Caso de uso: Empleo de tecnologías J2EE para el desarrollo de una plataforma

Rafael Pedraza Carmona
rpedraza@properly.es

Alberto Planas Domínguez
SEIRC/CESEAND
aplanas@bic.es

Antonio Navarro González
navarro@properly.es

Benjamín de la Fuente Ranea
benjamin@properly.es

Jose David Fernández Rodríguez
josedavid@properly.es

Abstract

En esta presentación se pretende explicar la experiencia que ha supuesto el desarrollo de una plataforma para la gestión de ofertas y demandas tecnológicas, desarrollada para dos organismos dependientes de la Consejería de Innovación, Ciencia y Empresa de la Junta de Andalucía. El objetivo principal de esta plataforma es servir de nexo de unión entre aquellas empresas, grupos de investigación u otros organismos que ofrecen algún producto con marcado carácter innovador y aquellas otras que pueden ser consumidores de estas tecnologías.

Las características más reseñables de este proyecto son el uso en exclusiva de tecnologías de código abierto, el empleo de las especificaciones J2EE para el módulo de servidor (basado en JBoss, Tomcat, Axis, Lucene,...), el innovador sistema de seguridad que facilita el acceso segmentado a la información, un workflow para el control de distintos grupos de trabajo y el desarrollo de una interface de cliente Java basado en la plataforma Eclipse 3.

La conjunción de todas estas tecnologías, con el lenguaje Java como hilo conductor, puede suponer un importante punto de referencia para otros desarrolladores que pretendan alcanzar un alto nivel tecnológico, basándose para ello en la potencia y seguridad que aportan los desarrollos de código abierto.

Keywords: J2EE, JBoss, JavaCC, Eclipse, Lucene, Hylafax, Software Libre, Seguridad.

1 Salto tecnológico con software libre

Properly Software cuenta en la actualidad cuenta con una plantilla de diez personas, de las que la mitad pertenecen al Departamento de Desarrollo. Los primeros productos se realizaron exclusivamente con tecnología de Microsoft (Visual Basic for Applications); si bien al principio nos beneficiamos de la facilidad y simplicidad de esta tecnología, pronto nos vimos en un callejón sin salida: acorralados por unas necesidades y especificaciones crecientes en complejidad y al mismo tiempo limitados por unas herramientas que no nos proporcionaban soluciones cuando pretendíamos obtener algún resultado más allá de los casos de uso más simples. Problemas de escalabilidad y la obsolescencia de la tecnología empleada no nos deja otra alternativa que pegar el salto tecnológico.

A la hora de tomar la decisión de hacia donde mover nuestro marco de desarrollo los factores determinantes para elegir J2EE y software libre, fueron razones económicas (soluciones de Oracle, IBM y BEA son excesivamente caras en un principio) y tecnológicas (posibilidad de acceder al código y modificarlo, creciente madurez de la tecnología, masa crítica de desarrolladores, Java como lenguaje común para aplicaciones de servidor y de cliente, presencia de J2EE en la industria frente a otras soluciones).

Esta apuesta de futuro tenía que como consecuencia el destinar recursos en tiempo y económicos a investigación, partida para la que hasta ese momento no había existido presupuesto.

2 Nuestro primer cliente: descripción del proyecto

Gracias a nuestro trabajo de investigación durante más de un año, tenemos la opción de presentar una oferta al Instituto de Fomento de Andalucía para el desarrollo de una aplicación para la gestión de entidades y transferencia de tecnología, así como herramientas que

permitan obtener conclusiones estadísticas de los datos cruzados.

Este proyecto consta de cuatro apartados principales:

- **Gestión de entidades**. Consideramos entidades las empresas, los centros tecnológicos, grupos de investigación y organismos. De cada entidad se guarda información sobre su localización, una descripción codificada de su actividad, personas de contacto, proyectos en los que participan y derechos de propiedad.
- **Gestión de documentos de información tecnológica**. Aquí se almacenan y clasifican documentos en formato XML validados por sus respectivos esquemas XML y se transforman a HTML mediante trasnformadores XLST.
- **Herramienta de distribución de información**. El objetivo de ésta es la entrega selectiva de aquellos documentos tecnológicos a aquellas entidades que lo requieran o que puedan serles de interés. Los medios de envío son el correo electrónico y el fax.
- **Gestión de expedientes de patentes y marcas**. Este apartado está relacionado con las entidades y almacena la información relativa a las diversas actuaciones que se llevan a cabo con las mismas en relación al patentado o registro de tecnologías.

Del análisis de requerimientos de la plataforma a desarrollar determinamos las siguientes necesidades principales:

- **Seguridad**: Además de los servicios de seguridad proporcionados por el sistema operativo (firewall, cifrado de comunicaciones por SSL) y de la base de datos, nuestra plataforma aporta la posibilidad de definir políticas de particionamiento basadas en los datos (seguridad semántica).
- **Variedad de clientes**: Serán consumidores de los servicios de la plataforma clientes web y clientes ricos conectados por Internet con múltiples escenarios de velocidad de conexión, proxies, firewalls y routers.
- **Extensibilidad**: Se prevé la incorporación de nueva funcionalidad al sistema conforme se produzca la implantación de los servicios existentes.
- **Escalabilidad**: Se plantea la necesidad de atender solicitudes de clientes en un número creciente y al mismo tiempo se espera un incremento considerable de la información manejada.

3 Tecnologías de servidor

La tecnología J2EE abarca desde JDBC hasta JSP y los EJBs. Toda esta cantidad de tecnologías está muy bien documentada en la literatura técnica. Nosotros hemos decidido usar EJBs, SOAP, JAAS, JNDI, JMX y Mbean. Usamos el contenedor de aplicaciones libre JBoss 3.2.x.

Hemos tratado de seguir el estandar a traves de sus BluePrints, pero en los escenarios donde JBoss o AXIS nos proporcionaba una alternativa más limpia o eficiente, no hemos dudado en hacer uso de ellas. Estos casos pueden resumirse en:

- **Consultas dinámicas.** El cliente debe poder lanzar consultas atendiendo a múltiples criterios de búsqueda. Los mecanismos que proporciona J2EE (findBy y select) no son lo suficientemente flexible. JBoss propone una alernativa: *DinamycQL*. Podemos construir la sentencia SQL que deseemos a partir de los datos suministrados por el cliente.
- **Modelo de Seguridad.** El modelo basado en roles no permite expresar restricciones basadas en datos. Poder impedir el acceso de un usuario a las entidades de una provincia determinada no es expresable con una política de roles (ni declarativa ni programada). Diseñamos un evaluador de expresiones (usando gramáticas JJTree en JavaCC) que son invocados a nivel de Beans por los *SecurityProxy* de JBoss. Estos autorizan (o no) al usuario a retirar este dato.
- **SOAP por HTTPS.** Modificando WSDL4Java y parametrizando los stubs generados por esta herramienta de AXIS, logramos que el protocolo XML-RCP SOAP viaje por un canal cifrado.
- **Extensión del modelo de paso de parámetros de configuración.** La especificación EJB facilita un medio de proporcionar valores de configuración publicados en el directorio JNDI a las beans, sin embargo este mecanismo requiere de un redeploy de la aplicación ante un cambio de estos valores. Para resolver esta problemática, además de otras asociadas al tipo de datos que podemos gestionar, hemos desarrollado un mecanismo mediante una MBean que lee los valores de configuración desde un fichero XML.

3.1 Seguridad

3.1.1 Requerimientos

Un requisito de la aplicación que estamos desarrollando es la de disponer de un mecanismo que permita controlar las acciones que un usuario puede realizar sobre los datos del sistema. Necesitamos restringir el acceso de un usuario a un conjunto de campos de un bean de entidad y necesitamos limitar el conjunto de beans de entidad accesibles por ese usuario.

La primera restricción acotará las acciones que se pueden realizar en un campo determinado p.ej: el usuario U puede tener acceso de lectura a un campo pero no lo puede modificar. Este tipo de limitaciones casan perfectamente con el concepto de seguridad descrita en la especificación J2EE [1]. En este modelo el usuario tiene o no tiene derecho de llamada sobre un método de un bean atendiendo a lo declarado en el fichero descriptor del deploy.

Por contra, limitar el conjunto de beans de entidad sobre los que un usuario debe tener conocimiento no puede ser expresado por medio de la seguridad declarativa de J2EE. El estándar no nos proporciona ninguna forma de indicar que un usuario concreto, al solicitar la lista de entidades de nuestro sistema mediante la llamada al método getEntityList() localizado en un SLSB, esta nos devuelva solo aquellas entidades que pertenezcan a una provincia determinada. Es decir, no tenemos la capacidad de expresar el dominio de datos de los usuarios.

En la Tabla 1 podemos ver un resumen de las acciones que debemos controlar y el receptor de dicha acción.

Tabla 1: Acciones básicas de un usuario

Acción	Receptor
Acceso (S)	Entidad
Lectura (R)	Dato miembro
Escritura (W)	Dato miembro
Consulta (X)	Dato miembro

Las operaciones R,W y X se realizan sobre campos de una entidad. Podemos leer el contenido de un campo, modificarlo o lanzar una consulta con este campo como criterio de búsqueda. La operación S determina qué entidades pertenecen al dominio de datos de un usuario.

Podemos transformar estas acciones a predicados de la forma expuesta en la Tabla 2. De esta manera un usuario U puede leer el contenido de un campo F de una entidad E si S(U,E) ^ R(U, F).

El predicado S, para poder ser evaluado sobre E necesita de un conjunto de cláusulas o expresiones de restricción de U sobre E. Es decir, tenemos que indicar el conjunto de condiciones que una vez evaluadas nos digan si E pertenece al dominio de este usuario.

Tabla 2: Predicados

Acción	Descripción
S(U, E)	Entidad E en dominio
R(U, F)	Permiso de lectura de F
W(U, F)	Permiso de modificación de F
X(U, F)	Consultar por campo F

Repasados los requerimientos de nuestro sistema de seguridad vemos que necesitamos de:

1. Una gramática para expresar restricciones de dominio.
2. Un evaluador de restricciones.
3. Una descripción de los metadatos donde podemos poner restricciones semánticas y permisos.
4. Un lugar estratégico en la arquitectura de nuestro servidor donde determinar los permisos de usuario y rechazar o aceptar la solicitudes del mismo.
5. Un modelo de datos donde alojar metadatos, restricciones y usuarios.

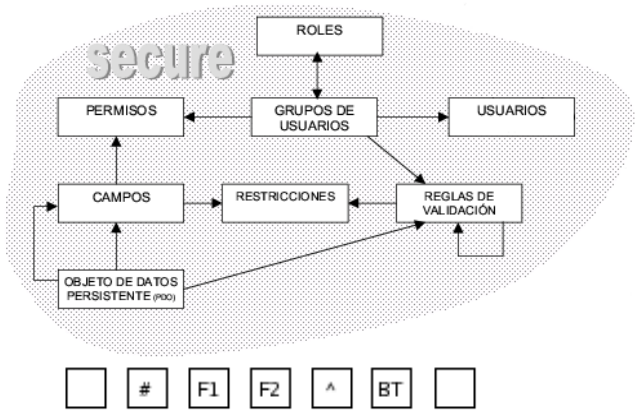

Figura 1: Compilación de las restricciones.

3.1.2 Restricciones y Gramáticas

Para poder expresar las restricciones de dominio de manera adecuada hemos diseñado una gramática sencilla que sea fácil de evaluar y de convertir a clausulas WHERE en EJBQL por razones de optimización de las consultas.

Hemos usado la herramienta JJTree de JavaCC [2] para generar árboles AST evaluables mediante un recorrido en postorden del mismo. Realmente el procedimiento de evaluación se ha optimizado traduciendo el árbol AST a una lista evaluable por medio de una pila (Fig. 1).

La gramática propuesta permite expresar restricciones del tipo:

#contato.apellidos LIKE "Delgado%"

#facturacion > #empleados * 1000

Tenemos variables que vienen prefijadas por el símbolo #, operadores, constantes, expresiones regulares, fechas, booleanos, números y listas. Es decir, es lo suficientemente completa para expresar un conjunto importante de restricciones.

Para simplificar el uso por parte del administrador encargado de definir estas restricciones de manera dinámica, hemos incorporado la variable sin nombre (#) para indicar 'el campo actual'. De esta manera la expresión ***(# > 10) AND (# < 100)*** tiene un significado diferente si se aplica al campo 'número de empleados' o 'edad'.

Con JJTree definimos los tokens que deberá encontrar el analizador lexicográfico, la gramática que el parser descendiente recursivo de JavaCC reconocerá, y las reglas de creación del árbol AST. Especificar una gramática en JavaCC es sencillo siempre que mantengamos en mente algunas reglas sencillas como la de ir definiendo las reglas de producción en orden inverso a la precedencia de los operadores. Es decir, debemos indicar primero las reglas de producción que tienen operadores de más baja precedencia.

Recorrido el árbol en postorden (subárbol izquierdo - subárbol derecho – raíz), y puesto los nodos del árbol de manera lineal, la tarea del evaluador queda simplificada.

3.1.3 Modelo de datos

Disponemos ya de una herramienta para evaluar restricciones. Para realizar su tarea el evaluador necesita de los valores de las variables de su expresión. Estos valores deben encontrarse en los beans de entidad sobre los cuales estamos imponiendo restricciones. Estas a su vez dependen del usuario que realiza la acción.

Vemos pues que necesitamos formalizar y almacenar las relaciones y los datos (y metadatos) del modelo de seguridad a través de un modelo de datos. Usaremos diagrama Entidad Relación de la Figura 2 para guiarnos.

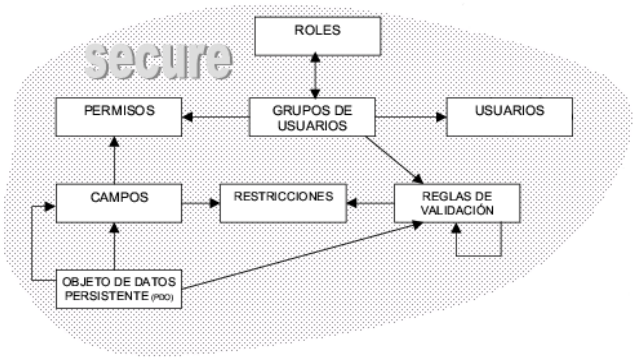

Figura 2: Diagrama ER de la seguridad.

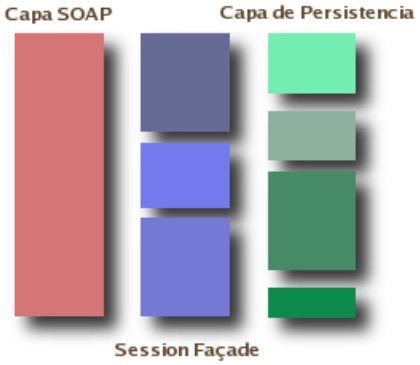

Figura 3: Arquitectura general del sistema.

Hay restricciones sobre una entidad y sobre campos de la entidad. Necesitamos recopilar en el sistema toda la información que tengamos sobre los objetos susceptibles de ser controlados. Hemos denominado a dichos objetos del sistema, PDO (Persistent Data Object, Objetos de Datos Persistentes). Cada PDO tiene un conjunto de campos. Así por ejemplo, el PDO de un Contacto tiene los campos Nombre, Dirección y Teléfono sobre los que podremos poner restricciones y permisos.

Estos metadatos pueden generarse automáticamente usando herramientas como XDoclet [3] o bien introducidos en el sistema de manera manual. Por desgracia generan una evidente redundancia, puesto

que los datos securizables son los mismos que tenemos en los Beans de Entidad (CMP y BMP), que a su vez tienen una contrapartida en el modelo de datos global de la aplicación.

Necesitamos almacenar los datos de los usuarios (login, nombre, contraseña...) para poder activar el mecanismo de autentificación y autorización de J2EE. Cada usuario pertenece a un grupo de usuarios y este a su vez dispone de varios roles dentro del sistema.

Almacenamos los datos correspondiente a los permisos de lectura, escritura y consulta de los campos y las restricciones que determinan el dominio de datos de un usuario.

Los permisos se relacionan con un grupo de usuarios y los campos de los PDOs. Por tanto para cada grupo/campo indicamos los permisos R,W,X.

Las restricciones de dominio asocian el grupo de usuario con el conjunto de restricciones sobre un PDO en concreto. Ahora tenemos una estructura donde recuperar la lista de restricciones que tenemos que evaluar para saber si un usuario del sistema tiene o no tiene permiso de acceder a un Bean de Entidad en concreto.

3.1.4 JBoss y los SecurityProxy

Llegados a este punto disponemos de un evaluador de restricciones y de un sistema que nos devuelve la lista de restricciones a evaluar cuando un usuario quiere acceder a una entidad. La cuestión ahora es en qué lugar de la arquitectura de nuestro sistema sería más conveniente realizar estas evaluaciones para aceptar o denegar la acción.

La Figura 3 es una sobresimplificación de la arquitectura del sistema. Vemos tres oportunidades donde colocar nuestro mecanismo de seguridad.

Si lo colocamos en la capa más externa de todas, la capa SOAP, por cada petición de un usuario, el sistema de seguridad deberá atravesar toda la arquitectura para recuperar los permisos, las restricciones y los datos necesarios para evaluar dichas restricciones. Es evidente que es poco óptimo y complejo. Además estamos incorporando lógica en la capa de comunicaciones.

Incorporar la seguridad semántica dentro de la Session Façade es factible. Estamos completamente dentro del servidor, dentro del segmento de nuestra aplicación donde recaen las decisiones de negocio. El problema está en que la seguridad quedaría difuminada y repartida en cada uno de los métodos de esta capa. La dificultad en el mantenimiento e implementación sería desaconsejable. Si bien tenemos a nuestra disposición patrones de diseño como los Decorators y los Proxy que pueden ayudarnos, quizás hay una alternativa mejor.

Si lográramos poner la seguridad en la capa de persistencia, donde encontramos lo BMPs y CMPs de nuestro sistema, la seguridad sería óptima. Ningún acceso a los datos (excepto claro está, accediendo directamente a la base de datos) podría saltarse las comprobaciones de seguridad, y la situaríamos al mismo nivel que la seguridad nativa de J2EE, mano a mano y complementándose.

Para lograr este nivel de integración necesitaremos echar mano de los interceptores. Aquí AOP nos puede ayudar, pero decidimos, puesto que usamos JBoss 3.2.x, acceder a las facilidades en la arquitectura de interceptores que nos brinda este contenedor J2EE de código libre.

Los SecurityProxy [4] son unos interceptores a nivel de beans que permiten separar la seguridad del resto de los procedimientos. Este tipo de proxy es llamado antes de cada invocación a los métodos del bean, es por tanto una gran oportunidad para abortar esta llamada en caso de que no se cumplan las restricciones de seguridad de nuestro sistema. Hay dos tipos de SecurityProxy, uno que tiene dos métodos (uno que se ejecuta para cada llamada a los métodos Home (create, findBy y selects) y otro para los métodos del objeto (postCreate, set, get y remove)) y otro tiene el mismo interfaz que el CMP o BMP que estamos interceptando.

Este modelo tiene la venta de poder navegar por las relaciones de manera natural. Podemos establecer una restricción sobre un elemento relacionado con el que entramos tratando. Así puedo decir, por ejemplo, que un usuario del sistema no puede acceder a las empresas que tenga al menos un cliente de nombre Raul. Restricciones así demuestran la potencia del planteamiento del modelo, si bien presenta un problema: demasiadas intercepciones pueden penalizar el rendimiento.

3.1.5 Optimizando el modelo

Si disponemos de un conjunto de restricciones expresadas en un lenguaje similar al EJBQL, al menos en lo que respecta a la cláusula WHERE ¿por qué no construir una expresión que aplicada a cualquier

consulta del sistema nos traiga el subconjunto de datos accesibles por el usuario?

De esta manera limitamos el númer de excepciones asegurando siempre la corrección del modelo. Aun sin esta optimización el sistema impide desde su base el acceso a los datos no autorizados.

Esta optimización hay que entenderla como tal: es la única separación del modelo de seguridad, ya que hasta ahora todo el asunto estaba localizado en los interceptores.

4 Tecnologías auxiliares

Además de las tecnologías que nos proporciona J2EE, hemos necesitado usar otras herramientas para cumplir con las especificaciones de la plataforma desarrollada. Esta tarea la hemos llevado a cabo mediante el mecanismo de integración por Mbeans proporcionado por JBoss. Los casos en los que ha sido necesaria esta integración son:

4.1 Lucene

El modelo de datos de Lucene gira alrededor del concepto de documento: un conjunto estático de campos de texto (accesibles por nombre), que constituye la unidad de indexación. En la compilación del índice se le añaden documentos y en la búsqueda por palabras se obtienen los documentos que satisfacen las condiciones de las consultas.

Bajo esta funcionalidad de alto nivel hay un sofisticado sistema cuyo conocimiento resulta fundamental cuando se requiere (como en nuestro caso) modificar el sistema en cierto grado para adaptarlo a nuestras necesidades. En este sentido, ha resultado fundamental la posibilidad de consultar el código fuente para poder comprender Lucene de un modo más completo.

Para el sistema de notificación de modificaciones en los documentos indexados se dispone de un acceso a la capa de persistencia usando el patrón de diseño *Session façade*. Aprovechamos la característica de proxys de JBoss para interceptar las llamadas de actualización a los beans. Este enfoque sugiere una aproximación a la programación orientada a aspectos.

4.2 Hylafax

Hylafax es un servidor para el envío y recepción de faxes, de código libre y amplio uso. Acepta peticiones de transmisión vía sockets mediante un protocolo derivado de FTP. Para facilitar la comunicación de nuestra aplicación Java con Hylafax existen librerías también de código libre. Nuestra interacción con el servidor Hylafax se reduce a dos puntos muy concretos: el envío de faxes y la determinación de si han sido enviados correctamente.

Utilizamos HTML2PS para la transformación de nuestros documentos HTML (generados a partir de los documentos de información tecnológica mediante transformadores XSLT) en documentos PostScript.

Para la comprobación del estado de los envíos usamos una técnica de *polling* a intervalos regulares de tiempo.

5 Tecnologías de cliente

Inicialmente consideramos dos alternativas como framework para la creación de nuestra aplicación de escritorio. Una era *NetBeans Platform*, que permite la construcción de aplicaciones Java con componentes Swing. Descartamos esta solución debido a cuestiones de rendimiento, consumo de memoria y aspecto general.

La otra alternativa, y la elegida, era Eclipse. Se basa en el uso de componentes nativos del sistema operativo cuando estos están disponibles (SWT) y un modelo muy sencillo de datos para los componentes JFace.

5.1 Eclipse Rich Client Platform

Con RCP disponemos de todo un framework para la escritura de aplicaciones de escritorio. Con el término Rich Client Platform, la comunidad Eclipse se refiere al mínimo conjunto de plugins que son necesarios para construir una aplicación con un interfaz de usuario. Este conjunto mínimo de plugins se reduce a tan solo dos: org.eclipse.ui y org.eclipse.core runtime, sin embargo podemos usar el resto del API ofrecida por Eclipse.

5.2 SWT (Standard Widget Toolkit)

SWT es una librería para crear interfaces de usuario en Java. Se caracteriza por su integración con los componentes gráficos del sistema operativo sobre el que corre, lo que supone una apariencia unificada con el resto de programas de esa plataforma y una velocidad de ejecución nativa. Como contrapartida, se plantean

Referencias

[1] Sun Microsystems, Inc. J2EE Patfrom Specification. *http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf*

[2] Java Compiler Compiler. *https://javacc.dev.java.net/*

[3] XDoclet. *http://xdoclet.sourceforge.net*

[4] Customized EJB security in JBoss. JavaWorld 15/02/2002 *http://www.javaworld.com/javaworld/jw-02-2002/jw-0215-ejbsecurity.html*

Editado por Abraham Otero Quintana.

ISBN 84-689-0035-4