación y estudio de Frameworks de la capa presentación

Este apartado contiene un resumen de la investigación de los diferentes frameworks existentes basados en la plataforma JEE. En la investigación prestaremos especial atención en analizar los diferentes patrones de diseño que se pueden utilizar al crear un Framework. De una visión general pasaremos a un análisis en profundidad de dos de los Frameworks más usados en la capa de presentación: Struts2 y Spring.

Un framework, traducido al castellano por marco de trabajo, es un conjunto de tecnologías, conceptos y prácticas para hacer frente a un tipo común de problema. Los frameworks proporcionan una estructura que facilita a los desarrolladores la creación de programas, reduciendo por un lado los costes, como el número de líneas de código, y por otro forzando una estructura que hace el software más robusto, flexible, mantenible, escalable y seguro. El software obtenido es de calidad y sigue los estándares actuales de desarrollo.

La estructura del Framework queda determinada en su mayor parte por los denominados patrones de diseño. Estos son soluciones para problemas típicos y recurrentes que se pueden encontrar a la hora de desarrollar una aplicación.

Veremos que la plataforma JEE, usada en este proyecto, tiene su propio catalogo de patrones de diseño.

2.1 Plataforma de desarrollo JEE

JEE es una plataforma de programación en lenguaje Java, para el desarrollo de aplicaciones empresariales distribuidas.

La plataforma JEE tiene una arquitectura distribuida por capas. La lógica de la aplicación queda dividida en componentes de acuerdo a su funcionalidad. Los diferentes componentes pueden ser instalados en maquinas distintas dependiendo de la capa en la que pertenece.

Una arquitectura típica de una aplicación JEE tiene una capa cliente, una capa web, una capa de negocio y una capa EIS. La maquina cliente ejecuta la capa cliente, en forma de aplicación cliente o pagina web en un navegador web. El servidor JEE despliega la capa Web y la capa de negocio. La capa EIS (Enterprise Information Systems) contiene el servidor de base de datos.

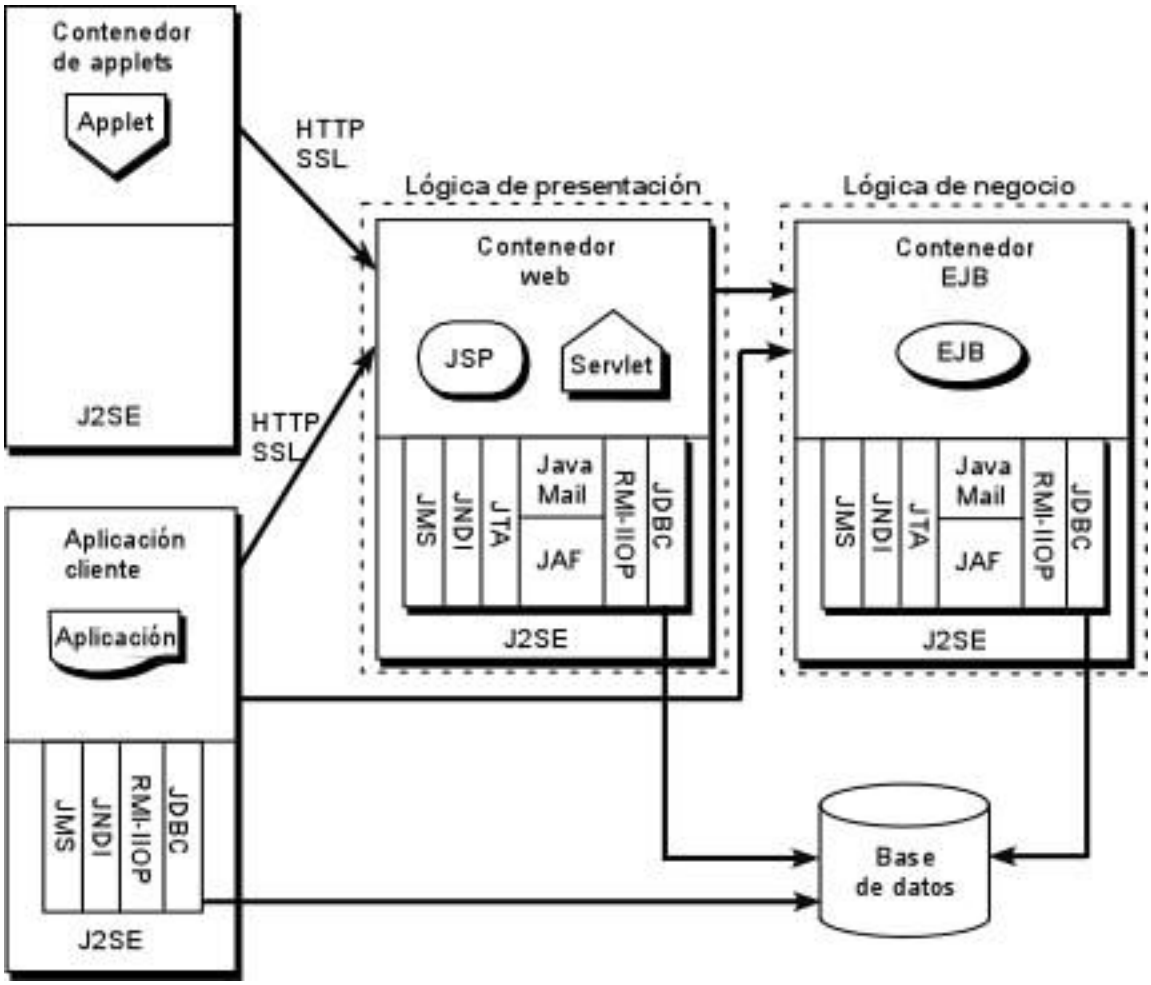
JEE por otro lado es un conjunto de especificaciones y tecnologías. Las siguientes son las más destacadas:

- Java Servlet, API para dar servicio a peticiones HTTP
- JavaServer Page (JSP), que son paginas HTML con código java incrustado. Internamente son traducidas a Servlets en el contenedor de aplicaciones JEE.

- JavaServer Pages Tag Library
- Java Message Service (JMS), API para crear, enviar, recibir y leer mensajes.
- Java Transaction (JTA), API estándar para gestión de transacciones
- JavaMail, API para gestionar mails
- XML, estándar para archivos descriptores
- JPA JDBC, conectores de bases de datos
- Java Naming and Directory Interface (JNDI), servicios de directorio para descubrir y buscar objetos
- Java Authentication and Authorization Service (JAAS)

Las aplicaciones JEE se pueden desplegar en un servidor de aplicaciones o en un contenedor web que cumpla con el estándar JEE.

Los servidores de aplicación son una implementación de la especificación JEE, y por lo tanto existen varios, creados por diferentes compañías. Todos comparten el grueso de las especificaciones JEE, pero se diferencian en funcionalidad y en la implementación. Los servidores de aplicaciones más utilizados en la actualidad son: Glassfish, JBoss, IBM WebSphere y TomEE.



Por otro lado, están los contenedores Web (a no confundir con los servidores web que solo atienden peticiones HTTP). Estos implementan un subconjunto de las funcionalidades ofrecidas por los servidores de aplicaciones. Los contenedores web ejecutan código java en forma de Servlets y JSPs para cada petición HTTP que hace el cliente. Tomcat es el más utilizado en la actualidad.

El Framework a desarrollar en este PFC se centra en la capa de presentación, y se basará principalmente en las tecnologías JSP, Servlet y Taglibs. Tomcat 8 será el contenedor web escogido para desplegar las aplicaciones que usen nuestro Framework.

2.2 Patrones de diseño

2.2.1 Descripción

Los patrones describen, con algún nivel de abstracción, una solución experta a un problema que se repite en situaciones similares. Esta solución experta es por lo tanto una solución probada y beneficiosa para estos problemas recurrentes. El conjunto de patrones seleccionados y usados en un Framework se interrelacionan creando una estructura abstracta, que va mas allá del código fuente (puede llegar a ser de cualquier lenguaje de programación).

A continuación analizaremos patrones, en especial los más utilizados en la plataforma JEE. Cuando se describe un patrón de diseño hay que hacerlo de acuerdo a sus principales características: contexto, problema y solución.

Antes de adentrarnos en el análisis, debemos introducir el patrón de arquitectura MVC (Modelo Vista Controlador).

2.2.2 MVC

MVC es un patrón que separa el código en tres capas diferentes, la capa Modelo, la capa Vistas y la capa Controladores. Cada capa tiene unas responsabilidades diferentes y está desacoplada de las demás.

Capa Modelo:

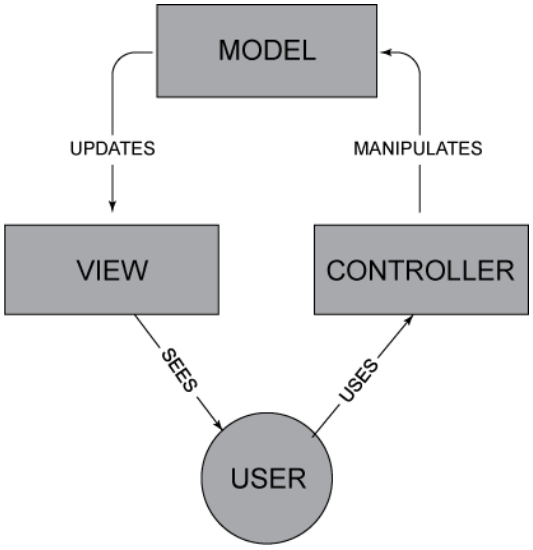
En esta capa es donde se realiza el trabajo con los datos, ya sea para acceder a la información o actualizarla. Se puede implementar de numerosas formas esta capa, aunque la mayoría de veces encontraremos aquí servidores de bases de datos con su configuración, nadie nos impide tener los datos en formato de archivo de texto plano. Es habitual también que haya unos bloques de código denominados Objetos de Acceso a Datos (DAO Data Access Objects) que son la interfaz para acceder a los modelos desde otras capas del MVC.

Capa Controladores:

Esta capa contiene el código y la estructura necesaria para gestionar las acciones solicitadas en los diferentes puntos de la aplicación, como visualizar un elemento, realizar una tarea, buscar información… Se puede ver como un enlace entre las vistas y la lógica de negocio.

Capa Vistas:

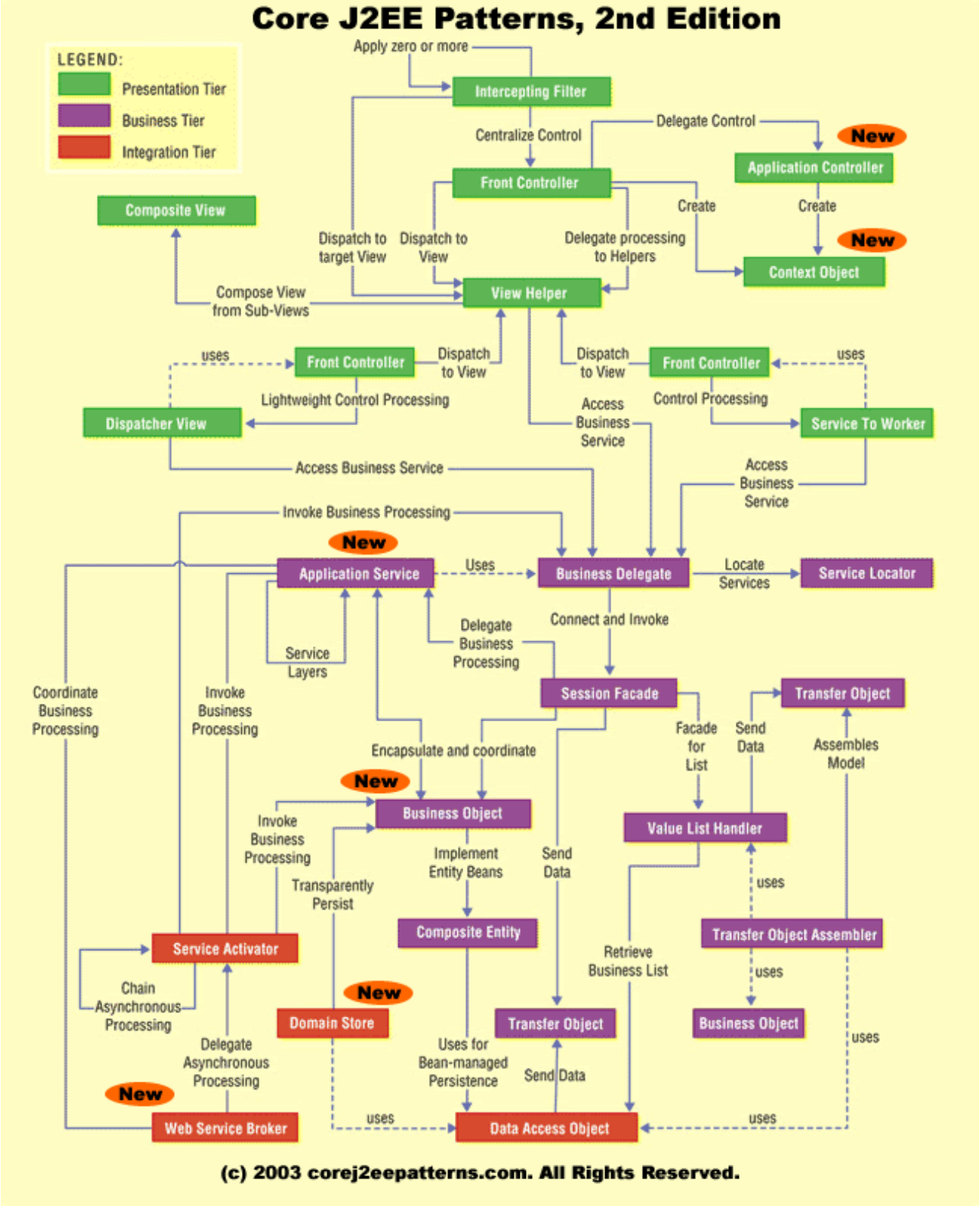
En esta capa se implementa la interfaz de usuario. Contiene todo lo necesario para que el usuario interactúe con la aplicación. La vista trabaja con datos pero no realiza operaciones con ellos, solo los presenta. Es la capa de Modelo quien lee, computa y modifica los datos.



MVC design pattern - www.ibm.com

2.2.3 Catalogo de patrones de diseño en JEE

A lo largo del tiempo se han realizado muchos proyectos JEE en Sun Java Center y para cada problema recurrente de los proyectos se han anotado las mejores soluciones. Comparando las soluciones y poniendo en común sus características, se ha elaborado el catalogo de patrones de diseño en JEE. Este catalogo podría cambiar en los próximos años, con nuevas soluciones abstractas, o reemplazando unas por otras.


Core J2EE Patterns, 2nd Edition
LEGEND:
Presentation Tier
Business Tier
Integration Tier
Apply zero or more
Intercepting Filter
Centralize Control
Delegate Control
New
Application Controller
Front Controller
Composite View
Create
Create
New
Dispatch to target View
Dispatch to View
Delegate processing to Helpers
Context Object
Compose View from Sub-Views
View Helper
uses
Front Controller
Dispatch to View
Dispatch to View
Front Controller
uses
Lightweight Control Processing
Control Processing
Dispatcher View
Access Business Service
Service To Worker
Access Business Service
Access Business Service
Invoke Business Processing
New
Application Service
Uses
Business Delegate
Locate Services
Service Locator
Service Layers
Delegate Business Processing
Connect and Invoke
Coordinate Business Processing
Invoke Business Processing
Session Facade
Transfer Object
Facade for List
Send Data
Assembles Model
Encapsulate and coordinate
New
Business Object
Value List Handler
Invoke Business Processing
Implement Entity Beans
Send Data
Transparently Persist
uses
Composite Entity
Transfer Object Assembler
Service Activator
Retrieve Business List
Chain Asynchronous Processing
uses
New
Domain Store
Transfer Object
Business Object
uses
Delegate Asynchronous Processing
Uses for Bean-managed Persistence
Send Data
New
Web Service Broker
uses
Data Access Object
uses
(c) 2003 corej2eepatterns.com. All Rights Reserved.

Al describir los diferentes patrones de diseño del catalogo haremos a veces mención a información específica del contenedor web escogido para nuestro Framework, Apache Tomcat 8.

2.2.3.1 Intercepting filter

2.2.3.1.1 Descripción

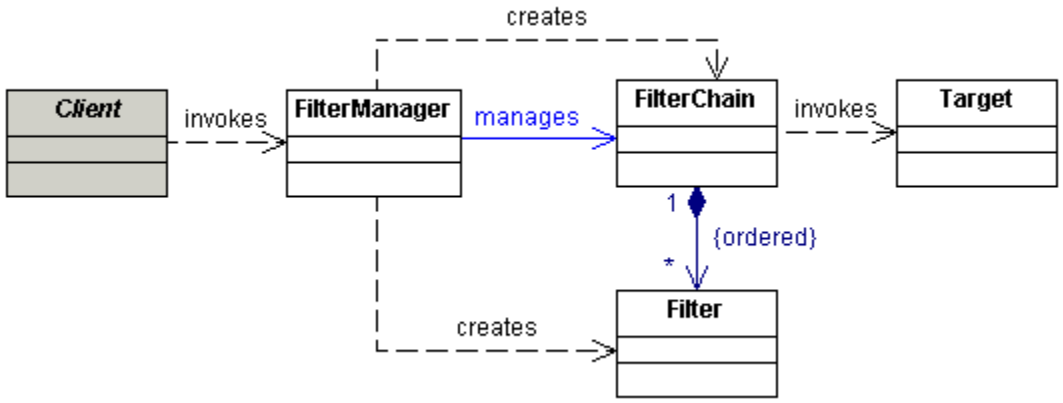
Este patrón de diseño surge de la necesidad de ejecutar bloques de código para cada petición del cliente. Las necesidades más habituales son comprobar la sesión del cliente, si está autorizado para acceder al recurso solicitado, determinar la codificación de datos por parte del cliente, entre otras. Algunas peticiones requieren un pre-procesamiento mas especifico como validar los campos del formulario enviado. Este patrón evita copiar y pegar todos estos bloques de código en cada Servlet/JSP. En el contenedor web se puede configurar una cadena de filtros que se ejecutan antes de la petición original.

En el caso de Tomcat podemos hacer esto en el archivo web.xml . Veamos por ejemplo

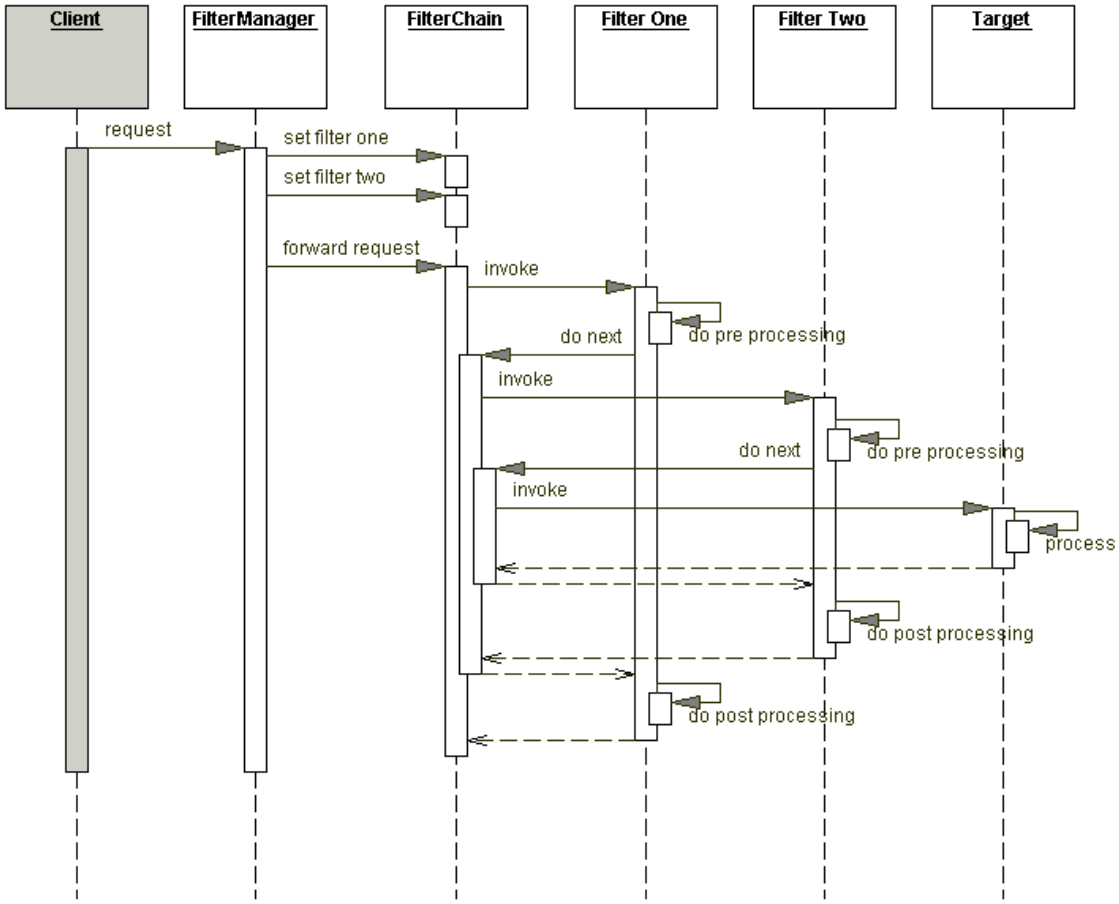
```
<filter>
<filter-name>MyFilter</filter-name>
<filter-class>com.uoc.MyFilter</filter-class>
<init-param>
<param-name>my-param</param-name>
<param-value>my-param-value</param-value>
</init-param>
</filter>
<filter-mapping>
<filter-name>MyFilter</filter-name>
<url-pattern>*Formularios1.jsp</url-pattern>
</filter-mapping>
```

La configuración anterior hará que el servidor ejecute el método `doFilter` de la clase `com.uoc.MyFilter` antes de acceder a los JSPs que terminan por Formularios1.jsp

2.2.3.1.2 Diagrama de clases



2.2.3.1.3 Diagrama de secuencia

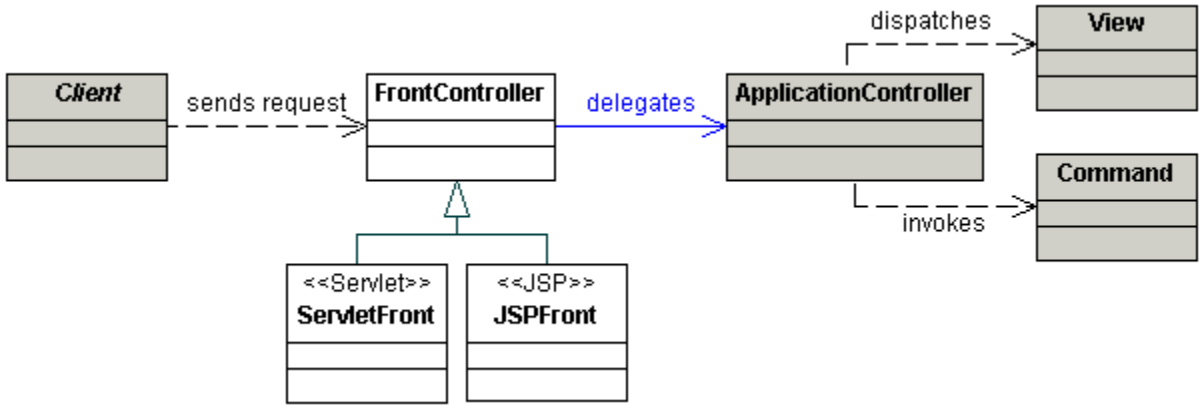


2.2.3.2 Front Controller

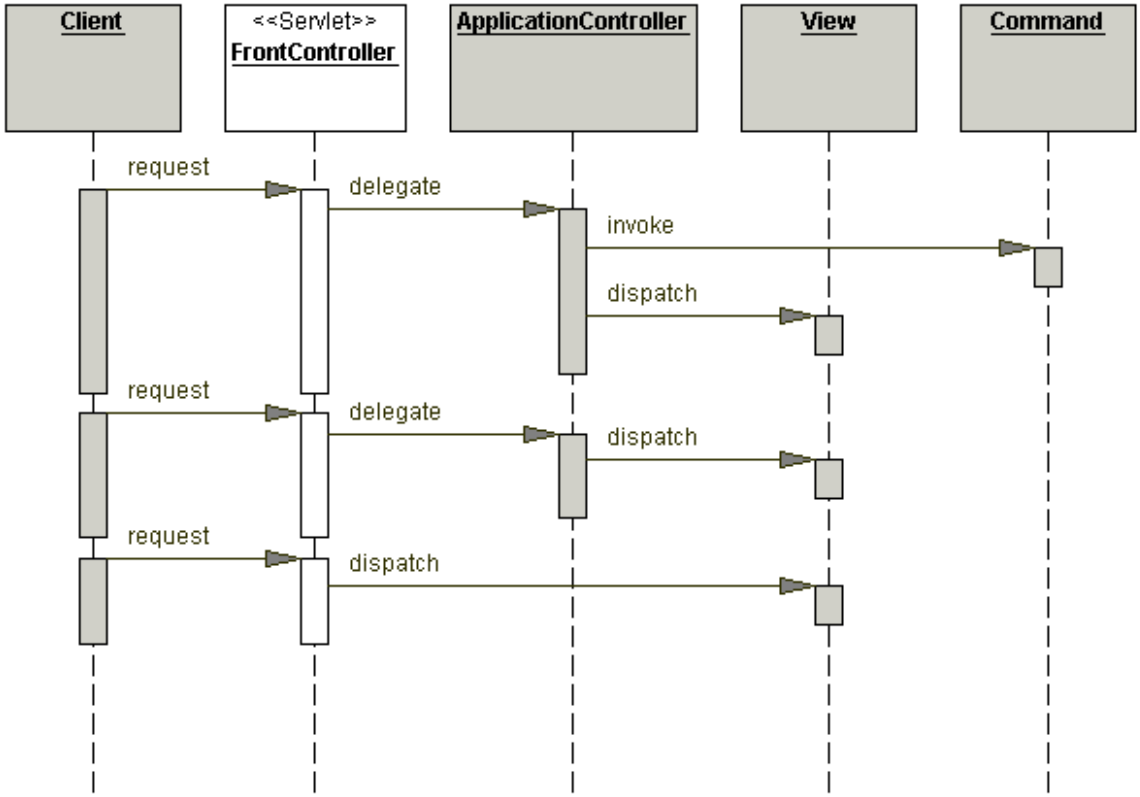
2.2.3.2.1 Descripción

Utilizar este patrón de diseño implica que el cliente interactuará con la aplicación pasando siempre por el mismo punto de control. Este único punto de acceso facilita la gestión de seguridad de acceso a los recursos, el manejo de errores y tracking de la interacción del usuario con la aplicación. El patrón evita por otra parte la duplicación de código y hace más fácil mantener la aplicación ya que los cambios no se tienen que realizar en números lugares.

2.2.3.2.2 Diagrama de clases



2.2.3.2.3 Diagrama de secuencia

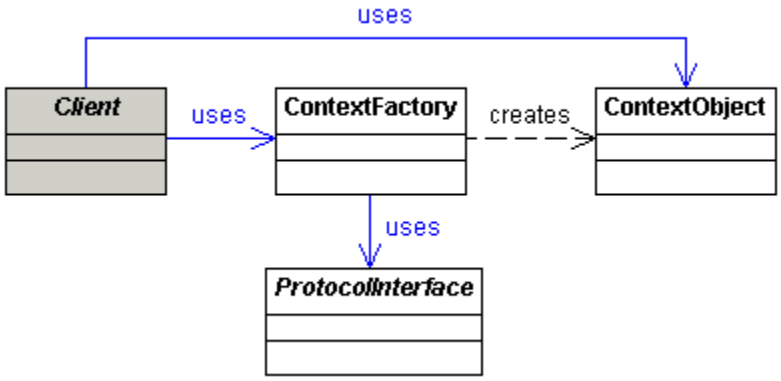


2.2.3.3 Context Object

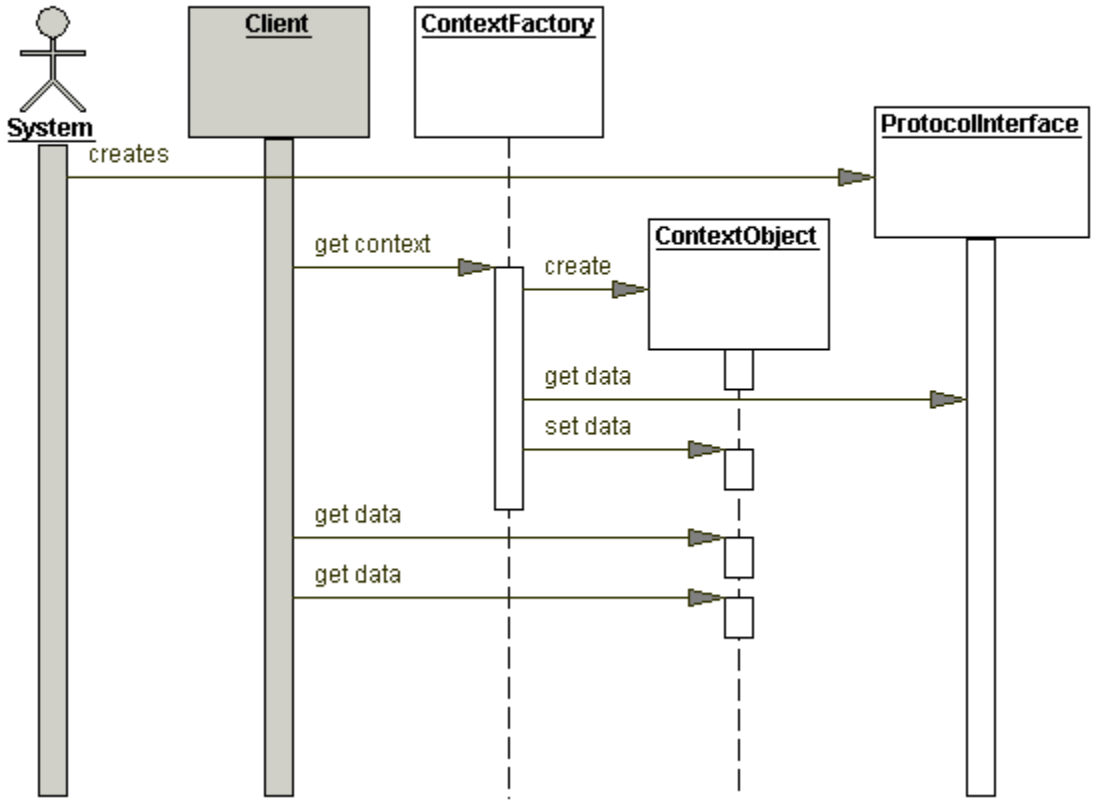
2.2.3.3.1 Descripción

Context Object es un patrón bastante simple que consiste en encapsular la información en un objeto independiente del protocolo y compartirlo en diferentes puntos (o componentes) de la aplicación.

2.2.3.3.2 Diagrama de clases



2.2.3.3.3 Diagrama de secuencia

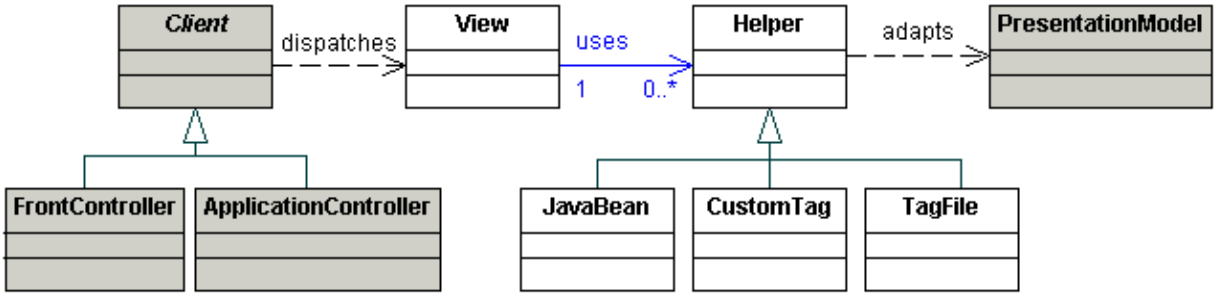


2.2.3.4 View helper

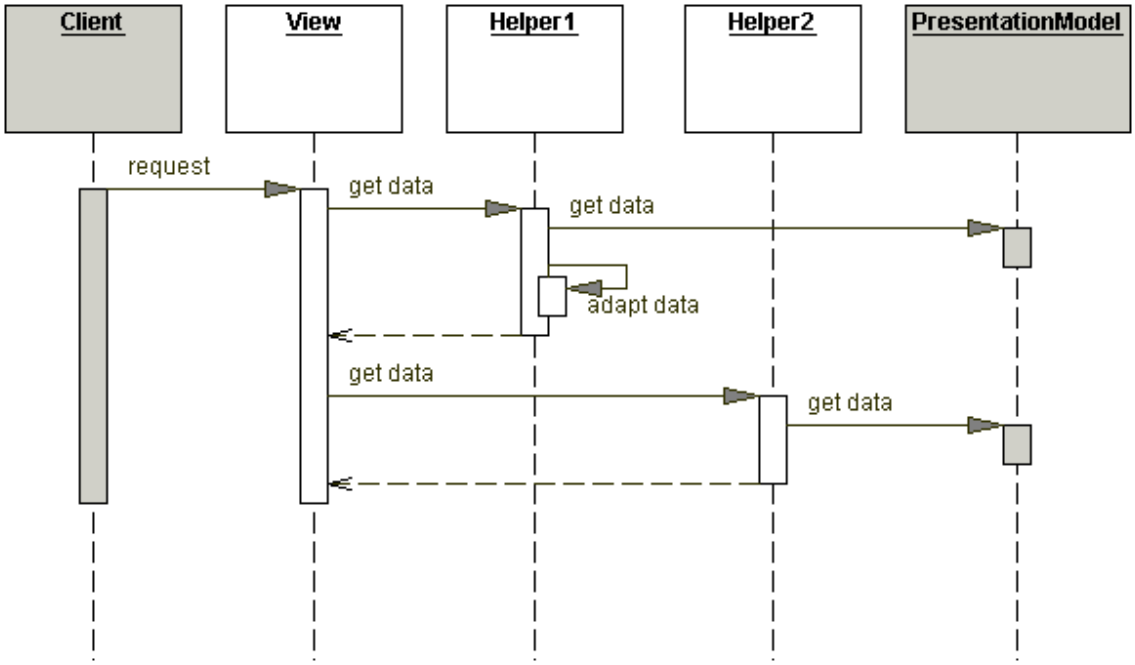
2.2.3.4.1 Descripción

Las vistas contienen el código que muestra la información final al usuario. Estas pueden delegar sus responsabilidades de procesamiento a unas clases de ayuda. Estas son View Helper y almacenan el modelo de datos intermedio de la vista, son adaptadores de los datos de negocio. Los helpers se implementan como JavaBeans o como CustomTag (o Tagfile). Un ejemplo de proceso relacionado con la lógica de formateo de un View Helper podría generar una tabla HTML con datos derivados del modelo.

2.2.3.4.2 Diagrama de clases



2.2.3.4.3 Diagrama de secuencia



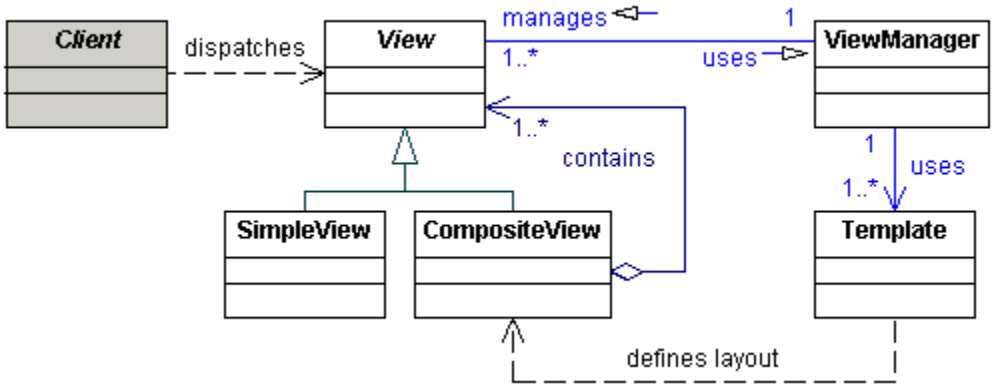
2.2.3.5 Composite View

2.2.3.5.1 Descripción

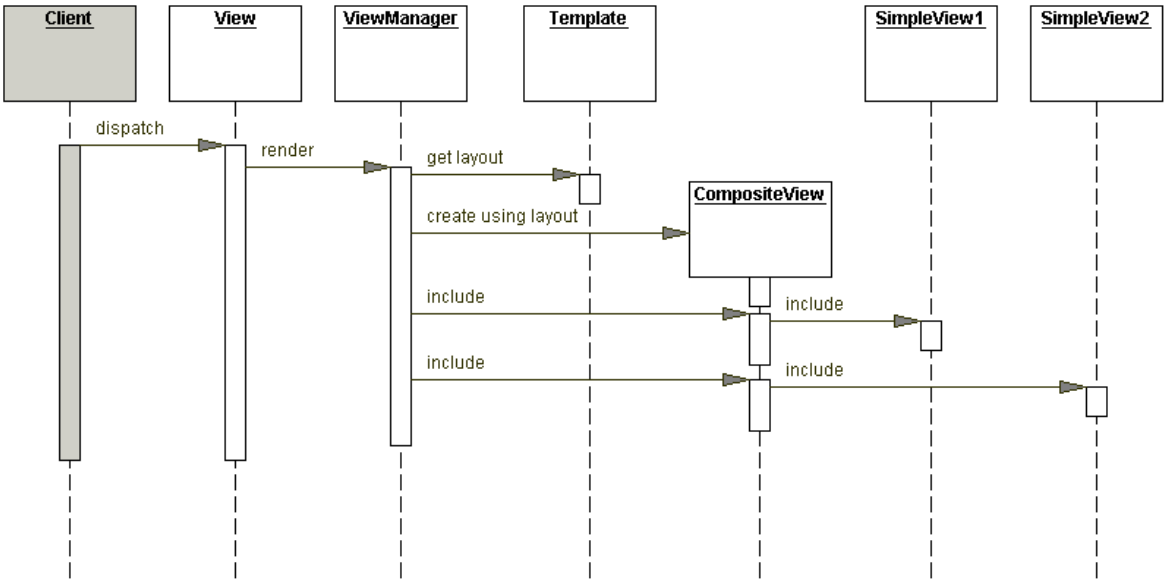
Este patrón lleva a programar modularmente, creando porciones atómicas de una vista y usándolas en vista compuestas. Así tendremos vistas más fáciles de mantener y reutilizaremos código.

Se puede implementar los módulos con JavaBean View Management aunque lo más normal es que se usen pequeños JSPs incluidos en la vista compuesta gracias al tag <jsp:include>.

2.2.3.5.2 Diagrama de clases



2.2.3.5.3 Diagrama de secuencia

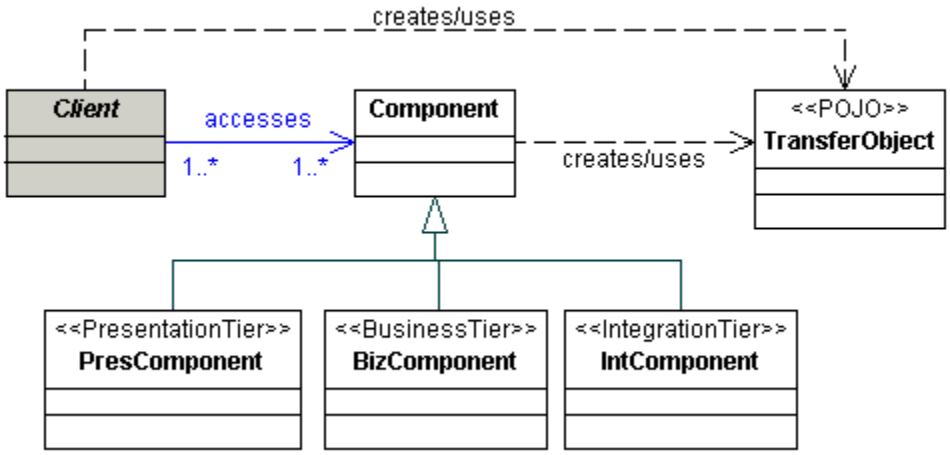


2.2.3.6 Transfer object

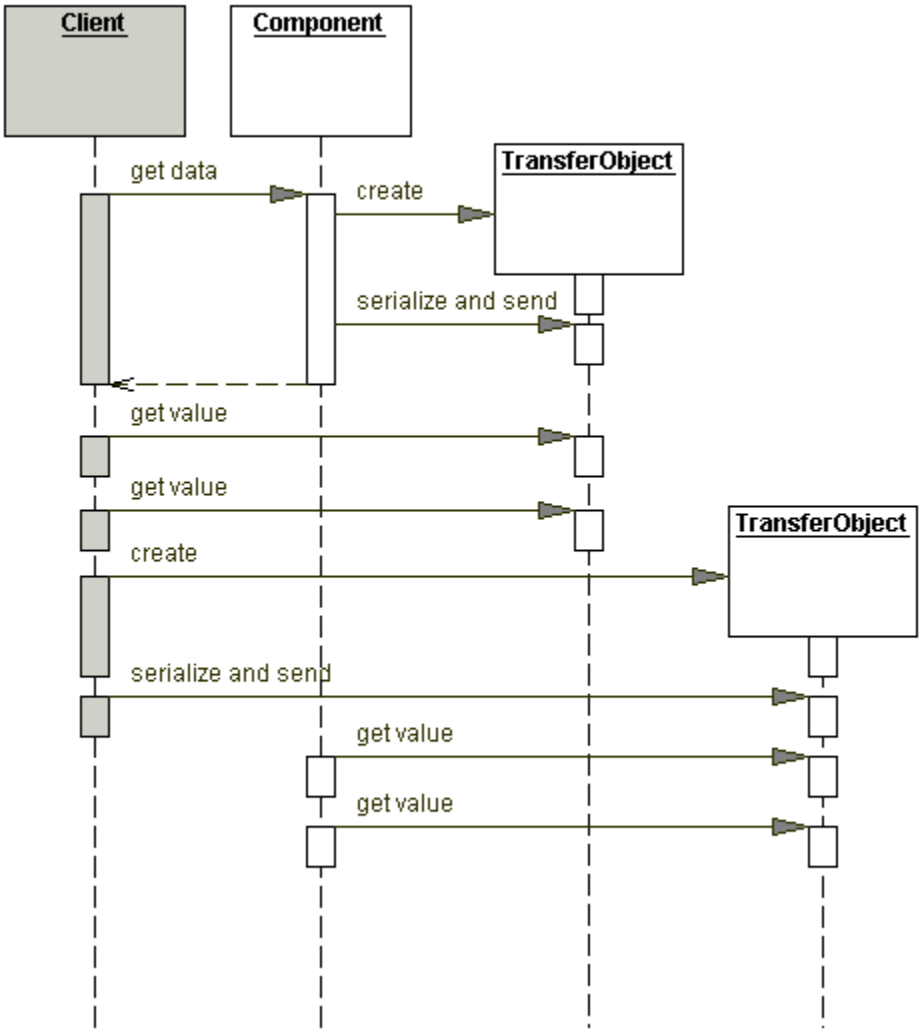
2.2.3.6.1 Descripción

Este patrón de diseño es bastante sencillo. Lo único que pretende es transferir múltiples datos de una capa a otras, reduciendo los costes de múltiples accesos remotos en un solo objeto.

2.2.3.6.2 Diagrama de clases



2.2.3.6.3 Diagrama de secuencia



2.2.3.7 Dispatcher View

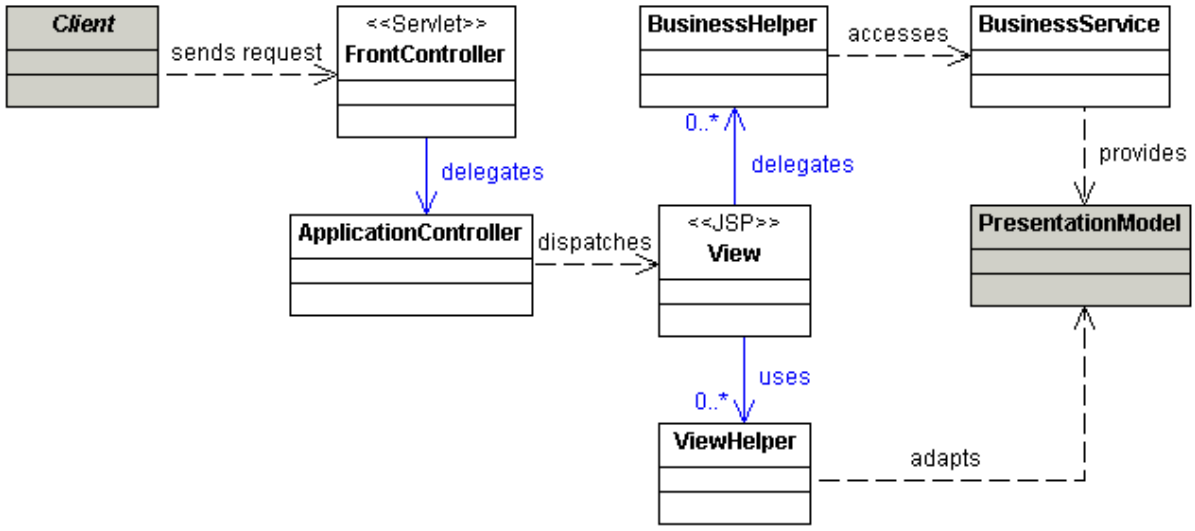
2.2.3.7.1 Descripción

El patrón Dispatcher view es un macro-patrón ya que es una combinación de otros patrones. Dispatcher view describe la combinación de los patrones Front Controller y View Helper con un componente dispatcher.

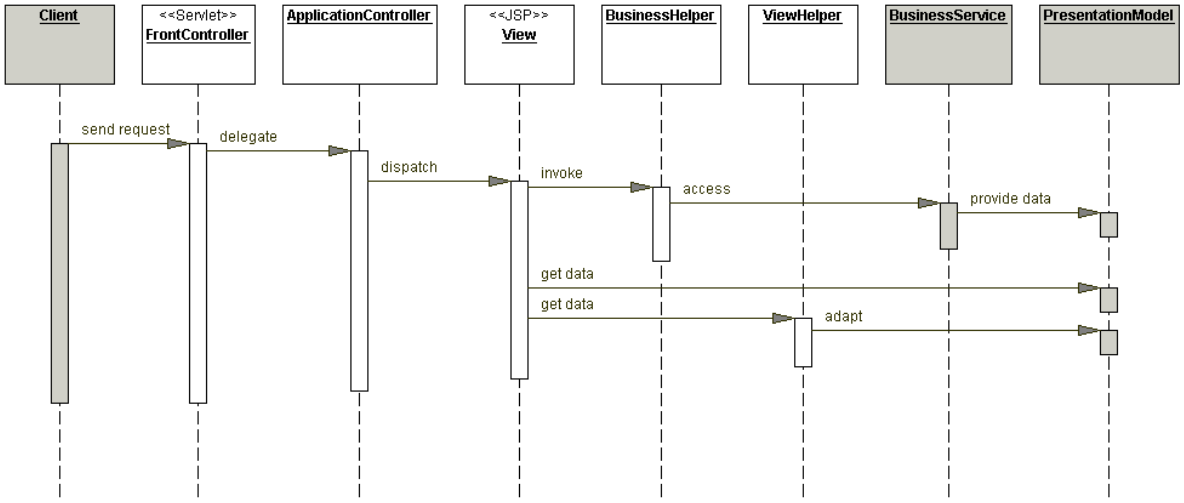
El controlador es un punto inicial para manejar la petición. Un dispatcher controla la vista y la navegación, escogiendo la siguiente vista a mostrar. En la vista se presenta al cliente la información que proviene del modelo, y son los helpers que se ocupan de ello adaptándolo.

Como hemos visto los helpers pueden ser en forma de Value Bean que lo que hacen es almacenar un modelo intermedio o en forma de Taglibs. El Business Service es quien maneja las peticiones de la capa de negocio.

2.2.3.7.2 Diagrama de clases



2.2.3.7.3 Diagrama de secuencia



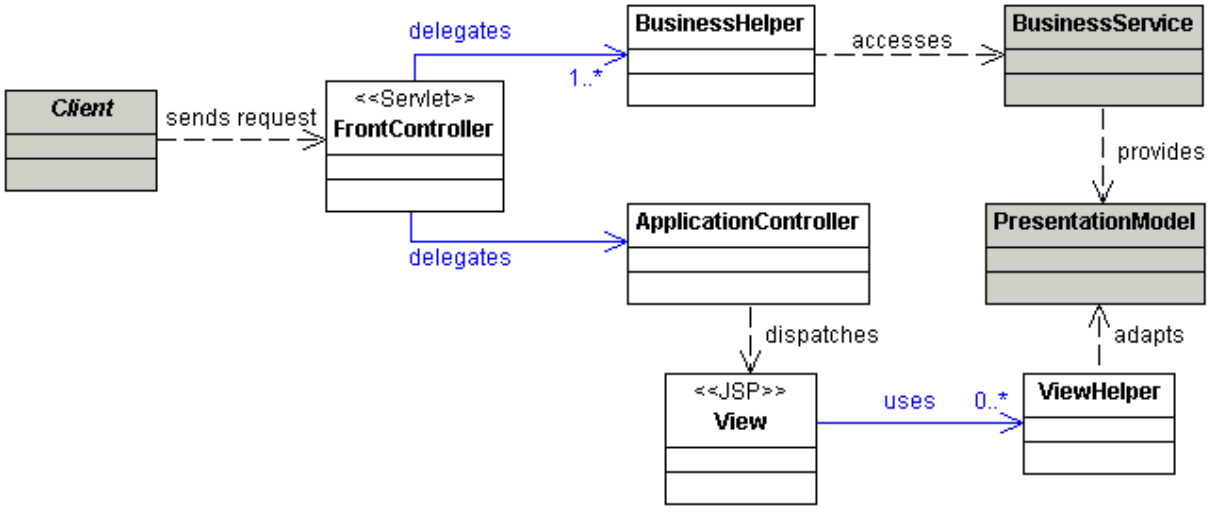
2.2.3.8 Service to worker

2.2.3.8.1 Descripción

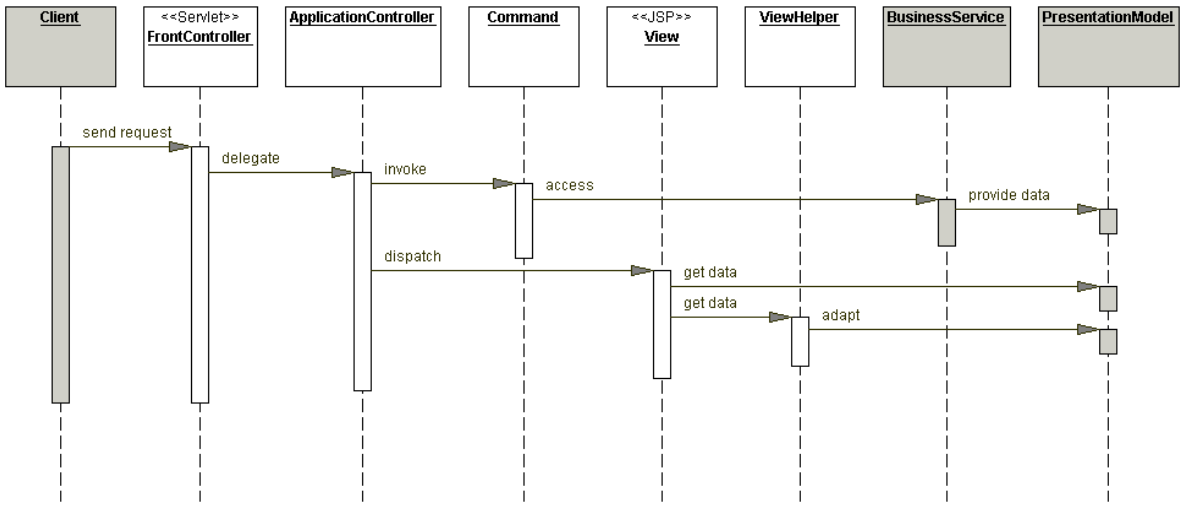
El patrón Service to worker es un macro-patrón ya que es una combinación de otros patrones. Se compone de un controlador, un dispatcher, una vista y sus View Helpers.

Es muy parecido al patrón Dispatcher view. Dispatcher View, a diferencia de Service to Worker, relega la recuperación de contenido al momento de procesar al vista mientras que el otro lo hace en el controlador. El dispatcher de Dispatcher View es también más limitado que el de Service to Worker, ya que la elección de la vista ya está incluida en la petición.

2.2.3.8.2 Diagrama de clases



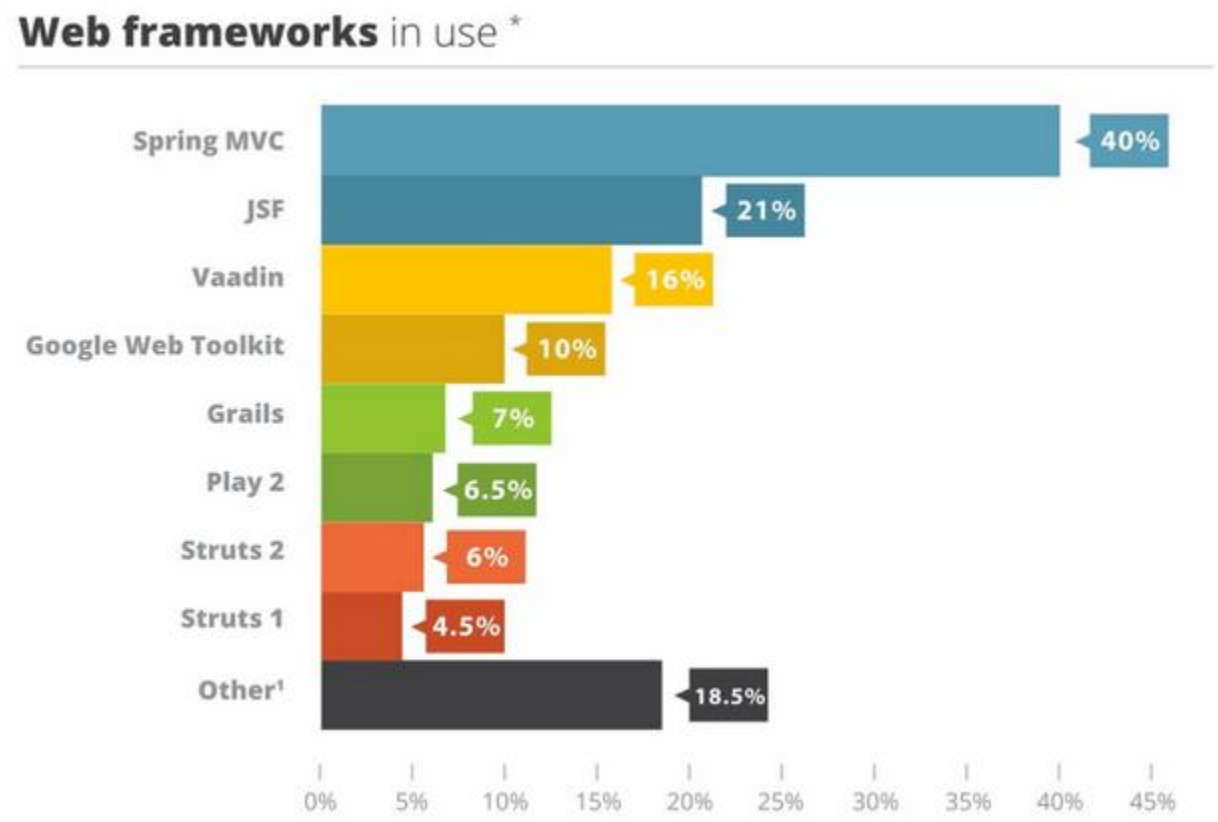
2.2.3.8.3 Diagrama de secuencia



2.3 Frameworks estudiados

Después de estudiar la plataforma JEE, los Framework y los de patrones de diseño en los cuales se basan; analizaremos ahora un Framework real de la capa de presentación.

Como podemos ver en la siguiente imagen, estos son los Frameworks MVC Java más usados (post de agosto 2014).



http://unpocodejava.wordpress.com/category/frameworks

Vemos que Spring MVC es el más utilizado, este puesto lo ostenta desde hace años.

Struts y JSF son los otros dos que también llevan tiempo en el mercado.

2.3.1 Descripción corta

2.3.1.1 Spring MVC

Cuando se habla de Spring hay que entender que se refiere a un conjunto de módulos. La principal característica de Spring está en su core y es el uso de inversión de control, también llamado inyección de dependencias. Esto es que la creación de objetos la lleva a cabo un contenedor externo inyectándolos a otros objetos. Uno de módulos es Spring MVC que es el que define un marco de trabajo por capas. Spring MVC está diseñado para girar en torno a un dispatcher Servlet que maneja peticiones hacia handlers.

2.3.1.2 JSF

JSF es el Framework MVC desarrollado por el Java Community Process. Tiene un conjunto de APIs para presentar componentes de interfaz de usuario, validar las entradas, controlar la navegación entre las páginas y soportar la internalización. Contiene también dos bibliotecas de etiquetas personalizadas para JSPs.

2.3.1.3 Vaadin

Vaadin es un Framework relativamente nuevo. Este Framework permite crear Single Page Applications, esto son aplicaciones web que solo tienen un punto de entrada y no tienen ninguna redirección. Se cargan todas las librerías al inicio, y los recursos y componentes se cargan de forma dinámica a partir de la interacción del usuario. El lado servidor se implementa en Java y no contiene JSPs, ni Taglibs como otros Frameworks.

2.3.1.4 Google Web Toolkit

Este Framework creado por Google permite crear aplicaciones web programando en Java. El código Java se compila y genera código Javascript y HTML que se ejecuta en el navegador cliente. Este código es compatible con la mayoría de navegadores.

2.3.1.5 Grails

Los proyectos Grails se despliegan en los contenedores web de Java y se programan en Groovy y Java. Con Grails la creación de aplicaciones web es muy rápida. Usa el “convention over configuration” para funcionar, y por lo tanto no necesita archivos de configuración (la configuración es solo necesaria si no se sigue las convenciones. Por ejemplo si hay un modelo llamado Persona, en base de datos se espera una tabla llamada persona, si no existe habrá

que configurar la tabla para el model Persona). Grails utiliza JEE como arquitectura base y Spring para estructurar la aplicación vía inyección de dependencias.

2.3.1.6 Play2

Play es un Framework open source programado en Scala y Java. Como Grails usa el "convention over configuration". Como diferencias a otros Frameworks Java se puede destacar que es un Framework sin estado y tiene una arquitectura modular.

2.3.1.7 Struts

Para construir el Framework objeto de este PFC, nos basaremos en gran medida en el Framework Struts2. En el siguiente punto lo analizaremos en profundidad.

2.3.2 Struts 2 por dentro

Struts 2 es un framework de presentación y se puede integrar con otros frameworks como Hibernate y Spring.

El framework ya incluye una biblioteca de etiquetas web pero se puede agregarle funcionalidad mediante el uso de plugins. Su núcleo es un filtro llamado FilterDispatcher (el nombre proviene de versiones anteriores, la clase que realmente implementa el filtro es StrutsPrepareAndExecuteFilter). Es el punto de entrada al Framework y a partir de él se invocan todas las rutinas hasta presentar el contenido al usuario.

Cada petición al FilterDispatcher pasa por tres elementos: los interceptores, las acciones y los resultados.

2.3.2.1 Interceptores

En un punto anterior hemos hablado del patrón de diseño interceptor. Struts2 hace uso de él, y lo utiliza para ejecutar funciones comunes a los Actions (que son acciones que se llevan a cabo en las peticiones del cliente). Podemos activar diferentes interceptores para cada Action según necesidad; estos se ejecutaran antes y después del Action. A parte de los interceptores incluidos en Struts, que cubren la mayoría de las necesidades, podemos crear nuestros propios interceptores. Podemos también cambiar el orden de los interceptores.

A continuación se muestra un listado de los diferentes interceptores de Struts2. Fuente: http://www.javatutoriales.com/2011/06/struts-2-parte-1-configuracion.html

Interceptor	Name	Description
Alias Interceptor	alias	Converts similar parameters that may be named differently between requests.
Chaining Interceptor	chain	Makes the previous Action's properties available to the current Action. Commonly used together with <result type="chain"> (in the previous Action).
Checkbox Interceptor	checkbox	Adds automatic checkbox handling code that detect an unchecked checkbox and add it as a parameter with a default (usually 'false') value. Uses a specially named hidden field to detect unsubmitted checkboxes. The default unchecked value is overridable for non-boolean value'd checkboxes.
Cookie Interceptor	cookie	Inject cookie with a certain configurable name / value into action. (Since 2.0.7.)
CookieProvider Interceptor	cookieProvider	Transfer cookies from action to response (Since 2.3.15.)
Conversion Error Interceptor	conversionError	Adds conversion errors from the ActionContext to the Action's field errors
Create Session Interceptor	createSession	Create an HttpSession automatically, useful with certain Interceptors that require a HttpSession to work properly (like the TokenInterceptor)
DebuggingInterceptor	debugging	Provides several different debugging screens to provide insight into the data behind the page.
DeprecationInterceptor	deprecation	When devMode is on, logs all unknown or deprecated settings
Execute and Wait Interceptor	execAndWait	Executes the Action in the background and then sends the user off to an intermediate waiting page.
Exception Interceptor	exception	Maps exceptions to a result.
File Upload Interceptor	fileUpload	An Interceptor that adds easy access to file upload support.
I18n Interceptor	i18n	Remembers the locale selected for a user's session.
Logger Interceptor	logger	Outputs the name of the Action.
Message Store Interceptor	store	Store and retrieve action messages / errors / field errors for action that implements ValidationAware interface into session.
Model Driven Interceptor	modelDriven	If the Action implements ModelDriven, pushes the `getModel` Result onto the Value Stack.
Scoped Model Driven Interceptor	scopedModelDriven	If the Action implements ScopedModelDriven, the interceptor retrieves and stores the model from a scope and sets it on the action calling `setModel`.
Parameters Interceptor	params	Sets the request parameters onto the Action.
Prepare Interceptor	prepare	If the Action implements Preparable, calls its `prepare` method.
Scope Interceptor	scope	Simple mechanism for storing Action state in the session or application scope.
Servlet Config Interceptor	servletConfig	Provide access to Maps representing HttpServletRequest and HttpServletResponse.
Static Parameters Interceptor	staticParams	Sets the `struts.xml` defined parameters onto the action. These are the <param> tags that are direct children of the <action> tag.
Roles Interceptor	roles	Action will only be executed if the user has the correct JAAS role.
Timer Interceptor	timer	Outputs how long the Action takes to execute (including nested Interceptors and View)
Token Interceptor	token	Checks for valid token presence in Action, prevents duplicate form submission.
Token Session Interceptor	tokenSession	Same as Token Interceptor, but stores the submitted data in session when handed an invalid token
Validation Interceptor	validation	Performs validation using the validators defined in *action*-validation.xml
Default Workflow Interceptor	workflow	Calls the `validate` method in your Action class. If Action errors are created then it returns the `INPUT` view.
Parameter Filter Interceptor	N/A	Removes parameters from the list of those available to Actions
Profiling Interceptor	profiling	Activate profiling through parameter
Multiselect Interceptor	multiselect	Like the checkbox interceptor detects that no value was selected for a field with multiple values (like a select) and adds an empty parameter

Para poder activar más de un interceptor a la vez, Struts2 ofrece pilas de interceptores ya configuradas.

2.3.2.2 Acciones

Los Actions son las clases encargadas de realizar la lógica de la petición. Las URLs están mapeadas a las acciones. Estas clases tiene un método, por defecto execute(), que contiene el código de la acción y retorna un String o Object Result. Por defecto se retorna el valor "SUCCES".

2.3.2.3 Resultados

Cada resultado de las acciones tiene configurado hacia donde se redirige al usuario: a un JSP o un archivo en flujo de bytes.

Los interceptores pueden llegar a impedir que se ejecute el Action, y son ellos mismos quien redirigen al usuario.

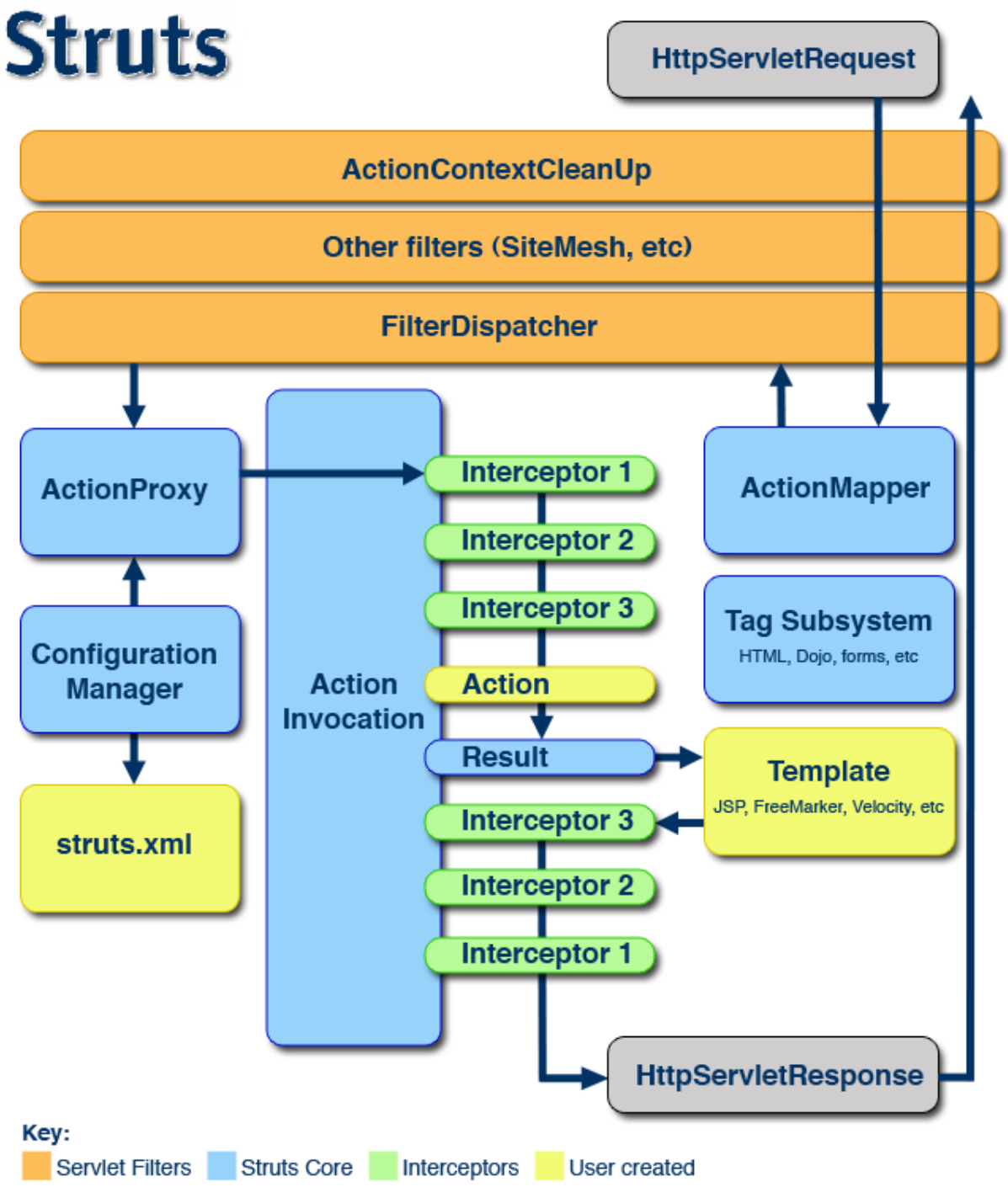
2.3.2.4 Configuración

En la siguiente tabla se muestran los archivos de configuración de Struts2. Fuente: https://struts.apache.org/docs/configuration-files.html

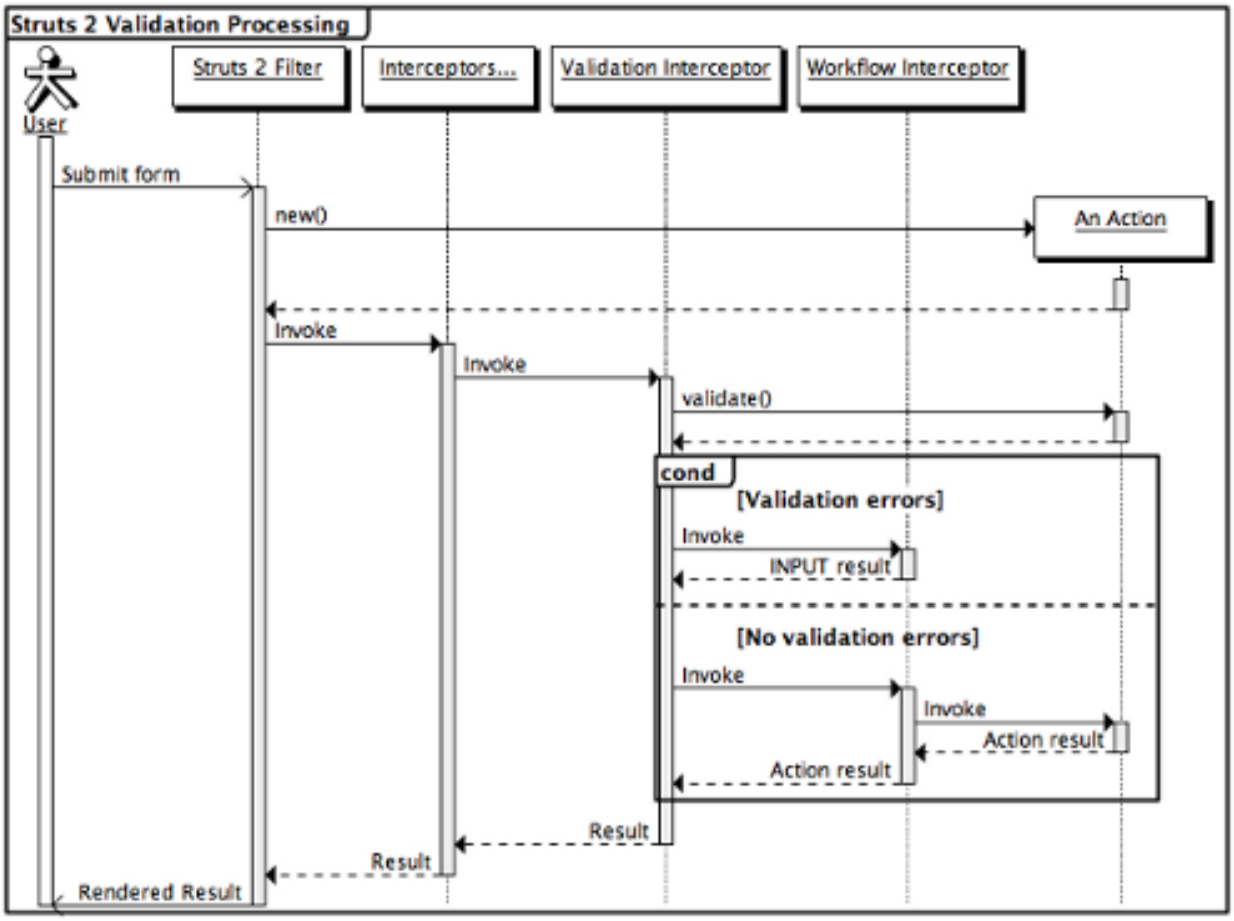
File	Optional	Location (relative to we-bapp)	Purpose
web.xml	no	/WEB-INF/	Web deployment descriptor to include all necessary framework components
struts.xml	yes	/WEB-INF/classes/	Main configuration, contains result/view types, action mappings, interceptors, and so forth
struts.properties	yes	/WEB-INF/classes/	Framework properties
struts-default.xml	yes	/WEB-INF/lib/struts2-core.jar	Default configuration provided by Struts
struts-default.vm	yes	/WEB-INF/classes/	Default macros referenced by `velocity.properties`
struts-plugin.xml	yes	At the root of a plugin JAR	Optional configuration files for Plugins in the same format as `struts.xml`.
velocity.properties	yes	/WEB-INF/classes/	Override the default Velocity configuration

2.3.2.5 Diagrama resumen

A continuación se ve un diagrama que ilustra el funcionamiento en conjunto de los elementos descritos anteriormente.



Podemos ver también un diagrama UML de secuencia que ilustra un caso más específico y concreto de funcionamiento de Struts2.


Struts 2 Validation Processing
User
Struts 2 Filter
Interceptors...
Validation Interceptor
Workflow Interceptor
An Action
Submit form
new()
Invoke
Invoke
validate()
cond
[Validation errors]
Invoke
INPUT result
[No validation errors]
Invoke
Invoke
Action result
Action result
Result
Result
Rendered Result

3 Análisis y diseño del Framework de presentación CVF

Después de toda esta búsqueda y asimilación de conocimiento, pasamos a un caso práctico de desarrollo creando nuestro propio Framework. Su principal diferencia y mejora respecto a otros Frameworks es su enfoque multiusuario. Para mostrar un caso real de uso del Framework recién creado, desarrollaremos un prototipo de una aplicación.

En este punto no se implementa nada, solo analizaremos y diseñaremos.

3.1 Descripción

El Framework que elaboraremos en este PFC será bastante parecido a Struts2. Cambiaremos ciertas partes y le añadiremos la funcionalidad de manejar concurrencia entre usuarios.

Las aplicaciones de gestión ofrecen a los usuarios listados, formularios para añadir y modificar elementos, generar informes como tareas básicas. Una problemática que no se resuelve fácilmente es la de la concurrencia entre varios usuarios. ¿Qué pasa si un usuario esta listando unos elementos y otro los está modificando? En un momento dado habrá una inconsistencia en los datos que se muestran por pantalla. Si son muchos los usuarios a interactuar simultáneamente puede llegar a ser un verdadero problema.

Este Framework a parte de facilitar al programador la creación de páginas con listados y formularios, aporta un componente configurable que atienda la concurrencia entre usuarios. Inicialmente solo ofreceremos dos modos de concurrencia: notificación al usuario que ha habido cambios en los datos que se están mostrando y actualización de los datos de forma automática.

El Framework lo llamaremos CVF, de Concurrent View Framework.

El Framework solo se ocupará de la capa de presentación, dejando de lado la capa de modelo. De hecho la capa de persistencia lo podemos implementar tanto en Hibernate, como JDBC o cualquier otra tecnología; queda fuera de las especificaciones del Framework. Solo damos por hecho que hay unos services que retornan JavaBeans o listas de JavaBeans que son los modelos intermedios que usaran lo View Helpers. Los Services por habito atacarán DAOs (Data Access Objects) pero queda también fuera de este Framework.

A continuación se mencionan los patrones de diseño que se usaran en el Framework. Como estructura general se podría decir que sigue el patrón de diseño Service to Worker.

3.2 Patrones de diseño

3.2.1 Filter Dispatcher

Como punto de entrada a CVF nos planteamos si usar un Front Controller o un Filter Dispatcher. Optamos por lo segundo ya que nos asegura que no habrá acceso a JSPs directamente por URL. Todas las peticiones pasarán por fuerza por el Filter. Es único punto de entrada nos permite mantener una gestión de logs de utilización de la aplicación.

3.2.2 Interceptores

Tendremos interceptores que se ejecuten antes y después de los Actions. Tendremos tres subconjuntos que se activaran según la terminación de las URLs.

- Terminados por .view ej: http://localhost/app/content1.view

  Se ejecutarán el interceptor de control de sesión, y el interceptor de control de permisos

- Terminados por .action ej: http://localhost/app/form1.action

  Aparte de los anteriores se ejecutaran los interceptores de checkboxes, de validación de formularios

- Terminados por .monitor ej: http://localhost/app/beans.monitor

3.2.3 Composite View

Las vistas podrán ser compuestas. En principio nos planteamos usar los tags <jsp:include> aunque finalmente se haremos uso de Taglibs.

Así index.jsp podría estar compuesto de una cabecera, un menú, y el contenido. Cado uno será un taglib y el contenido será obtenido por una petición al Filter Dispatcher que llamara a un Action y este redirigirá a un JSP adjuntándole el contenido en forma de JavaBean.

3.2.4 View Helpers

Las vistas utilizaran View Helpers para mostrar el contenido. Los Views Helpers los implementaremos como TagLibs. En un principio tendremos los siguientes TagLibs:

- <section> creará una sección con el contenido de otra petición al Framework.
- <mantField> genera inputs de tipo texto, combos, checkboxes, calendarios, etc... según del tipo de datos
- <navigation> genera botones de navegación. Hace peticiones al servidor y carga contenido en un contenedor.
- <mantList> genera listados tabulados a partir de una colección de objetos.

3.2.5 Componentes de monitorización de concurrencia

3.2.5.1 Server-side

A cada vez que se ejecuta un Action y este modifica el modelo (después de un cambio de formulario por ejemplo), un interceptor añade a una lista el elemento del modelo modificado. Esta lista es una cola y espera que el cliente la lea y desencole.

3.2.5.2 Client-side

El cliente tendrá un componente de monitorización de concurrencia implementado en Javascript. Este se comunicará con el servidor para obtener la lista de elementos del modelo modificados. Esto se puede hacer por pooling via AJAX o con un WebSocket permanente.

Para cada elemento de la lista, el componente verificará si hay un objeto registrado a la espera de notificación. Si no hay objeto registrado, el elemento se descarta, sino se procesa la petición de notificación.

Los objetos, que puede ser campos de texto, listas, combos, checkboxes, etc.., han de poder registrarse y descartarse del componente de concurrencia. Los View Helpers se registran al componente al cargarse y se descartan del componente al descargarse.

4 Implementación del Framework de presentación CVF

Este apartado consta con la descripción de la implementación del Framework de la capa de presentación CVF y una aplicación de prueba. Se entregan para ambos el código fuente y los binarios.

La aplicación de prueba la he donado a la Cruz roja Andorrana. A día de esta entrega la aplicación esta solo en su fase inicial, solo para demostrar el uso del Framework, pero en los próximos meses pasará a ser una aplicación de uso real para la gestión de usuarios de la Cruz roja.

4.1 Estructura de carpetas

- **bin/CreuRojaGestio/CreuRojaGestio.war**

  Archivo para desplegar la aplicación de prueba en Tomcat

- **bin/cvf/cvf-1.0.1.jar**

  Archivo JAR con el Framework. Es el archivo que ha de importar cualquier aplicación que quiera usar el Framework.

- **src/CreuRojaGestio/**

  Carpeta con todo el código fuente de la aplicación de prueba. Es el contenido del proyecto creado con Eclipse. Se puede importar en otro Eclipse

4.2 Manual del Framework CVF

Para usar el Framework CVF en una aplicación hay que añadir las líneas siguientes al archivo web.xml utilizado por Tomcat para desplegar la aplicación web:

```
<filter>
	<filter-name>CvfFilterDispatcher</filter-name>
	<filter-class>com.achavero.cvf.core.CvfFilterDispatcher</filter-class>
</filter>

<filter-mapping>
	<filter-name>CvfFilterDispatcher</filter-name>
	<url-pattern>/*</url-pattern>
</filter-mapping>
```

y

```
<jsp-config>
  <taglib>
        <taglib-uri>http://cvf.achavero.com/taglibs/views</taglib-uri>
        <taglib-location>/META-INF/mytags.tld</taglib-location>
  </taglib>
</jsp-config>
```

Así tendremos el filter dispatcher en funcionamiento y todos los Taglibs del framework disponibles para ser usados en los JSPs del programa desarrollado.

También hay que crear un archivo con el nombre cvf.xml en la raíz del proyecto y configurarlo correctamente. En este archivo se configuran todas las opciones del framework y se definen las peticiones que tendrá la aplicación y su secuencia.

Por último y imprescindiblemente hay que importar la librería en formato JAR. La librería está adjunta a este documento en la carpeta bin/cvf/ .

4.2.1 Contenido del JAR cvf.jar

El JAR contiene todos los archivos que componen el Framework: sus clases java y los TagFiles ofrecidos para las vistas. El Framework está compuesto por los siguientes packages:

- **com.achavero.cvf.core** : Todas las clases que definen el core del Framework. El uso de cada clase se puede conocer consultando su documentación. El usuario que utilice el Framework solo necesitará usar las clases abstractas CvfInterceptor y CvfRequestCommand; el resto son de uso interno.
- **com.achavero.cvf.interceptors** : Aquí encontramos los interceptores ya creados dentro del Framework. El usuario necesitará configurar el cvf.xml para utilizarlos.
- **com.achavero.cvf.view**: Están las clases de ayuda para las vistas. Desde anotaciones para los beans, clases de uso para los TagLibs etc..
- **com.achavero.utils**: Clases de utilidad
- **META-INF/tags**: Aquí están todos los TagFiles que componen los View Helpers. Detallamos a continuación los más importantes:
    - **mantField.tag** : crea un campo de formulario a partir de un JavaBean y una propiedad. Según el tipo de la propiedad creará un campo diferente y una validación diferente. Irá a buscar la etiqueta multiidioma en las anotaciones del JavaBean, así como la longitud máxima, etc..
    - **mantList.tag** : crea un listado a partir de una colección de JavaBeans y las propiedades (una columna por cada propiedad)

- **navigation.tag** : crea botones para la navegación. Carga URLs o el resultado de formularios en secciones de la página mediante AJAX.
- **section.tag** : crea secciones en las cuales cargar contenido

4.2.2 Configuración cvf.xml

A continuación se hace una descripción de los campos de XML de configuración con ejemplos. Un DTD definiría mejor la estructura y posibles valores del XML, y sobre todo forzaría al usuario a generar un archivo de configuración valido.

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

<cvf>

Aquí se configuran aspectos globales del framewrok
<configuration>

La página por defecto
<page name="default" uri="default_page.jsp" />

La página a mostrar en caso de no encontrar la pagina
<page name="not_found" uri="/error/not_found.jsp" />

La página a mostrar en caso de que haya un error en la configuración del framework
<page name="cvf_xml_config_error" uri="/error/xml_config_error.jsp" />

Lista de carpetas que no se procesan por el framework CVF y por la tanto son tratados
por el Tomcat como lo haría sin el framework.
<excluded type="directory" value="carpetaNoCVF" />

</configuration>

Aquí se configuran los interceptores
<interceptors>

Este sería un interceptor que ya implementa el core del framework CVF, solo registra
el tiempo necesario para servir la petición.
El atributo intercepting puesto a auto, activa este interceptor por defecto en cual-
quier petición
<interceptor name="timeEllapsed"
class="com.achavero.cvf.interceptors.TimeEllapsedInterceptor" intercepting="auto"/>

Este sería un ejemplo de interceptor implementado por la aplicación cliente del fra-
mework
El atributo intercepting puesto a manual, obliga al usuario a configurar en que peti-
ciones si se activará este interceptor manualmente
<interceptor name="ejemplo" class="com.package.ejemplo.interceptor" intercep-
ting="manual"/>
```

```
</interceptors>

Aquí se define un componente, se pueden define tantos componentes como se quiera,
solo son agrupaciones lógicas
<component name="nombre componente">

Aquí se configura que la petición http://host:port/app/requestEjemplo1.view
Se redirige a http://host:port/app/ejemplo1.jsp, sin ejecutar ninguna instrucción.
<request name="requestEjemplo1" type="view">
	<result>/ejemplo1.jsp</result>
</request>

En este ejemplo se ejecuta el commando request2 y al devolver success redirige a
ejemplo2SuccesPage.jsp
<request name="requestEjemplo2" type="view" class="com.package.ejemplo.request2">
	<result name="success">/ejemplo2SuccesPage.jsp</result>
</request>

En este ejemplo se ejecuta la acción request3 y según el resultado se redirige a un
jsp o a otro request
<request name=" requestEjemplo3" type="action" class=" com.package.ejemplo.request3">
	<result name="error">/ejemplo3Succes.view</result>
	<result name="success">/ejemplo3SuccesPage.jsp </result>
</request>

</component>

</cvf>
```

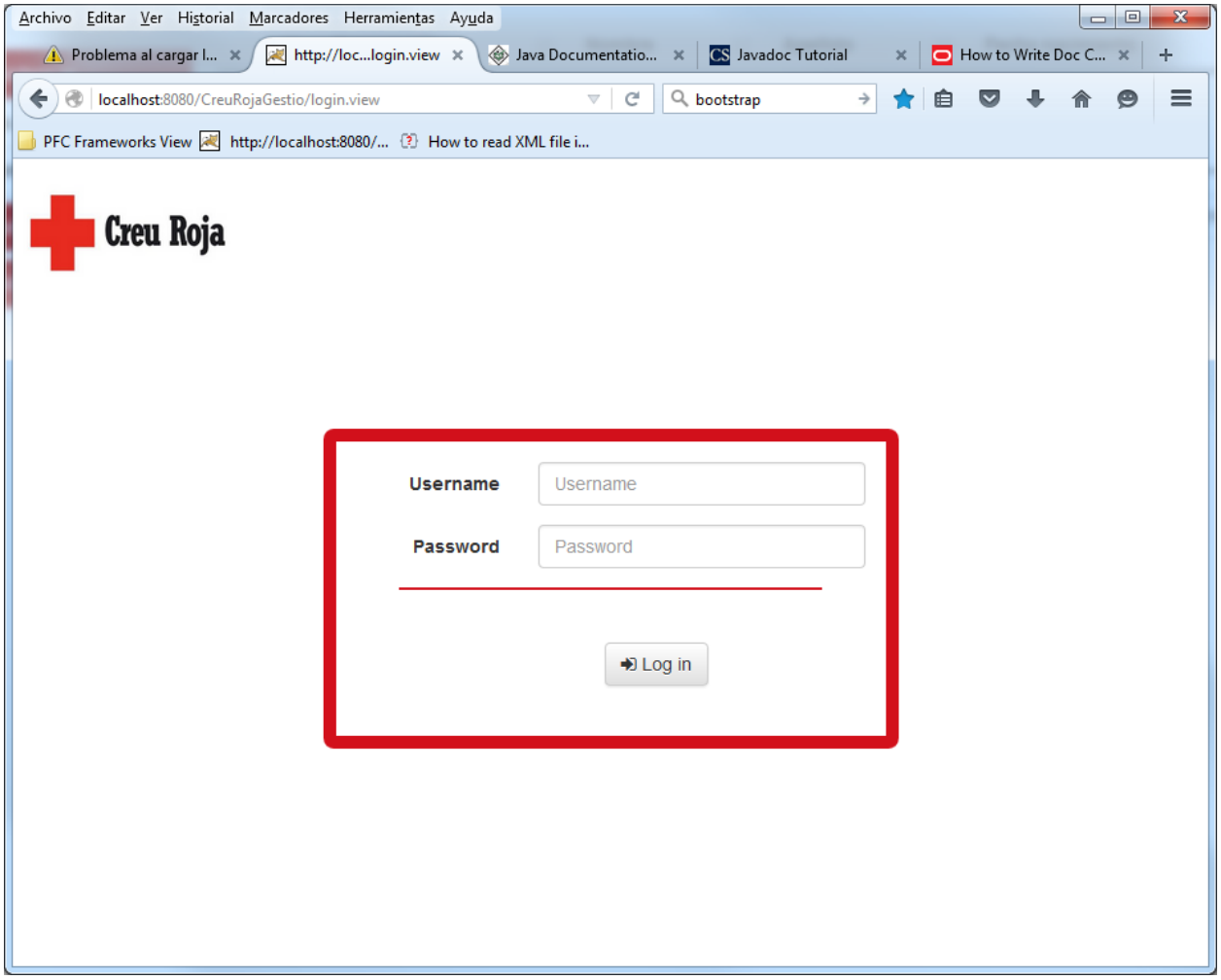
4.3 Aplicación de prueba - Creu Roja Gestio

Para probar el funcionamiento del framework tenemos que hacer un prototipo que use las funciones que proporciona.

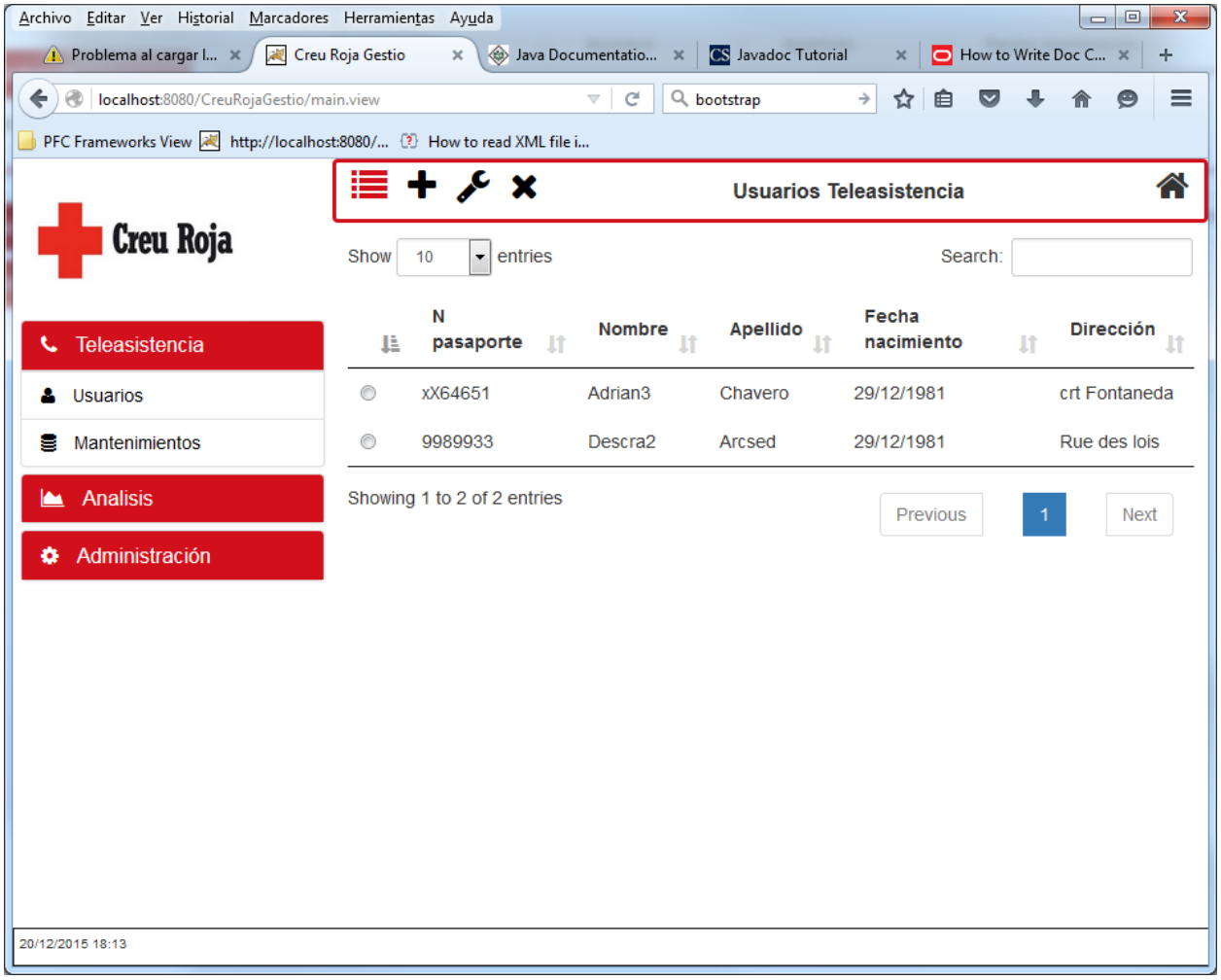
Soy voluntario desde hace un año de la Cruz Roja Andorrana, y hablando con la responsable de la delegación ya hace tiempo convenimos que haría una aplicación para la gestión de los usuarios de la entidad. Viendo la posibilidad de avanzar el PFC a la vez que esta aplicación, el prototipo que muestra el uso del framework será la aplicación Creu Roja Gestio. A momento de la entrega de este PFC la aplicación no estará terminada, pero en los meses que siguen será usado por usuarios de verdad.

La aplicación solo implementa el login y dos puntos de menú: el mantenimiento de logins y de el mantenimiento de usuarios de la teleasistencia.

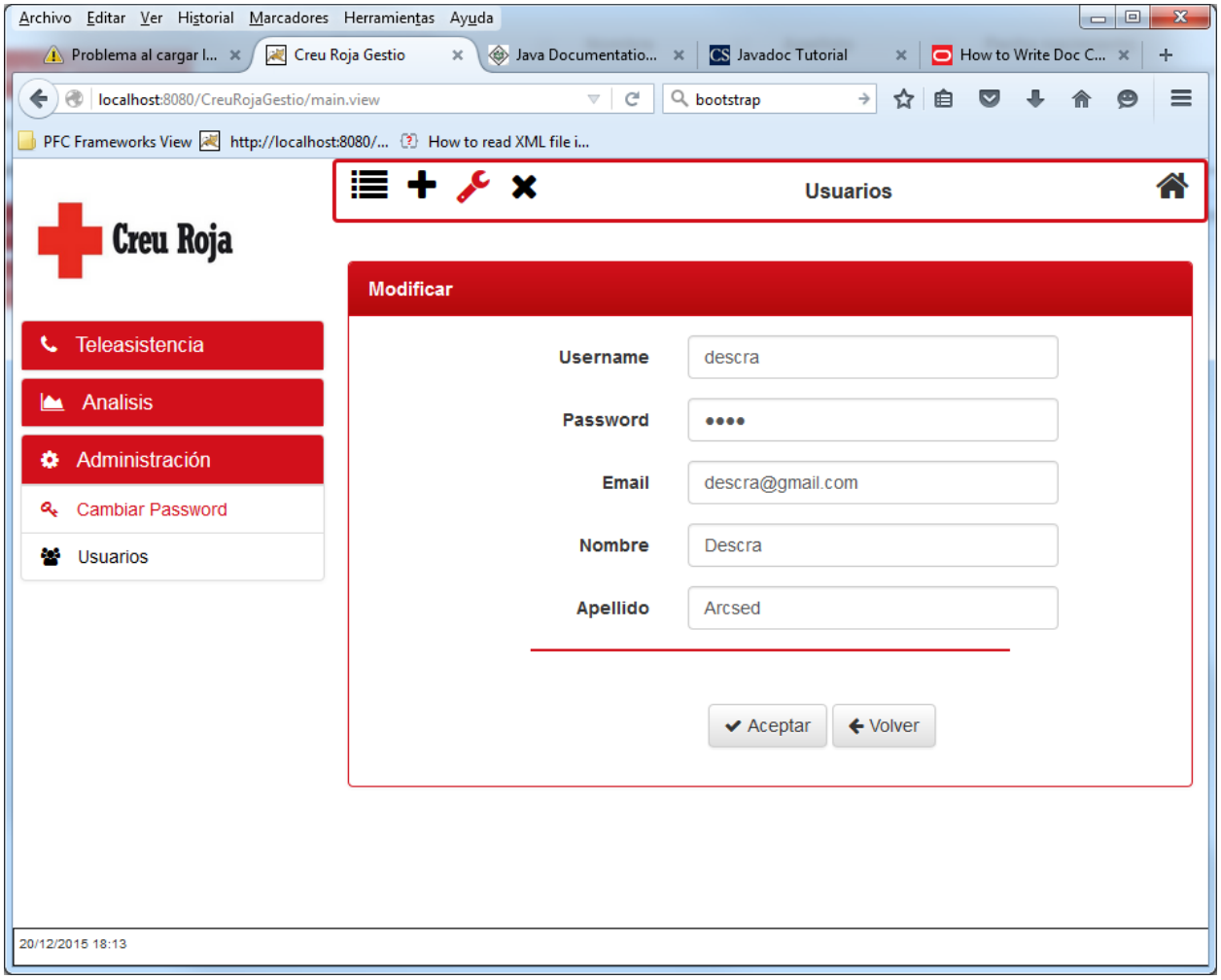
- Login


Archivo Editar Ver Historial Marcadores Herramientas Ayuda
Problema al cargar l...
http://loc...login.view
Java Documentatio...
Javadoc Tutorial
How to Write Doc C...
localhost:8080/CreuRojaGestio/login.view
bootstrap
PFC Frameworks View
http://localhost:8080/...
How to read XML file i...
Creu Roja
Username
Username
Password
Password
Log in

- Teleasistencia -> Usuarios (Listar, añadir, modificar y eliminar)



- Administración -> Usuarios (Listar, añadir, modificar y eliminar)



La configuración del web.xml es la siguiente:

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

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                      http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1"
  metadata-complete="true">
  <display-name>Creu Roja Gestio</display-name>

  <filter>
        <filter-name>CvfFilterDispatcher</filter-name>
        <filter-class>com.achavero.cvf.core.CvfFilterDispatcher</filter-class>
  </filter>

  <filter-mapping>
        <filter-name>CvfFilterDispatcher</filter-name>
        <url-pattern>/*</url-pattern>
  </filter-mapping>
```

```
  <listener>
    <listener-class>com.achavero.cvf.core.CvfServletContextListener</listener-class>
  </listener>

  <jsp-config>
    <taglib>
          <taglib-uri>http://cvf.achavero.com/taglibs/views</taglib-uri>
          <taglib-location>/META-INF/mytags.tld</taglib-location>
    </taglib>
  </jsp-config>

</web-app>
```

La configuración del cvf.xml es la siguiente:

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

<cvf>

<configuration>
<page name="default" uri="login.view" />
<page name="not_found" uri="/error/not_found.jsp" />
<page name="cvf_xml_config_error" uri="/error/xml_config_error.jsp" />

<excluded type="directory" value="test" />

</configuration>

<interceptors>
<interceptor name="timeEllapsed" class="com.achavero.cvf.interceptors.TimeEllapsedInterceptor" intercept-
ing="auto"/>
<interceptor name="logged" class="ad.creuroja.gestio.interceptors.LoggedInterceptor" intercepting="auto"/>
</interceptors>

<component name="CreuRojaGestioMain">
<request name="main" type="view">
        <result>/main/main.jsp</result>
</request>
<request name="mainContent" type="view">
        <result>/main/mainContent.jsp</result>
</request>
<request name="menu" type="view">
        <result>/main/menu.jsp</result>
</request>
<request name="headerContent" type="view">
        <result>/main/headerContent.jsp</result>
</request>
<request name="footer" type="view">
        <result>/main/footer.jsp</result>
</request>
<request name="about" type="view" class="ad.creuroja.gestio.requests.AboutView">
        <result name="success">/main/about.jsp</result>
</request>
</component>

<component name="teleAssistMant">
<request name="teleAssistUserList" type="view"
class="ad.creuroja.gestio.requests.teleAssist.TeleAssistUserListView">
        <result name="success">/teleAssist/teleAssistUserList.jsp</result>
</request>
<request name="teleAssistUserAdd" type="view">
        <result>/teleAssist/teleAssistUserAddMod.jsp</result>
</request>
<request name="teleAssistUserMod" type="view"
class="ad.creuroja.gestio.requests.teleAssist.TeleAssistUserModView">
        <result name="error">teleAssistUserList.view</result>
        <result name="success">/teleAssist/teleAssistUserAddMod.jsp</result>
```

```
</request>
<request name="teleAssistUserAdd" type="action"
class="ad.creuroja.gestio.requests.teleAssist.TeleAssistUserAddAction">
        <result name="error">/teleAssist/teleAssistAddMod.jsp</result>
        <result name="success">teleAssistUserList.view</result>
</request>
<request name="teleAssistUserMod" type="action"
class="ad.creuroja.gestio.requests.teleAssist.TeleAssistUserModAction">
        <result name="error">/teleAssist/teleAssistAddMod.jsp</result>
        <result name="success">teleAssistUserList.view</result>
</request>
<request name="teleAssistUserDel" type="action"
class="ad.creuroja.gestio.requests.teleAssist.TeleAssistUserDelAction">
        <result name="error">teleAssistUserList.view</result>
        <result name="success">teleAssistUserList.view</result>
</request>
</component>

<component name="loginModule">
<request name="login" type="view">
        <interceptor-ref name="logged" exclude="true" />
        <result>/main/login.jsp</result>
</request>
<request name="login" type="action" class="ad.creuroja.gestio.requests.LoginAction">
        <interceptor-ref name="logged" exclude="true" />
        <result name="failed">/main/login.jsp</result>
        <result name="success">main.view</result>
</request>
<request name="loginList" type="view" class="ad.creuroja.gestio.requests.administration.LoginListView">
        <result name="success">/administration/loginList.jsp</result>
</request>
<request name="loginAdd" type="view">
        <result>/administration/loginAddMod.jsp</result>
</request>
<request name="loginMod" type="view" class="ad.creuroja.gestio.requests.administration.LoginModView">
        <result name="error">loginList.view</result>
        <result name="success">/administration/loginAddMod.jsp</result>
</request>
<request name="loginAddMod" type="action"
class="ad.creuroja.gestio.requests.administration.LoginAddModAction">
        <result name="errorAdd">/administration/loginAddMod.jsp</result>
        <result name="errorMod">/administration/loginAddMod.jsp</result>
        <result name="successAdd">loginList.view</result>
        <result name="successMod">/administration/loginAddMod.jsp</result>
</request>
<request name="loginDel" type="action" class="ad.creuroja.gestio.requests.administration.LoginDelAction">
        <result name="error">loginList.view</result>
        <result name="success">loginList.view</result>
</request>
</component>

</cvf>
```

Todas las clases de los request implementan la clase CvfRequestCommand ofrecida por el Framework y están en el package ad.creuroja.gestio.request.

Todos los interceptores implementan la clase CvfInterceptor y están en el package ad.creuroja.gestio.interceptors .

La aplicación de prueba usa Hibernate para la persistencia de datos. Los modelos se encuentran en el package: ad.creuroja.gestio.models .

La lógica de negocio se implementa en services que están en el package ad.creuroja.gestio.services.impl .

4.4 Puntos pendientes

La aplicación web Creu Roja Gestio dejará de ser un prototipo en los próximos meses. La coordinadora de la cruz roja andorrana irá detallando el resto de secciones que precisará la aplicación. A medida que la aplicación crezca, probablemente el framework tendrá que ampliarse en funcionalidad. Con los esfuerzos necesarios es posible que el framework CVF lo utilice en mis proyectos de software por venir.

El Framework aun que funcional, tiene ciertos puntos importantes que no se han implementado. El archivo de configuración cvf.xml necesita de un DTD. Así el usuario escribirá forzosamente un XML correcto.

La gestión de excepciones precisa también de ser terminada.

Una de las partes importantes de este framework es el componente de concurrencia entre usuarios, que describimos durante el análisis y diseño. La implementación de este componente no está incluida en el PFC pero está por venir.

5 Conclusiones

Este proyecto de final de carrera me ha ofrecido la oportunidad de consolidar mis conocimientos en ingeniería de software. El PFC pasa por todas las etapas de desarrollo de un software, tanto para el marco de trabajo como para el prototipo.

La definición de requisitos de usuario y la planificación no ocupan un parte importante de este proyecto pero también han tenido un peso pedagógico ya que se sitúan en el marco de un proyecto real y no en uno teórico.

El análisis y diseño del framework han sido lo más instructivo, sobre todo por el estudio de los patrones de diseño y el framework Struts 2. Los patrones de diseño son conceptos que podré reutilizar en muy diversas ocasiones durante mi vida profesional dado su naturaleza abstracta y genérica.

La implementación no ha sido especialmente difícil pero sí bastante laboriosa. El hecho de implementar el framework por un lado y el prototipo por otro le confiere una gran diferencia respecto a los muchos otros software que he implementado en mi vida. En muchas ocasiones he estado implementando ambos en paralelo.

La memoria termina dando a este proyecto de final de carrera su lado más académico, brindando una oportunidad de aprender a realizar trabajos en el ámbito de la informática de forma más formal y profesional.

Por último añadir que me siento orgulloso que el proyecto no termine aquí, ya que el prototipo Creu Roja (Andorra) Gestio pasará a ser un programa real y utilizado por usuarios de verdad. En los próximos meses iré programando las secciones que solicite la coordinadora de la cruz roja Andorrana y ampliando el framework CVF para dar nuevas funcionalidades. Con los esfuerzos adecuados espero terminar el framework en condiciones y reutilizarlo en otros proyectos venideros.

6 Glosario

PFC: Proyecto de Final de Carrera

Framework: conjunto de tecnologías que sirve para desarrollar y unir los diferentes componentes de un proyecto. Esta infraestructura digital está compuesta por programas, bibliotecas, lenguajes interpretados y herramientas.

Prototipo: representación de un sistema, aunque no es un sistema completo, posee las características del sistema final o parte de ellas.

Diagrama de Gantt: herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

JavaBeans: clase Java con las convenciones: un constructor sin argumentos, atributos privados, sus propiedades accesibles mediante métodos get y set y serializable.

API: interfaz de programación de aplicaciones es un conjunto de subrutinas, funciones y procedimientos que ofrece una biblioteca para ser utilizado por otro software.

HTTP: Hypertext Transfer Protocol, protocol usado en cada transacción de la World Wide Web

XML: siglas de eXtensible Markup Language, es un lenguaje de marcas utilizado para almacenar datos en forma legible.

DTD: siglas de Document Type Definition, descripción de estructura y sintaxis de un documento XML.

JEE: Java Enterprise Edition, plataforma de programación para desarrollar y ejecutar aplicaciones en lenguaje de programación Java.

JSP: Java Server Page, páginas HTML con código Java incrustado, internamente son traducidas a Servlets en el contenedor de aplicaciones JEE.

Servlet: Clase Java, para generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

JTA: Java Transaction API, para la gestión de transacciones.

JDBC: Java Database Connectivity, API Java para ejecutar instrucciones sobre una base de datos.

Taglib: componentes reutilizables en paginas JSP.

MVC: Model View Controller, patrón de arquitectura que separa el código en tres capas diferentes. Cada capa tiene una responsabilidad diferente y esta desacoplada de las demás.

DAO: Data Acces Object, componente de software con una interfaz para acceder a los datos.

Bootstrap: HTML, CSS y JS web framework.

Hibernate: framework open source para la persistencia en programas Java

Tomcat: contenedor de Servlets por Apache Software Foundation

Eclipse: IDE para programar en Java

Mysql: Sistema Gestor de Base de Datos multihilo y multiusuario de Oracle Corporation.

7 Bibliografía

Documentación

http://www.arquitecturajava.com/el-concepto-de-java-proxy-pattern/

http://www.arquitecturajava.com/para-que-sirven-los-spring-modules/

http://www.arquitecturajava.com/spring-mvc-requestmapping/

http://www.oracle.com/technetwork/articles/java/mvc-2280472.html

http://corej2eepatterns.com/

http://www.javaworld.com/article/2940438/java-web-development/jsf-2-3-aims-to-be-the-default-mvc-framework-for-java-ee.html

http://programacion.net/articulo/catalogo_de_patrones_de_diseno_j2ee_i_capa_de_presentacion_240/4

http://corej2eepatterns.com/InterceptingFilter.htm

http://www.wisdomjobs.com/e-university/j2ee-tutorial-230/j2ee-patterns-overview-280/j2ee-patterns-1576.html

http://www.tutorialspoint.com/struts_2/struts_examples.htm

http://www.javatutoriales.com/2011/06/struts-2-parte-1-configuracion.html

http://www.vogella.com/tutorials/JavaXML/article.html

https://es.wikipedia.org/wiki/Wikipedia:Portada

http://www.oracle.com/es/index.html

Software

https://www.datatables.net/ç

https://www.mysql.com/

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

http://hibernate.org/orm/

https://eclipse.org/

https://maven.apache.org/

http://tomcat.apache.org/

http://www.nomagic.com/products/magicdraw.html

8 Anexos

8.1 Instalación/Ejecución framework CVF

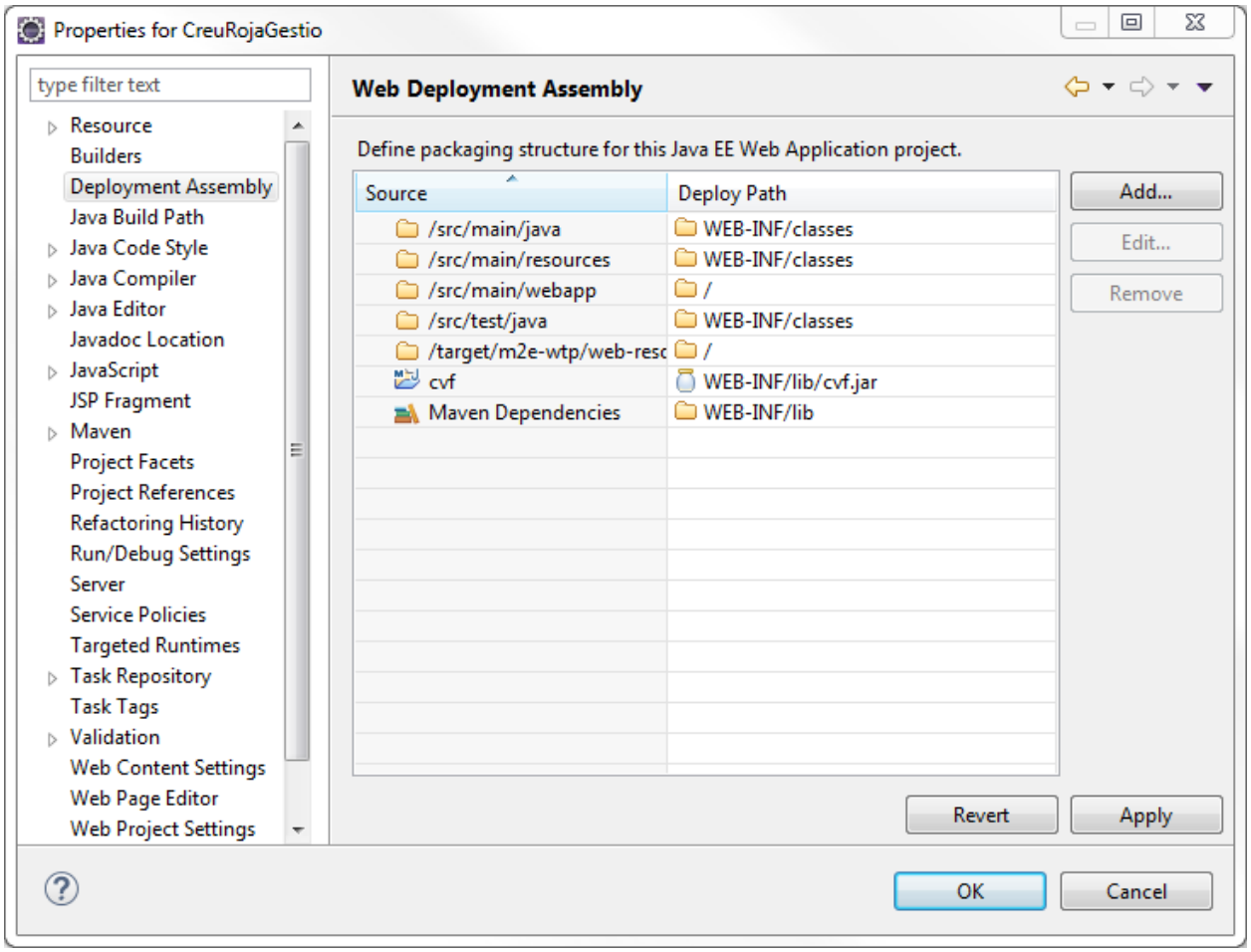
El uso del framework es muy sencillo como visto en el manual del framework en el punto anterior. Hay que importar el archivo JAR que contiene todos los archivos necesarios y crear un XML con el nombre cvf.xml en la raíz del proyecto (en este XML se configura el proyecto para que use el framework). También hay que modificar el web.xml utilizado por Tomcat para desplegar la aplicación web añadiendo las líneas:

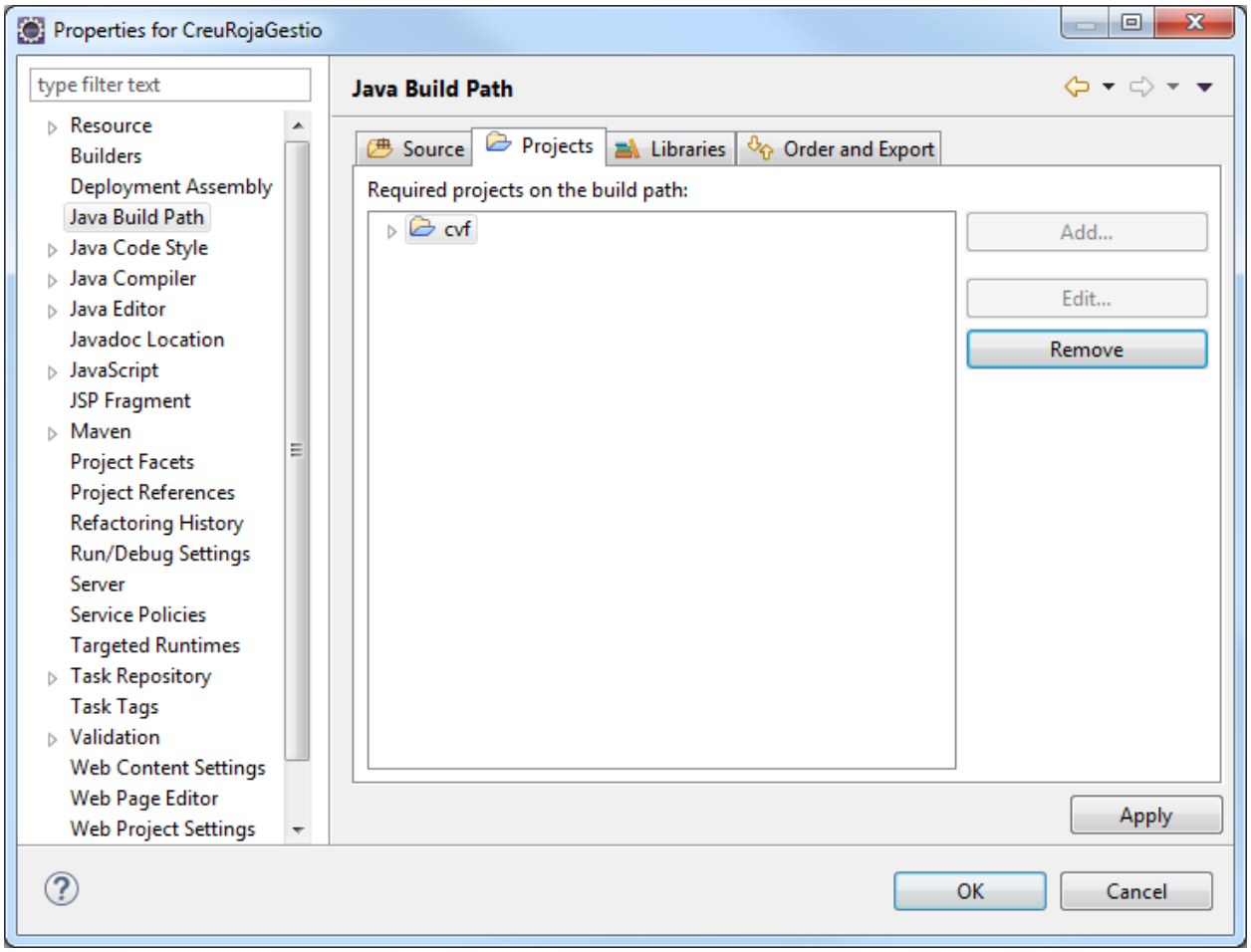
```
<filter>
      <filter-name>CvfFilterDispatcher</filter-name>
      <filter-class>com.achavero.cvf.core.CvfFilterDispatcher</filter-class>
</filter>

<filter-mapping>
      <filter-name>CvfFilterDispatcher</filter-name>
      <url-pattern>/*</url-pattern>
</filter-mapping>
```

y

```
<jsp-config>
  <taglib>
        <taglib-uri>http://cvf.achavero.com/taglibs/views</taglib-uri>
        <taglib-location>/META-INF/mytags.tld</taglib-location>
  </taglib>
</jsp-config>
```

Para modificar el Framework resulta muy incomodo tener que publicar el JAR a cada cambio y importarlo al proyecto a cada vez, así que durante el desarrollo hay que eliminar el JAR de la aplicación y configurar el proyecto debidamente.


Properties for CreuRojaGestio
type filter text
Resource
Builders
Deployment Assembly
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
JavaScript
JSP Fragment
Maven
Project Facets
Project References
Refactoring History
Run/Debug Settings
Server
Service Policies
Targeted Runtimes
Task Repository
Task Tags
Validation
Web Content Settings
Web Page Editor
Web Project Settings
Web Deployment Assembly
Define packaging structure for this Java EE Web Application project.
Source
Deploy Path
/src/main/java
WEB-INF/classes
/src/main/resources
WEB-INF/classes
/src/main/webapp
/
/src/test/java
WEB-INF/classes
/target/m2e-wtp/web-resc
/
cvf
WEB-INF/lib/cvf.jar
Maven Dependencies
WEB-INF/lib
Add...
Edit...
Remove
Revert
Apply
OK
Cancel



De esta forma podremos desarrollar el proyecto en paralelo al framework CVF.

8.2 Instalación/Ejecución aplicación Creu Roja Gestio (Andorra)

Primero hay que desplegar la aplicación, eso se puede hacer copiando el WAR

\bin\CreuRojaGestio\CreuRojaGestio.war a la carpeta webapps del Tomcat.

La aplicación necesita de una base de datos Mysql. Debemos crearla antes de intentar entrar en la aplicación:

mysql> create database CreuRojaGestio;

La aplicación es de acceso restringido, solo la pantalla de login es accesible sin estar autenticado. Para crear un usuario con login, podemos ir a la dirección:

http://localhost:8080/CreuRojaGestio/test/addLoginRootMaster.jsp

Esto creará un login con username: root y con password: master

Ya se puede entrar en la aplicación:

http://localhost:8080/CreuRojaGestio/login.view