tecnim@p Sevilla 2006
30 mayo a 2 junio
060.es
24x7x365

Comunicación

209

HACIA UNA ARQUITECTURA CON JAVASERVER FACES, SPRING, HIBERNATE Y OTROS FRAMEWORKS

Juan Medín Piñeiro
Consultor
Thales

Antonio García Figueras
Jefe Sección Análisis y Desarrollo
IMSERSO

Juan Medín Piñeiro
Antonio García Figueras

Palabras clave

JavaServer, Faces, JSF, Spring, Hibernate, Acegi, Java, framework, arquitectura, Imserso, J2EE, MVC, UML, IDE, DAO.

Resumen de su Comunicación

Uno de los aspectos más interesantes de Java es el ritmo al que evolucionan tanto sus tecnologías como las arquitecturas diseñadas para soportarlas. Cada poco tiempo aparecen nuevas librerías, herramientas, frameworks, patrones, etc, y se hace muy difícil elegir entre ellos y, sobre todo, combinarlos de una manera eficiente que aproveche toda su flexibilidad y potencia.

El presente documento explica la satisfactoria experiencia del IMSERSO en la adopción de varias de las últimas tecnologías más útiles disponibles en Java, cubriendo todos los aspectos, desde los puramente técnicos hasta los metodológicos –así como los problemas que han surgido- para lograr este objetivo.

Este proceso ha sido una evolución hacia un sistema totalmente modular, conformado por frameworks especializados líderes en su área (JavaServer Faces, Spring, Hibernate, etc.), con una arquitectura que facilita un desarrollo conducido por el modelo y con una separación limpia en capas -en donde pueden participar desarrolladores especializados-. Permite además introducir tecnologías existentes (EJB's, Web-Services, RMI, etc) de una forma limpia y sin tener que tocar el código existente.

El objeto final de esta presentación es ofrecer una guía clara y sencilla para todas aquellas personas que cuentan con un equipo propio de desarrollo y que no están obteniendo el máximo provecho del mismo o que desean realizar una migración a las últimas tecnologías disponibles para aprovechar su potencia.

Juan Medín Piñeiro
Antonio García Figueras

HACIA UNA ARQUITECTURA CON JAVASERVER FACES, SPRING, HIBERNATE Y OTROS FRAMEWORKS

1. Introducción

El Imserso (Instituto de Mayores y Servicios Sociales) es una entidad Gestora de la Seguridad Social adscrita al Ministerio de Trabajo y Asuntos Sociales. En el plano informático da servicio a unos 500 usuarios internos y soporte a diversos centros repartidos por todo el Estado que dependen de él. Cuenta con un Área de Informática formada por 50 personas, 15 de las cuales pertenecen a desarrollo. Se ha trabajado durante muchos años con un entorno de programación estructurada empleando diferentes lenguajes: Cobol, Clipper, Access, Ingres 4GL. Como servidor principal disponemos de un Sun Fire 15K con 9 dominios y sistema de cluster junto con varios servidores NT. Contamos con dos gestores de bases de datos relacionales: Ingres (a desaparecer) y Oracle 9i.

Hace tres años se decidió cambiar el paradigma de programación a objetos y se eligió como lenguaje Java. Se comenzó a formar a los desarrolladores en Java, JSP y servlets. Durante dos años se realizaron con éxito diferentes aplicaciones con estas tecnologías apoyándose en una serie de utilidades propias que se habían creado a nivel interno para facilitar el trabajo. A medida que se iba adquiriendo experiencia se comenzaron a percibir las posibilidades de emplear nuevas tecnologías y la necesidad de mejorar el entorno de desarrollo. Es por esto que se decidió valorar otras alternativas.

En octubre del 2005 se evaluó una nuevo framework de presentación, JavaServer Faces, que resolvía varios de los problemas que existían aportando una librería comercialmente estándar y aceptada por la comunidad de código abierto.

Para aprovechar este nuevo framework se precisaban otras modificaciones: el estilo de desarrollo, heredado de años de programación estructurada no reflejaba algunos de los elementos propios de la programación orientada a objetos y era necesario acometer también esos cambios. Esto llevó y, la vez, permitió utilizar otros frameworks complementarios para gestionar la persistencia, la seguridad y la integración entre componentes.

2. Justificación de la arquitectura elegida

2.1. Elección de capas y componentes

El objetivo principal de la arquitectura es separar, de la forma más limpia posible, las distintas capas de desarrollo, con especial atención a permitir un modelo de domino limpio y a la facilidad de mantenimiento y evolución de las aplicaciones. Otros elementos importantes han sido la facilidad del despliegue y el empleo de las mejores tecnologías disponibles en la actualidad –en contraposición al continuismo con opciones que se consideran anticuadas a día de hoy-.

Se deseaba una arquitectura que permitiese trabajar en capas y que sirviese tanto para las aplicaciones en la intranet como en Internet, así como disponer de la flexibilidad necesaria para poder emplear un cliente ligero (navegador web, Wap) o un cliente pesado (Swing, SWT, etc). Era fundamental no tener que rescribir ningún código y que las capas comunes fuesen reutilizadas sin cambios en ambos casos.

Para lograr esto se eligió el patrón MVC (Modelo-Vista-Controlador) que permite una separación limpia entre las distintas capas de una aplicación.

Para la capa de presentación (la vista) se buscaba un framework que nos proporcionase una mayor faci-

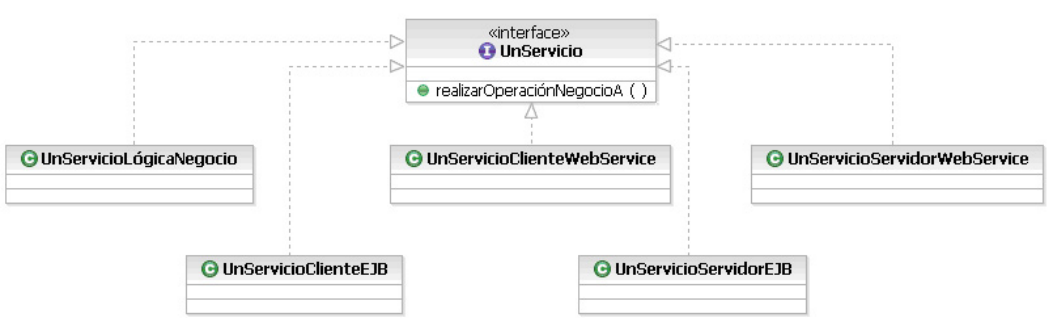
lidad en la elaboración de pantallas, mapeo entre los formularios y sus clases en el servidor, la validación, conversión, gestión de errores, traducción a las diversas lenguas autonómicas y, de ser posible, que facilitase también el incluir componentes complejos (menús, árboles, ajax, etc) de una forma sencilla y –sobre todo- fácil de mantener. Para esta capa se ha elegido JavaServer Faces[1].

En la capa de negocio y persistencia, se decidió desde el primer momento no emplear EJB's por su elevado coste de desarrollo y mantenimiento así como su falta de flexibilidad y coste en tiempo para los cambios. Se optó por una solución basada en servicios (no necesariamente servicios web, aunque permitiendo su integración de forma limpia) que trabajaban contra un modelo de dominio limpio. La persistencia de las clases se sustenta en DAO's (Objetos de Acceso a Datos), manteniendo aislada la capa de persistencia de la capa de negocio. Tanto los servicios como los DAO's así como el propio modelo son realmente POJOs (clases simples de Java), con la simplicidad que conllevan y sin dependencias reales con ningún framework concreto. Para realizar esta integración se ha elegido Spring[1].

Para la capa de persistencia se pensó en utilizar alguna herramienta ya existente, que permitiese realizar el mapeo objeto-relacional de una forma cómoda pero potente, sin tener que implementarlo directamente mediante JDBC. Esto último conllevaría, por ejemplo, un esfuerzo importante en un caso de cambio de base de datos (como ha ocurrido), en la gestión de la caché, la utilización de carga perezosa, etc. La herramienta elegida finalmente fue Hibernate[1].

Para la de seguridad se buscaba más potencia que la actualmente definida para los contenedores web que, aunque es ampliada por cada fabricante, se vuelve dependiente del mismo. Se buscaba una solución que fuese totalmente independiente de cada contenedor concreto, común a todos ellos y que ofreciese una gran versatilidad (posibilidad de trabajar contra LDAP, base de datos, controlar el número de sesiones del usuario, etc). Acegi[1]. cumplía todos estos puntos y fue seleccionado para esta función.

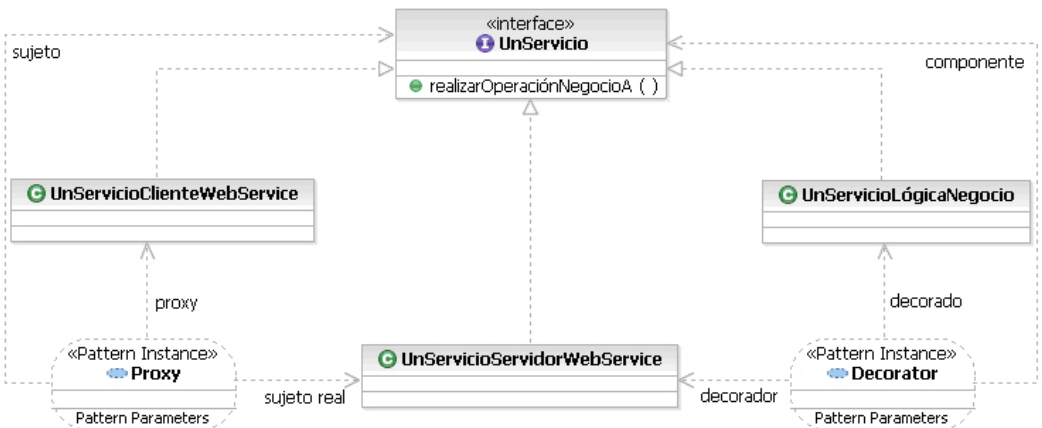
Las jerarquías son sencillas y limpias. Spring gestiona todos estos objetos y sus dependencias.



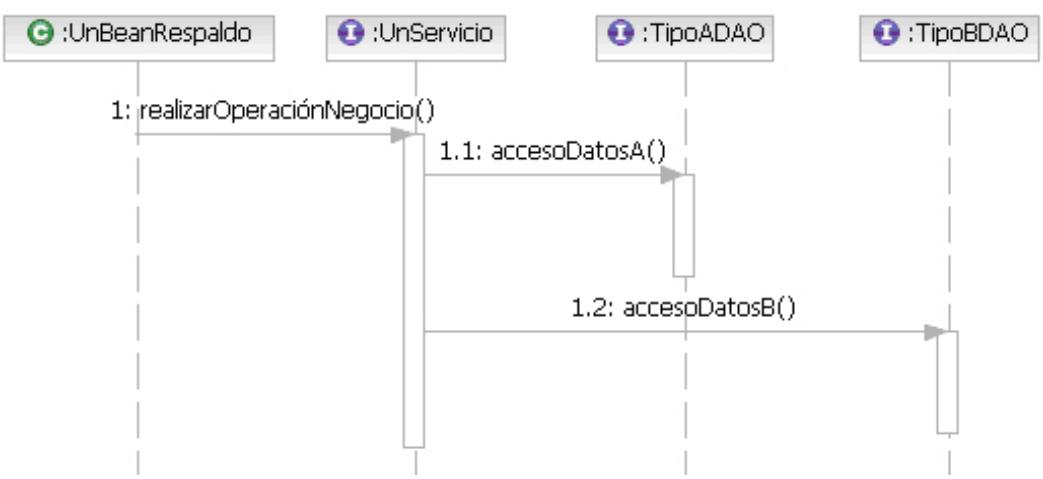
El uso de interfaces limpios permite incluir funcionalidad extra sin cambiar la lógica existente

[1] En el punto 4 se comenta con detalle cada framework.

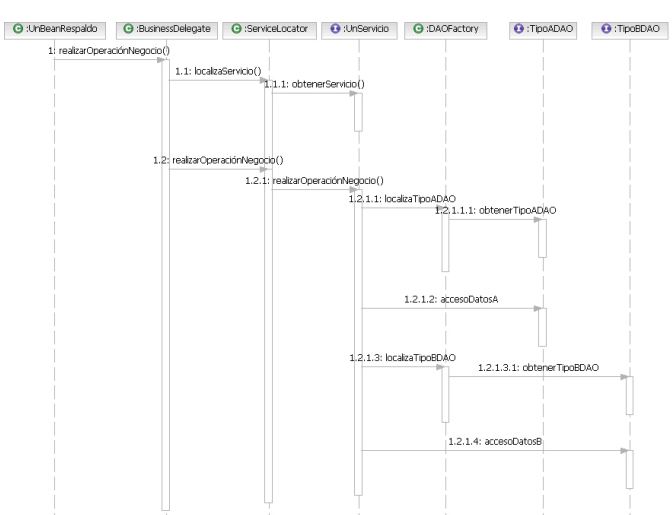
(Un ejemplo sobre como invocar una operación de negocio sin rescribir ni una sola línea de código. Spring favorece este tipo de escenarios.)



Invocación de una operación de negocio en una arquitectura con Spring



Invocación de la misma operación de negocio en una arquitectura clásica



2.2. Beneficios de la arquitectura diseñada

La primera ventaja se deriva de la modularidad del diseño. Cada una de las partes empleadas (JSF para la vista, Spring para la integración, Hibernate para la persistencia, Acegi para la seguridad) es intercambiable de forma sencilla y limpia por otras soluciones disponibles. Por ejemplo, para la vista se emplea JavaServer Faces, pero nada impide emplear también una aplicación de escritorio mediante Swing o SWT sin tener que tocar ni una sola línea de código de las restantes capas. Es más, nada impediría que se pudiese disponer de una aplicación con una parte de la capa de presentación en JSF y otra parte, para otro tipo de usuarios, en Swing, ambas funcionando a la vez y compartiendo todo el resto del código (lógica de negocio, persistencia, integración, seguridad, etc).

De igual forma, si se desean cambiar elementos de la capa de persistencia empleando otro framework para el mapeo diferente de Hibernate –o sencillamente no utilizar ninguno- tan sólo serían necesarios cambios en esa capa.

De la misma manera se podrían sustituir cualquiera de las otras capas. El diseño se ha hecho reduciendo al mínimo posible las dependencias entre ellas.

3. Frameworks empleados

JavaServer Faces

JavaServer Faces es el estándar presentado por Sun para la capa de presentación Web. Forma parte de la especificación J2EE 5 -que deberán cumplir todos los servidores de aplicaciones- y se erige como una evolución natural de los frameworks actuales hacia un sistema de componentes.

Es un estándar sencillo que aporta los componentes básicos de las páginas web además de permitir crear componentes más complejos (menús, pestañas, árboles, etc). Ya hay disponibles diferentes implementaciones de la especificación, tanto comerciales como de código abierto, así como librerías de componentes adicionales que amplían la funcionalidad de esos componentes iniciales.

JSF ha sido acogida por la comunidad como “el framework que hacía falta”. Muchos de los proyectos de código abierto y las compañías con más influencia lo han identificado como el framework de presentación web del futuro. JBoss ha integrado directamente JSF con EJB3 mediante el proyecto Seam (abanderado además por Gavin King, el líder del equipo de desarrollo Hibernate). IBM lo ha incorporado como mecanismo de presentación estándar para su entorno, desarrollando no sólo el soporte completo en su IDE, sino nuevos componentes. Oracle es otra de las compañías que más ha apostado por esta tecnología, ofreciendo la que posiblemente sea la mayor oferta de componentes propios.

Dentro de los “pesos pesados” en Java, nombres que han guiado a la industria en el pasado como Matt Raible, Rick Hightower, David Geary, el mencionado Gavin King y por supuesto el propio creador de Struts y ahora arquitecto de JSF, Craig McClanahan.

Estas son las principales razones por las que se descartaron otros frameworks:

Struts.- Aunque es el framework que ofreció un camino hacia el MVC en Java para desarrollos Web, es también uno de los que más ha acusado el paso de los años en tiempo de desarrollo y flexibilidad. Hoy existen frameworks que comparten los mismos principios pero con más potencia, elegancia y flexibilidad.

Spring MVC.- Una de las alternativas a Struts que ha incorporado una lógica de diseño más sencilla y que cuenta con todo el abanico de librerías de Spring. No obstante sigue también la misma filosofía y no ofrece

nuevas mejoras que sean comparables a JSF.

Tapestry.- Reconocido como uno de los frameworks más potentes para la web, es también uno de los más complejos y de los que tienen una curva de aprendizaje más pronunciada. Ofrece un sistema de componentes al igual que JSF pero no dispone de la comunidad, la documentación, el soporte y –por supuesto- la estandarización de JSF.

MyFaces

MyFaces es un proyecto de la fundación Apache que ofrece una implementación en código abierto de JavaServer Faces, así como un amplio conjunto de componentes adicionales. Entre ellos se dispone de un menú, árboles, pestañas, componentes para gestionar el estado de los diálogos, etc. A lo largo del tiempo se van incorporando componentes cada vez más potentes y ya se encuentran en pruebas componentes basados en Ajax.

Realmente el uso de MyFaces no impide el uso de otras librerías de componentes, así que su uso dependen tan solo de la utilidad que encontremos en la funcionalidad que ofrece.

Spring

Spring es un conjunto de librerías "a la carta" de entre las que podemos escoger aquellas que faciliten el desarrollo de nuestra aplicación. Entre sus posibilidades más potentes está su contenedor de Inversión de Control (Inversión de Control, también llamado Inyección de Dependencias, es una técnica alternativa a las clásicas búsquedas de recursos vía JNDI. Permite configurar las clases en un archivo XML y definir en él las dependencias. De esta forma la aplicación se vuelve muy modular y a la vez no adquiere dependencias con Spring), la introducción de aspectos, plantillas de utilidades para Hibernate, iBatis y JDBC así como la integración con JSF.

Es uno de los proyectos más sorprendentes en el panorama actual en Java en el grado en que ayuda a que los diferentes componentes que forman una aplicación trabajen entre sí, pero no establece apenas dependencias consigo mismo. Esta es la primera característica de este framework. Sería posible retirarlo sin prácticamente cambiar líneas de código. Lo único que sería necesario es, lógicamente, añadir la funcionalidad que provee, ya sea con otro framework similar o mediante nuestro código.

A nivel de soporte de la comunidad, Spring es uno de los proyectos con más actividad, con desarrollos dentro y fuera del propio framework (de hecho Acegi, el sistema de seguridad empleado, fue realizado por colaboradores ajenos al framework). Actualmente dispone de soporte comercial a través de Interface21, la empresa creadora, así como otros fabricantes que dan soporte en su área.

Otras alternativas que fueron descartadas como contenedor IoC

Avalon.- Fue una de las primeras opciones consideradas y es uno de los contenedores IoC más antiguos. Sin embargo, lejos de ser una ventaja, esto se nota mucho en su diseño, mucho menos flexible y elegante que Spring. Su comunidad es realmente pequeña y no se han encontrado muchas referencias a usos en aplicaciones reales.

Picocontainer y Nanocontainer:- pequeños contenedores de IoC que no disponen –con muchísima diferencia- de la funcionalidad y conjunto de librerías de Spring. Parecen útiles para ser empleados en applets y en aplicaciones J2ME, pero no en una aplicación de tipo Enterprise.

HiveMind.- Una de las opciones más interesantes después de Spring. De la mano de Apache es un completo contenedor IoC. Se han visto sin embargo tres grandes áreas en donde Spring es superior: conjunto de librerías, apoyo de la comunidad a todos los niveles y elegancia de uso.

Acegi

Acegi es un gestor de seguridad que está diseñado fundamentalmente para ser usado con Spring y en el que destaca su versatilidad. Acegi proporciona una capa que envuelve diversos estándares de seguridad presentes en Java y ofrece una forma unificada de configuración a través de un descriptor en XML.

Cubre la capa web y la de negocio. A nivel Web captura todas las peticiones mediante la implementación de un filtro y a nivel de métodos mediante interceptación a través de AOP. En ambos casos permite aplicar los criterios de seguridad que trae de serie o añadir nuevas opciones de forma sencilla implementando los interfaces diseñados a tal fin.

Acegi se ha elegido como opción frente a los sistemas propietarios de los diferentes vendedores por su universalidad de uso –no es necesario cambiar nada si se cambia de proveedor en los servidores- así como por su potencia –que engloba los API's de seguridad de Java-.

Hibernate

Hibernate es un motor de persistencia de código abierto. Permite mapear un modelo de clases a un modelo relacional sin imponer ningún tipo de restricción en ambos diseños. Cuenta con una amplia documentación, tanto a nivel de libros publicados como disponible gratuitamente en su Web. A nivel comercial está respaldado por JBoss, que proporciona servicios de soporte, consultoría y formación en el mismo.

Actualmente es el rey indiscutible de la persistencia. Desde su versión 1.0, el motor no ha parado de evolucionar, incorporando todas las nuevas ideas que se iban incorporando en este campo.

Hoy en día, en su versión 3.1, ya soporta el estándar EJB 3 (su autor es uno de los principales integrantes del JCP que está definiendo esta especificación) por lo que ya se puede elegir desarrollar aplicaciones empleando EJB 3 -que correrán en cualquier contenedor de EJB's que soporte J2EE 5- o aplicaciones independientes sin . Esto no solo blinda la inversión de tiempo en Hibernate de cara al futuro, sino que, de ser útil, nos hace totalmente independientes del mismo.

Otras opciones valoradas fueron:

iBatis.- iBatis no es exactamente un mapeador objeto-relacional, o al menos en el sentido puro del término. Es un sistema para mapear objetos contra una base de datos relacional mediante DAO's en donde todo el código en SQL debe ser escrito (no dispone de los mapeos que ofrecen a Hibernate la potencia que le ha hecho famoso). Es más sencillo que Hibernate y parece un proyecto muy interesante cuando hay que trabajar contra un modelo de base de datos ya existente si esta es muy compleja o si tiene mucha peculiaridades o defectos de diseño.

Torque.- Otra de las opciones de mapeo por medio de DAO's. No se encontraron muchos proyectos –fuera de la esfera del propio Torque- que lo utilizasen ni dispone de la potencia de Hibernate (a nivel de mapeos y utilidades como su doble caché, carga perezosa, etc). Además su comunidad es, comparada con la de Hibernate, extremadamente reducida.

JUnit

Estándar actual en Java en código abierto para la creación de tests unitarios. Su limpieza de diseño, la amplia documentación existente y lo estandarizado de su uso lo hacen la mejor opción para este tipo de

tests.

Actualmente, para pruebas unitarias, JUnit es prácticamente la opción universal.

C3PO

C3PO es un pool de conexiones de código abierto. Su uso está muy estandarizado, la documentación disponible sobre él es abundante y tiene un desarrollo continuo por parte de la comunidad.
Siendo un simple pool de conexiones, C3PO es sencillo de emplear y rápido de configurar. Es soportado por Hibernate de forma trivial y su uso es transparente.

Crystal Reports

Crystal Reports es la herramienta líder en el mercado para la generación de informes. Dispone de componentes que facilitan la presentación de informes en web mediante JavaServer Faces, facilitando su integración dentro del API.

Se evaluó la posibilidad de usar Jasper Reports, pero su potencia no es comparable a Crystal Reports –con el que además el equipo de desarrollo ya contaba con experiencia-.

Ant

Ant es una herramienta de código abierto para ensamblar aplicaciones y, de forma general, realizar cualquier operación repetitiva del desarrollo que se pueda mecanizar mediante un script.

Otra opción muy interesante -disponible también por parte de Apache- es Maven. El problema es que uso no está muy extendido y, en esencia, ambos tienen las mismas funciones aunque con distinta filosofía.

CVS

CVS es el conocido sistema de control de código concurrente. Aunque en la actualidad se está comenzando a sustituir por Subversion, sigue siendo la opción de referencia y la más probada para gestionar un repositorio de código.

CVS es el sistema casi universal para guardar código concurrente (aunque en la esfera de Microsoft se use sobre todo SourceSafe) y hasta que el Subversión –el proyecto que parece ser su relevo- adquiera su madurez sigue siendo la opción de referencia.

Log4Java

Log4Java, otra de las librerías de código abierto de Apache, es la opción de referencia para gestionar todos los aspectos de los logs de desarrollo. Aunque se ha visto reemplazada en el API de Java por las nuevas funcionalidades del log en el JDK 1.5 su uso –ampliamente extendido en todas las áreas- y su probado comportamiento en grandes aplicaciones la mantienen todavía como la más usada.

Log4Java es empleado de forma casi universal como gestor de logs. Si bien ya se comienza a emplear el API para logs de Java, su funcionalidad es muy similar a Log4Java.

XDoclet2

XDoclet2 es una herramienta para la generación de código o XML a partir de "doclets", marcas que se

incluyen en los comentarios de un programa. De esta forma, a partir de los comentarios de clases, métodos o variables, es posible generar el XML asociado para Hibernate. Esto facilita el desarrollo y, de forma análoga, documenta el código.

La otra opción sería XDoclet 1, pero la creación de nuevas plantillas en XDoclet2 (empleando Velocity como gestor de las plantillas en vez del pseudo lenguaje que era el XML de XDoclet 1) lo hacen más interesante de cara a añadir nuevas funcionalidades.

4. Entornos de desarrollo

El desarrollo en Java se inició en NetBeans, un entorno de desarrollo gratuito realizado con el apoyo de Sun y uno de los primeros IDEs disponibles en código abierto para Java. NetBeans 3.6 fue empleado con éxito durante los primeros años para el desarrollo de páginas web basadas en JSP puro.

A partir de la transición a JavaServer Faces se buscó otro IDE de características similares pero que pudiese ofrecer un editor visual para JSF y otras herramientas útiles para el desarrollo. Continuando con los productos de Sun se evaluó el Java Studio Creator, IDE liderado por el propio arquitecto de JSF y comercializado por Sun. Su versión 1 era realmente una beta y contaba con numerosos bugs de todo tipo que hacían muy complicado el desarrollo en el día a día. Además de esto, no contaba con ningún soporte de herramientas avanzadas (refactoring, seguimiento y navegación por código, editores XML, CVS, etc, etc). A pesar de disponer de un buen editor WYSIWYG se abandonó en poco tiempo en busca de alternativas.

Un IDE que se evaluó durante este periodo fue el IDEA de IntelliJ. Este es sin lugar a dudas el IDE más potente en cuanto a lo que es el trabajo con Java puro (dispone de potentes analizadores de código, la más completa colección de refactorings, plantillas para el desarrollo, etc, etc). Desafortunadamente, el soporte para J2EE era muy pobre en todos los sentidos (ni siquiera ahora, en el momento de escribir este, documento soporta JSF) y fue descartado de inmediato.

El siguiente IDE fue Eclipse en conjunción con las WebTools. El IDE era extremadamente sólido, con soporte para muchas características avanzadas de edición en Java y con un ritmo de desarrollo muy rápido (cada 6 semanas se presenta una nueva versión). Las WebTools, a pesar de estar en esa época en fase beta, eran también muy maduras (mucho más que el Java Studio Creator en su versión beta). El problema es que ninguno de los dos soportaba JSF –en general el soporte en Eclipse para J2EE era muy pobre-.

Pensando en mantener Eclipse como base se buscaron añadidos comerciales que pudieran suplir las carencias de cara a JSF. Se probó MyEclipse, una conocida extensión que aportaba nuevos añadidos para JSF así como el Exadel Studio, otra ampliación comercial. Ambas incorporaban toda la potencia de Eclipse pero, en la época, todavía no contaban con un soporte maduro de JSF, especialmente de cara a un editor real con componentes JSF.

Más tarde se probó la herramienta Rational Software Architect (RSA), en su versión 6. Este IDE está basado en Eclipse 3.0 y es la opción comercial que ofrece IBM (la empresa que cedió la base de código de Eclipse como código abierto). Se trataba, con mucho, de la versión más madura y potente para la gestión tanto de JSF como de los demás elementos necesarios en el desarrollo de una página web. En su versión Architect se incluía además soporte para la generación de UML, conversiones UML a código, perfiladores de rendimiento, soporte para varios servidores de aplicaciones, soporte para la elaboración de modelos relacionales, soporte para JUnit, etc.

Aunque esta herramienta tenía también algunos puntos negativos (siempre está varias versiones por detrás de Eclipse, no tiene soporte para JDK 1.5, tiene bastantes bugs, etc) se trata de la mejor opción para el desarrollo en Java / J2EE.

Otra de las ventajas de RSA es que, al basarse en Eclipse, dispone de muchos plugins para realizar todo tipo de funciones. Este ha sido por tanto el IDE elegido para el desarrollo de futuras aplicaciones en Java.
Los requisitos de RSA 6 son muy elevados: 2GB de memoria, un procesador rápido (en máquinas a 3Ghz en ocasiones se ejecuta con una cierta lentitud) y un disco duro con una buena tasa de transferencia. Se hace también muy deseable una pantalla de alta resolución, especialmente durante el tiempo de modelado.

5. Posibles áreas de evolución

Existen diferentes opciones que se están evaluando en la actualidad y que complementarían el desarrollo actual:

Herramientas de generación de código / MDA

La definición de las clases que componen el dominio puede ser generada a partir de la información suministrada por los modelos UML empleados en el diseño. Esto ayudaría además a generar la mayor parte de los ficheros de configuración de Hibernate (que es siempre una tarea tediosa). Para este fin se está evaluando uno de los proyectos de código abierto de MDA (Model Driven Architecture), AndroMDA. Esta herramienta permite generar no solo el modelo de clases, sino los servicios, DAO's , flujo de la aplicación, operaciones CRUD, etc.

Esta herramienta se ajusta perfectamente a la arquitectura que se está empleando, potencialmente reduciendo el tiempo de desarrollo a la vez que se realiza una documentación estándar de la misma.

Desarrollo de aspectos para cubrir algunas áreas transversales al desarrollo como logging, seguridad, etc.

El uso de aspectos simplificaría muchas áreas que la programación, incrementando la reutilización de código y mejorando las posibilidades que ofrece la programación orientada a objetos clásica.

Con la llegada de Spring el coste de integración se ha reducido de forma drástica. Los aspectos se pueden emplear como simples clases que Java, que se inyectan en el código existente mediante los mecanismos definidos en Spring.

Integración de frameworks que proporcionan funcionalidad añadida

Como se ha comentado en otros puntos, existe un número creciente de nuevas librerías y herramientas basadas en Hiberate, JSF o Spring. De entre ellos se han identificado como útiles:

Shale.- Evolución del framework Struts. Está construido sobre JSF y aporta un mayor control sobre el ciclo de vida, la gestión del flujo de la aplicación, los estados de diálogo, integración con Spring, etc. No reemplazaría a JSF, sino que añadiría nuevas funcionalidades.

Facelets.- Sistema simplificado de presentación, en donde es posible diseñar de forma libre una página web y luego asociarle los componentes JSF precisos. Aporta mayor libertad al diseñador y mejora los informes de errores que tiene JSF entre otras cosas.

Tobago.- Ampliación del proyecto MyFaces (conjunto de componentes extra para JSF) que incrementa el número de componentes disponibles alejándose del diseño en HTML a favor de un sistema más sencillo y eficiente

Seam.- Integración de Hibernate o un contenedor EJB3 con JSF mediante anotaciones. Es uno de los frameworks emergentes más interesantes –liderado por Gavin King, el creador de Hibernate y englobado dentro de JBoss- que intenta simplificar y acelerar el desarrollo de aplicaciones, proponiendo una forma alternativa de trabajar con la vista y la capa de persistencia. El propio Seam emplea Facelets, descrito arriba.

Spring 2.- La mayor revisión hasta la fecha de este framework. Incluye soporte para anotaciones, ha incrementado enormemente el soporte para programación orientada a aspectos –siendo actualmente una de las opciones más potentes en este área-, nuevas posibilidades para vincular los objetos con los que se trabaja, etc, etc.

6. Problemas durante el desarrollo

Integración de Crystal Reports

Uno de los peores problemas vino de la mano de Crystal Reports, para el que además se disponía de un contrato de soporte durante el periodo de pruebas.

Tanto a nivel de desarrollo con JSP o servlets como más tarde con JavaServer Faces, Crystal se ha caracterizado por la falta de documentación, la ausencia de foros especializados, la mala calidad del soporte (que hacía que algunas consultas llevasen hasta semanas), la ausencia de información durante el desarrollo (trazas, herramientas de seguimiento, etc), falta de compatibilidad con el resto de las librerías disponibles, etc.

Pese a todo, mediante prueba y error, se ha logrado integrar de forma correcta y limpia. De esta forma se han podido aprovechar las ventajas que proporciona Crystal Reports: la comodidad de tener un buen editor de diseño de informes, su potencia y la limpia integración con componentes JSF.

Se podría decir que, en nuestra opinión, Crystal Reports es un buen producto con un mal soporte.

Escasa documentación en Español

Al tratarse de una tecnología emergente que lleva poco tiempo en el mercado –aunque existe mucha documentación de todo tipo en inglés- apenas hay material en castellano. Probablemente en poco tiempo comience a estar disponible.

Curva de aprendizaje

El problema de toda nueva tecnología –y en especial un cambio tan radical como el llevado a cabo- viene de la dificultad de, no solo comprender los nuevos frameworks, sino llegar a dominarlos y aprovechar toda la potencia y funcionalidades que ofrecen.

A pesar de ello, los frameworks de código abierto (JSF, Hibernate, Spring, etc) favorecen en gran medida la resolución de dudas y problemas a través de foros muy participativos y existe amplia documentación tanto en forma impresa como en Internet (a diferencia de algunos productos comerciales como Crystal Reports).

Localización de los componentes necesarios de JSF

Uno de los elementos que proporcionan mayor riqueza pero que a la vez requiere un cierto trabajo es localizar –y aprender a usar- los componentes gráficos que se desean para el desarrollo. La oferta en JSF es

amplia (y sigue creciendo) y consume un cierto tiempo encontrar los componentes que mejor se adaptan a nuestras necesidades e investigar su funcionamiento e integración.

7. Implantación y formación

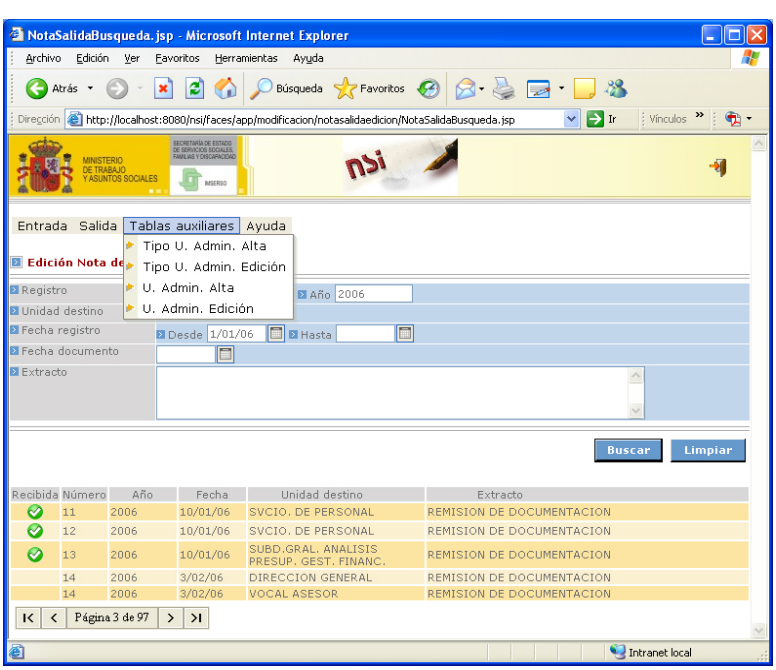
Se ha creado un pequeño equipo para definir la arquitectura, seleccionar los componentes y evaluar las distintas opciones disponibles en el mercado, realizar una aplicación con calidad de producción de prueba, identificar los problemas más recurrentes y preparar toda la infraestructura necesaria.

Durante el proceso de implantación se definieron los estilos CSS, las plantillas de los formularios, se crearon los componentes comunes a todas las aplicaciones (menú con soporte para roles, localización, carga dinámica de opciones, etc), el flujo de aplicación general y la integración con las diferentes librerías necesarias. También fue necesario crear el nuevo entorno de seguridad e implementar las librerías precisas. Se pasó de tener los perfiles de seguridad en base de datos a un modelo basado en LDAP, mucho más fácil de gestionar y mejor localizado.

Una vez finalizado y probado se pasó a definir la formación del equipo de desarrollo. Se identificaron dos áreas: metodológica (el nuevo entorno exigió un conocimiento mucho más profundo de programación orientada a objetos) y tecnológica (todos los frameworks empleados eran nuevos y era necesario formar a los desarrolladores en ellos).

En la formación metodológica se vio la necesidad de realizar cursos de UML, análisis y diseño. En relación a la formación tecnológica se aprovechó la ventaja de la separación limpia en capas para dar formación, de forma independiente, en cada una de ellas (JavaServer Faces, Hibernate, etc)

La creación de una aplicación completa y finalizada ha permitido disponer de referentes en todos los elementos necesarios para la construcción de sucesivas aplicaciones y como apoyo a la formación. Formularios de entrada de datos, búsqueda, mapeo objeto-relacional en Hibernate, integración de informes dentro de las aplicaciones mediante componentes de Crystal Reports, scripts de ANT para diversas tareas o XML's de configuración para Spring o Acegi están directamente disponibles como plantillas y como guías.



Ejemplo de una captura de pantalla de una aplicación desarrollada en JSF.

Todos los elementos que aparecen en ella son componentes gestionados desde el servidor.

8. Bibliografía y Enlaces

Título	Author
Hibernate In Action	Christian Bauer, Gavin King
Hibernate Reference Documentation	Equipo Hibernate
Core JavaServer Faces	David Geary
JavaServer Faces in Action	Kito D. Mann
JavaServer Faces	Hans Bergsten
JavaServer Faces Specification	Craig McClanahan, Ed Burns, Roger Kitain, editors
Spring in Action	Craig Walls, Ryan Breidenbach
Spring Framework Reference Documentation	Equipo Spring
Acegi Reference Documentation	Equipo Acegi
Core J2EE Patterns	Deepak Alur, John Crupi, Dan Malks
Building J2EE applications with the Rational Unified Process	Peter Eeles, Kelli Houston, Wojtek Kozaczynski
Patterns of enterprise application architecture	Martin Fowler
Refactoring: Improving the Design of Existing Code	Martin Fowler
UML and Patterns 3rd Edition	Craig Larman
UML 2 Toolkit	Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado
UML Distilled 2nd Edition	Martin Fowler
Ingeniería del Software	Benet Campderrich Falgueras

Web sites:

Páginas oficiales

Java ServerFaces	http://java.sun.com/j2ee/javaserverfaces/
Spring	http://www.springframework.org
Hibernate	http://www.hibernate.org
Crystal Reports	http://www.businessobjects.com
Acegi	http://acegisecurity.org/
MyFaces	http://myfaces.apache.org/
XDoclet2	http://xdoclet.codehaus.org/
Ant	http://ant.apache.org/
Log4Java	http://logging.apache.org/log4j/docs/
JUnit	http://www.junit.org/index.htm
Rational Software Architect	http://www-306.ibm.com/software/awdtools/architect/swarchitect/
UML	http://www.omg.org

Recursos

Java ServerFaces	http://www.jamesholmes.com/JavaServerFaces/
Spring	http://www.springhub.com/
Rational Software Architect	http://www-306.ibm.com/software/awdtools/architect/swarchitect/index.html

209

Comunidades

Java ServerFaces	http://forum.java.sun.com/forum.jspa?forumID=427
Spring	http://forum.springframework.org/
Hibernate	http://forum.hibernate.org/
Crystal Reports	http://support.businessobjects.com/forums/default.asp
Rational Software Architect	http://www-128.ibm.com/developerworks/forums/dw_rforums.jsp