Enero | 09

Framework para la capa de presentación de aplicaciones web.

José María Foces Morán
Ingeniería Informática

Josep María Camps Riba

14 de enero de 09

Índice

Índice de figuras.

Agradecimientos.

Gracias a mi tutor, Josep María Camps Riba por su acierto en la dirección de este trabajo –*logró que diera lo mejor de mi.*

Gracias a mi esposa, María Elisa, por su inspiración y paciencia.

Soli Deo Gloria.

Resumen.

La *arquitectura de interacción con el usuario* en aplicaciones web constituye la representación web de un gran número de servicios. El uso del modelo MVC-2 se ha transformado en la norma a la hora de diseñar aplicaciones web, lo cual, ha ocasionado la aparición de variados marcos de trabajo que articulan las aplicaciones web. En este trabajo final de carrera se diseña un marco de trabajo basado en MVC-2 y los patrones J2EE Front Controller, Application Controller, Service to worker, Context object, Command helper, View helper y otros, con el objetivo de comprender la complejidad de la *arquitectura de interacción con el usuario* web y de las tecnologías subyacentes. El lenguaje de programación y todas las plataformas empleadas son estandar y open source. El framework diseñado, denominado *magis*, ayuda a los desarrolladores organizando la presentación de contenidos, el acceso a las funciones y servicios de negocio, el flujo de páginas, el procesamiento de formularios, el manejo de errores y la gestión de estado.

Introducción

Descripción del TFC

El presente trabajo de fin de carrera consiste en el diseño y la implementación de un *framework* con el cual se puede generar la capa de presentación completa de una aplicación web. En concreto, se diseña y construye la funcionalidad básica de un *framework* similar a Struts. Se complementa el trabajo con un *webapp* a modo de ejemplo que ilustra la funcionalidad construida.

La ingeniería de software de hoy en día se basa en gran medida en *frameworks* orientados a objetos que promocionan la reutilización de diseños y código fuente. Estos sistemas de software están relacionados con un dominio concreto, en nuestro caso los entornos gráficos de usuario para clientes ligeros, en particular navegadores web. Con estos sistemas se pretende mejorar la productividad del diseño de aplicaciones web ofreciendo al programador un esqueleto de aplicación web, el cual, es susceptible de adaptarse a cada aplicación web concreta.

Todo *framework* contiene *puntos de flexibilidad* denominados *hot spots*, los cuales son clases o métodos abstractos que el programador debe implementar. Para instanciar un *framework* el programador ha de incluir el código de aplicación para cada *hot spot*. Al final, obtenemos una aplicación web en la que, a diferencia del uso de librerías, *el código del framework* no mutable, —denominado *frozen spots*—, es el que efectúa llamadas al código que constituye los *hot spots*. El código de cualquier objeto de librería es invocado por el código de la aplicación, justo al revés de la situación explicada anteriormente, a lo cual nos referimos con el término *inversión*

de control. Resumiendo, el código del framework usa el código de los *hot spots* mediante callbacks, los cuales, son disparados por la recepción de eventos.

El tipo de *framework* que me propongo diseñar es del tipo *white box*. Estos frameworks, en líneas generales, consisten en un conjunto de clases abstractas e interfaces que el programador conoce bien. El programador crea una instancia del *framework* creando nuevas clases mediante herencia o asociación. De este modo, todos aquellos aspectos que en el dominio considerado han de permanecer fijos, constantes estarán capturados en los métodos de las clases abstractas y, todos aquellos aspectos que varían de aplicación a aplicación son insertados por el programador mediante los mecanismos comentados. Los *frameworks* del tipo *black box* no requieren un gran conocimiento de la arquitectura del software por parte del programador y, la creación de instancias del *framework*, en este caso, se produce a través de *scripts* de configuración y de herramientas de instanciación que crean el código fuente de la aplicación usando la configuración como base.

El paradigma de desarrollo que emplearé será el de la *orientación a objetos* conjuntamente con la aplicación de *patrones de análisis* para el modelado del problema y *patrones de diseño* donde se puedan emplear soluciones probadas. Está comprobado que el uso de patrones de diseño es una gran ayuda en la documentación de *frameworks* dada su complejidad y el nivel de abstracción empleado en su lenguaje. El patrón de diseño más notable en el dominio de GUI's es el patrón conocido originalmente como MVC (Model-View-Controller) y es el que emplearé, teniendo en cuenta que, al tratarse de un framework para la capa de presentación de una aplicación web, habrá de sufrir algunas modificaciones dada la naturaleza del protocolo http, el cual soporta las transferencias de texto HTML entre el cliente y el contenedor de servlets: este protocolo es un protocolo sin estado. La forma de patrón mixto MVC que emplearemos se denomina MVC 2.

El patrón MVC 2, pensado para aplicaciones web, se diferencia del patrón MVC en los siguientes aspectos:

- La vista no es un observador (patrón Observer) del modelo en el mismo sentido en que lo es en MVC: no se registra en el modelo para que pueda recibir notificaciones de cambio de estado. La vista, una página jsp, cuando recibe una request, su servlet calcula su aspecto, el navegador lo renderiza y, cuando este proceso termina, el servlet de la jsp finaliza y, por tanto, no hay ninguna parte de la vista que pueda recibir eventos de ningún tipo.
- El objeto estrategia (patrón Strategy) es el servlet controlador, sin embargo, no está compuesto con la vista en el sentido clásico. Sin embargo, sí que una vista activa, dependiendo de la información de estado recibida en el momento en que se crea, puede decidir cambiar de controlador.
- El patrón composite *puede* estar presente en MVC 2 si el sistema de objetos de la vista está organizado en forma de árbol con posibilidad de notificaciones en cascada, como es el caso de JFC y su sistema de componentes reutilizables para la vista.

Las decisiones de diseño más importantes la tomaré de acuerdo con el estudio los *frameworks* disponibles en el momento actual y las tendencias, en concreto, el

diseño de la vista podría ser completamente reutilizable como JFC o estar basado en JavaBeans y JSP como es el caso de Struts.

Objetivos generales y específicos.

Me propongo diseñar una mezcla de clases abstractas que constituyan el esqueleto de una aplicación web cualquiera. Las clases concretas tendrá que proveerlas el programador de la aplicación. El *framework* contendrá las partes inmutables de la aplicación, mientras que las partes que diferencian una aplicación de otra formarán parte de las clases concretas; esta es una forma de separar lo que varía de lo que permanece fijo, lo cual, dota al diseño de una gran flexibilidad y promueve la reutilización. Un *framework* constituye un arquitectura de software muy compleja que, además, será difícil de documentar y de probar.

La documentación del *framework* es esencial para que un programador pueda comprenderlo. Para usar un *framework* de caja blanca como el propuesto un programador debe comprender la organización de las clases del mismo, porque, para insertar sus clases en el kernel del *framework* tendrá que aplicar herencia y composición a las clases provistas. Para lograr una comprensión adecuada del *framework* me propongo emplear patrones de diseño extensivamente, no copiarlos al pie de la letra y acompañarlos de las justificaciones de su uso correspondientes.

Las pruebas de un *framework* son ostensiblemente más complejas que las de una aplicación, particularmente si resulta necesario *depurar código,* porque, contamos con código perteneciente a los *hot spots* y código perteneciente a los *frozen spots* y, a pesar de que mediante precondiciones, etc. puede controlarse la conducta del *debugger,* esto no es en absoluto trivial.

Planificación con hitos y temporización.

Las tres fases más comúnmente empleadas en el ciclo de vida de un *framework* son el análisis del dominio, el diseño del *framework* y la instanciación de éste.

En la fase de *análisis del dominio* descubrimos los requisitos del dominio (GUI's para aplicaciones web) y posibles requisitos futuros. Para la captura de estos requisitos me propongo realizar una prospección de las tecnologías y conceptos que se aplican en el momento actual en al dominio mencionado y analizar experiencias anteriores documentadas, sistemas de software similares (JSF, Struts, Spring MVC y otros frameworks para la capa de presentación de webapps), experiencia personal en el desarrollo de un webapp basado en jsp, servlets y pojo/jdbc y normas industriales relevantes que apliquen al dominio considerado.

En la fase de *diseño del framework* se definen las abstracciones del *framework* y se modelan los *hot spots* y los *frozen spots.* Emplearé UML para realizar este modelado aplicando extensivamente *patrones de diseño*, en particular el patrón estructural mixto MVC 2 y, subsiguientemente *Observer, Strategy* y *Composite*.

En la fase de instanciación implementaré los *hot spots* para hacer posible la construcción de una aplicación web basada en el modelo de 3 capas, aunque, en su

momento estudiaré la posibilidad de usar un servidor de aplicaciones y realizar un sistema de software correspondiente a un modelo de 4 capas.

Las fases mencionadas se pueden ajustar perfectamente a la metodología RUP (Rational Unified Process), en concreto, aplicaré *pequeños incrementos* que me permitan comprobar el progreso del sistema evitando a toda costa análisis extensivos que no responden a los requisitos del sistema o diseños excesivamente complejos a o fallidos.

- **PEC 2 (5/noviembre)**: Entrega de un estudio detallado de las **tecnologías** y tendencias actuales en el dominio objeto de estudio, la **especificación** de requisitos y una versión del **análisis** muy próxima a la definitiva o la definitiva.
- **PEC 3 (17/diciembre):** Diseño orientado a objetos usando patrones de diseño y la implementación de una prueba de concepto del kernel del framework junto con un webapp basado en el kernel, jsp, JavaBeans y Pojo/jdbc(modelo).
- **PEC 4 (14/enero/09):** Entrega de la memoria del proyecto, el proyecto de *framework software* completo y al menos un webapp de prueba.

Productos obtenidos

Los productos resultantes incluyen el conjunto de componentes que conforman el *framework* y una aplicación web de ejemplo. De ambos productos, se entrega el código fuente de las clases, interfaces, páginas web y ficheros de configuración. Todos los ficheros fuente están comentados y, aquellos métodos o secciones del código de especial relevancia han sido comentados de forma más detallada y extensa. Junto con los fuentes, se ha incluido también la documentación javadoc.

Descripción del resto de capítulos de la memoria

En los capítulos siguientes se explica el proceso de estudio, análisis, diseño, e implementación seguido.

- **Estudio sobre frameworks.** En este capítulo se lleva a cabo un estudio de los frameworks actuales que cubren la capa de presentación y cuales son las tendencias actuales en el ediseño de estos sistemas de software.
- **Especificación de requisitos.** En este capítulo se explica cómo se llevó a cabo la recogida de requisitos y el análisis orientado a objetos de los mismos.
- **Diseño e implementación del framework.** En este capítulo se explica el desarrollo de los componentes básicos de Magis.

- **Implementación final y conclusiones.** En este capítulo se explica de forma somera las dificultades de diseño y realización de los componentes finales de Magis junto con las conclusiones finales.

Estudio sobre frameworks y especificación de requisitos

Introducción.

En el apartado de la PEC 1 dedicado a los objetivos de este TFC definíamos *framework* y tratábamos de clasificar los diferentes tipos de frameworks. Ahora, estudiaremos los *frameworks* para la capa de presentación de aplicaciones web que están disponibles en el presente, de este modo, podremos conocer los límites actuales de estas tecnologías y, así, estaremos en disposición de formular una *especificación de requisitos* que resulte relevante y realista teniendo en cuenta los condicionantes típicos que afectan al desarrollo de un TFC, uno de los cuales, el tiempo disponible, es limitado e inextensible.

Este escenario de partida es particularmente representativo de una disciplina como la *ingeniería de software* en la que no contamos con herramientas de "cálculo" de diseños de software; generalmente los ingenieros de software combinan un poco de teoría con una extensísima experiencia. Este TFC, el diseño de un *framework,* deberá satisfacer un conjunto de requisitos funcionales y no funcionales y cumplir ciertos criterios técnicos y otros no técnicos y, como tal, será representativo de la forma en la que la sociedad progresa y se desarrolla: reutilizando los diseños ya probados en escenarios de aplicación futuros. Recordemos que, en la PEC1, definimos *framework*, de forma resumida como un conjunto de clases abstractas e interfaces que constituyen el esqueleto de un conjunto de aplicaciones software posibles: la *comunalidad* queda capturada en las clases e interfaces y la variabilidad resulta ser

responsabilidad del desarrollado que usa el *framework*. En este estudio sobre los *frameworks* más importantes en el momento actual nos centraremos en seis de ellos, los que hemos considerado sobresalientes por una variedad de motivos que explicaremos en su momento, pero, queremos resaltar que el número de estos supera los cuarenta y, consecuentemente nos preguntamos porqué existe un número tan elevado. Sin ofrecer una justificación exhaustiva, propondremos que el hecho de no existir un "cálculo de diseños de software", las comunidades de desarrolladores y algunos desarrolladores individuales, con especial talento y pasión, han producido un gran número de soluciones que reflejan los compromisos que surgen en cualquier diseño: flexibilidad, simplicidad, facilidad, potencia, etc.

Frameworks disponibles en la actualidad.

En la Wikipedia podemos encontrar una tabla de los frameworks que se encuentran disponibles en el momento actual, la figura siguiente es sólo la primer página:

Please help improve this incomplete table by expanding it. Further information might be found on the talk page or at requests for expansion. (September 2007)

Project	Language	Ajax	MVC framework	MVC Push/Pull	i18n & l10n?	ORM	Testing framework(s)	DB migration framework(s)	Security Framework(s)	Template Framework(s)	Caching Framework(s)	Form Validation Framework(s)
AIDAWeb	Smalltalk	Yes, Prototype, script.aculo.us	Yes	Yes		Yes, Gemstone/S, GLORP, ...	Yes, SUnit		Yes			
Ajile	JavaScript	Yes	Yes	Push & Pull	Yes		Yes, jsUnit			Yes	Yes	
Akelos PHP Framework	PHP	Yes, Prototype, script.aculo.us	Yes, Active record pattern	Push	Yes	Yes, Active record pattern	Yes, Unit Tests	Yes		Yes	Yes	Yes
Apache Struts	Java	Yes	Yes	Push	Yes	Yes	Yes, Unit Tests			Yes, Jakarta Tiles framework		Yes, Jakarta Validator framework
Struts2	Java	Yes	Yes	Push & Pull	Yes	Yes	Yes, Unit Tests			Yes		Yes
Aranea MVC	Java	Yes		Pull	Yes	Yes						
BFC	.NET	Yes	Yes, but not mandatory	Push & Pull	Yes	Yes, through active data dictionary	Yes, Unit Tests	Yes, SQL Server, Oracle, DB2, Sybase, MySQL	Yes, security groups and rules	Yes	Yes, metadata and result sets	Yes, data dictionary-driven
CakePHP	PHP	Yes, Prototype, script.aculo.us	Yes, Active record pattern	Push	Yes, Development branch	Yes, Active record pattern	Yes, Unit Tests	Yes	Yes, ACL-based	Yes	Yes, Development branch	Yes
Camping	Ruby	No	Yes	Push	No	Yes, Active record pattern	Yes, via Mosquito	Yes	No	Yes	No	No
Catalyst	Perl	Yes, multiple (Prototype, Dojo...)	Yes	Push in its most common usage	Yes	Yes, multiple (DBIx::Class, Rose::DB...)	Yes[1]		Yes, multiple (ACL-based, external engines...)	Yes, multiple (Template::Toolkit, HTML::Template, HTML::Mason...)	Yes, multiple (Memcached, TurckMM, shared memory,...)	Yes, multiple (HTML::FormValidator,...)
CJAX	PHP	Yes,	Yes	Push & Pull	Yes	Yes, (framework extension)	Yes Unit Tests		Yes, framework extension	Yes, framework extension	Yes, framework extension	Yes, framework extension
CherryPy	Python				Yes		No, because unittest and doctest are standard Python modules			Yes, CherryTemplate	Yes	
Click Framework	Java	Yes	Yes	Pull	Yes	Yes, integrates with Hibernate and Cayenne				Yes, Velocity and JSP		Yes, built-in validation
CodeIgniter	PHP	Yes, Framework extension	Yes, Modified Active record pattern	Push	Yes	Yes, framework extension	Yes, Unit Tests	No	Yes	Yes	Yes	Yes
ColdBox Framework	ColdFusion	Yes, various libraries	Yes	Push & Pull (via Viewlets)	Yes	Yes, Transfer & Reactor	Yes, Unit Tests	No	Yes, via plugins or interceptors		Yes ColdBox Cache Manager and externally pluggable.	Yes

Fig. 1. La primera página de la lista de Web frameworks de la Wikipedia.

Uno de los frameworks de más éxito es Struts, un proyecto de la fundación Apache (Apache Software Foundation, ASF), todavía hoy sigue siendo uno de los más demandados en las ofertas de trabajo (Ve fig. 2 a continuación). Struts ha servido de base a otros frameworks de la ASF, tales como Struts 2, JSF, Tapestry, Wicket, etc.

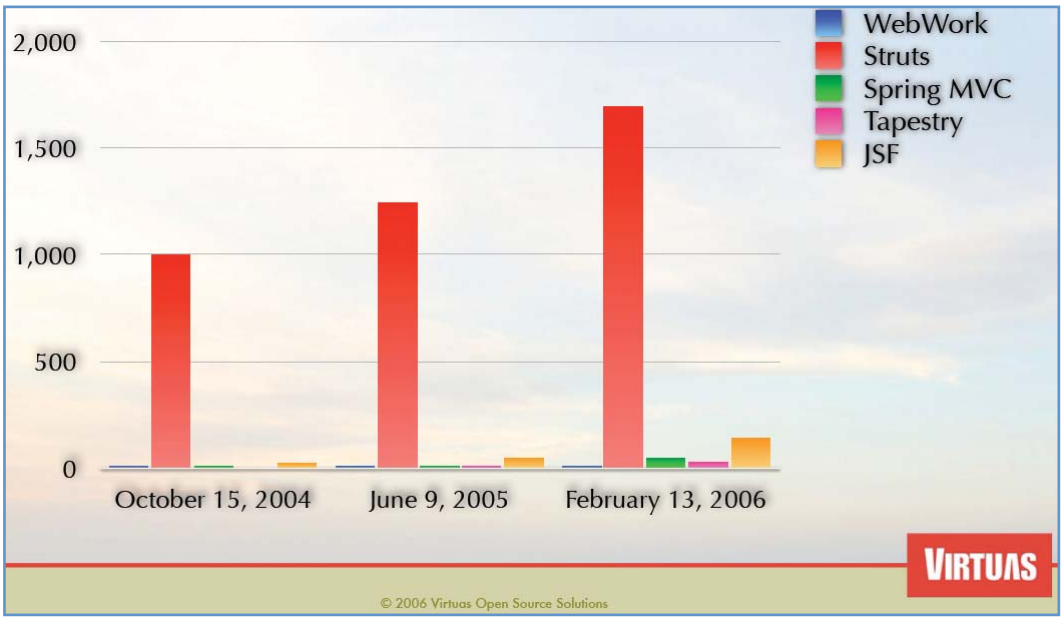

Fig. 2. Ofertas de empleo y web frameworks.

Además de los frameworks mencionados estudiaremos también Spring MVC y Stripes.

La elección de un framework.

Hay un cierto número de observaciones de alto nivel que pueden ser aplicadas a la elección de un framework. En principio **un framework puede incluir una librería de componentes o, ambos elementos pueden encontrarse separados**, es decir, en este último caso el framework ofrecería la columna vertebral de la aplicación y la infraestructura y, la librería de componentes provee servicios básicos construidos sobre esta infraestructura. Puede resultar difícil distinguir si un determinado producto constituye un framework, un API, con librería de componentes o no, etc. pero, lógicamente, al final lo que cuentan son los usos y ventajas que ofrece el producto. Los aspectos más notables que pueden ayudar a establecer la adecuación de un framework son los siguientes:

- **Licencia.** Un factor limitador de lo que se puede hacer y lo que no con un determinado software -sus limitaciones legales. Los dos tipos de licencia más comunes son la licencia de software abierto y la propietaria. En el último caso, habrá que tener muy en cuenta el precio y los posibles *royalties* que pueden afectar a cada aplicación producida.
- **Complejidad.** Un framework debe ser simple para que pueda ser aprendido con facilidad, pero, al mismo tiempo deberá ofrecer suficientes características que permitan un uso efectivo del mismo. Un framework siempre supone implícitamente una teoría del dominio de aplicación, en nuestro caso la capa de presentación web; es conveniente conocer esa teoría antes de lanzarse a evaluar sus características específicas, de hecho, si ofrece un gran número de características pero los patrones de alto nivel en los que se basa son fallidos o su estructura no escala adecuadamente, puede que estas deficiencias no se detecten hasta una fase avanzada en el ciclo de vida.

- **Patrones de diseño.** Un buen framework no es simplemente un conjunto de clases abstractas a las que podemos aplicar herencia, si no un diseño: un conjunto de patrones que define cómo las instancias colaboran entre sí para construir una aplicación final. Es esencial conocer los patrones y las relaciones entre ellos, mucho más importante que saber cómo funciona o cómo construir aplicaciones con él. Es esencial, por tanto, contar con diagramas UML que documenten el diseño en el que se basa el framework.
- **Disponibilidad de ejemplos.** Al menos se debe poder contar con una aplicación final y relevante.
- **Documentación.** La documentación es una parte esencial de cualquier framework y, no basta con la documentación basada en JavaDoc, se ha de disponer de los siguientes documentos:
  - Lista de características.
  - Libro de consulta rápida (Cookbook).
  - Documentación de diseño: Los patrones de diseño utilizados en el framework y las justificaciones de su uso.
  - Status de la documentación.
  - Documento de orientación acerca del futuro desarrollo del framework.
  - Código fuente.
- **Soporte.** Hay que conocer que tipos de soporte se encuentran disponibles en el mercado, quién lo provee, cuáles son los procedimientos, cuál es el coste, cuál es el nivel de los expertos, cuál es el tiempo de respuesta, etc.
- **Estándares.** Se han de considerar tanto estándares formales como estándares de facto.
- **Soporte de componentes.** Si el framework soporte una librería de componentes complementaria o no y, en el primer caso, cuál es su enfoque de integración con componentes.
- **Plantillas o transformaciones.** En el caso de frameworks para la capa de presentación de aplicaciones web nos encontramos con frameworks que usan plantillas y frameworks que usan transformaciones. Los motores de presentación basados en **plantillas** incluyen marcadores especiales (tags) en el lenguaje de marcado, estas etiquetas, en tiempo de ejecución estarán asociadas con el contenido dinámico para el que fueron pensadas. Si las etiquetas están correctamente diseñadas, los diseñadores gráficos podrán usan DreamWeaver o FrontPage, por ejemplo, para el diseño de las páginas web sin problemas con los tags. El enfoque basado en **transformaciones** controla la interfaz de usuario aplicando una serie de cambios o transformaciones a la salida xml producida por la aplicación. Cuando el usuario pide una página dinámica, la lógica de la aplicación se ejecuta como antes y produce el contenido dinámico en XML, el cual, es transformado usando un motor **XSLT** y traducido a HTML, el cual, finalmente será entregado al navegador web.

Finalmente podríamos preguntarnos porqué adoptar un framework y no diseñarlo "en casa". En este caso debemos considerar un cierto número de factores como los anteriores, particularmente, algunas organizaciones optan por construir si las aplicaciones han de incorporar *secretos empresariales*.

Objetivos del estudio y método.

Realizamos este estudio para comprender qué área del espacio de problemas de desarrollo web motiva el diseño de un nuevo framework, no pretendemos un estudio estadístico exhaustivo de las ganancias de productividad derivadas del uso de los diferentes frameworks. Comenzamos con el framework que es nuestra base de estudio, Struts, explicando cuáles son sus características más notables, y, continuamos con Struts 2, JSF, Tapestry, Wicket y, finalizamos con Spring MVC y Stripes. Al ser Struts la base conceptual de este TFC, le dedicaremos una atención especial y un estudio más exhaustivo que al resto.

Struts. Struts forma parte del proyecto Jakarta de la ASF, del cual también podemos resaltar Tomcat, Ant y Velocity. El desarrollo inicial de Struts lo llevaron a cabo 30 desarrolladores entre los años 2000 y 2001. El arquitecto principal es Craig. R. MacClanahan, quien también fue el arquitecto principal de Tomcat y del paquete de Web Services para Java. En el presente, Craig MacClanahan es líder de desarrollo de JSF y de J2EE en Sun Microsystems.

Struts es software abierto bajo la licencia ASF, al igual que otros proyectos esenciales para el desarrollo web liderados por empresas "osadas" como Sun, IBM y Apple que promueven la colaboración y el conocimiento compartido y que producen un beneficio social que se extiende a mercados completos: estos frameworks basados en software abierto son una muestra palpable de la filosofía *yo-gano-tu-ganas*.

La pieza central de Struts es un controlador MVC2. La arquitectura de este controlador responde a los patrones J2EE denominados Front Controller (Ver fig. 3) y Application Controller (Ver fig. 4). La realización del patrón Front Controller la lleva a cabo el programador extendiendo struts.ActionServlet (un HttpServlet , en ocasiones una JSP). Este servlet actúa como único punto de control central para la capa de presentación de la aplicación web que se está desarrollando. Todas las solicitudes que se entregan al WebApp pasan por el controlador frontal, el cual, gestiona su despacho a otras partes de la aplicación. No es muy común implementar solamente el controlador frontal, normalmente, organizamos el controlador central para que lleve a cabo sus responsabilidades delegando en otro patrón denominado Controlador de la Aplicación, que en Struts recibe la denominación *request processor*.

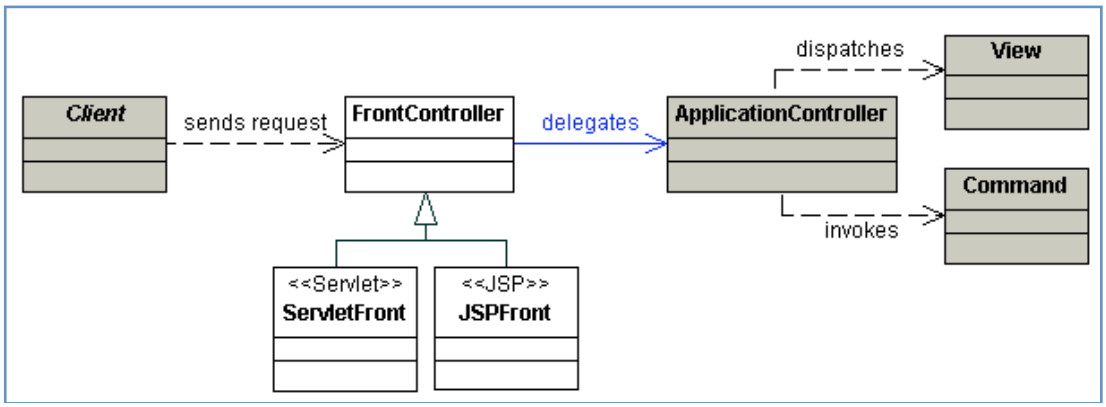

Fig. 3. Patrón J2EE front controller.

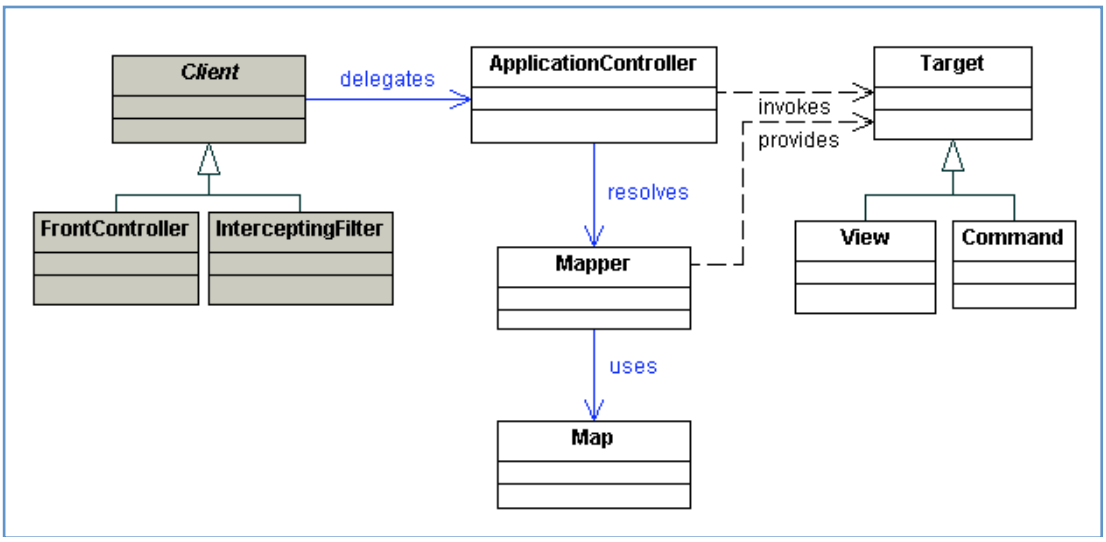

Fig. 4. Patrón J2EE application controller.

La estructura controlador frontal/controlador de aplicación confiere a Struts las siguientes características:

- Control declarativo. Creación de correspondencias entre solicitud, objetos que validan los parámetros de la solicitud, objetos que invocan al modelo y vistas que calculan la presentación. Estas correspondencias se construyen declarativamente en XML, el controlador las lée al inicio y mantiene una copia de ellas en memoria, la cual, se emplea para resolver cada solicitud.
- Despacho automático de solicitudes. Un objeto *action* cuyas responsabilidades consisten en invocar el modelo y devolver al *controller* un objeto ActionForward que indica qué *view* ha de desplegar los resultados devueltos por el modelo o bien un error. Esta característica confiere a Struts otra capa de abstracción entre el controlador y la view (bajo acoplamiento).
- DataSources. Struts puede ofrecer gestión de fuentes de datos.
- Etiquetas (Custom Tags). Struts contiene un gran número de tags Jsp muy útiles.
- Internacionalización.

- Validación declarativa. La validación no se ha de realizar obligatoriamente en los *beans* de formulario, programáticamente, puede realizarse declarativamente en un fichero XML, lo cual, promueve la reutilización a través de configuración.
- Manejo global de excepciones. Se trata de un mecanismo de control de errores que se especifica declarativamente.
- Plug-ins. El framework de validación de Struts, por ejemplo, se activa a través de un plug-in que el desarrollador puede decidir incorporar o no hacerlo.

Como aspecto negativo de Struts resaltamos que se trata de un framework de aplicación más que un *framework de componentes de interfaz de usuario*. JSF, por ejemplo, aplica el patrón *composite* a la view, de forma que cada componente de ésta está formado por otros componentes y la entrega de notificaciones y los eventos se propagan en una jerarquía de componentes de forma parecida a como ocurre con Java 2 Swing.

Struts 2. Struts 2 constituye la nueva generación de frameworks MVC-2 que ofrecen una mejor productividad gracias a que usan menos configuración XML, más convenios inteligentes y una arquitectura modular con un grado bajísimo de acoplamiento. Struts 2 presenta las siguientes características:

- En Struts 2, a diferencia de Struts una acción no necesita extender clase alguna o implementar ninguna interfaz, se puede considerar como un POJO (Un objeto Java sin características especiales).
- El acceso a los datos de la *view* y la transferencia a la *action* se emplean setters y getters.
- Las transformaciones de los datos las lleva a cabo Struts 2 y el desarrollador puede definir nuevas conversiones.
- Permite la inyección de dependencias (patrón de diseño) lo que posibilita la integración con las beans gestionadas de Spring, por ejemplo.
- Las configuraciones de este framework están disponibles como anotaciones Java, lo cual libera al desarrollador de la escritura de XML.
- Los interceptores juegan un papel central en los servicios básicos de Struts

Struts sigue siendo el framework número 1 en las ofertas de empleo, Struts 2 va creciendo poco a poco, pero, otros frameworks como JSF están creciendo mucho dado su altísimo potencial debido al prestigio de Sun, la calidad del software y el abanico de posibilidades que ofrece junto a su integridad conceptual. Revisaremos JSF a continuación.

JSF. Java Server Faces es un framework basado en componentes diseñado por Sun Microsystems. JSF es un intento de proveer un framework parecido a Swing pero como GUI Web. Los widgets Swing son componibles, orientados a eventos y pueden cambiar su apariencia de acuerdo con el *look-and-feel*. JSF hace uso de MVC-2 (Ver fig. 5 a continuación), la adaptación del patrón arquitectónico MVC a los GUIs basados en aplicaciones web.

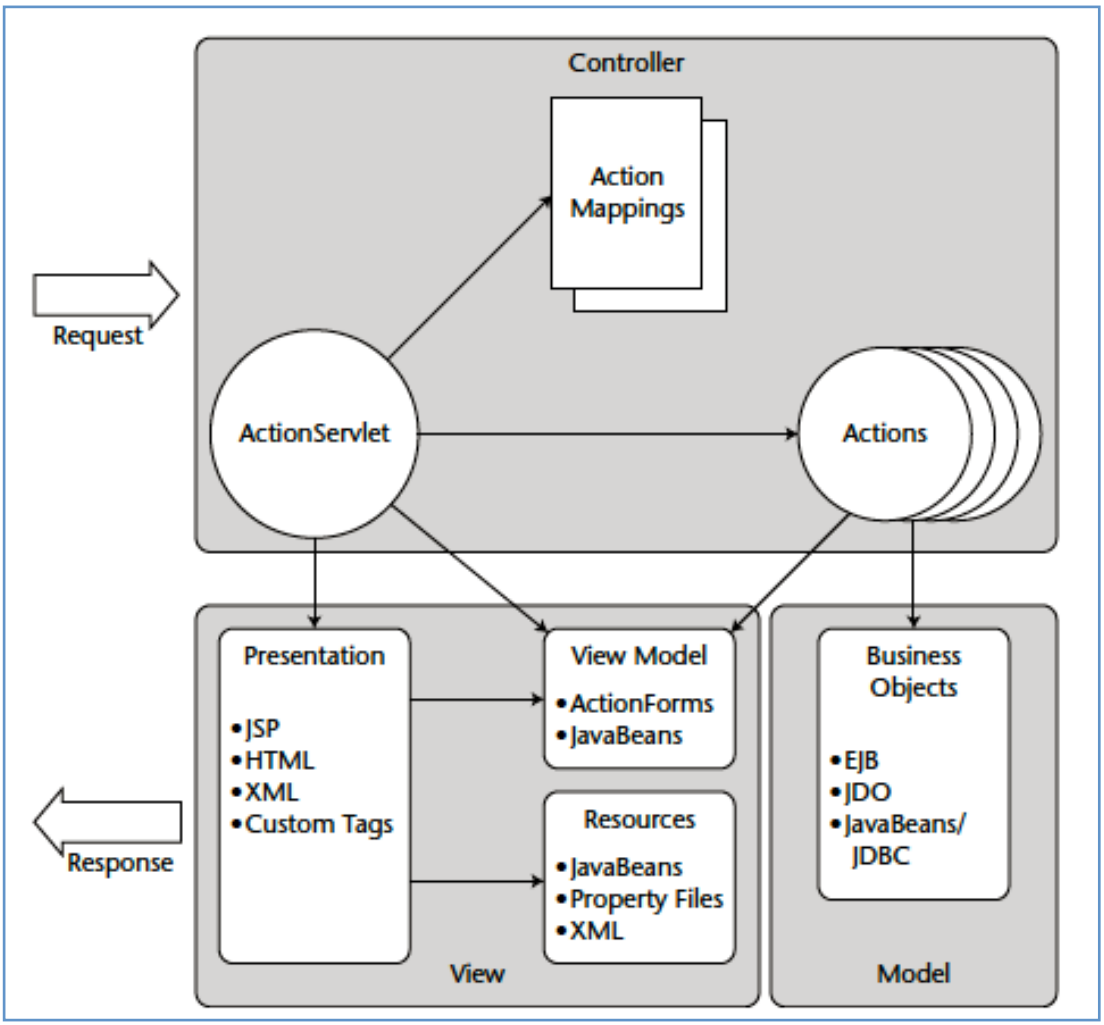

Fig. 5. El modelo MVC-2 de JSF.

En JSF la *view* está constituida por un árbol de componentes, en cambio, la *view* en Struts está basada en páginas JSP. El árbol de componentes JSF se refiere a una jerarquía de componentes GUI que forman parte de una página JSP u otra forma de presentación. Estos componentes siguen el patrón de diseño llamado *composite*. En JSF, a diferencia de Swing, no se distingue entre componentes de nivel alto y componentes normales. JSF provee la interfaz UIComponente que es implementada por todos los componentes. JSF contiene un cierto número de componentes básicos tales como UIForm y UIInput, pero, el desarrollador puede crear los componentes que desee implementando UIComponent o extendiendo UIComponentBase. JSF nos permite anidar componentes para conformar grandes componentes compuestos.

A diferencia de Struts, en JSF encontramos una noción de *evento* bien definida. Los eventos JSF siguen el modelo de eventos basados en JavaBeans que incluye clases evento y clases receptoras de eventos (listeners) fuertemente tipadas. El conjunto normal de componentes JSF emiten eventos *Value Changed* y *Action;* cualquier otro componente, ya sea GUI o residente en el servidor puede registrarse para recibir eventos procedentes de componentes del GUI web.

A modo de resumen, sobre JSF, podemos decir que se trata de la tecnología oficial de presentación en la capa web de JEE. JSF incluye un conjunto de componentes GUI

predefinidos, un modelo de programación basado en eventos y la posibilidad de añadir componentes de otras fuentes. JSF ha sido concebido como un *framework* extensible y fácil de usar, es rápido, es estándar y ofrece un framework de navegación muy rico.

Tapestry. Este framework ha sido desarrollado como proyecto en ASF. Tapestry está basado en componentes , el framework se ocupa de la validación de las entradas, internacionalización, gestión de estado y persistencia, correspondencia de parámetros de la solicitud y construcción de URLs. Las características más notables de tapestry son las siguientes:

- Es un framework robusto, y puesto que está basado en estándares corre en cualquier contenedor.
- La cantidad de configuración XML es mínima. En Tapestry se crea la aplicación en términos de objetos y métodos y propiedades de esos objetos.
- El desarrollo con Tapestry resulta amigable y ha sido diseñado teniendo presente la escalabilidad y la seguridad e incluye soporte para Ajax y excepciones.
- Tapestry mejora ostensiblemente la productividad de los desarrolladores con su exclusiva característica de recarga de clases. Con Tapestry uno cambia el código fuente y puede visualizar los resultados inmediatamente: no hay que redesplegar el webapp ni rearrancar el contenedor. Los informes de excepciones son detallados e incluyen incluso sugerencias.
- Sorprendentemente, las páginas de Tapestry son páginas XHTML, simplemente.
- Se integra fácilmente con Hibernate, Spring y otros.

Wicket. Este framework ha sido desarrollado como proyecto en ASF y, parece estar basado de algún modo en Tapestry. Las características más notables de este framework son las siguientes:

- Modelo de *componentes* que recuerda al de Swing (De forma parecida a JSF y a Tapestry).
- Java + XHTML: se puede apalancar una gran cantidad de conocimiento.
- Separación de responsabilidades muy clara: Java + XHTML en la *view*.
- Seguro. Por defecto los URLs no manifiestan información sensible relativa a la sesión.
- Soporte de *clustering* escalable y transparente.
- Soporte transparente del botón de retorno (back).
- Componentes reutilizables en la view. Se pueden crear grupos de componentes como una unidad reutilizable.
- Validación de formularios simple, flexible y localizable (internacionalización).
- Manejo transparente de los atributos de la sesión.
- Extensibilidad a través de factorías y métodos factoría.

- Manipulación programática de los atributos.
- Conversiones automáticas.
- Componente árbol que permite la navegación y la selección de nodos, directamente después de la instalación.
- Gran número de ejemplo.

Spring MVC (SMVC). Este framework basado en software abierto resultó de la publicación de un libro titulado Expert One-on-One J2EE Design and Development en 2002 escrito por Don Johnson. Hoy en día está siendo desarrollado por unos 40 miembros, algunos de ellos trabajando a tiempo completo para Interface21.

La clase central es un DispatcherServlet que despacha solicitudes a clases que gestionan la solicitud (HttpRequest), incluye correspondencias entre las solicitudes y sus manejadores, resolución de las vistas, internacionalización así como soporte de subida de ficheros. Un diferencia fundamental con Struts es que SMVC cuenta con un conjunto de clases abstractas controlador, por ejemplo, si no contamos con un formulario, no es necesario implementar un controlador de formulario como en Struts.

SMVC permite usar cualquier objeto Java como comando u objeto formulario: no hay necesidad de implementar ninguna interfaz o extender ninguna clase específica del framework. El acoplamiento de datos es muy flexible y los problemas de concordancia de tipos se tratan como errores de validación no como errores de sistema. Otras características destacables de SMVC son las siguientes:

- La resolución de la vista es extremadamente flexible.
- Se puede integrar con cualquier renderizador de la vista directamente: JSP, Velocity.
- La integración de otros frameworks basados en MVC-2 es muy fácil a través de un mecanismo basado en plug-ins.
- Separación clara de roles: *controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver, etc.*
- Configuración directa y muy potente.
- Adaptabilidad: se puede usar cualquiera de las clases controlador disponibles: normal, comando, formulario, *wizard,* multiacción o uno customizado.
- Código de la capa de negocio reutilizable.
- Los errores de falta de concordancia de tipos son errores de validación, a nivel de aplicación, no errores del sistema.
- La correspondencia entre manejadores y la resolución de la vista son customizables.
- Integrable con cualquier tecnología de renderización de la vista (JSP, Velocity, etc).
- La resolución de temas y del lenguaje nacional es customizable.

- Librería de tags JSP customizadas, lo que facilita el desarrollo de páginas JSP.

Stripes. Stripes es un framework basado en software abierto que trata de simplificar al máximo el desarrollo de aplicaciones web: según sus autores, Struts padece de ciertos fallos arquitectónicos y, SMVC, siendo mucho mejor, aún requiere grandes cantidades de configuración y parece que exige que eldesarrollador aprenda un nuevo lenguaje. Se trata de un framework basado en *acciones* típico, aunque muy avanzado y simple. Las características más notables de este framework son las siguientes:

- Las ActionBeans son autodescubiertas por el framework y son configuradas a través de anotaciones java (A partir de Java 5).
- Contiene un potente motor de enlace capaz de construir complejas redes de objetos a partir de la *request http*.
- Fácil sistema de validación y de conversión.
- Soporta la recepción de múltiples eventos desde un mismo formulario.
- Subida de ficheros transparente.
- Soporta desarrollo incremental, no es necesario escribir la ActionBean para probar una página JSP.
- Evita la dispersión de puntos de configuración permitiendo al desarrollador concentrar la configuración de los objetos junto con el propio código de las clases a medida que escribe los fuentes gracias al uso de anotaciones java.
- "Un mix de ordenador Apple, tele Sony y un BMW 335xi con acabado M".
- Aprovecha otras características notables de Java 5 como *generics* y las anotaciones que ya hemos mencionado, supuestamente para evitar la configuración a través de ficheros externos. Esta característica no siempre será deseable, porque obliga a que la configuración del webapp la lleva a cabo un desarrollador siempre impidiendo que ciertos cambios los lleva a cabo, por ejemplo, un administrador de sistemas o aplicaciones.

Especificación de requisitos.

Introducción.

Uno de los requisitos no funcionales del presente TFC es su modelado usando el patrón arquitectónico MVC. La parte más volátil de una aplicación de software interactiva es la interfaz de usuario por ser ésta la parte que se comunica directamente con los usuarios y, por tanto, es también la parte más visible a éstos. Además de los cambios diarios que tienen lugar en los requisitos del sistema, nos encontramos que frecuentemente el sistema de software ha de poder ser ejecutado en múltiples sistemas operativos, lo cual, ocasiona que ha de soportar múltiples aspectos distintos, es decir, que complica rápidamente lo que en principio se pudiera haber considerado simplemente como una interfaz de usuario.

En las aplicaciones monolíticas, las cuales presentan un alto grado de acoplamiento, realizar cambios de la naturaleza explicada puede tornarse una tarea extremadamente difícil y producir verdaderas cascadas de errores. Normalmente, la solución adoptada consiste en mantener diferentes versiones de las aplicaciones dependiendo de los requisitos de presentación impuestos por cada grupo de usuarios y sus plataformas de ejecución. Finalmente, estos escenarios ocasionan que las versiones que satisfacen los diferentes requisitos de presentación pierdan su sincronía en cuanto a requisitos funcionales también. Se hace necesario por tanto el empleo de un enfoque radicalmente distinto al de las aplicaciones monolíticas.

La solución ofrecida por el diseño desacoplado basado en el patrón MVC ofrece una gran flexibilidad al dividir la aplicación en los componentes vista, controlador y modelo debidamente interrelacionados entre sí mediante interfaces uniformes. El uso de MVC se recomienda si se da cualquiera de las condiciones siguientes:

- Se desea ofrecer diferentes representaciones de la misma aplicación.
- Se desea ofrecer diferentes aspectos para la interfaz de usuario sin afectar al resto de la aplicación.
- Los eventos ocasionados por la interacción con el usuario deben actualizar los datos inmediatamente o bien otros componentes de la interfaz de usuario sin afectar al resto de la aplicación.
- Se desea reutilizar algunos componentes de la interfaz de usuario independientemente de los datos de la aplicación.

Las consecuencias más notables, derivadas del uso de esta patrón, son las siguientes:

- Pueden existir múltiples vistas asociadas a un mismo modelo.
- Diferentes *look-and-feel.*
- Las views y los controllers son *observadores* del modelo.
- EL MVC promueve la reutilización de componentes de la interfaz de usuario.
- El uso del MVC en ciertas aplicaciones conlleva una alto grado de *overhead.*
- Algunos frameworks organizan la view aplicando el patrón *composite.*

Las aplicaciones web utilizan como transport el protocolo http, un protocolo sin estado lo cual conduce a una modificación del protocolo MVC denominada MVC2. Cuando un servlet recibe una solicitud http, éste interactúa con el modelo y determina cual es la vista (p. ej. una página jsp) que ha de presentar los resultados contenidos en la respuesta http. Como parte de la generación de la respuesta, la vista puede necesitar interactuar con el modelo. El escenario que tiene como base tecnológica las páginas jsp y los servlets se puede resumir en los siguientes aspectos:

- Normalmente encontramos un único servlet controlador para la aplicación web completa. Este servlet se transforma, entonces, en el único punto de acceso central de la aplicación, lo cual ayuda a la gestión de la seguridad, de los usuarios y del flujo de operaciones. A este tipo de servlet se le denomina front controller siguiendo la denominación del patrón de diseño front controller.
- El front controller normalmente utiliza componentes de ayuda en los cuales delega el procesamiento de comandos, el flujo de la aplicación, el cálculo de la vista, etc. El despacho de solicitudes en Struts, por ejemplo, es delegado en un RequestDispatcher.
- Los componentes de la vista puede que no interactúen directamente con el modelo, entonces, es el controlador o uno de sus componentes de ayuda el que recibe los resultados en forma de objetos java (JavaBeans o Data Transfer Objects) y los envía a la view para que esta calcule el código HTML final que

será renderizado por el navegador web. El envío suele hacerse en la sesión http o en el objeto request http enviado.

Conviene que tengamos presente que en MVC2 no hay ningún mecanismo que permita que los cambios en el modelo se propaguen inmediatamente a las vistas y controladores registrados. Cada vista se reconstruye desde cero para cada respuesta, no son, por tanto actualizadas. Una vez que el controlador recibe una solicitud http, no puede interactuar con esa vista de ningún modo.

El modelo que nos proponemos seguir en este TFC es el que emplea Struts, el modelo MVC2. Este es un requisito no funcional esencial y, aunque propiamente afecta a los modelos dinámicos de datos propios de la fase de diseño, ha sido incluido en el análisis como si fuera un requisito más. Tomando como base el siguiente diagrama estructural de Struts, generaremos un diagrama estático de análisis del TFC, el cual, capturará los conceptos fundamentales del Framework derivados de la siguiente lista de requisitos procedentes de Struts:

- **Control declarativo**. Posibilidad de creación de mapas declarativos entre el URL de un solicitud http, sus objetos de validación, objetos de invocación del modelo y la vista final.
- **Despacho automatizado de solicitudes http**. La ejecución de una acción debe producir un objeto que representará la vista a la que hay que efectuar el despacho. Esto crea otra capa de abstracción y reduce el acoplamiento controlador-view.
- **Validación declarativa**. Evitar al máximo realizar las validaciones de cada campo de los formularios a través de código java, el Framework debe poder detectar qué validador declarativo corresponde a cada campo de un formulario.
- **Tratamiento de errores declarativo.** A base de objetos error, el Framework debe encapsular los errores y las excepciones producidas.

Además, especificaremos algunos otros requisitos de importancia que iremos perfilándolos en cada iteración:

- **Custom tags.** Librería de componentes de la view que permita la reutilización de los mismos y facilita el diseño de páginas web sin necesidad de recurrir a scriptlets.
- **Plug-ins.** Construir una interfaz Plugin que permita mejorar la aplicación añadiéndole funcionalidad a medida que se necesite, siguiendo el principio OCP (Open for Extension, Closed for modification).
- **Internacionalización.**
- **Gestión de fuentes de datos.**

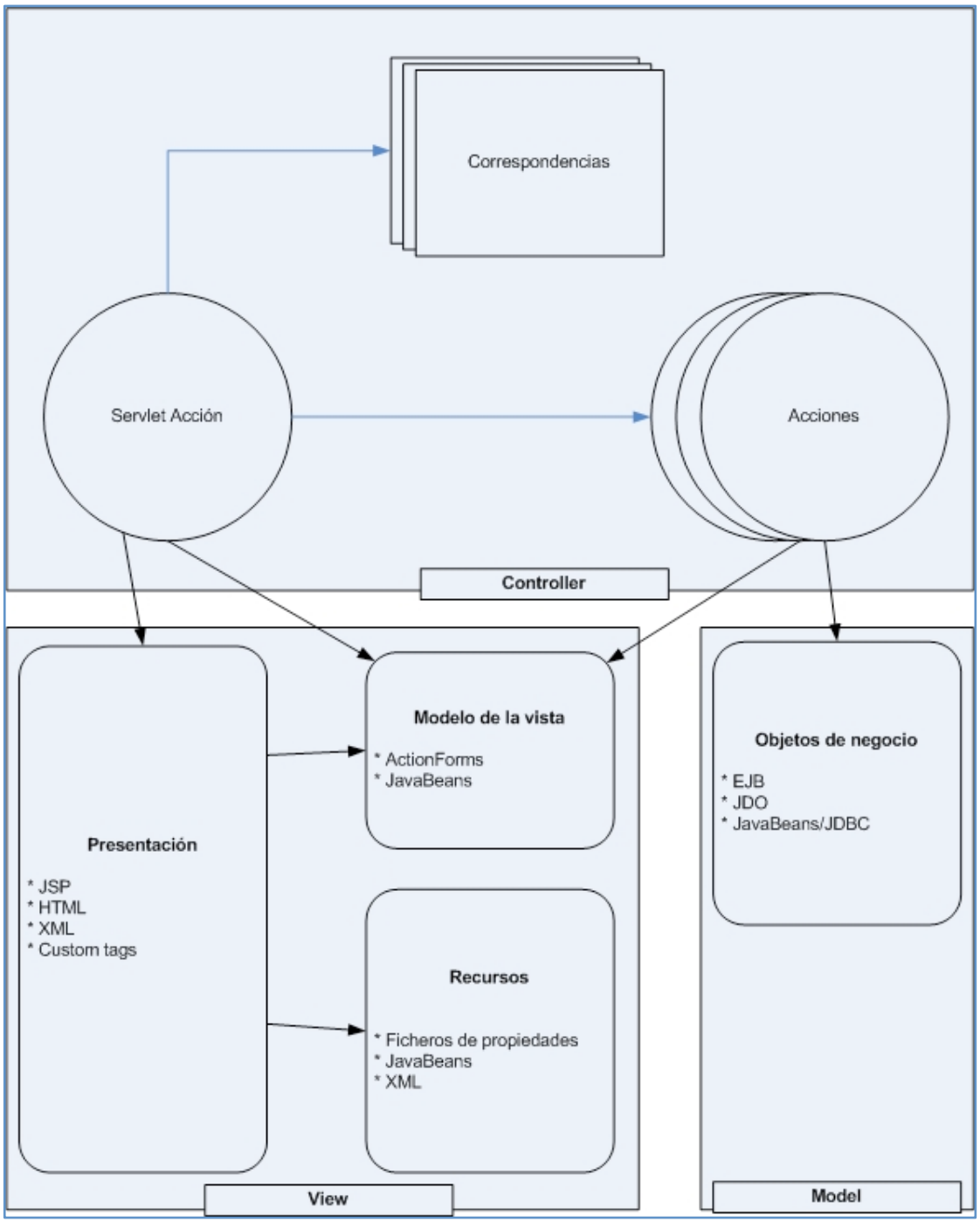

Fig. 6. Diagrama estructural MVC2 de Struts, el framework de referencia.

Fig.: Modelo conceptual del framework.

El siguiente diagrama refleja el conocimiento que poseemos acerca del dominio en el que se aplica el software framework objeto del TFC. Dada la naturaleza de software para construir software inherente al framework, incluso en el análisis de requisitos tenemos que tener en cuenta los requisitos no funcionales, particularmente aquellos que tienen que ver con las tecnologías subyacentes: lenguaje de programación Java, servlets y JSPs, etc. El framework cuya funcionalidad básica se desea emular es Struts.

El diagrama estático de análisis contendrá una clase para cada concepto del dominio, el cual estará basado en el modelo MVC anteriormente explicado. Debemos insistir en la importancia que cobra la lista de requisitos no funcionales definidos más arriba y deducidos a partir de una serie de características sobresalientes de Struts y algunos otros frameworks analizados basados en MVC2. Los conceptos capturados en el siguiente diagrama estático de análisis no serán exclusivamente del dominio (Desarrollo de aplicaciones web) sino relativos a la tecnología subyacente. De hecho, el siguiente diagrama puede ser considerado una forma de diagrama de diseño incipiente:

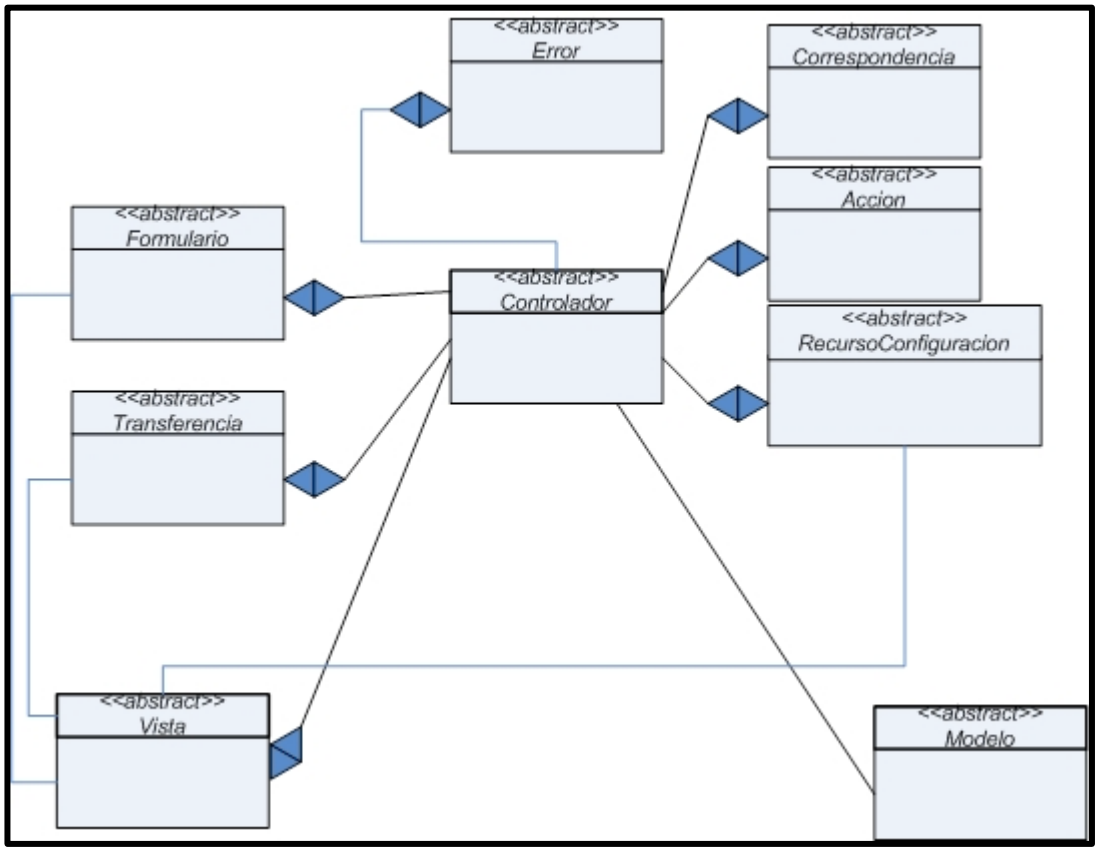

Fig. 7. Modelo conceptual del framework.

Diseño e implementación del núcleo básico del framework.

Introducción.

Este documento explica el diseño orientado a objetos y la implementación del núcleo básico del Framework para la capa de presentación. Este núcleo básico es el resultado de la primera iteración. Previsiblemente, el resultado final surgirá de una segunda iteración, la cual, partirá de una especificación y análisis de requisitos complementaria de la entregada en la PEC 2 y basada en la experiencia resultante de esta primera iteración.

El propósito del *diseño* consiste en definir la estructura del software del Framework, la cual, deberá satisfacer los requisitos de partida. Esta estructura deberá ofrecer el comportamiento requerido junto con ciertas restricciones de rendimiento. Se ha procurado que esta estructura sea simple y que, en la medida de lo posible, evite la duplicación de esfuerzos. También se ha procurado aplicar la tecnología de orientación a objetos de forma que facilite la extensibilidad del software, esto es, en general su mantenimiento. El grado de cumplimiento de este objetivo, previsiblemente será ostensiblemente mejor en la segunda, y final, iteración. Concretando un poco más, se ha procurado derivar clases a partir de abstracciones (Clases abstractas e interfaces) y no directamente de implementaciones; otro ejemplo es la encapsulación de la creación de objetos en *factorías de objetos*: se ha procurado aplicar el *principio de inversión de dependencias.*

A lo largo del diseño del núcleo antes mencionado se han efectuado algunas revisiones orientadas a evaluar su complejidad y su adecuación a las especificaciones.

Particularmente, se han evaluado *críticamente* la aplicación de ciertos patrones de diseño y sus diversas estrategias. Por ejemplo, en el caso de la factoría de comandos (**CommandInterningFactory**) se han aplicado los patrones **Command** y **Interning** después de descartar **FlyWeight** por considerarse éste demasiado restrictivo. El aspecto más crítico considerado en las revisiones ha sido el hecho de si el diseño realizado describe la estructura de un Framework cuya implementación, pruebas y mantenimiento resultarían fáciles. Otros compromisos que han sido necesarios tienen que ver con la integración del Framework con la tecnología JSP y Servlets, por ejemplo, la creación del contexto lógico debería ser responsabilidad del servlet **FrontController** o deberíamos dejar la puerta abierta a aplicar en ciertas **request** un **FrontController** basado en JSP y, por tanto, la creación del contexto lógico del webapp resultante queda encargada a un **ServletContextListener** que crea ese contexto y lo exporta al **ServletContext** del webapp. Se ha optado por la segunda posibilidad, aunque, ésta implique una dispersión mayor del diseño al ofrecer una mayor flexibilidad de cara a futuras modificaciones.

Se ha procurado hacer un uso lo más extensivo posible de *patrones de diseño*. Estos patrones nos permiten aprovechar la experiencia pasada de otros diseñadores, ganando velocidad en el proceso de diseño y manteniendo unos niveles aceptables de calidad. Cada patrón ofrece una ventaja que puede afectar positivamente al rendimiento y al proceso de cambio del software. El patrón arquitectónico y mixto conocido como MVC (Model-View-Controller) constituye el patrón central de este trabajo y, este documento se basa fundamentalmente en él: todos los patrones aplicados pretenden ofrecer un diseño probado de algún aspecto del patrón MVC. La pieza central de este trabajo es un **Controller** que contiene la funcionalidad básica de Struts 1.0. La aplicación de patrones de diseño nos proporciona un vocabulario que agilizará las discusiones acerca de diferentes aspectos del software y, aunque pueden aumentar la complejidad de éste, sin embargo, normalmente se comprueba que ese coste es aceptable al considerar la mejora de flexibilidad. Se ha procurado que la aplicación de patrones no contravenga el *principio de sustitución de Liskov*.

La estrategia empleada en la implementación ha sido *bottom-up:* en primer lugar se han diseñado aquellas clases básicas de las que dependen otras clases. La implementación del núcleo básico entregada será sometida a una revisión exhaustiva de su rendimiento, consumo de recursos, coherencia con los requisitos funcionales y su integridad en cuanto a su ejecución en un contenedor multi-hilo. En cuanto a este último e importante aspecto hemos de resaltar que, esta implementación, la cual no contiene la funcionalidad de autenticación de usuarios, no usa la sesión (HttpSession) y que las estructuras de datos compartidas se exportan al contexto de la **request**, la cual es inherentemente *thread-safe*. Finalmente, las estructuras de datos compartidas en el *servlet context* y, por tanto, disponibles a todo el webapp, son de sólo lectura y son creadas por un **ServletContextListener**, por lo cual son, también, *thread-safe*. Al incorporar los requisitos de autenticación en la siguiente iteración, realizaremos la implementación teniendo en cuenta aplicar accesos sincronizados a la **HttpSession**.

Consideraciones preliminares. La misión fundamental que cumplen los componentes de los que está constituida una aplicación web es gestionar las

interacciones con el usuario. Esta gestión se puede descomponer en los siguientes pasos:

1. Procesar la **request** http.
2. Invocar los componentes de servicio asociados a esa *request*.
3. Generar el código XHTML necesario para la página web que contenga los resultados dinámicos.

Estos pasos se corresponden con el patrón MVC, mencionado en la introducción. El **controlador** maneja la *request,* el **modelo** representa los servicios y, la **view** dibuja el contenido dinámico en forma de páginas web. Este patrón se encuentra en el núcleo del presente *Framework*. Este patrón permite la **separación** de la presentación, el control de flujo y la lógica de negocio presente en el *back-end*. El asilamiento de estas tres capas se logra aplicando MVC, en concreto, en J2EE se lleva a cabo mediante un controlador central implementado mediante un servlet.

Conceptos centrales de Magis: framework básico basado en un servlet front controller.

El servlet **controlador** *puede* ser el punto único de entrada de todas las *requests,* procesando tanto links como formularios. Todas las aplicaciones web se basan en esta capacidad esencial del controlador, por tanto, su diseño ha de llevarse a cabo distinguiendo claramente entre los aspectos de éste que varían y los que no lo hacen y aislándolos apropiadamente, lo cual redundará en una mayor extensibilidad de este componente esencial. Un aspecto esencial del diseño es *automatizar tantas facetas del procesamiento de requests como sea posible*. En el diseño del controlador se ha procurado hacer que el caso más normal sea muy simple y, para ello, se ha aplicado un alto grado de automatización y configuración. El desarrollador que usa el *Framework* puede sobrescribir los elementos automatizados y configurados cuando ello sea necesario en casos de uso *particulares*. El elemento esencial de flexibilidad en el controlador es el uso del patrón **Command**, el cual, nos permite que el controlador central delegue en un Command la gestión de la ejecución de un servicio de la capa de negocio.

El uso del patrón **Command** no es el único aspecto que confiere fñlexibilidad al controlador, también se ha aplicado el patrón **Application Controller**. Este patrón, permite al **Front Controller** fijarse en el tratamiento de la **request** y delegar en el **Application Controller** la gestión de los comandos y de la vista de destino resultante.

La implementación más eficaz de los comandos (**patrón Command)** en un run-time multi-hilo consiste en objetos sin estado, *thread-safe*, que permitan que las instancias puedan ser almacenadas y reutilizadas. Magis crea una sola instancia de cada *command* en el momento en que se recibe una *request* cuyo *mapping* está asociado con él. La unicidad del *command* en el sistema está garantizada por una *factory* de *commands* la cual, aplicando una estrategia particular del **patrón Interning,** en esta estrategia, la factory mantiene un mapa privado de instancias de *command:* si se necesita un *command,* la factory comprueba si ya existe, si es así, devuelve la

referencia a ese *command* y, si no, crea uno utilizando un *wrapper* de un HashMap de mappings creada por el ServletContextListener usando un objeto consumidor de **magis-config.xml**. El conumidor xml está basado en apache.commons.Digester. El patrón *command*, nos permite aislar la invocación del servicio de negocio de los procesos de gestión de esa invocación y de la view, las cuales, son responsabilidades del *Application Controller*. En resumen, hemos aplicado el **patrón Factory Method** conjuntamente con **Interning,** obteniendo además las ventajas derivadas de **Singleton.** La clase central se denomina **CommandInterningFactory**.

Los objetos creados por un *Interning* han de ser inmutables o casi inmutables, por tanto, es factible aplicar este patrón. Ciertamente, la implementación de los *command* ha de hacerse en forma de objetos sin estado y *thread-safe*.

Los *command* acceden a los servicios de negocio y preparan el acceso a las páginas que habrán de presentar los resultados al usuario. El servlet *Front Controller* se encarga de que los datos necesarios para la presentación de resultados en la página a la que se va a despachar, estén disponibles en el contexto de la *request*. En Magis, esto se lleva a cabo declarativamente, opcionalmente en cada *mapping*. El *Command* encapsula los objetos resultantes de la ejecución de los servicios en un objeto denominado *ResponseContext,* el cual, se comparte en el *request context* con la página a la que se despacha. Si el *Command* recibe los datos resultado de las invocaciones de los servicios en un DTO (**Patrón Data Transfer Object**), puede envolver éste en una *bean ResponseContext*.

Las responsabilidades de la arquitectura del **controlador MVC** engloban todos los flujos de procesamiento necesarios:

1. Determinar el *evento* generado por el usuario y la acción que es necesario llevar a cabo.
2. Crear el objeto *evento*.
3. Crear la clase *Command* correspondiente a la *request*.
4. Llevar a cabo las validaciones necesarias de los datos provistos en la *request* y manejar los posibles errores derivados.
5. Invocar los servicios de la capa de negocio y manejar los errores derivados.
6. Gestionar el *contexto de usuario*.
7. Calcular la página *siguiente*.
8. Redirigir la *request* a la página siguiente.

La creación de una *abstracción* de la *request http* nos permite traducir la acción generada por el usuario de su forma original http a en evento de negocio. Este *evento de negocio* encapsula los datos incluidos en la request, ya sean estos navegación pura o un formulario. La forma elegida en Magis es una *Java Bean* diseñada de acuerdo con el **patrón Context Object** (clase abstracta ContextBean), la cual es escrita por el desarrollador y –en tiempo de ejecución- autorrellenada por el controlador empleando *Java Reflection*. En la próxima iteración, un requisito no funcional es usar Apache Commons Beanutils; este Framework será usado también para dar la posibilidad al desarrollador de auto crear los Context Beans como

Commons.beanutils.DynaBean. En la próxima iteración estableceremos un requisito funcional nuevo que tiene que ver con el soporte de parámetros multivalor en la request y, por tanto, en el formulario de entrada. La abstracción realizada nos permite hacer transparentes al sistema los detalles de protocolo que permiten el acceso a los datos provistos por el usuario.

La correspondencia entre la *request* y el *command* se fija declarativamente en la sección <mappings> de magis-config.xml. Estas correspondencias permiten que el controlador aplique mecanismos simples para calcular la página siguiente, los cuales dependen del status de la operación –*command*- efectuado o, el salto directo a un destino.

Las validaciones de la *context bean* efectuadas en el *command* tienen como objetivo evitar viajes de ida y vuelta capa de negocio, los cuales, pueden comportar utilizaciones innecesarias de los recursos de red. En Magis, la validación se lleva a cabo programáticamente en la subclase ContextBean provista por el desarrollador. En la próxima iteración, sería deseable incluir alguna capacidad de validación especificada declarativamente mediante la utilización de apache.commons.validator, por ejemplo o quizás un posible Framework básico de validación declarativa.

Cuando la request llega a la página de destino, ésta espera encontrar alguna forma de contexto de usuario que le permita acceder a los resultados y presentarlos. Otros aspectos que han de ser gestionados por el controlador son la abstracción de URLs físicos, los cuales son representados como metadatos; determinar cuál se la página siguiente que hay que desplegar y, efectuar la redirección a la página calculada. Al proceso explicado debemos añadirle la capacidad de generar mensajes de error y que las páginas jsp de destino sean capaces de acceder a los mismos y desplegarlos. En Magis se ha optado por representar los errores en forma de ArrayList y, transmitirlos en una bean (Errors) en el contexto de la request. Generalmente, es una buena práctica limitar el uso del contexto de la HttpSession excepto donde sea *necesario*, porque, de otro modo comprometeríamos la escalabilidad de la aplicación web.

Resumen de los patrones más notables aplicados. La aplicación de patrones de diseño es esencial en el desarrollo de frameworks, entre otras razones ya comentadas en el *estudio de frameworks* realizado en la pec 2 porque nos permiten organizar el software de formas bien probadas, ayudan a su comprensión y por tanto facilitan partes esenciales del ciclo de vida del software. En Magis, siguiendo esta recomendación se han aplicado patrones a una gran variedad de situaciones que explicamos a continuación.

- **Context object.** Compartir los parámetros presentes en la **http request** haciendo que los detalles estructurales de ésta resulten transparentes en otros contextos dentro de la capa de presentación. El controller de Magis solicita la creación de un objeto **ContextObject** cuando recibe una nueva request y determina declarativamente qué instancia concreta de **ContextBean** ha de solicitar basado en los mappings.

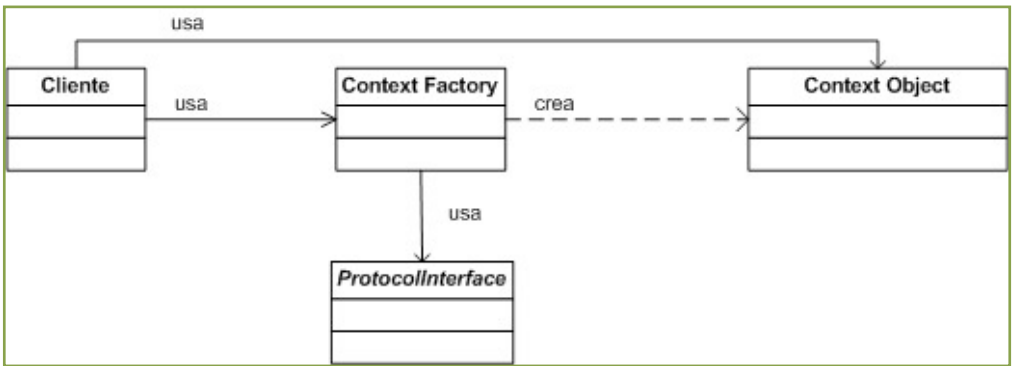

Fig. 8. Patrón Context Object.

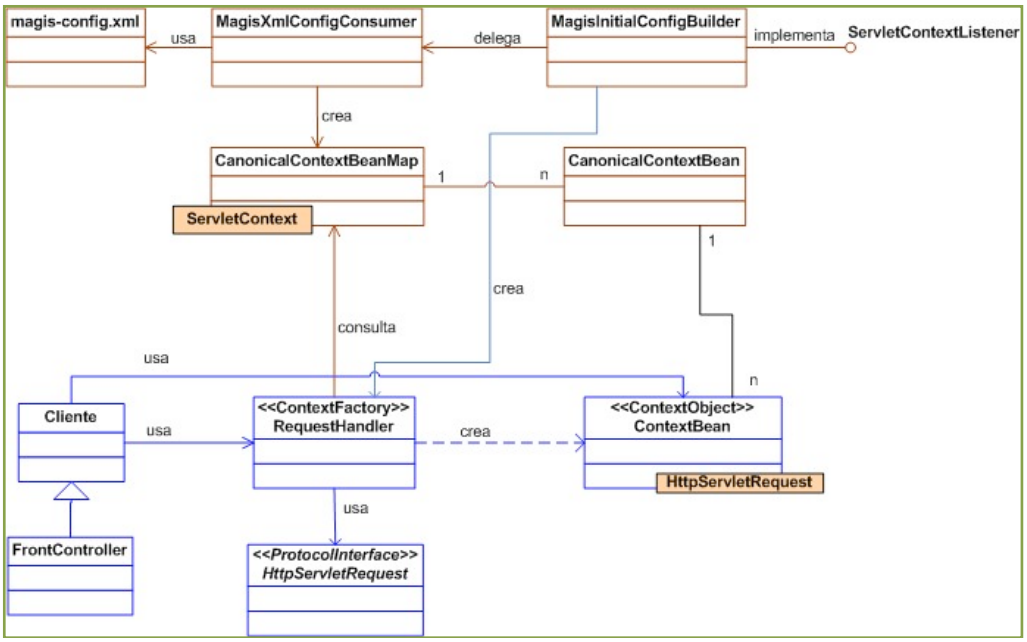

Fig. 9. Aplicación de los patrones Factory Method, Observer y Context Object con estrategia request context en Magis.

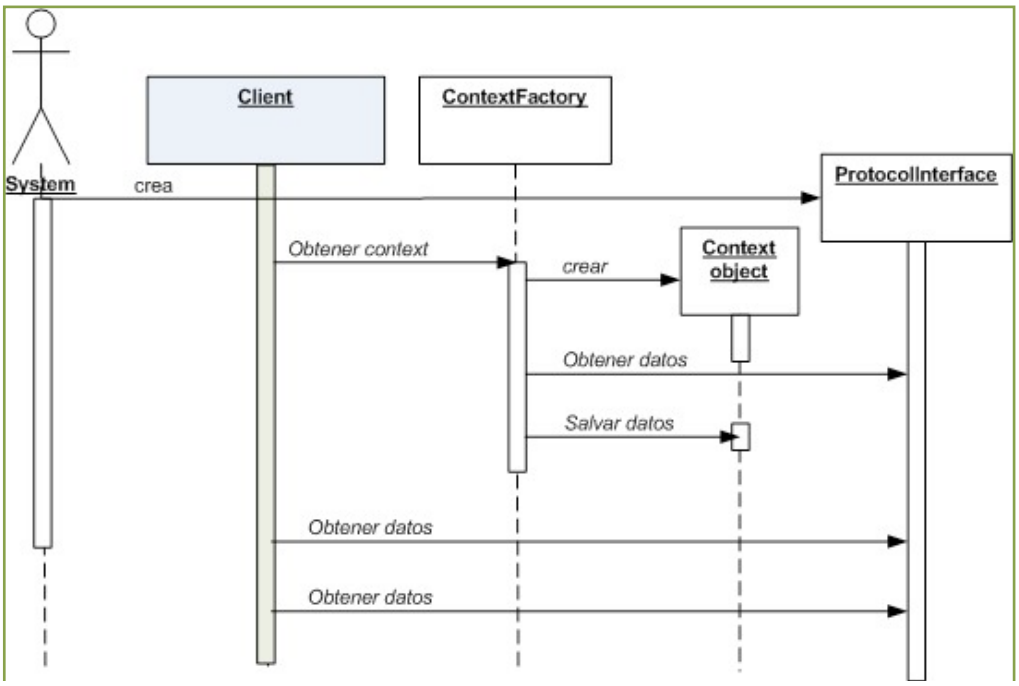

Fig. 10. Diagrama de secuencia, patrón Context Object.

- **Patrón Front Controller.** Punto inicial de contacto para el manejo de todas las requests que tengan que ver entre sí. El control está centralizado, no duplicado. Centralizando la lógica de control, el Front Controller ayuda a reducir la cantidad de lógica incluida directamente en las views. Típicamente, este patrón se aplica con el patrón Application Controller que, explicamos su uso en Magis, más adelante. Básicamente, aplicamos este patrón para llevar a cabo las operaciones de manejo de protocolo y transformación de contexto.

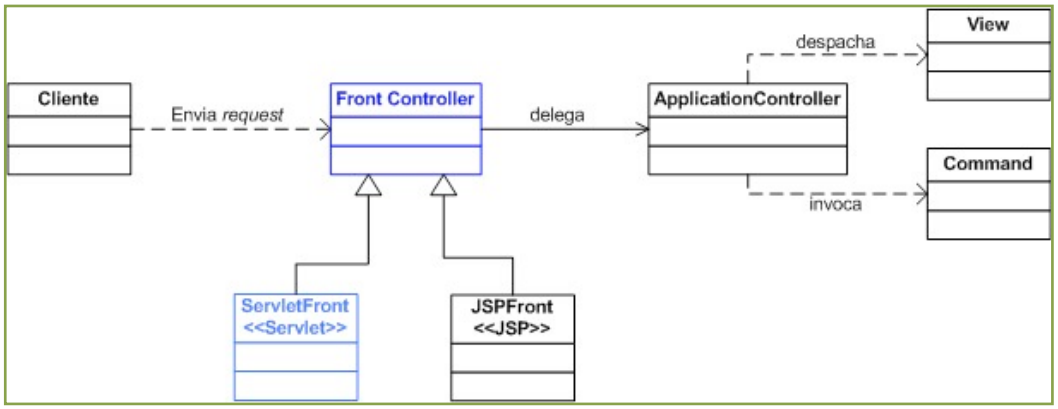

Fig. 11. El patrón Front Controller con estrategia ServletFront (En azul) y su relación con Application Controller

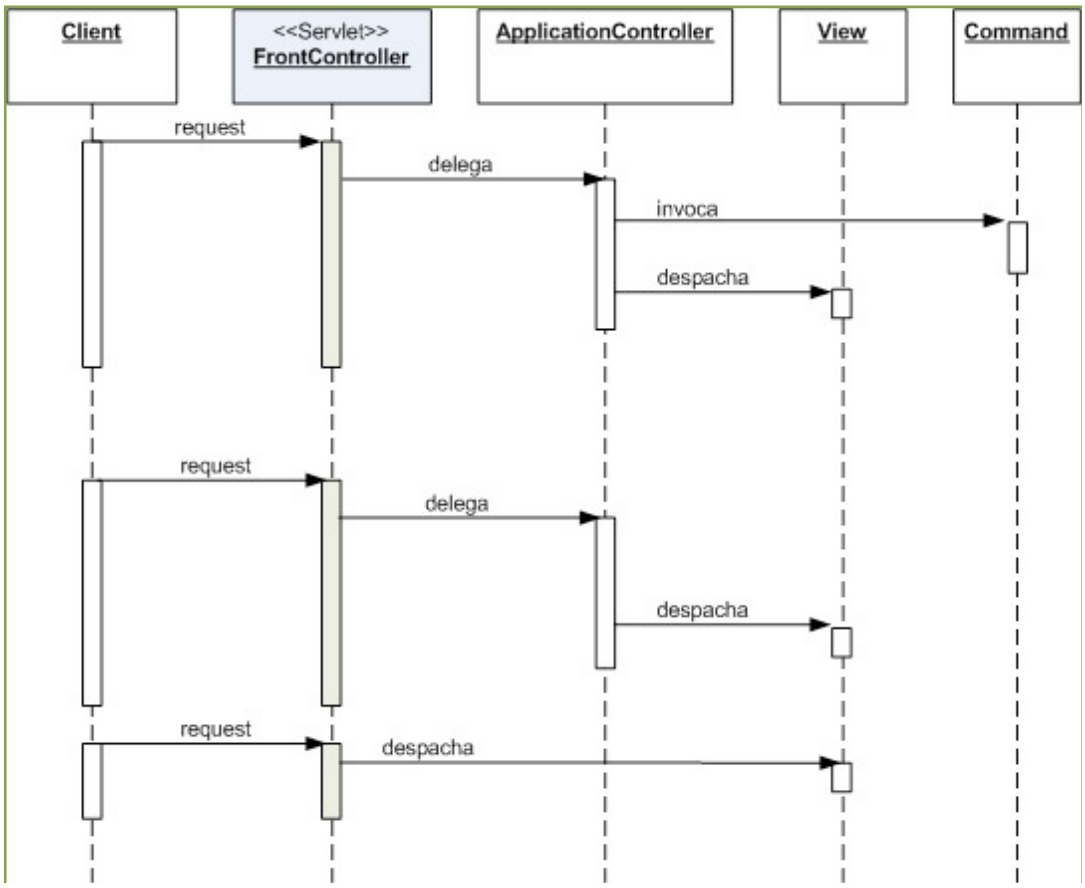

Fig.12. Diagrama de secuencia que ilustra la solicitud de tres requests, cada una de las cuales sufre una secuencia de operaciones distinta, especificada declarativamente en Magis a través de los mappings.

Application Controller. Es importante suprimir la gestión de comandos y de views del Front Controller a medida que éste crezca, de este modo, los tratamientos relativos a protocolo permanecen en el Front Controller y aquellos que tienen que ver con los comandos y con las views pasan al Application Controller. Hay dos estrategias de este patrón que se entremezclan en Magis: Command Handler y View Handler. El patrón Context Object, explicado anteriormente, facilita la comunicación entre el Front Controller (FC) y el Application Controller (AC). De hecho, aunque no es así en Magis, el AC suele ser una Context Object Factory. En Magis, las Context Beans se solicitan a una ContextBeanFactory que las busca en una estructura CanonicalContextBeanMap que se crea en tiempo de arranque del webapp mediante un ServletContextListener cuyo nombre es MagisInitialContextBuilder. Cada ContextBean se obtiene a partir de su CanonicalContextBean, a su vez, obtenida desde el MappingsWrapper procedente del consumo de magis-config.xml.

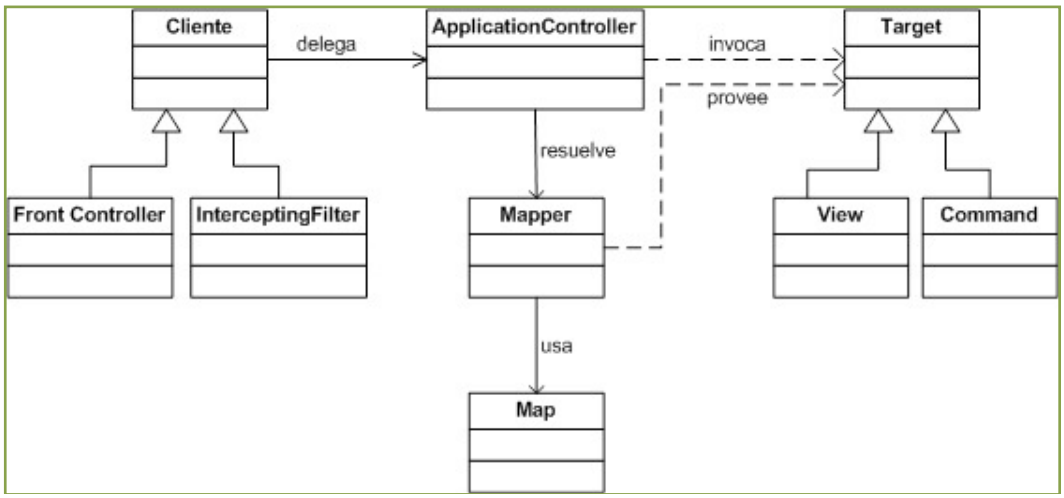

Fig. 13. Patrón Application Controller.

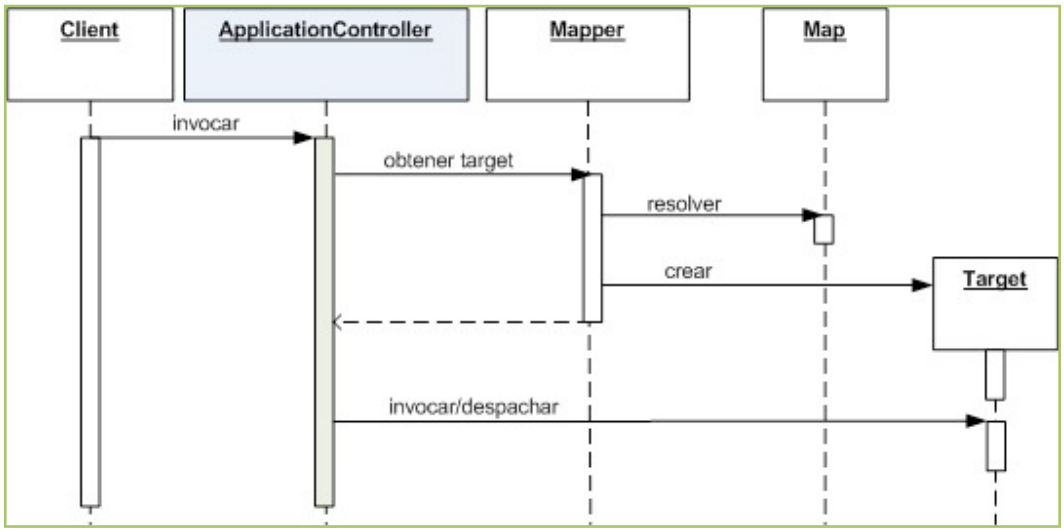

Fig. 14. Diagrama de secuencia, Application Controller.

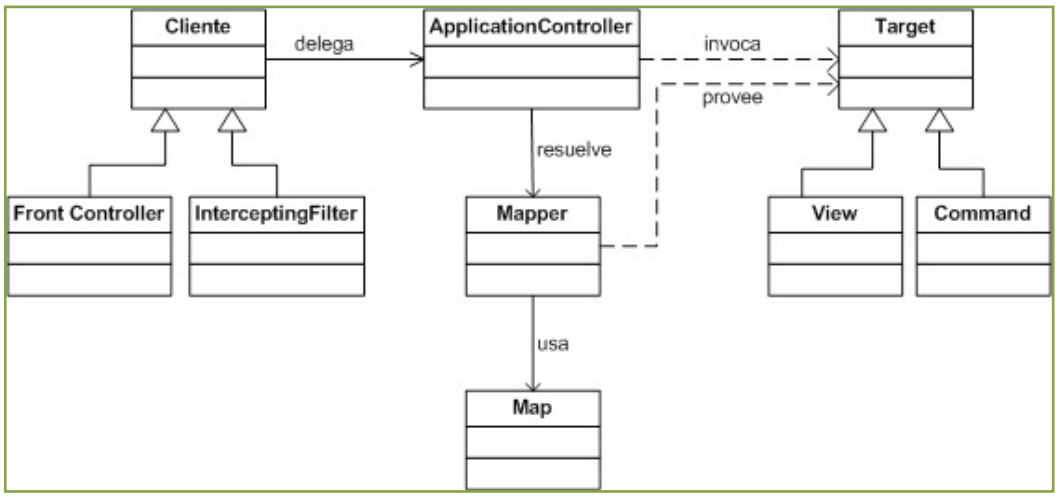

Fig. 15. Patrón Application Controller con estrategia view handler, aplicado en Magis.

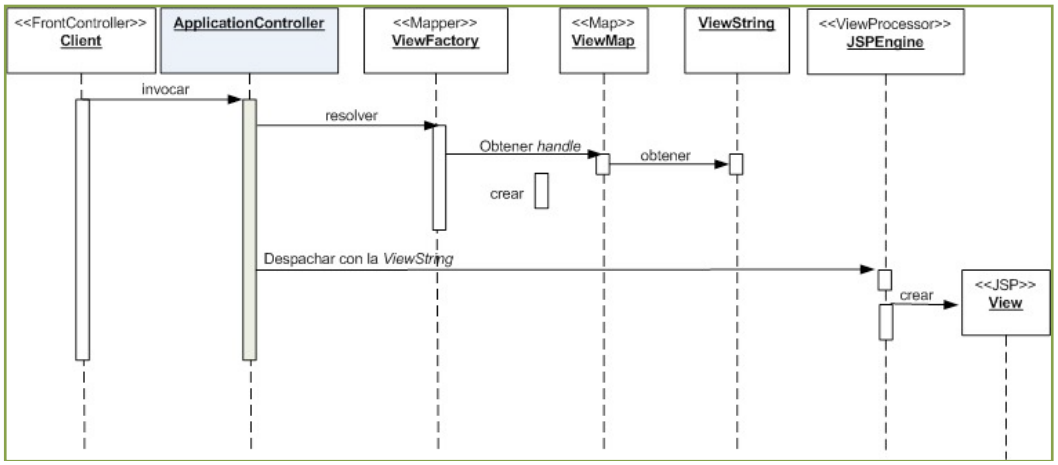

Fig. 16. Diagrama de secuencia correspondiente a una request de ejemplo del patrón Application Controller con View Handler strategy.

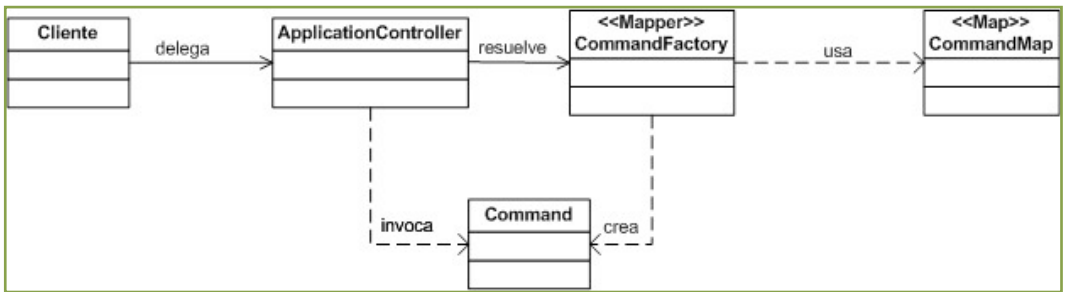

Fig. 17. Patrón Application Controller con Command Handler strategy.

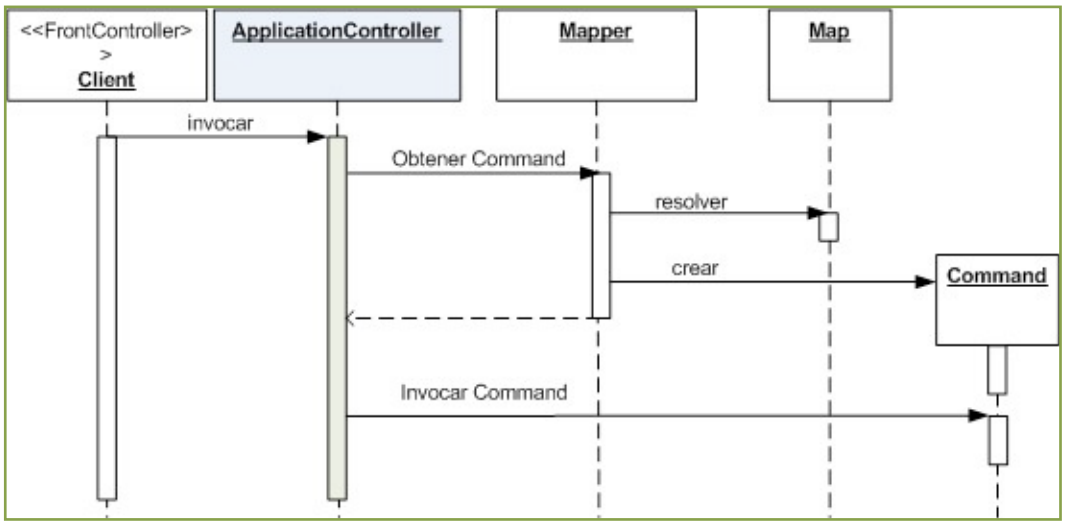

Fig. 18. Diagrama de secuencia correspondiente a una request de ejemplo del patrón Application Controller con Command Handler strategy.

En Magis aplicamos ambas estrategias de este patrón, de forma un tanto entremezclada.

- **Patrón Service to Worker.** Se trata de un patrón mixto, está compuesto de otros patrones que ya han sido explicados. El control centralizado de las requests y la creación de las view se llevan a cabo con Front Controller, Application Controller y View Helper.

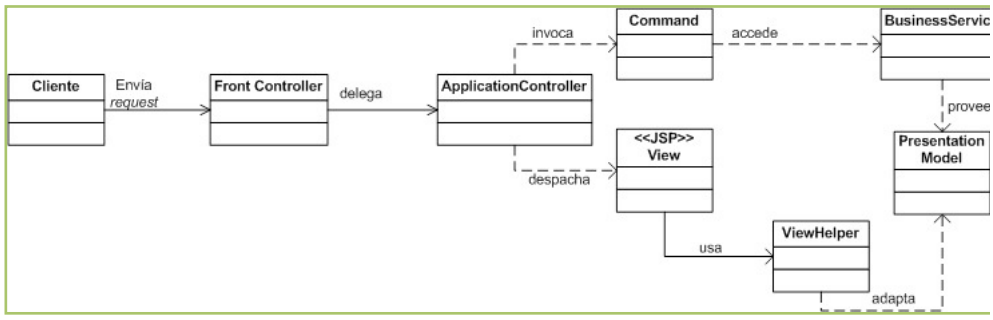

Fig. 19. Service to Worker con estrategia Command.

- **Patrones Service to Worker, Interning, Command y Singleton**. Estos patrones juegan un papel central en Magis. Interning representa una factoría de comands que garantiza la unicidad de cada command mediante un mapa interno. Cuando Interning recibe un zapping, busca el Command, si no existe lo instancia y lo guarda en el mapa para su uso posterior. La CommandInterningFactory es el mecanismo correcto para instanciar commands, que además nos garantiza su unicidad (Singleton). Los objetos command han de ser sin-estado y seguros-con-hilos.

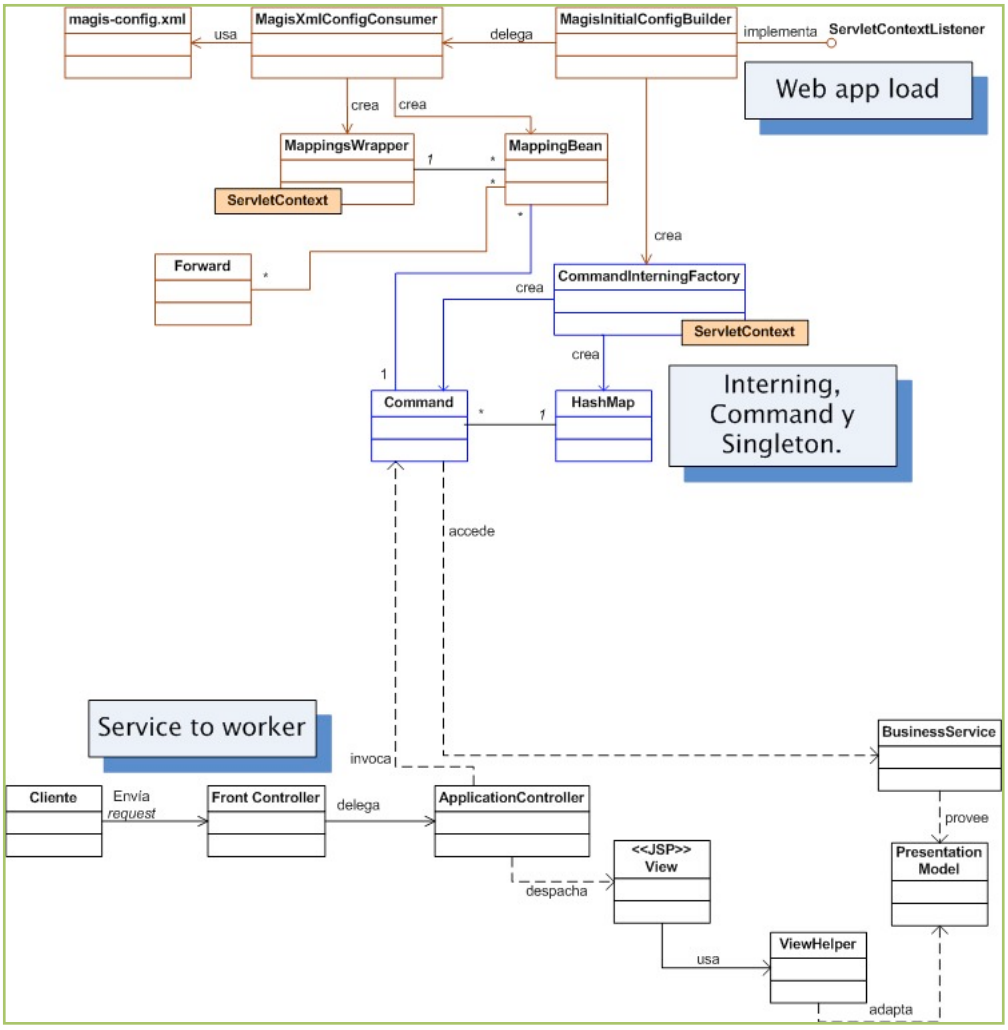

Fig. 20. Patrones Service to Worker, Interning, Command y Singleton en Magis.

Implementación final y conclusiones

Grafos de objetos de configuración en Magis.

Los webapps basados en Magis necesitan una configuración específica del contenedor, la cual, como es conocido se realiza en el fcihero /WEB-INF/web.xml. Magis necesita fijar un *Context Listener* que consuma el fichero de configuración de Magis, magis-config.xml. Del consumo del fichero mencionado, resulta un grafo de objetos que los diferentes componentes emplean para llevar a cabo sus funciones. Entre ellos se encuentra la factoría de objetos *Command* que además actúa como *Interning*. No sólo fabrica objetos sino que, una vez creados, no los destruye y, cada vez que una request necesita un determinado *Command* se lo solicita y la *Interning factory* devuelve la referencia correspondiente extraída de su *hashmap* de *commands*. Se trata de un mapa de lectura y escritura que se encuentra en un contexto, el *servlet context*, el cual no es *thread-safe*. Hay dos aspectos que tenemos que explicar, uno de ellos tiene que ver con la integridad de la propia estructura y el otro con la intergidad de cada command.

En cuanto al primer aspecto, esto es, la integridad de la *interning factory* en el proceso de creación de objetos, hemos garantizado que la creación es *thread-safe* sincronizando el método estático CommandInterningFactory.getInstance(). En cuanto al carácter thread-safe de los Command fabricados por la interning factory se establece que el desarrollador no puede usar variables globales. En general se ha procurado usar la request siempre que ello resulta suficiente, ya que, la request es inherentemente *thread-safe*. En los casos en los que se accede a estructuras compartidas en contextos no *thread-safe* como el contexo del servlet y la sesión, se han creado bloques sincronizados sobre el objeto servlet context en el primer caso y sobre el objeto sesión en el segundo.

El controlador de Magis.

Se ha tratado de que la implementación del controlador fuera lo más completa posible; alguna de sus facetas, sin embargo, no ha sido posible ofrecer una implementación funcionalmente rica. Tal es el caso de los procesos de autenticación y de autorización. Esta implementación simplista, sin embargo, confiere estructura al controlador para que cualquier persona que lo estudie comprenda, simplemente leyendo el texto fuente, qué hace funcionalmente un *Application Controller*.

Los forwards en Magis ofrecen la opción de enviar al destino una OCB *(Output Context Bean)* la cual, ha sido previamente llenada con el objeto DTO retornado por el objeto de negocio. El envío se hace en la sesión, para que la OCB pueda sobrevivir los ciclos posteriores de validación o de reintentos derivados de errores en la ejecución de algún comando posterior. Las OCB llevan datos que se desplegarán en la view, la cual, en Magis se realiza con páginas jsp. De este modo, el desarrollador puede expresar declarativamente qué bean recibe cada destino final de una request, si

lo desea. Las páginas jsp de destino, si usan los tags de Magis pueden desplegar resultados simplemente buscando cual es el nombre del atributo de la sesión donde se encuentra la bean y ejecutando el getter indicado en su parámetro 'name'. A este respecto, hay que observar que la estructura de cada uno de los *hot spots* de la clase abstracta *Command* podría todavía extraerse una gran comunalidad en las operaciones del código, es decir, hay oportunidades de encapsular aspectos comunes y dejar los aspectos variables al desarrollador. Nos referimos en concreto al método postService(), el cual, previsiblemente podría automatizarse ostensiblemente.

Programación orientada a objetos.

Se ha procurado aplicar buenos principios de programacion orientada a objetos conjuntamente con los patrones de diseño, si bien es verdad que, en ocasiones, no se habrá logrado del todo. En particular, no siempre hemos encontrado formas simples y eficientes de aplicar el principio de inversión de dependencias,esto es, que ninguna variable debe contener referencias a clases concretas, que ninguna clase debe derivarse de ninguna clase concreta y finalmente, que ningún método debería sobreescribir un método implementado por cualquiera de sus clases base. Está suficientemente aceptado incluso en prestigiosos libros sobre OO que es conveniente aplicar estos principios de forma flexible y sensata. Con frecuencia, si se ha intentado programar contra interfaces en vez de contra implementaciones y aprovechar la herencia de interfaz, que es múltiple en Java y, no la herencia de implementación.

Glosario.

La terminología empleada en este TFC es la que comunmente se emplea en contextos informáticos, a excepción de los nombres de los componentes de Magis. El esquema conceptual de Magis es parecido al de Struts, pero, los nombres de los conceptos empleados pretenden ser fieles a los patrones de diseño que representan. Esto es muy conveniente porque los patrones de diseño se han convertido en una especie de *lingua franca* que nos facilita la transimisión de los conceptos y de otros aspectos, incluso subjetivos, que conlleva el uso de patrones.

Concepto	Acrónimo	Comentarios
Application Controller	AC	El componente central de Magis.
Command	C	Patrón Command, diseñado como clase abstracta con 5 callbacks.
Context Beans	CB	Patrón Context Object
Data Transfer Object	DTO	Patrón Data Transfer Object. Se emplea sobre todo cuando los objetos de negocio son remotos ocn el objeticvo de minimizar el ancho de banda empleado.
Input context bean	ICB	Patrón context object aplicado por el controlador cuando rellena la bean con los parámetros de la request.
Output context bean	OCB	Patrón context object aplicado cuando un Command salva un OCB que representa los datos dinámicos que deberá renderizar el destino.
Error		Asocia un índice de error con un mensaje concreto
Errors		Lista de errores acumulados en una secuencia de operaciones del controlador.
Front Controller	FC	Patrón Front Controller
Forward bean	FB	Asocia un nombre lógico con un destino y una OCB.
Global forward bean	GFB	Una bean que liga un nombre a un destino globales, es decir, no particulares de un determinado Mapping.
Mapping		La estructura de datos central empleada por el controlador de Magis: liga una request uri con su

		posible ICB, con un Command y algún forward
Service data	SD	Beans empleadas para transportar datos entre un Command y un objeto de negocio (BO) y al revés.
Service data object	SDO / DSO	Bean empleada para comunicar un BO con su command.

Bibliografía

Abi-Antoun, Marwan (octubre 2007) "Making Frameworks Work: a Project Retrospective" *22 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, ACM (pág.: 1004-1017) [en línea] [Fecha de consulta: 4 de enero de 2009] http://portal.acm.org/citation.cfm?id=1297969

Alur, Deepak; Crupi, John; Malks, Dan (2001) *Core J2EE Patterns: Best Practices and Design Strategies* (1st ed.) Palo Alto: Prentice Hall.

Bates, Bert; Sierra, Kathy; Basham, Bryan (2004) *Head First Servlets and JSP: Passing the Sun Certified Web Component Developer Exam* (1st ed.). Sebastopol: O'Reilly Media, Inc. (Head First Series).

Booch, Grady; Rumbaugh, James; Jacobson, Ivar (1999) *El lenguaje unificado de modelado* [The Unified Modeling Languaje User Guide] (1. ª ed. en español). Madrid: Addison Wesley Iberoamericana.

Bosh, Jan et al. (2000) "Object-Oriented Framework-based Software Development: Problems and experiences" *ACM Computing Surveys* (Vol. 32, n. º: 1, artículo 3) [en línea] [Fecha de consulta: 4 de enero de 2009] http://portal.acm.org/citation.cfm?id=351936.351939

Brooks, Frederick P., Jr. (1995) *The Mytical Man-Month: Essays on Software Engineering* (Anniversary ed.). Indiana: Addison Wesley Longman, Inc.

Brooks, Frederik P. Jr. (1987) "No Silver Bullet: Essence and Accidents of Software Engineering" *Computer* IEEE Computer Society (Vol. 20, n. º 4, pág. 10-19) [en línea] [Fecha de consulta: 4 de enero de 2009] http://portal.acm.org/citation.cfm?id=26440.26441

Chan, Patrick (1999) *The JavaTM Developers Almanac 1999* () Massachusetts: Addison Wesley Longman, Inc. (The Java Series).

Chan, Patrick; Lee, Rosanna; Kramer, Douglas (1998) *The Java Class Libraries* (1st ed.) Massachusetts: Addison Wesley Longman, Inc.

Christensen, Henrik B. (28-30 junio 2004) "Frameworks: Putting Design Patterns into Perspective" *ACM SIGCSE Bulletin* (vol. 36, n. º 4, pág. 142-145) ACM [en línea] [Fecha consulta 28 de diciembre de 2008] http://portal.acm.org/citation.cfm?doid=1026487.1008035

Fayad, Mohamed E. (marzo 2000) "Introduction to the Computing Surveys' Electronic Symposium on Object-Oriented Application Frameworks" *ACM Computer Surveys* (Vol. 32, n. º: 1, pág.: 1-9) [en línea] [Fecha de consulta: 4 de enero de 2009]

http://portal.acm.org/citation.cfm?id=351936.351937&jmp=cit&coll=GUIDE&dl=GUIDE&CFID=17102360&CFTOKEN=19175552#CIT

Fayad, Mohamed E.; Schmidt, Douglas C.; eds. (octubre 1997) "Object-Oriented Application Frameworks" Comunications of the ACM (Vol. 40, n. º: 10, pág.: 32-38) [en línea] [Fecha de consulta: 4 de enero de 2009] http://portal.acm.org/citation.cfm?id=262793.262798&coll=GUIDE&dl=GUIDE&CFID=17103020&CFTOKEN=83581402

Fernández-Conde, Carlos J.; González-Calero, Pedro A. (julio 2002) "Domain Analysis of Object-Oriented Frameworks in FrameDoc" *Proceedings of the 14th international conference on Software engineering and knowledge engineering, ACM International Conference Proceeding Series* (Vol. 27, pág.: 27-33) [en línea] [Fecha de consulta: 4 de enero de 2009] http://portal.acm.org/citation.cfm?id=568760.568767&type=series

Flanagan, David (1997) *JavaTM Examples in a Nutshell: A Tutorial Companion to Java in a Nutshell.* (1.st ed.). Sebastopol: O'Reilly & Associates.

Freeman, Eric; Freeman, Elisabeth (2004) *Head First Design Patterns* (1st ed.). Sebastopol: O'Reilly Media, Inc. (Head First Series).

Froehlich, Garry et al. (marzo 2000) "Choosing an Object-Oriented Domain Framework" *ACM Computing Surveys* (Vol. 32, n. º: 1, artículo 17) [en línea] [Fecha de consulta: 4 de enero de 2009] http://portal.acm.org/citation.cfm?id=351936.351953&coll=GUIDE&dl=GUIDE&CFID=17103020&CFTOKEN=83581402

Froelich, Garry et al. (mayo 1997) "Hooking into Object-Oriented Application Frameworks" *Proceedings of the 1997 International Conference on Software Engineering* (pág.: 491-501) Department of Computer Sciences, University of Alberta [en línea] [Fecha de consulta: 4 de enero de 2009] http://www.cs.ualberta.ca/~softeng/papers/icse10.pdf

Gómez, Cristina et al. (2003) *Diseño de sistemas software en UML* (1. ª ed.). Barcelona: Edicions UPC.

Hansen, Tod (1997) "Development of Succesful Object-Oriented Frameworks" *1997 ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (Addendum)* (pág. 115 - 119) ACM [en línea] [Fecha consulta 28 de diciembre de 2008] http://portal.acm.org/citation.cfm?id=274567.274591

Horstmann, Cay (2008) *Java Concepts: compatible with Java 5 & 6* (5th ed.) Hoboken: Jonh Wiley & Sons, Inc.

Horstmann, Cay S.; Cornell, Gary (2000) *Core JavaTM 2: Volumen II – Advanced Features.* (). Palo Alto: Prentice Hall.

Hou, Daqing (mayo 2001) "Supporting the Deployment of Object-Oriented Frameworks" *Proceedings of the 23rd International Conference on Software Engineering ICSE 2001* (pág.: 791 – 792) IEEE [en línea] [Fecha de consulta: 4 de

enero de 2009]
http://ieeexplore.ieee.org/Xplore/login.jsp?url=/stamp/stamp.jsp?arnumber=919180&isnumber=19875

Hüni, Herman; Jonhson, Ralph; Engel, Robert (octubre 1995) "A Framework for Network Protocol Software" *ACM SIGPLAN Notices* (Vol. 30, n. º: 10, pág.: 358-369) [en línea] [Fecha de consulta: 4 de enero de 2009] http://portal.acm.org/citation.cfm?id=217875

Husted, Ted; et. al (2003) *Struts in Action: Building web applications with the leading Java Framework* () Greenwich: Managing Publications Co.

Iverson, Will (2005) *Apache Jakarta Commons: Reusable JavaTM Components* () Crawfordsville: Pearson Education, Inc.

Johnson, Ralph E. (1998) "How to Design Frameworks" *Object-oriented Programing and Design Lecture Notes*, Department of Computer Science, University of Illinois at Urbana-Champaign [en línea] [Fecha de consulta: 4 de enero de 2009] http://st-www.cs.uiuc.edu/users/johnson/cs497/notes98/day18.pdf

Jonhson, Ralph E.; Foote, Brian (June/July 1998) "Designing Reusable Classes" *Journal of Object-Oriented Programming* (vol. 1; n. º 2, pages 22-25) [en línea] [Fecha de consulta 28 de diciembre de 2008] http://www.laputan.org/drc/drc.html

Lee, Kang B.; Song, Eugene Y. (agosto2005) "Object-Oriented Application Framework for IEEE 1451.1 Standard" *IEEE Transactions on Instrumentation and Measurement* (Vol. 54, n. º: 4, pág.: 1527 - 1533) [en línea] [Fecha de consulta: 4 de enero de 2009]
http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/19/31498/01468564.pdf?arnumber=1468564

Liskov, Barbara (2001) *Program Development in Java: Abstraction, Specification, and Object-Oriented Design* (1st ed.) Massachusetts: Addison Wesley Professional.

Mahmoud, Qusay H. (marzo 2003) "Servlets and JSP Pages Best Practices" *Sun Developers Network, Technical Articles and Tips* [artículo en línea] [Fecha de consulta: 28 de diciembre de 2008]
http://java.sun.com/developer/technicalArticles/javaserverpages/servlets_jsp/

Markiewicz, Marcus Eduardo; Lucena, Carlos J. P. (Summer 2001) "Object Oriented Framework Development" *ACM Crossroads student magazine* (vol. 7.4) [en línea] [Fecha de consulta 28 de diciembre de 2008] http://www.acm.org/crossroads/xrds7-4/frameworks.html

McLaughlin, Brett D.; Pollice, Gary; West, David (2006) *Head First Object- Oriented Analisys and Design* (1st ed.). Sebastopol: O'Reilly Media, Inc. (Head First Series).

McLaughlin, Brett; Flanagan, David (2004) *Java 5.0 Tiger: A Developer's Notebook* (1st ed.). Sebastopol: O'Reilly Media, Inc.

MIT 6.170 Lectures 12-14 (2, 3 y 10 de octubre de 2001) *Design Patterns* [en línea] *MIT Opencourseware Lecture Notes; Electrical Engineering and Computer Science; Laboratory in Software Engineering, Fall 2001* [Fecha de consulta: 28 de diciembre de 2008] http://mitocw.udsm.ac.tz/OcwWeb/Electrical-Engineering-and-Computer-Science/6-170Laboratory-in-Software-EngineeringFall2001/LectureNotes/index.htm

Molin, Peter (1996) "Applying the Object-Oriented Framework Technique to a Family of Embedded Systems" *Research Report, Blekinge Institute of Technology* [en línea] [Fecha de consulta: 4 de enero de 2009] http://www.bth.se/fou/forskinfo.nsf/alfs/6d17588c63a76383c12568a3002ca9cc

Pallavi, Tadepalli; Cunnigham, H. Conrad (febrero 2005) "Using Function Generalization with Java to Design a Cosequential Framework" *Proceedings of the ALAR Conference on Applied Research in Information Technology, Acxiom Laboratory for Applied Research,* (pág.: 95-101) [en línea] [Fecha de consulta: 4 de enero de 2009] http://john.cs.olemiss.edu/~hcc/papers/functionGenJava.pdf

Pilone, Dan; Miles, Russ (2007) *Head First Software Development* (1st ed.). Sebastopol: O'Reilly Media, Inc. (Head First Series).

Robinson, Mike; Finkelstein, Ellen (2004) *Jakarta Struts for Dummies®* (1st ed.) Hoboken: Wiley Publishing, Inc.

Royce, Walker (1995) *Software Project Management: a unified Framework.* (1st ed.) Massachusetts: Addison Wesley Longman, Inc. (The Addison-Wesley object technology Series).

Sedgewick, Robert (2003) *Algorithms in Java* (3rd ed.) Massachusetts: Pearson Education, Inc.

Sierra, Kathy; Bates, Bert (2003) *Head First EJB: Passing the Sun Certified Business Component Developer Exam* (1st ed.). Sebastopol: O'Reilly Media, Inc. (Head First Series).

Srinivasan, Savitha (Febrero 1999) "Design Patterns in Object Oriented Frameworks" *Computer magazine* (vol. 32, n. º 2, pag. 24-32) [en línea] IEEE Computer Society [Fecha consulta 28 de diciembre de 2008] http://www2.computer.org/portal/web/csdl/magazines/computer#4

Tulach, Jaroslav (2008) *Practical API Design: Confessions of a JavaTM Framework Architect* (1st ed.) New York: Springer Verlag New York, Inc.

Van der Linden, Peter (1998) *Just JavaTM and Beyond* (3rd ed.) Palo Alto: Prentice Hall (The Sunsoft Press Java Series).

Zobel, Justin (2004) *Writing for Computer Science* (2nd ed.). London: Springer Verlag London Limited.

Anexos.

Texto fuente del controlador *BasicApplicationControllerImpl.java:*

```
/* PRINCIPIO */
/*****************************************************************************
 * UOC. TFC "Framework para la capa de presentación de aplicaciones web."     *
 * José María Foces Morán, jfoces@uoc.edu                                     *
 *                                                                            *
 * BasicApplicationControllerImpl.java                                        *
 *                                                                            *
 * El controlador MVC del framework. Representa un controlador parecido al    *
 * controlador de Struts 1.0; incluye la funcionalidad básica de éste. Esta   *
 * clase es central y sigue el patrón Application Controller, cuyos objetivos *
 * son básicamente tratar la request y seleccionar el comando adecuado; estas *
 * responsabilidades quedan descargadas del servlet Front Controller.         *
 *                                                                            *
 *****************************************************************************/

package magis.ac;

import javax.servlet.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletResponse;
import java.util.*;
import java.io.*;
import javax.servlet.http.HttpSession;
import magis.command.Command;
import magis.mappings.*;
import magis.servlets.FrontController;
import magis.command.CommandInterningFactory;
import magis.contextbeans.ContextBeanFactory;
```

```
import magis.contextbeans.ContextBean;
import magis.errors.*;
import magis.system.Globals;
import java.util.Locale;
import javax.servlet.http.HttpServletRequest;
import magis.globalforwards.GlobalForwardInterningFactory;
import magis.mappings.ForwardBean;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.commons.logging.impl.Jdk14Logger;

/**
 *
 * @author Chema
 */
public class BasicApplicationControllerImpl implements ApplicationController {

    /*
     *
     * El siguiente bloque de variables globales pertenecen a cada instancia
     * del controlador, NO SON VARIABLES GLOBALES DE UN SERVLET, las cuales
     * serían no thread-safe, es decir, el servlet FC crea una instancia de
     * este BasicApplicationControllerImpl como variable local de su método
     * processRequest(), por tanto, en cada hilo (para cada request) el servlet
     * posee una NUEVA INSTANCIA de BasicApplicationControllerImpl y, de este
     * modo no aparecen interacciones debidas a la ejecución con hilos.
     *
     * Lógicamente, el acceso de escritura a estructuras de datos r/w exportadas
     * al Servlet Context debe hacerse sincronizando tomando como lock una
     * referencia al Servlet Context.
     *
     */
```

```
    private static Log log =
LogFactory.getLog(BasicApplicationControllerImpl.class);

    private PrintWriter out = null;

    private HttpServletRequest request = null;

    protected FrontController servlet = null;

    protected AppControlBean appControlBean = null;

    protected ServletContext servletContext = null;

    protected ResourceBundle resBundle = null;


    public void init(FrontController servlet)

            throws ServletException {


        this.servlet = servlet;


        //MagisUtils.configureLogging(log);

        try {

            ((Jdk14Logger) log).getLogger().setLevel(java.util.logging.Level.ALL);


            ((Jdk14Logger) log).getLogger().addHandler(

                    (java.util.logging.FileHandler)
Class.forName("java.util.logging.FileHandler").newInstance());


            System.out.println("{ Controller init(): JDK1.4 logging configurado con
éxito }");


        } catch (Exception e) {


            System.out.println("Controller init(): Problemas al configurar el
logging/JDK1.4:: " + e);


        }


    }


    public void process(HttpServletRequest request, HttpServletResponse response)

            throws IOException {
```

```
        System.out.println("Controller process() called.");

        log.info("Controller process() called.");



        this.request = request;

        out = response.getWriter();

        servletContext = servlet.getServletContext();



        appControlBean = (AppControlBean)
servletContext.getAttribute(Globals.CONTROLLER_KEY);



        MappingsWrapper mapper = (MappingsWrapper)
servletContext.getAttribute(Globals.MAPPINGS_KEY);

        ContextBeanFactory contextBeanFactory = (ContextBeanFactory)
servletContext.getAttribute(Globals.CONTEXTBEANFACTORY_KEY);

        CommandInterningFactory cif = (CommandInterningFactory)
servletContext.getAttribute(Globals.COMMANDS_KEY);

        GlobalForwardInterningFactory globalFwdInterningFactory =
(GlobalForwardInterningFactory)
servletContext.getAttribute(Globals.GLOBALFORWARDS_KEY);

        resBundle = (ResourceBundle)
servletContext.getAttribute(Globals.RESOURCEBUNDLE_KEY);

        Errors errors = null;

        HttpSession session = request.getSession();




        //session contains a valid _username attribute?

        if (!authenticatedInSession(request)) {


            /* User not-logged-in, let's see whether this request comes from the

             * "login uri", i.e., the controller-configured parameter that specifies

             * the login command request's uri: loginuri.

             */


            log.info("User not logged in.");


            String loginUri = appControlBean.getLoginUri();
```

```
            String requestUri = magis.utils.MagisUtils.getUriFromRequest(request);



            if (!requestUri.equalsIgnoreCase(loginUri)) {

                /*

                 * User is not logged-in and has submitted a non-login request,

                 * we will display the global-forward error page.

                 *

                 */


                log.info("User not logged in and request received is not for login.
Forwarding to general error-page.");


                addSingleErrorToRequest("user.error.notauthenticated", request);


                //Send to global-forward error page

                ForwardBean fb =
globalFwdInterningFactory.findGlobalForward(Globals.GLOBFWD_ERRORPAGE_KEY);

                ultimateConvey(request, response, fb, Globals.DISPATCHFORWARD_KEY);

                return;

            }


        /*

         * Exit from this nested-if structure means that the user is not logged-in

         * but she has accessed the controller-configured login page (loginuri
param)

         * which has allowed her to send in a request with uri "/login.magis". This
request

         * will be honored in the normal way, vía its mapping's command, etc.

         */

        } else {


            log.info("User already in session.");


        }
```

```
        //multipart/caching/response/+

        primaryRequestProcessing(request, response);


        //Obtener el uri de esta request:

        String uri = magis.utils.MagisUtils.getUriFromRequest(request);

        if (uri == null) {

            addSingleErrorToRequest("system.error.missinguri", request);

            ForwardBean fb =
globalFwdInterningFactory.findGlobalForward(Globals.GLOBFWD_ERRORPAGE_KEY);

            ultimateConvey(request, response, fb.getUri(),
Globals.DISPATCHFORWARD_KEY);

            return;

        }

        log.info("Request's uri: {" + uri + "}");


        //Obtener el mapping correspondiente al uri:

        Mapping mapping = mapper.findMappingBean(uri);

        if (mapping == null) {


            log.error("No mapping found for uri {" + uri + "} Redirecting to global
error page.");


            this.addSingleErrorToRequest("system.error.missingmapping", request);


            ForwardBean fb =
globalFwdInterningFactory.findGlobalForward(Globals.GLOBFWD_ERRORPAGE_KEY);


            String dpf;

            if (fb.isRedirect()) {

                dpf = Globals.DISPATCHREDIRECT_KEY;

            } else {

                dpf = Globals.DISPATCHFORWARD_KEY;

            }


            ultimateConvey(request, response, fb.getUri(), dpf);

            return;
```

```
        } else {

            log.info("Uri's mapping found and placed into the request context:\n{" +
mapping + "}");

            request.setAttribute(Globals.OWNMAPPING_KEY, mapping);

        }




        /* Si contextbeanname != null:

         * pedir una context bean a la factory

         */

        ContextBean contextBean = null;

        String beanLogicalName = mapping.getInputCBName();

        if (beanLogicalName != null) {

            contextBean = contextBeanFactory.getContextBean(mapping);

            log.info("Context bean instantiated.");



            //Convertir los parametros de la request en un mapa

            HashMap<String, String[]> reqParamMap = (HashMap<String, String[]>)
request.getParameterMap();

            log.info(printRequestParameterMap(reqParamMap));




            /* Poblar la context bean (contextBean) a partir del mapa de parámetros

             * extraido de la request.

             */

            contextBean.populate(reqParamMap);

            log.info("Context Bean populated:\n" + contextBean);



            /* Si validate == true:

             * validar el formulario

             */

            if (mapping.isValidate()) {
```

```
                errors = contextBean.validate();
                if (errors.size() > 0) {

                    log.error(printValidationErrors(errors));

                    /*
                     * Guardar la cb en el scope indicado en magis-config.xml con
                     * el nombre de atributo indicado en el mismo.
                     */
                    placeBeanIntoContext(mapping.getScope(), beanLogicalName,
contextBean);

                    /*
                     * Multiples Errores en la sesión
                     */
                    addMultipleErrorsToRequest(errors, request);
                    log.info("Form bean validation failed, error messages:\n" +
errors);

                    ultimateConvey(request, response, mapping.getFrom(),
Globals.DISPATCHFORWARD_KEY);
                    return;
                } else {
                    log.info("Form bean validation passed.");
                }

            } else {
                log.info("Form bean validation not requested.");
            }

            placeBeanIntoContext(mapping.getScope(), beanLogicalName, contextBean);

        }
```

```
        /* Ver si se trata de un directForward:
         */
        ForwardBean directForward = mapping.getDirectForward();
        if (directForward.getUri() != null) {
            log.info("Direct forward: {" + directForward + "}");
            ultimateConvey(request, response, directForward.getUri(),
Globals.DISPATCHFORWARD_KEY);
            return;
        }

        /*
         * Ver si se trata de un redirect
         */
        ForwardBean directRedirect = mapping.getDirectRedirect();
        if (directRedirect.getUri() != null) {
            log.info("Direct redirect: {" + directRedirect + "}");
            ultimateConvey(request, response, directRedirect.getUri(),
Globals.DISPATCHINCLUDE_KEY);
            return;
        }

        //Ha de ser un Command:
        if (mapping.getCommandClass() != null) {

            /* Usando el mapping, obtenemos el command
             * y lo ejecutamos:
             */
            Command command = cif.getCommand(mapping);
            if (command != null) {
                log.info("Command for business call invocation: {" + command + "}");

                ForwardBean forward =
                        command.execute(request, response, session, mapping, uri,
servletContext, contextBean);
```

```
                log.info("Session arguments check:");

                Enumeration enu = session.getAttributeNames();

                String attr;

                while (enu.hasMoreElements()) {

                    attr = (String) enu.nextElement();

                    log.info("Session argument: " + attr);

                }

                log.info("LOGGED USER: " +
session.getAttribute(Globals.LOGGEDUSER_KEY));



                if (forward.isRedirect()) {

                    ultimateConvey(request, response, forward.getUri(),
Globals.DISPATCHREDIRECT_KEY);

                    return;

                } else {

                    ultimateConvey(request, response, forward.getUri(),
Globals.DISPATCHFORWARD_KEY);

                    return;

                }



            } else {

                //explain error comando not found error en xml....

                //ultimateConvey(request, response, global fwd to error page,
Globals.DISPATCHFORWARD_KEY);

                return;

            }




        } else {



            log.error("Mapping has a null directForward/directInclude AND has a null
command.");

            addSingleErrorToRequest("system.error.nodestination", request);

            ForwardBean fb =
globalFwdInterningFactory.findGlobalForward(Globals.GLOBFWD_ERRORPAGE_KEY);

            ultimateConvey(request, response, fb, Globals.DISPATCHFORWARD_KEY);
```

```
            return;

        }

    }

    /*
     * La sesión activa contiene un atributo _username, esto es, un usuario ha
     * hecho login con éxito y, por tanto, ha sido autenticado?
     */
    protected boolean authenticatedInSession(HttpServletRequest request) {
        boolean auth = false;

        HttpSession session = request.getSession();
        String username = (String) session.getAttribute(Globals.LOGGEDUSER_KEY);

        if (username != null) {
            auth = true;
        }

        return auth;
    }

    protected void addMultipleErrorsToRequest(Errors errorsToAdd, HttpServletRequest
request) {

        magis.errors.Errors errs = (magis.errors.Errors)
request.getAttribute(Globals.ERRORS_KEY);

        if (errs == null) {
            request.setAttribute(Globals.ERRORS_KEY, errorsToAdd);
        } else {
```

```
            for (Object o : errorsToAdd) {
                errs.add((magis.errors.Error) o);
            }
        }

    }

    protected void addSingleErrorToRequest(String key, HttpServletRequest request) {

        String errorString = resBundle.getString(key);

        if (errorString != null) {
            magis.errors.Error e = new magis.errors.Error(key, errorString);

            magis.errors.Errors errs = (magis.errors.Errors)
request.getAttribute(Globals.ERRORS_KEY);

            if (errs == null) {
                errs = new Errors();
                request.setAttribute(Globals.ERRORS_KEY, errs);
            }

            errs.add(e);

        }

    }

    /*
     * Los dos métodos siguientes efectuan el envio de la request al destino
     * inidcado y utlizando forward o redirect, dependiendo del caso concreto.
     *
     */
    protected void ultimateConvey(HttpServletRequest request,
```

```
            HttpServletResponse response,
            ForwardBean dest,
            String dispatchMethod) {

        ultimateConvey(request,
                response,
                dest.getUri(),
                dispatchMethod);

    }

    protected void ultimateConvey(HttpServletRequest request,
            HttpServletResponse response,
            String dest,
            String dispatchMethod) {

        if (dispatchMethod.equals(Globals.DISPATCHFORWARD_KEY)) {

            log.info("Dispatching (forward) to " + dest);

            try {
                RequestDispatcher view = request.getRequestDispatcher(dest);
                view.forward(request, response);
            } catch (ServletException ex) {
                log.error(ex);
            } catch (java.io.IOException ex) {
                log.error(ex);
            }

        } else if (dispatchMethod.equals(Globals.DISPATCHINCLUDE_KEY)) {

            log.info("Dispatching (include) to " + dest);
            try {
```

```
                RequestDispatcher view = request.getRequestDispatcher(dest);
                view.include(request, response);
            } catch (ServletException ex) {
                log.error(ex);
            } catch (java.io.IOException ex) {
                log.error(ex);
            }

        } else {
            log.info("Redirecting to " + dest);
            try {
                response.sendRedirect(dest);
            } catch (IOException ex) {
                log.error(ex);
            }
        }

    }

    protected void placeBeanIntoContext(String scope, String beanLogicalName,
ContextBean contextBean) {
        /* Guardar la context bean en request/session según
         * el parámetro scope de magis-config.xml y usando el
         * nombre propio de la bean (contextbenaname)
         */

        if (scope.equalsIgnoreCase("request")) {

            request.setAttribute(beanLogicalName, contextBean);
            log.info("Context bean saved into the request's context.");

        } else if (scope.equalsIgnoreCase("session")) {
            HttpSession session = request.getSession();
```

```
            synchronized (session) {

                session.setAttribute(beanLogicalName, contextBean);

            }


            log.info("Context bean saved into the session's context.");

        }


    }


    protected void primaryRequestProcessing(HttpServletRequest request,
HttpServletResponse response) {




        //Multi-part:

        request = decorateRequestForMultipart(request);


        //Fijar el tipo de la response

        setResponseType(response);


        //Locale:

        setLocale(request);


        //Caching:

        setCaching(response);


        //hook:

        customProcessing(request, response);




    }


    private String printValidationErrors(Errors err) {

        StringBuffer sb = new StringBuffer();
```

```
        sb.append("Context bean validation error list:\n{\n");

        for (Object e : err.getErrors()) {
            sb.append((magis.errors.Error) e + "\n");
        }

        sb.append("}");
        return sb.toString();
    }

    private String printRequestParameterMap(HashMap<String, String[]> reqParamMap) {
        StringBuffer s = new StringBuffer();

        Iterator paramNames = ((Set<String>) reqParamMap.keySet()).iterator();

        s.append("Request parameter map built:\n");

        while (paramNames.hasNext()) {
            String ss = (String) paramNames.next();
            String vals[] = reqParamMap.get(ss);
            for (int i = 0; i < vals.length; i++) {
                s.append("{" + ss + " = " + vals[i] + "}\n");
            }
        }

        return s.toString();
    }

    /* Obtener la lista de parámetros de la request, como comprobación.
     * USADO SÓLO EN PRUEBAS INICIALES.
     */
    private String printRequestParameterList(HttpServletRequest request) {
        StringBuffer s = new StringBuffer();
```

```
        Enumeration paramNames = request.getParameterNames();

        s.append("Request parameter list:\n-----------------------\n");

        while (paramNames.hasMoreElements()) {
            String ss = (String) paramNames.nextElement();
            String vals[] = request.getParameterValues(ss);
            for (int i = 0; i < vals.length; i++) {
                s.append("{" + ss + " = " + vals[i] + "}\n");
            }
        }

        return s.toString();
    }


    /*
     *
     * Este método envuelve las requests "multipart" en un wrapper que ayuda
     * a que su uso posterior no produzca errores.
     *
     * La clase MultipartRequestWrapper() utilizada proviene de Struts 1.0 y
     * se ha usado según los términos de su licencia (Ver texto fuente incluido
     * en la clase correspondiente (Apache Software Foundation). Es el único
     * texto funete no original incluido en este proyecto.
     *
     */

    protected HttpServletRequest decorateRequestForMultipart(HttpServletRequest
request) {

        if ("POST".equalsIgnoreCase(request.getMethod())) {
            String ct = request.getContentType();
```

```
            if ((ct != null) && ct.startsWith("multipart/form-data")) {
                request = new MultipartRequestWrapper(request);
                log.info("multipart/form-data request decorated succesfully");
            } else {
                log.info("Non-multipart/form-data request, avoiding request-
decoration");

            }
        }

        return request;
    }

    protected void setResponseType(HttpServletResponse response) {

        String ct = appControlBean.getContentType();

        if (ct != null) {
            response.setContentType(ct);
            log.info("Response type set to {" + ct + "} per magis-config.xml
instructions.");
        } else {
            ct = Globals.DEFAULT_RESPONSE_TYPE;
            response.setContentType(ct);
            log.info("Response type set to default value {" + ct + "}");
        }

    }

    protected void setLocale(HttpServletRequest request) {

        HttpSession session = request.getSession();
```

```
        if (appControlBean.getLocale() == false &&
                session.getAttribute(Globals.LOCALE_KEY) == null) {

            Locale locale = request.getLocale();

            if (locale != null) {

                synchronized (session) {
                    session.setAttribute(Globals.LOCALE_KEY, locale);
                }

            }
        }


        log.info("request's locale set into the session: {" +
request.getSession().getAttribute(Globals.LOCALE_KEY) + "}");
    }

    protected void setCaching(HttpServletResponse response) {


        if (appControlBean.getNoCache()) {
            response.setHeader("Pragma", "No-cache");
            response.setHeader("Cache-Control", "no-cache,no-store,max-age=0");
            response.setDateHeader("Expires", 1);
            log.info("Caching configuration, page-caching not allowed.");

        } else {
            log.info("Caching configuration, page-caching allowed.");
        }
```

```
    }

    protected void customProcessing(HttpServletRequest request, HttpServletResponse
response) {

        //hook

        log.info("Custom processing -empty- hook called.");

    }

    public void cleanUp() {

        log.info("Application controller cleanUp() called.");

    }
}

/*FIN*/
```