ServEngine

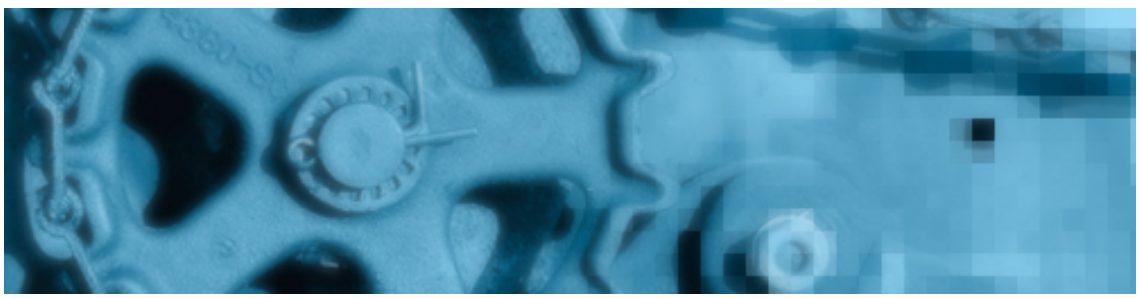
***Framework* para el desarrollo de servicios en línea**

José Ignacio de Córdoba Álvaro

Enginyeria en Informàtica

Josep Maria Camps Riba

16 de Junio de 2006



Dedico este proyecto a mi abuelo Luis

Prefacio

A lo largo de mis últimos años como desarrollador de portales corporativos, sistemas de comercio electrónico y comunidades virtuales he observado la demanda por parte del cliente de un conjunto de servicios muy similares a todos los proyectos, así como la necesidad de gestionar de forma simple y flexible los usuarios del sistema y sus limitaciones y permisos de acceso a dichos servicios.

A pesar del continuo desarrollo de proyectos de este tipo, no he encontrado un framework o librería que permita, tanto a programadores como integradores finales, reutilizar componentes con estructura J2EE [1] en sus aplicaciones. (JSR-168 crea los conocidos portlets [2], pero además de ser un enfoque muy poco extendido se solapa con el concepto de portal integrado y no ofrece servicios como autenticación de usuarios o sesión única. Además, por su alta complejidad, su uso a integradores web está vedado)

Es por ello que el desarrollo de este Proyecto Fin de Carrera presenta para mí una oportunidad de analizar posibles opciones existentes, crear una estructura para el desarrollo de aplicaciones web junto con varios módulos ejemplo que hagan uso de la arquitectura presentada. He agrupado experiencia sobre código fuente de anteriores proyectos comerciales el cual he reescrito como una extensión de Struts y siguiendo el estándar J2EE 1.4, siempre con vistas al reciente JEE 5.0

Haciendo un estudio de “mejores prácticas (*best practices*) en metodología MVC y en especial bajo framework struts, he elegido la EJBs 2.0 como capa de persistencia y JSP como capa de vista, extendiendo Struts para integrar una capa de control única a varios servicios característicos de comunidad virtual (correo electrónico, foros, comercio electrónico...)

Además de la librería final, que he decidido llamar **ServEngine**, presento una aplicación web ejemplo completamente funcional que la utiliza.

Es mi intención publicar la librería en código abierto para facilitar el desarrollo de nuevos componentes, además de los que he creado y adaptado a lo largo del desarrollo del PFC. Por ello he traducido la mayoría de los comentarios JavaDoc a inglés, si bien en el código fuente he dejado partes en castellano.

---

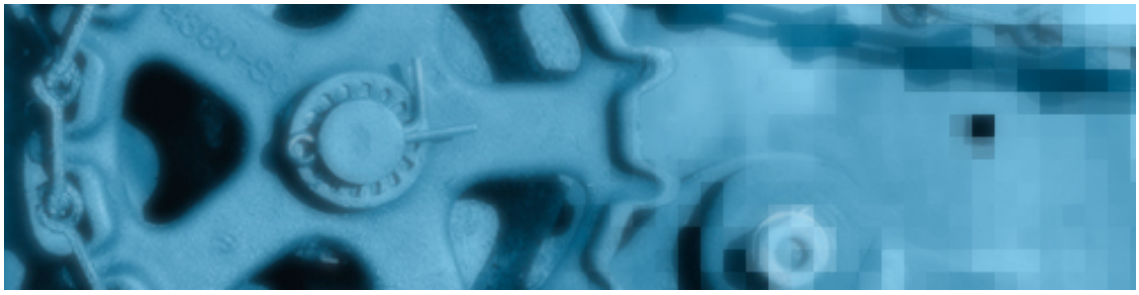
[1] http://java.sun.com/javaee

[2] http://www.jcp.org/en/jsr/detail?id=168

Índice

1. Introducción

La necesidad de un framework para desarrollo de portales



Podría estimar en el 80% la parte de código que, una vez especializado en el desarrollo de portales horizontales y verticales, comunidades virtuales y sistemas de comercio electrónico, suelo reutilizar para cada nuevo proyecto. Si bien puede parecer que dicha "reutilización" un enfoque válido, parece tener más sentido encapsular las partes comunes en módulos o componentes que ofrezcan el servicio comúnmente demandado de forma que ni siquiera sea necesario recompilarlo para su puesta en explotación.

1.1.- Justificación y contexto en el que se desarrolla el proyecto. Punto de partida y aportación del PFC.

Antes de plantearme el desarrollo y la integración de la librería objeto de este proyecto ya había seguido este enfoque mediante la agrupación de partes de lógica de negocio siguiendo el estándar J2EE. Inicialmente incluso desarrollé un sistema paralelo de beans de entidad y sesión siguiendo el modelo J2EE, puesto que en su momento y al menos en lo relativo a modelo de datos, la persistencia BMP ofrecida por las primeras versiones de J2EE no me parecía que ofreciera un avance importante (si cada bean de entidad se ocupa de gestionar su propia persistencia no hay lugar para optimizaciones o *cachés* elaborados por el *middleware* de turno). De hecho, mis primeras clases que representaban entidades disponían de un método de almacenamiento intermedio que reducían las llamadas JDBC a base de datos.

El modelo propuesto por J2EE dio un importante paso con la llegada de la persistencia gestionada por contenedor (CMP) y fue ahí cuando decidí a adoptar J2EE como estándar para el desarrollo de mis aplicaciones corporativas; tanto para proyectos propios como bajo demanda para clientes.

Aún así en ese momento no efectué una evaluación de patrones de desarrollo y, sin duda con mal criterio, seguí mi instinto a la hora de aplicar "mejores prácticas" en el desarrollo J2EE del momento. Esto llevo a acabar adoptando un estándar propio para la capa de presentación. La capa de control, que aparece en el patrón MVC, era todavía para mí algo etéreo que encajaba todavía de una forma muy difusa entre la presentación y el modelo. Quizá podemos hablar por ello de un híbrido poco elegante entre MVC modelo 1 y modelo 2.

1.2.- MVC ¿Realmente importante?

Tanto por aspectos relativos a "formación académica" como práctica", la necesidad de efectuar una separación entre capas a la hora de desarrollar aplicaciones complejas es sin duda imprescindible. Cuando se dispone de un equipo de desarrolladores gráficos con un conocimiento nulo o mínimo de conceptos de programación la unión de ambas capas sin duda acelera en un primer momento el desarrollo pero plantea problemas gravísimos a la hora de mantener o migrar aplicaciones. El ejemplo más claro de esto, en el entorno de aplicaciones corporativas desarrolladas en java, es la inserción de código java en páginas JSP. Sin duda el único que sale ganando con esta práctica es el desarrollador rápido de la empresa, que de repente se convierte en un gurú imprescindible en la organización puesto que toma un control absoluto de la capacidad de evolución del software desarrollada. Todo pasará por sus manos puesto que nadie será capaz de tomar control de la aplicación.

En el proceso de investigación sobre patrones de diseño he podido ver como algunos ingenieros pertenecientes a la empresa Pramati, desarrolladora del conocido contenedor J2EE homónimo no se encuentran nada cómodos con el patrón de desa-

Aún así y dando por supuesta la separación en capa de presentación y lógica de negocio ¿Qué necesidad existe de la capa de control? Es posible acceder a las EJBs de entidad mediante interfaces remotos o encapsular la lógica de negocio en servlets o javabeans clásicos que hagan la vez de clientes de acceso a los datos..

Cualquiera de estas dos aproximaciones es a todas luces más simple de desarrollar, pero tira por tierra el desacoplamiento entre capas y, sobre todo, no aprovecha las posibilidades de encapsulamiento de lógica de negocio en EJBs de sesión. Sin ir más lejos perdemos capacidad de distribución de carga (las EJBs de sesión por un lado pueden ponerse en explotación de forma distribuida en

distintos contenedores. El acoplamiento entre ellas mediante interfaces remotos o colas de mensajes nos da una tremenda flexibilidad en ese aspecto. Además, los contenedores J2EE congelan las EJBs que no reciben peticiones liberando recursos para otras EJBs que tengan más carga en sus procesos. De igual manera, tanto las EJBs de sesión con estado como las que no llevan información de sesión (*stateless*) son replicadas de forma automática por el contenedor para incluso dentro de la misma máquina virtual efectuar distribución de carga en distintos hilos.

La capa de control nos permite entonces enlazar de una forma estructurada las capas de modelo y vista, estableciendo una distribución de peticiones ordenada, tanto a nivel de código fuente como de ejecución.

Así pues, se analizará el modelo MVC como paso previo a la elección de un lenguaje de desarrollo y, dentro de este, un framework basado en dicho modelo y de código abierto y extensible.

1.2 Objetivos del PFC

En este proyecto se desarrollará **ServEngine**, un framework para el desarrollo de servicios de portales, comunidades virtuales y sistemas de comercio electrónico basado en el patrón MVC y, como se elegirá más adelante después de un proceso de análisis y comparación, utilizando Struts como base de desarrollo. Se extenderán las clases del framework que sean necesarias para ofrecer los servicios de seguridad, sesión y objetos comunes que faciliten la utilización de la librería como motor principal de gestión de usuarios y servicios, así como la posibilidad de agregar nuevos módulos de forma ligeramente acoplada y minimizando la generación de código nuevo mediante la reutilización de clases e interfaces comunes proporcionados por el “core” de la librería.

El proyecto incluye la publicación de la librería final como código abierto, examinando las distintas licencias existentes y poniendo a disposición de la comunidad de desarrolladores el código fuente y la documentación elaborada.

1.3 Enfoque y método seguido

Los pasos seguidos para el desarrollo del proyecto incluyen un análisis de requisitos, comparación de opciones de partida (tanto a nivel de patrón/metodología, como de lenguajes y frameworks concretos sobre los que basar el desarrollo), diseño del sistema, codificación, publicación del producto final en la comunidad de código abierto y desarrollo de una aplicación ejemplo que usa la librería. Los pasos que se seguirán son:

1.- Fase de análisis:

1.1.- Recopilación de requisitos

1.2.- ¿Qué hay actualmente?

1.3.- Proceso de selección de metodología.

1.4.- Proceso de elección de lenguaje / framework

2.- Diseño del sistema

2.1.- Elaboración de esquemas de clases, entidades y agrupación de lógica de negocio según el patrón elegido.

3.- Codificación del sistema

4.- Orientación en casos de uso de utilización de la librería

5.- Publicación del código abierto

6.- Recomendaciones sobre puesta en explotación de aplicaciones desarrolladas con ServEngine.

Nota importante: Inicialmente incluí un glosario tradicional al final de esta memoria, si bien lo he eliminado a causa de la práctica inexistencia de términos no conocidos por el desarrollador objeto de este documento. Todos los conceptos se encuentran explicados *in situ* y puesto que el documento se distribuirá como introducción a la librería de cara a la comunidad de desarrolladores, se presenta en PDF incluyendo los enlaces a webs externos.

1.4 Productos obtenidos

Este proyecto comprende:

- Memoria del proyecto

- Librería compilada lista para su puesta en explotación

- Clases de la capa de modelo: EJBs se sesión y entidad y archivos XML de configuración y puesta en explotación.

- Clases DTO (*data transfer objects*) asociadas a las entidades

- Clases de la capa Vista y Controlador, heredadas de las proporcionadas por Struts

- Aplicación ejemplo

- Archivo .ear:

    - Archivo .jar con las EJBs

    - Archivo .war con la aplicación web ejemplo

    - Documento XML de descripción persistencia para conexión contra base de datos MySQL para contenedor J2EE JBoss.

- Código fuente publicado bajo licencia LGPL.

Todo lo anterior se entrega junto con esta memoria y se pone en explotación en la red:

- Librería y documentación disponibles en sourceforge.net

- Aplicación web puesta en explotación en JBoss disponible en un servidor de aplicaciones on-line.

1.5 Desarrollo del proyecto

Los siguientes apartados abordan el desarrollo de este proyecto como se ha introducido anteriormente, con la siguiente estructura:

Capítulo 2: Análisis de requisitos para el desarrollo del framework

- Introducción a los servicios comunes de comunidad virtual
- Análisis de casos reales de desarrollos anteriores
- Enumeración de los servicios que deberá ofrecer el framework
- Comentarios

Capítulo 3: Análisis de librerías existentes para el desarrollo de portales

- El concepto portlets
- PHP Nuke, la opción de código abierto
- Opciones online
- Tuportal.com
- Opciones comerciales

Capítulo 4: Análisis y elección de metodología

Capítulo 5: Análisis y elección de framework base

Capítulo 6: Diseño del framework

- Modelo
- EJBs de sesión encapsulando lógica de negocio
- EJBs de entidad definiendo el modelo de datos
- Vista
- Extensión de struts
- Controlador
- Extensión de struts

Capítulo 7: Codificación

- Desarrollo en lenguaje Java (JSDK 5)
- Desarrollo de EJBs en J2EE 1.4
- Orientación de cara a la evolución con vistas a JEE 5

Capítulo 8: Orientación en casos de uso de utilización de la librería

- Desarrollo simple
- Desarrollo medio
- Desarrollo avanzado

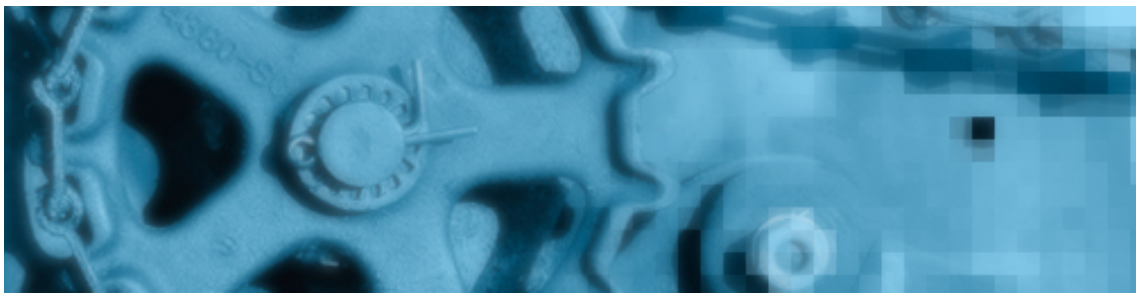
Capítulo 9: Publicación de código abierto

- Elección de licencia de código abierto
- Elección de hospedaje del proyecto

Capítulo 10: Puesta en explotación de aplicaciones desarrolladas con ServEngine

2. Análisis de requisitos

Necesidades que debe resolver ServEngine



Al desear desarrollar un framework o librería que debe ofrecer unos servicios comunes a futuras aplicaciones finales, no existe una lista cerrada de requisitos que el producto final deba tener. Es por ello que procedo a analizar una serie de casos prácticos de proyectos que he desarrollado a lo largo de mi reciente actividad profesional como desarrollador y consultor de aplicaciones web para recopilar los servicios comunes a todos ellos y tomarlos como base para la elaboración de los requisitos a implementar en el producto final objeto de este proyecto.

2.1 Casos reales de desarrollo de aplicaciones web con orientación de portal

2.1.1.- 123exitos.com. El portal de comercio electrónico de Operación Triunfo.

TickerMedia, empresa perteneciente grupo ValeMusic, había tenido dos experiencias negativas en la implementación de su tienda *online* para la venta de productos musicales y *merchandising*, tanto sobre Operación Triunfo como de otros artistas representados por la marca.

El principal problema radicaba en el altísimo nivel de carga (varios cientos de sesiones simultáneas) que debía soportar el sistema, muy por encima de las tradicionales soluciones basadas en Microsoft IIS o PHP, que complican la distribución de carga para una única aplicación. Se desarrollo un modulo J2EE de comercio electrónico con capacidad de soportar varios TPVs simultáneamente, así como requerimientos especiales de 123exitos, como el exportador de pedidos integrador con el software de mensajería de DHL.

Como frontal WEB se instaló un Apple XServe con doble procesador, que ejecutaba Apache con el módulo de conexión con el servidor de aplicaciones Tomcat funcionando en un servidor Linux. Como contenedor de aplicaciones se utilizó JBoss, funcionando también en servidores Linux. La base de datos elegida fue PostgreSQL, instalándose en un equipo independiente. Además de este componente, se incluyó el de noticias a la carta y gestor de contenidos, permitiendo al equipo de trabajo de TickerMedia el unificar toda la presencia WEB de su tienda en un único punto de gestión.

2.1.3.- Solomoda.com

Solomoda.com deseaba pasar de una web de exposición de contenidos a un portal de relación con los usuarios mediante la incorporación de un gran número de servicios dinámicos: correo web, boletines de noticias, foros, encuestas, tienda virtual, chat, diccionario, subastas... Personalización gráfica completa de cada servicio Provisión del servicio en modalidad ASP Implementación de Servengine ServEngine puso a disposición de SoloModa.com todos sus componentes, de forma que la estructura de SoloModa.com cambió completamente para pasar de una simple web de exposición de contenidos a un portal con toda una gama de servicios dinámicos destinados a fidelizar y multiplicar el número de usuarios. Todos los servicios (más de 20 en este caso) fueron personalizados individualmente por el webmaster de SoloModa.com en menos de dos meses.

SoloModa.com es un portal vertical de tamaño medio y destinado a un segmento de usuarios muy concreto por lo que la empresa no consideró necesario tener un servidor dedicado para su aplicación y escogió por ello la modalidad ASP (Application Service Provider) que usa los servidores de SKIOS.

2.1.4.- Cursos de verano de UCM

La organización de los Cursos de Verano de la Universidad Complutense de Madrid necesitaba un sistema integrado con capacidad para un gran número de transacciones que resolviera las siguientes áreas:

- Sistema dinámico de gestión de cursos, englobando las áreas de dichos cursos y los ponentes. El sistema debía permitir buscar de forma flexible por cualquier campo en los cursos e incluso mostrar individualmente los cursos en los que participaba cada ponente. La información debía estar en una base de datos estándar que permitiera su mantenimiento día a día.

- Sistema de matriculación para permitir a los interesados efectuar de forma completa el proceso de matriculación a través de Internet. Además, el sistema debía permitir la importación de los datos de matriculación en una base de datos interna de la organización.

- Gestión de noticias para posibilitar la inclusión de noticias con cualquier periodicidad en la página principal y otras áreas del portal, con un sistema de gestión de contenidos administrable por personal no técnico.

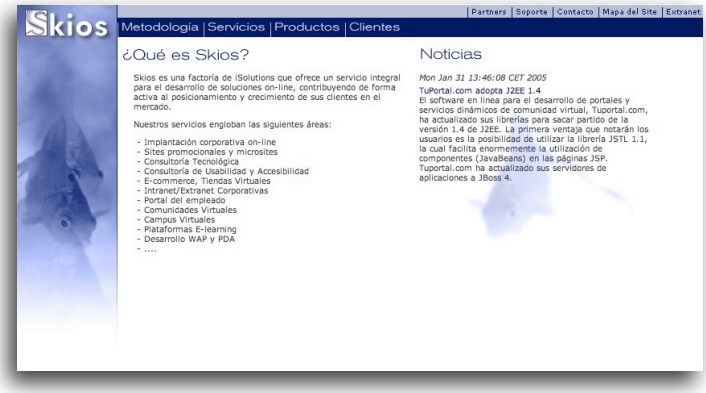
- Gestión de noticias, para posibilitar la inclusión de noticias con cualquier periodicidad en la página principal y otras áreas del portal, con un sistema de gestión de contenidos administrable por personal no técnico.

- Gestión de contenidos de prensa para medios: la organización deseaba disponer de un sistema sencillo de envío de información a los periodistas acreditados. La información era en múltiples formatos... Tanto gráficos como texto en formato .DOC y .PDF

- Gestión de boletín de novedades. Sistema de envío de un boletín periódico a un número grande de suscriptores interesados en las novedades de los cursos. Los usuarios debían poder darse de alta y baja desde el propio portal de forma automática para la organización de los cursos.

- El sistema debía estar operativo en un tiempo récord. Menos de dos meses desde la primera reunión de trabajo.

2.1.5.- Proveedor de servicios Internet SKIOS



SKIOS ISP ofrece sus servicios Internet desde principios de los noventa. Necesitaba ampliar los servicios tradicionales de hospedaje WEB y correo electrónico para dotar a sus clientes de mayor control sobre las cuentas, posibilidad de consulta multi plataforma (WEB/WAP/Voz) a la vez que una integración con su actual sistema de administración.

Se desarrollo un motor de servicios J2EE junto con componentes de Correo electrónico (WEB/WAP, ...), noticias a la carta y gestión de contenidos. Al utilizarse persistencia gestionada por contenedor (CMP), sin modificar ningún componente, las EJBs se asociaron a las tablas ya existentes en la base de datos de SKIOS (PostgreSQL). El componente de correo de ServEngine utiliza IMAP como repositorio de mensajes, por lo que se pudo integrar sin problemas con el software de SKIOS ISP (Cyrus IMAP) a la vez que con el sistema de autenticación de usuarios del mismo. El gestor de contenidos y noticias a la carta permiten publicar noticias sobre nuevos servicios de SKIOS, así como el gestor de boletines incluido en el motor de principal permite mantener a los usuarios informados en todo momento de los detalles de su cuenta. Gracias al gestor de preferencias de usuarios del motor, estos pueden de forma autónoma consultar las características de sus servicios y efectuar de forma ágil tareas como el cambio de contraseña de sus cuentas de acceso y correo.

2.1.7.- El Portal de la Publicidad

El portal de la Publicidad requería un desarrollo de comunidad virtual orientada a comercio electrónico de servicios de artes gráficas: Impresión de tarjetas, catálogos, trípticos.

Requería una plataforma de gestión de usuarios y productos unificada para 4 áreas diferenciadas, pero manteniendo un sistema único de autenticación, así como un repositorio único de productos ofrecidos. Como característica diferenciada resalta la necesidad de generación de documentos PDF dinámicos, por lo que era necesario integrar una librería de generación PDF en la lógica de negocio.

2.1.10.- Radio Total

Radio Total fusiona el concepto de radio total y la emisión de música en streaming. Ofrece 25 canales diferenciados por estilos de música y relacionados con las preferencias musicales del usuario. El sistema debe permitir el pago de suscripción a la modalidad de música de calidad así como el envío de correo según las preferencias de cada oyente.

Fue necesario desarrollar un módulo de gestión de la configuración del servidor streaming. Se utilizó para ello el Darwin Streaming Server de Apple, puesto en explotación en servidores Linux.

2.2.- Servicios comunes

De los anteriores proyectos ejemplo obtenemos una serie de servicios comunes a la mayoría de ellos, desde el punto de vista del desarrollo de componentes web:

- Servicio de control de acceso en modo usuario y administración a cada servicio.
- Gestión de usuarios: Campos adicionales y propiedades privadas.
- Autenticación de usuarios como una extensión de la autenticación estándar de struts para integrarse en el framework.
- Sistema de sesión única para todos los servicios.
- Control de entradas y accesos al sistema.
- Múltiples portales, tanto en la misma aplicación web como en aplicaciones web distintas, conectados a un único motor de lógica de negocio.
- Gestión de acceso a distintos servicios
- El sistema debe ofrecer un método de almacenamiento de propiedades de portales de cara a sistemas multiportal
- Para la aplicación multiportal, debe existir un método de categorías de portales para la elaboración de listados automáticos
- Notas y comentarios sobre portales
- Necesidad de unas clases abstractas para la extensión en el desarrollo de otros módulos
- Modelos para el desarrollo de capa de modelo (lógica de negocio encapsulada por completo en EJBs de sesión y modelo de datos en EJBs de entidad)
- Acciones Struts genéricas que marquen un patrón de desarrollo común.
- Control de acceso a las acciones de struts transparente y basado en los usuarios, roles y servicios gestionados en la capa de modelo.
- Capa de presentación extendida de Struts

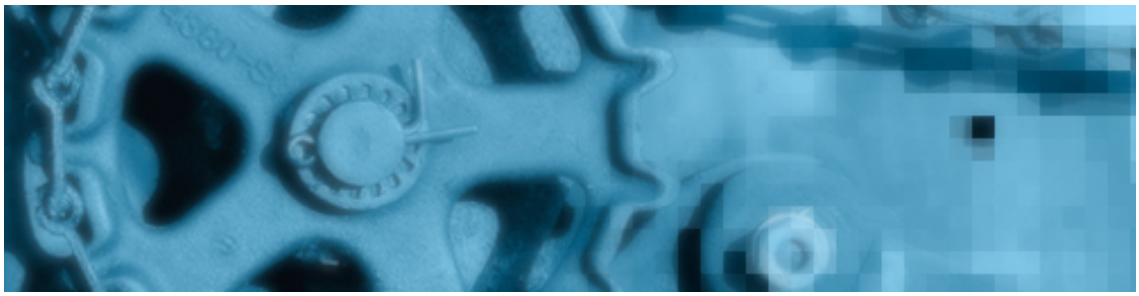
- Gestión de roles
- La estructura del sistema debe ser altamente modular
- Modelo de desarrollo extensible

Servicios adicionales:

- Gestión de salida de correo
- Gestión de archivos (FileManager)
- Control de limitación de acceso a servicios (para evitar uso abusivo en servicios como libro de visitas)

3. Análisis de opciones existentes

Productos existentes que unifican el concepto framework con librería de servicios



En el plano de librerías y entornos de desarrollo comerciales hay una gran cantidad de librerías disponibles. No entraré en detalle en estos productos, como Bea Weblogic Portal server, Oracle Portal Development kit o IBM Websphere portal puesto que se tratan de librerías que se distribuyen con unos costes por licencia que las sitúa fuera del ámbito de este proyecto. Aún así todas comparten una serie de características muy similares.

3.1.- Opciones comerciales

- Estos frameworks permiten integrar aplicaciones y fuentes de datos en los portales, efectuar tareas administrativas como la gestión de usuarios del portal.
- Servicios de presentación o vista, que facilitan el desarrollo del interfaz de usuario con un aspecto gráfico integrado y muy personalizable. Posibilidad de desarrollar interfaces para distintos dispositivos, como ordenadores portátiles o teléfonos móviles.
- Servicios de conectividad que permiten al portal acceder a datos corporativos, contenidos sindicados o fuentes externas de otras empresas..
- Gestión de personalización que permite a los usuarios del portal seleccionar qué aplicaciones y contenidos quieren ver del portal y como la información está organizada en sus páginas.
- Librería de Portlets incluidas para aplicaciones como correo electrónico, calendarios, sistemas de trabajo en grupo, noticias sindicadas. Los portlets conectan las aplicaciones y datos al portal.
- Interfaz API que permite crear nuevos portlets par añadir nuevas aplicaciones y fuentes de datos al portal. Todos incluyen un *toolkit*.
- Proceso de autenticación y sesión único a lo largo de aplicaciones distintas.
- Personalización por parte del usuario de forma individual.
- Comunidades de desarrolladores, generalmente orientadas al desarrollo de Portlets.

3.2.- PHP Nuke

PHP Nuke consiste en un gestor de contenidos que integra los instrumentos necesarios para crear un portal de información. A pesar de su orientación a la gestión de contenidos, PHP Nuke permite el desarrollo de portales de tipo intranet, sistemas de comercio electrónico, portales corporativos, servicios de noticias, sistemas de *e-learning*, ...

PHP Nuke tiene una gran base de usuarios que lo utilizan, pero la utilización de PHP como plataforma de desarrollo así como el no seguir un patrón de diseño no lo aconsejan para su utilización en aplicaciones corporativas de tamaño grande. Su escalabilidad es limitada en relación a las opciones comerciales anteriores o la opción que presentaría una librería de desarrollo de servicios de portales J2EE / MVC como la propuesta en este proyecto.

3.3.- Opciones *online*. Tuportal.com

Tuportal.com [3] es un servicio de creación de portales con orientación en línea. Esto significa que es posible crear un portal y dotarle de sus servicios sin necesidad de disponer de un servidor

[3] http://www.tuportal.com

propio o capacidad de desarrollo propio. Existen en la red una gran cantidad de servicios similares, cada uno con distinta orientación. Otro de ellos, similar a Tuportal.com es BraveNet [4].

En todos estos servicios no es posible la extracción del software para su instalación local en servidores propios, por lo que los desaconsejan para proyectos que vayan más allá de pequeñas aplicaciones.

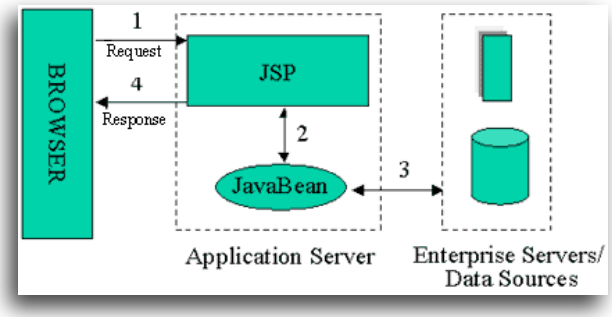
3.4.- Librerías con orientación de framework y de código abierto

A pesar de haber buscado alguna opción similar en concepto a la propuesta en este proyecto, no he podido encontrar un producto similar, de código abierto y que, siguiendo un patrón de programación y con una tecnología probada ofrezca un punto de partida para el desarrollo de servicios de comunidad virtual.

[4] http://www.bravenet.com

A controller defines application behavior. It dispatches user requests and selects views for presentation. It interprets user inputs and maps them into actions to be performed by the model. In a stand-alone GUI client, user inputs include button clicks and menu selections. In a Web application, they are HTTP GET and POST requests to the Web tier. A controller selects the next view to display based on the user interactions and the outcome of the model operations. An application typically has one controller for each set of related functionality. Some applications use a separate controller for each client type, because view interaction and selection often vary between client types.

4.- MVC Modelo 2 sobre Modelo 1



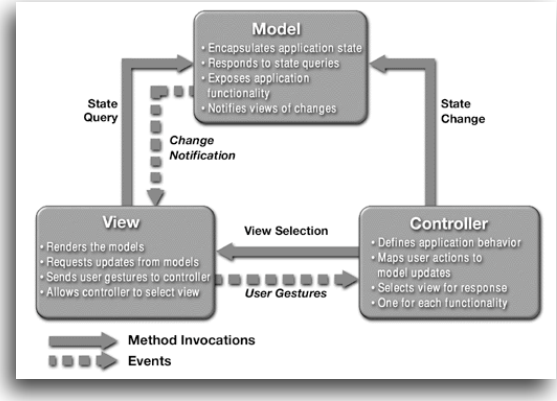
[5] En la arquitectura propuesta por el modelo 1, la página web dinámica (JSP en nuestro caso) es la responsable del procesado de la petición y la respuesta al cliente. Hay separación real de la presentación sobre el contenido porque todo el acceso a los datos se hace mediante beans. Aunque la arquitectura de modelo 1 puede ser válida para pequeñas aplicaciones, no interesa utilizarla en proyectos de complejidad media o alta. El uso de esta orientación generalmente lleva a una cantidad de scriptlets de código fuente “incrustados” en la página, sobre todo cuando hay una gran cantidad de procesamiento de las peticiones. El modelo 1 presenta problemas especialmente cuando diseñadores gráficos deben llevar a cabo el mantenimiento de las páginas web (que es lo común en proyectos grandes).

Como una aproximación intermedia entre MVC modelo 2 tenemos el encapsulado de la lógica de negocio relativa a los datos de petición y respuesta en javabeans cliente, que hacen la vez de *proxy-clients* al núcleo de negocio (contenedor J2EE). He aplicado este modelo durante años, si bien a raíz del desarrollo de este proyecto y la aplicación de la orientación de modelo 2 observo ahora como desaparecen problemas muy comunes como la no estandarización del proceso de inicialización de cada javabean cliente, su asociación a una página, petición o sesión o la gestión de incidencias o errores.

Encuentro en cualquier caso que este modelo mixto era más flexible de cara al número de páginas que se dispone para la presentación de resultados, puesto que una vez codificada la acción en modelo 2, esta maneja siempre un número fijo de posibles respuestas, cosa que podía gestionarse mediante etiquetas jsp en el modelo mismo.

[5] Fuente del gráfico: JavaWorld

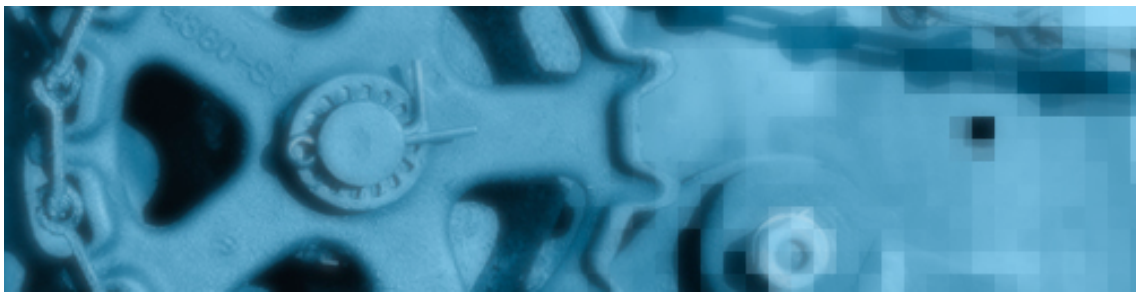
La arquitectura MVC modelo dos finalmente es una aproximación híbrida. Cuando se aplica en tecnología Java y, utilizando el *framework* Struts en concreto, combina los conceptos de servlets para el desarrollo del modelo de controlador y JSP para la capa de presentación (vista). Así fuerza la desaparición de lógica de negocio e incluso de los *proxy-clients* en la página web final (JSP) puesto que previamente el controlador lleva a cabo toda la lógica de cada petición, por contra al modelo mixto donde en primer lugar se instanciaban las javabeans cliente quedando por ello incrustados en la propia página JSP. [6]



[6] Fuente del gráfico: sun.com

5.- Opciones de implementación

MVC modelo 2



Ruby on Rails [7] es un framework de código abierto relativamente reciente en el mercado que está teniendo una impresionante aceptación. Parece probado que los tiempos de desarrollo para proyectos pequeños y medios son muy interiores a los necesitados si se sigue un framework MVC en Java, especialmente comparado con J2EE 1.x.

5.1.- Ruby on Rails vs. Struts

La elección de Ruby puede parece correcta, pero tiene una base instalada aún muy reducida en comparación con las opciones basadas en Java/J2EE. La ventaja de velocidad prometida sobre J2EE lleva a los desarrolladores de Rails a asegurar que se tarda 10 veces menos en hacer desarrollos Rails que Java.

En cualquier caso y después de instalar Ruby on Rails y seguir algunos de sus tutoriales, se llega a la conclusión de que solo su “aproximación filosófica” separa ambos frameworks. Rails prefiere el código explícito por encima de los ficheros de configuración. Eso no parece jugar en favor de Ruby para el desarrollo de componentes... Mientras de J2EE y, especialmente, gracias a la excitante evolución sufrida en el nuevo JEE 5 parece haberse liberado de la necesidad de los tradicionales archivos de configuración y múltiples interfaces asociados a cada componente. (Posteriormente se verá como esta nueva aproximación acelera los tiempos de desarrollo pero juega en contra de la reutilización de código compilado. Más bien la imposibilita y hace que se necesario volver a los descriptores XML y a esa figura etérea del “técnico en puesta en explotación”.

Por último, la utilización de un *framework* no dependiente de un auténtico lenguaje de programación como Java deja a Ruby on Rails en una posición claramente en inferioridad de cara aplicaciones complejas. En conversaciones de foros de desarrolladores con frecuencia se hace una pregunta a los “fans” de Rails: ¿Qué tal funciona Rails en los teléfonos móviles? Siguiendo esta lógica, Rails también cae ante la posibilidad de desarrollar una aplicación de escritorio o applet web independiente. El uso de Java nos permite especializarnos en conocimiento de un único lenguaje de programación para entornos tan distintos. Es por ello por lo que no contemplo la opción de desarrollar este proyecto basándome en Rails.

5.2.- Struts, el estándar de facto MVC y otros frameworks java con orientación MVC

El siguiente paso es el análisis y selección del framework base para el desarrollo del proyecto. De los frameworks de código abierto y desarrollados en java selección los siguientes para hacer el estudio: Struts, Sprint, WebWork, Tapestry y JSF. JSF no es en sí un framework completo, pero puede ser considerado como tal, especialmente después del espaldarazo que ha recibido al ser incluido en los paquetes de la nueva estructura JEE 5.

5.2.1.- Tiempo y base instalada de aplicaciones

Struts: Es el más veterano puesto que lleva disponible desde principios de 2001.

Spring, WobWork, Tapestry y JSF tienen un par de años escasos de vida, lo que les sitúa por un lado en inferioridad de condiciones en cuanto a su base de aplicaciones instalada y por otra, sin duda plantean enfoques más “modernos” que el “decano” Struts.

---

[7] http://www.rubyonrails.org

5.2.2.- Struts [8]

A favor: Estándar, mucha información y ejemplos, librería de *tags* muy completa

En contra: Las ActionForms se hacen pesadas de desarrollar., no ofrece un método para pruebas unitarias (*unit test*). StrutsTestCase solo resuelve la integración

5.2.3.- Spring [9]

A favor: Ciclo de vida de la petición y sesión más fácil de sobrecargar, mejor integración con otras tecnologías, más fácil de efectuar pruebas

En contra: Demasiados XML de configuración. Requiere reescribir los JSPs, demasiado flexible (no ofrece un “controller” padre del que heredar)

5.2.4.- WebWork [10]

A favor: Arquitectura sencilla fácil de extender, librería de tags muy personalizable basada en Velocity. Se ha fusionado con Struts conformando el nuevo Struts Action 2.0 .

En contra: Es reciente y tiene una base de desarrolladores todavía pequeña. Todavía poca documentación, validación de la parte cliente pobre. (Todos estos problemas se han resuelto en las últimas versiones y, en especial, ante la fusión con Struts)

5.2.5.- Tapestry [11]

A favor: Muy productivo, buena comunidad de desarrolladores

En contra: Documentación demasiado conceptual, curva de aprendizaje complicada. Poca frecuencia de actualizaciones.

5.2.6.- JSF [12]

A favor: El estándar JEE 5 lo incorpora, fácil de desarrollar

En contra: Demasiados *tags* propios: los desarrolladores gráficos no saben por donde coger las páginas integradas con JSF, es una tecnología todavía inmadura.

---

[8] http://struts.apache.org

[9] http://www.springframework.org

[10] http://www.opensymphony.com/webwork

[11] http://jakarta.apache.org/tapestry

[12] http://java.sun.com/javaee/javaserverfaces

Después de esta comparación y de examinar el código fuente y, sobre todo, la documentación sobre como extender estos frameworks, la decisión parece clara por Struts. Si bien es el más antiguo de los frameworks, la gran comunidad de desarrolladores y documentación disponible facilita mucho su uso para la creación de una librería de componentes. Hay que recordar que **no se esta estudiando qué framework utilizar para un desarrollo de una aplicación final**, sino para el desarrollo de una librería-framework.

5.3.- Portlets - El estándar JSR168

El desarrollo de una librería basada en el estándar JSR168 no cabe a causa de la orientación del concepto portlet. Comparándolo con Struts observamos claramente por qué:

Por una parte, JSR168 necesita un servidor de portales. No requiere sencillamente un servidor de aplicaciones web y, opcionalmente, contenedor EJB si se decide usar EJBs para la capa de modelo.

En cualquier caso, el uso de portlets parece solapado con Struts. De hecho el proyecto Apache ha desarrollado un puente entre ambas tecnologías. [13] IBM por su parte también resuelve la puesta en explotación de aplicaciones struts como portlets, si bien hay tres problemas importantes con la adopción de JSR168 (y probablemente unos cuantos menores):

- Los *frameworks* trabajan con objetos del API estándar de JSP (ServletContext, ServletRequest, ServletResponse), mientras que JSR-168 introduce clases nuevas: PortletContext, PortletRequest, and PortletResponse.
- El ciclo de vida de una petición a un portlet está dividido en dos partes, el procesado y el "renderizado" del mismo. Desde el punto de vista MVC esto no encaja correctamente (la primera parte del procesado de la petición debe ocurrir en el "*process*" mientras que la redirección a la página resultado debería encapsularse en la parte de "renderizado").
- Los desarrollos con Struts tienen control sobre la generación de las URLs para las acciones y las páginas. Struts se ejecuta en una "webapp". En un contenedor de portlets es el servidor de portales quien maneja las URLs, por lo que un portlet basado en Struts deberá interactuar, de nuevo, con el API de portlets para esto. De nuevo no trabajamos con clases de la librería estándar.

5.4.- Versión de Struts

Una vez elegido Struts como framework MVC es necesario elegir qué versión usaremos. Actualmente Struts ofrece tres distribuciones:

- Struts 1.2
- Struts (Action) 1.3
- Struts Shale

La versión 1.2 fue dividida en dos proyectos independientes; Struts Action y Struts Shale. El primero mantiene su orientación al modelo basado en "petición" (*request*) mientras que el segundo está enfocado a la adopción de JSF y Ajax integrado como modelo de presentación. El nivel de madurez de Action es sin duda superior al de Shale. El anuncio del equipo de desarrollo de Action sobre la fusión con WebWorks de cara a Struts Action 2.0 es un punto determinante.

Inicialmente comencé el desarrollo con Struts 1.2 puesto que se presenta como la "mejor" versión de struts (v1.2.9), si bien pocos meses después la versión 1.3 ha pasado por varias revisiones que junto con las siguientes características la convierten en la elegida (Struts Shale no dispone toda-

[13] http://portals.apache.org/bridges/multiproject/portals-bridges-struts/index.html

via de un nivel de madurez aceptable). Una ve más, recordar que Action 1.3 evolucionará mediante la fusión hacia Struts Action 2.0 (basado en el framework WebWorks).

Las principales novedades de Struts (Action) 1.3 son las siguientes:

- Composable Request Processor
- ActionDynaForm interfaces
- Arbitrary configuration properties
- Catalog and Command Elements
- Enhanced Global Exception Handlers
- Extends attribute for XML configurations
- "isCommitted" Exception Handling
- Postback Actions
- Wildcard ActionConfig properties

El cambio clave es el "**composable request processor**". El procesador de peticiones (requests) puede entenderse como el "*kernel*" del *framework* y, de hecho, ha sido extendido en este proyecto mediante la sobrecarga de uno de sus componentes. Los antiguos métodos del procesador de peticiones de la versión 1.2 son ahora "objetos comando" en una cadena de comandos flexible definida mediante un documento XML de configuración. En lugar de sobrecargar un objeto monolítico como anteriormente, ahora se sustituye la clase que representa un comando por la nueva implementación.

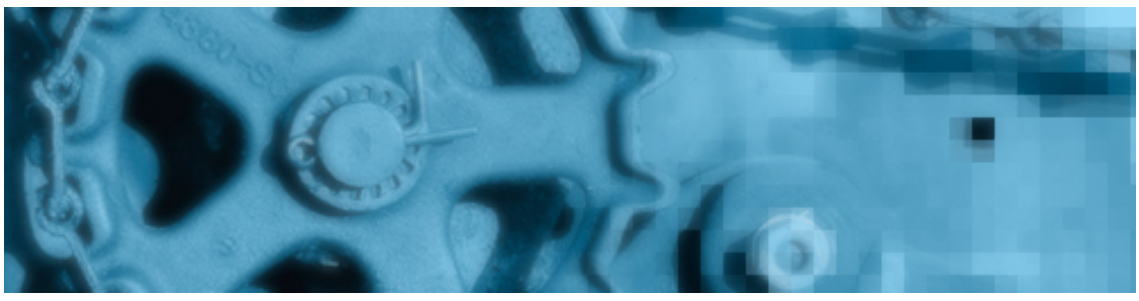
Gracias a esta arquitectura los comandos también pueden ser eliminados o añadidos si se necesita, para extender la cadena de procesamiento.

La elección de Struts Action presenta además algunas ventajas adicionales:

- Se integración mejor con las capas *View* y *Model* externos (y, por ejemplo, utilizaremos EJBs en la capa Modelo).
- Aunque no es el caso, es posible usar solo el *Controller* (utilizaremos también objetos de la capa Vista).

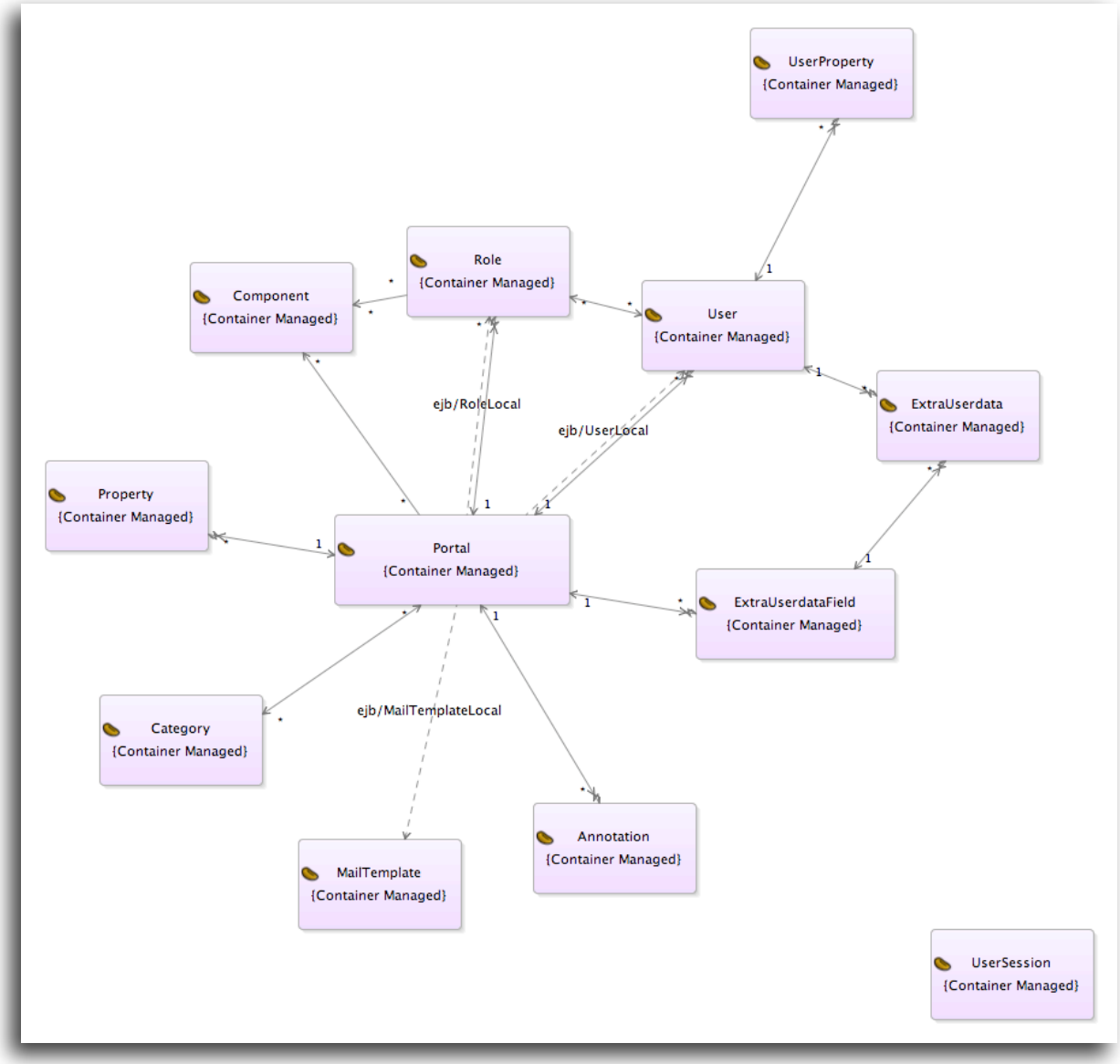
6.- Diseño del sistema

Arquitectura de la librería



6.1.- Estructura de datos

La estructura de entidades (representadas unívocamente en EJBs de entidad) que se utilizará como base en el framework es la siguiente:



Todas las EJBs de entidad disponen de su clase de transferencia (objeto DTO, *data transfer object*) que encapsula los datos permitiendo acceder así desde la capa de presentación desde y hacia la capa de lógica de negocio (EJBs de sesión). Las referencias desde estas EJBs de sesión a las entidades siempre se hace mediante interfaces locales. De esta manera y puesto que la capa de presentación se pone en explotación en una máquina virtual distinta (Tomcat) a la del contenedor de EJBs (JBoss) nos garantizamos que no hay posibilidad que incluso un desarrollador que no quiera seguir el modelo propuesto pueda caer en la tentación de acceder directamente mediante interfaces remotos al modelo de datos.

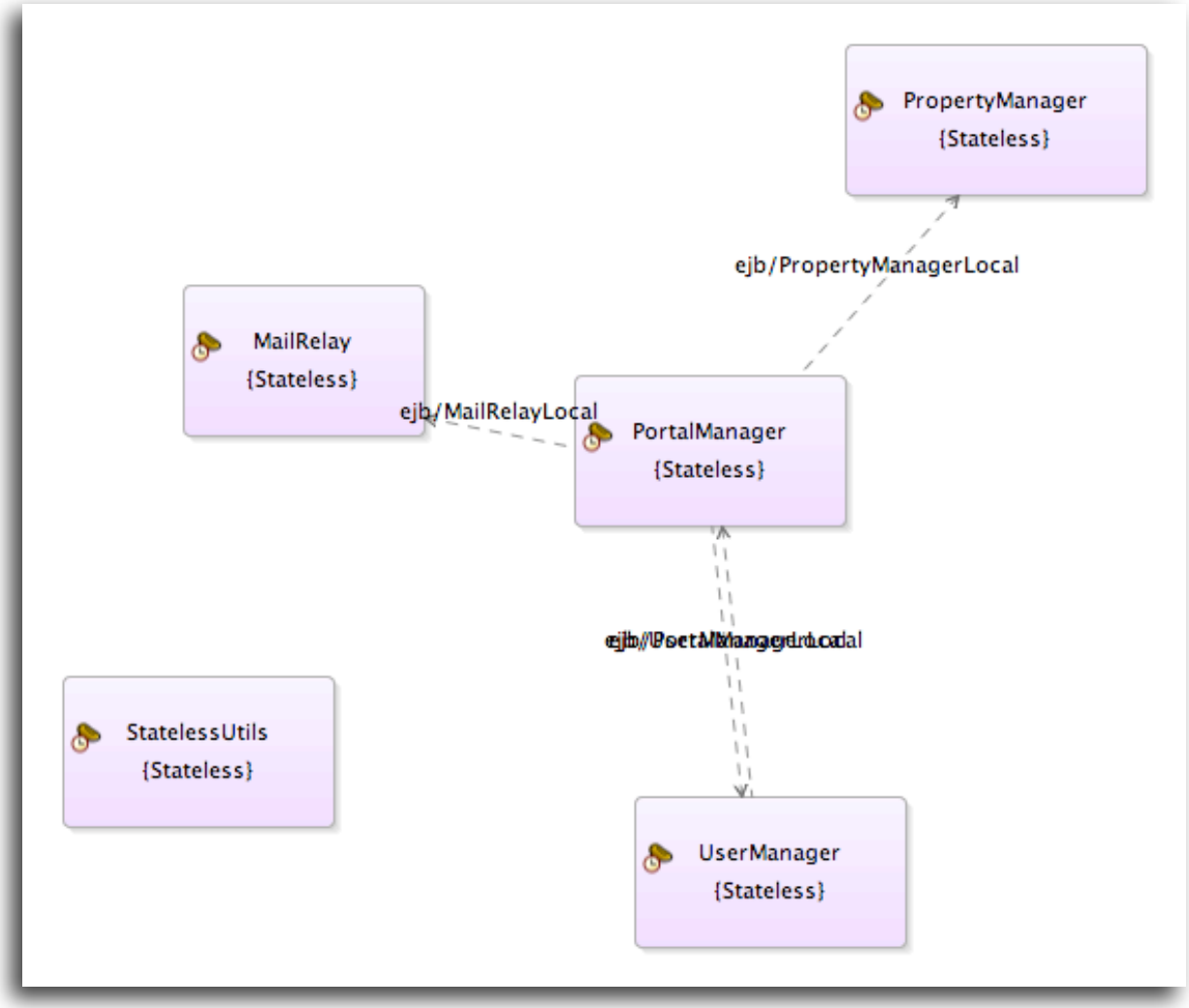
Cada EJB dispone de comentario JavaDoc en el código fuente de su clase principal pero a continuación describo cada una:

- - **Portal**: Entidad asociada a los portales que se ponen en explotación en cada instancia de nuestra aplicación. ServEngine permite trabajar en modo multi portal; de hecho siempre trabaja así incluso con un único portal.

- **User**: Representa un usuario de un portal. Un usuario solo puede serlo de un portal. Cuando un usuario se registra en el sistema, el objeto DTO equivalente (UserSBean.class) se vincula a la sesión. Los usuarios no autenticados se ejecutan como si fueran un usuario especial, invitado, que puede o no tener cada portal. Este usuario tiene el identificador “guest”.
- **Role**: Cada usuario puede tener asociados uno o varios roles. Cada componente (en principio, acción struts) puede quedar asociado a uno o varios roles. El sistema de permisos de Struts ha sido sobre cargado para sustituir la autenticación estándar por este modelo.
- **Component**: Representa una clase java. Puede ser un JavaBean que hace el papel de proxyclient contra las EJB de sesión que encapsulan la lógica de negocio o, en general, una acción Struts. (Clase que hereda de Action.class)
- **ExtraUserdataField**: Representa una campo adicional de datos de usuario. La EJB de usuario solo guarda identificador, contraseña cifrada, nombre, apellido, apellido2 y email. Cada portal puede configurar mediante esta entidad campos adicionales para sus usuarios.
- **ExtraUserdata**: El valor de un campo extra para un usuario.
- **Annotation**: Comentario privado sobre un portal. Los usuarios pueden verlo si el diseñador de la aplicación desea situar información de estas entidades en algún proceso del portal.
- **Property**: Propiedad de un portal. Esta entidad se ha reservado para su uso por parte de módulos desarrollados a partir de esta librería. Así es posible guardar datos bancarios del portal, información de colores configurables del interfaz, sufijo y otras características del servicio de correo electrónico web, etc.
- **UserProperty**: Es el equivalente a la entidad anterior, pero asociada a un usuario en lugar del portal. Esta información no es visible al usuario (a diferencia de los campos extra) siempre que el diseñador de la aplicación no decida lo contrario.
- **Category**: Si se desarrolla una aplicación multiportal con una gran cantidad de portales haciendo uso del servicio, mediante esta entidad es posible categorizar los portales y de esa manera, por ejemplo, ofrecer publicidad distinta dependiendo de la categoría de cada portal (Negocio, deportes, ocio, programación, ...)
- **MailTemplate**: El servicio de envío de mensajes puede tomar de esta entidad las plantillas para cada mensaje. Así, otros servicios (suscripción a boletines, forums, ...) disponen de un almacen único donde guardar las plantillas de texto y html para los mensajes mútiples que envíen de forma individual y en grupo a los usuarios.

6.2.- Estructura del modelo de negocio

La estructura del encapsulado del modelo de negocio en EJBs de sesión (siempre sin estado, *stateless*) es la siguiente[14]:



Usamos EJBs sin estado puesto que la información de la sesión se lleva a cabo en la capa de vista (*view*) mediante sesiones http. De esta manera solo se efectúa una instancia de cada EJB de sesión ahorrando memoria en el contenedor de EJBs.

- **PortalManager**: Encapsula toda la lógica de negocio relacionada con los portales dados de alta en el sistema.
- **UserManager**: Encapsula toda la lógica de negocio relacionada con los usuarios dados de alta en el sistema.
- **PropertyManager**: Encapsula la lógica de negocio relacionada con las propiedades del sistema de usuarios y portales. Podría haberse distribuido entre las dos primeras EJBs de sesión, pero opté por extraer los métodos indicados a esta EJB.
- **StatelessUtils**: Métodos de utilidad que no están en ninguna de las tres categorías anteriores
- **MailRelay**: Ofrece los distintos servicios de envío de correo.

[14] No se muestran todas las referencias locales entre *managers*

No se usan EJBs de mensajes puesto que complica la puesta en explotación de las aplicaciones (es necesario configurar el servicio JMS y las colas de mensajes, cosa dependiente de cada contenedor de EJBs). Además, me encuentro más cómodo con acoplamiento más fuerte entre EJBs de sesión que el que presenta la utilización de EJBs de mensaje.

Como he comentado anteriormente, las EJBs de entidad solo tienen interfaz local puesto que se accede a ellas localmente desde el correspondiente Manager (EJB de sesión). Dichos Managers tienen interfaz remoto puesto que se accede a ellos desde las acciones Struts mediante *lookups* remotos. Algunos presentan interfaz local siempre y cuando requieren invocaciones de métodos entre managers (por ejemplo, el manager de usuario requiere métodos del manager de portales y puesto que pertenecen al mismo paquete de EJBs y siempre se ejecutarán en la misma máquina virtual, es conveniente usar las referencias locales, unas 10 veces más rápidas que las equivalentes remotas.

Para pequeñas aplicaciones podría ser conveniente ejecutar la capa de presentación y la de negocio en una misma máquina virtual (como consigue por ejemplo JBoss con la inclusión de un Tomcat dentro del mismo contenedor de EJBs) pero eso obligaría a duplicar código que detectara si existe la EJB local antes de localizar la remota. He descartado esta posibilidad pero probablemente sea uno de los primeros asuntos a tratar en el foro SourceForge donde he publicado el código de esta librería.

Para ahorrar tiempo y entendiendo que las EJBs de cada paquete se sitúan siempre en el mismo contenedor, en la creación de cada Manager (EJB de sesión) se invocan las búsquedas (*lookups*) de los interfaces “home” de las entidades que se van a usar guardándose durante todo el ciclo de vida de la entidad de sesión. De esta manera siempre hay una referencia lista y se evita sobre carga de hacer un *lookup* para cada invocación de un método. En algunos contenedores de EJBs el comportamiento puede no ser el deseado y, por ejemplo, el contenedor “hiberna” una EJB que no se esté usando, cuando otra mantiene una referencia a ella. En las múltiples pruebas de explotación en JBoss no he detectado ningún problema por esto.

6.3.- Estructura de acoplamiento de objetos entre capas

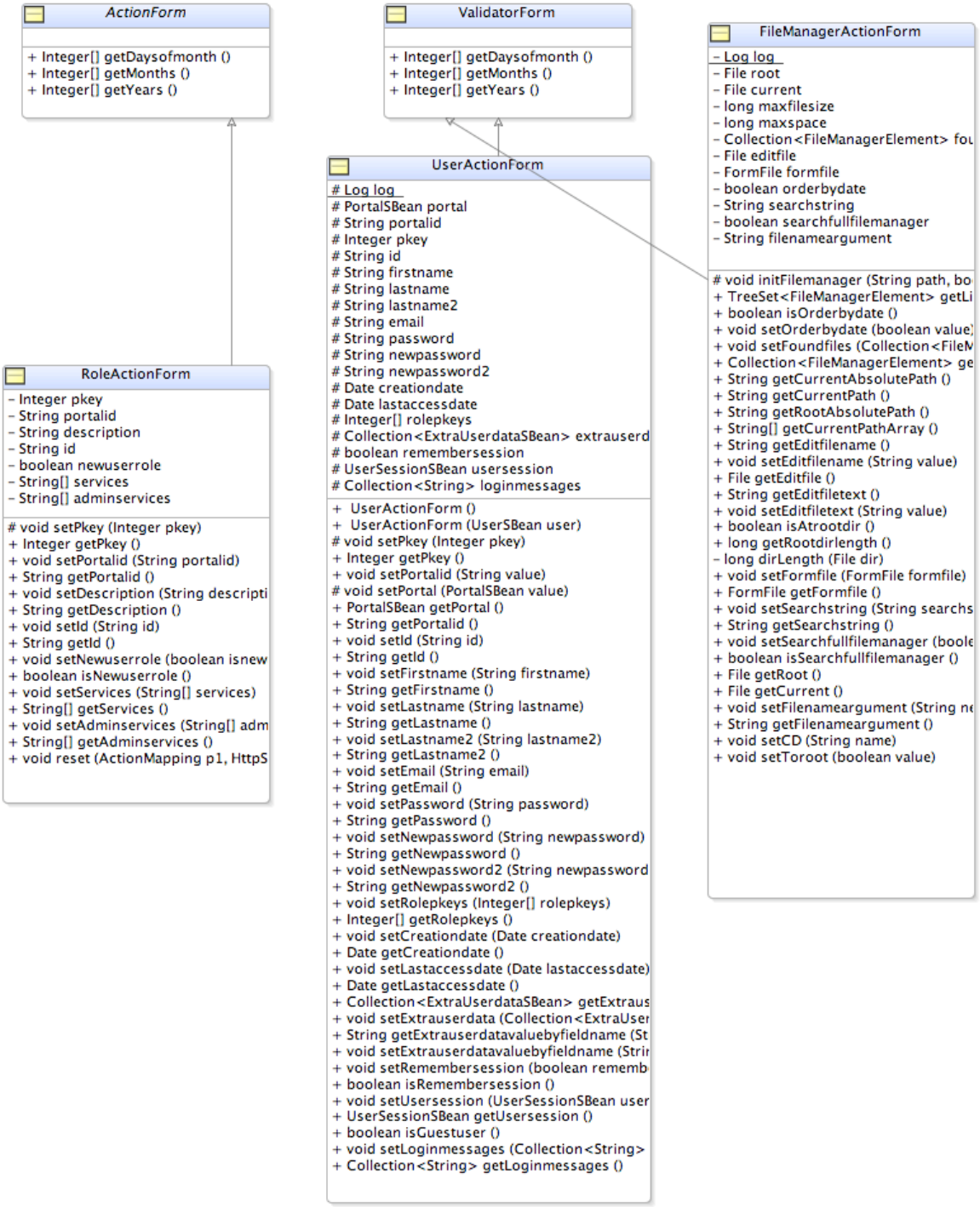
Propongo en ServEngine la siguiente estructura, tanto en la parte de la librería ofrecida, como de cara a futuros módulos desarrollados por terceros:

Página HTML Struts Action <-> Struts Action Form <-> Object DTO <-> Manager EJB <-> Entity EJB <-> Persistencia (MySQL) JBoss

Siguiendo el modelo MVC, las peticiones se ejecutan contra acciones Struts, las cuales procesan el ActionForm correspondiente al objeto (por ejemplo, UserActionForm). La acción struts correspondiente copia desde o hacia el correspondiente objeto DTO (las prácticas recomendadas de Struts indican que no se debe unificar el ActionForm con el objeto DTO. Al hacer un intento de unificación para evitar la duplicidad de clases me di cuenta, por ejemplo, que ello hace que el contenedor de EJBs, puesto que recibe los DTOs serializados, tenga que tener en su Classpath las clases de Struts. Por ello, he seguido la práctica recomendada con la estructura indicada.). El Manager (EJB de sesión) recibe así pues el objeto DTO y copia su información desde o hacia la EJB de entidad. [15]

[15] JEE5 supone un importante avance al facilitar la unificación DTO y EJB.

6.3.1.- ActionForms del framework



6.3.3.- Actions del framework

Como puede observarse, todas derivan de ClientAction (de ServEngine, derivado del homónimo de Struts) el cual además de algún método de conveniencia ofrece un interfaz remoto a los *managers* de Utilidades, Usuario y Portal. Si un ActionForm necesita acceso al servicio de correo, o a sus propios managers si hablamos de un módulo independiente, puede localizarlos (*lookup*) en el método initAction que debe ser implementado en toda Action de ServEngine (es un método abstracto de la clase padre ClientAction).

Siguiendo las prácticas recomendadas de Struts, agrupamos acciones en clases en lugar de desarrollar una clase por acción. El sistema usado para elegir qué acción se ejecuta es el estándar de struts (Dispatch Action) y se usará el parámetro “method” para indicar mediante introspección qué método se ejecuta.

6.4.- El *Controller*. Paquete com.servengine.struts

6.4.1.- Descripción de las extensiones sobre Struts 1.3 estándar:

- **ActionForm**: Hereda de la clase org.apache.struts.action.ActionForm incorporando algunos métodos útiles para la presentación de fechas en los formularios.

- **ValidatorForm**: Hereda de la clase org.apache.struts.validator.ValidatorForm incorporando algunos métodos útiles para la presentación de fechas en los formularios.

- **ClientAction**: Hereda de org.apache.struts.action.Action y es una de las clases más importantes de la librería. Toda clase Action debe heredar de esta, agrupando métodos que invoquen a la misma agrupación de logica de aplicación. La sobrecarga descrita a continuación efectúa la verificación de permisos para cada usuario segun sus roles.

- **AuthorizeAction**: Hereda de la clase org.apache.struts.chain.commands.servlet.AuthorizeAction y la sustituye como comando de autorización de ejecución de cada acción estándar de Struts. Efectúa la comprobación de roles conforme el método propuesto anteriormente y ejecuta el método o devuelve un error en caso de que el usuario no esté autorizado para ejecutar métodos de esa clase.

- **ExceptionHandler**: Hereda de la clase org.apache.struts.action.ExceptionHandler y debe configurarse en el documento XML para cada módulo Struts. Su funcionalidad actual consiste en capturar la excepcion UnidentifiedUserException, que puede ser lanzada en cualquier momento tanto en la capa de modelo como en la de vista, y en lugar de mostrar error, redirigir a la respuesta correspondiente (dependiendo de si se trata de un usuario administrador o no)

6.4.2.- Modelo de implementación de acciones:

Las acciones para el Controller están agrupadas en cuatro o cinco clases usando el método "*dispatch*" que proporciona Struts (se propone esta estructura para desarrollo de otros módulos) de la siguiente manera (X representa el nombre del módulo):

- Actions: Acciones genéricas no asociadas con un ActionForm (entidad en el modelo).
- AdminActions: Acciones de administración genéricas no asociadas con un ActionForm (entidad en el modelo).
- XActions: Acciones asociadas a un ActionForm (XActionForm).
- XAdminActions: Acciones de administración asociadas a un ActionForm (XActionForm).
- XUnidentifiedActions: Acciones genéricas que se ejecutarán aunque el usuario no esté autentificado

Puede examinarse el código fuente y la documentación JavDoc del paquete com.servengine.user para observar un ejemplo de esta propuesta. La distribución de acciones queda asi muy estructurada facilitando el mantenimento posterior.

6.4.3.- Seguridad

He decido no basarme en extendido modelo de delegación de permisos, User Delegation Model, si bien individualmente por componentes los usuarios con permiso de administración del modulo principal se convierten automáticamente en "delegadores" de dicho permiso. En cualquier caso hay una propuesta de implementación de este modelo que a pesar de resultarme un poco barroca puede ser interesante de examinar [16]

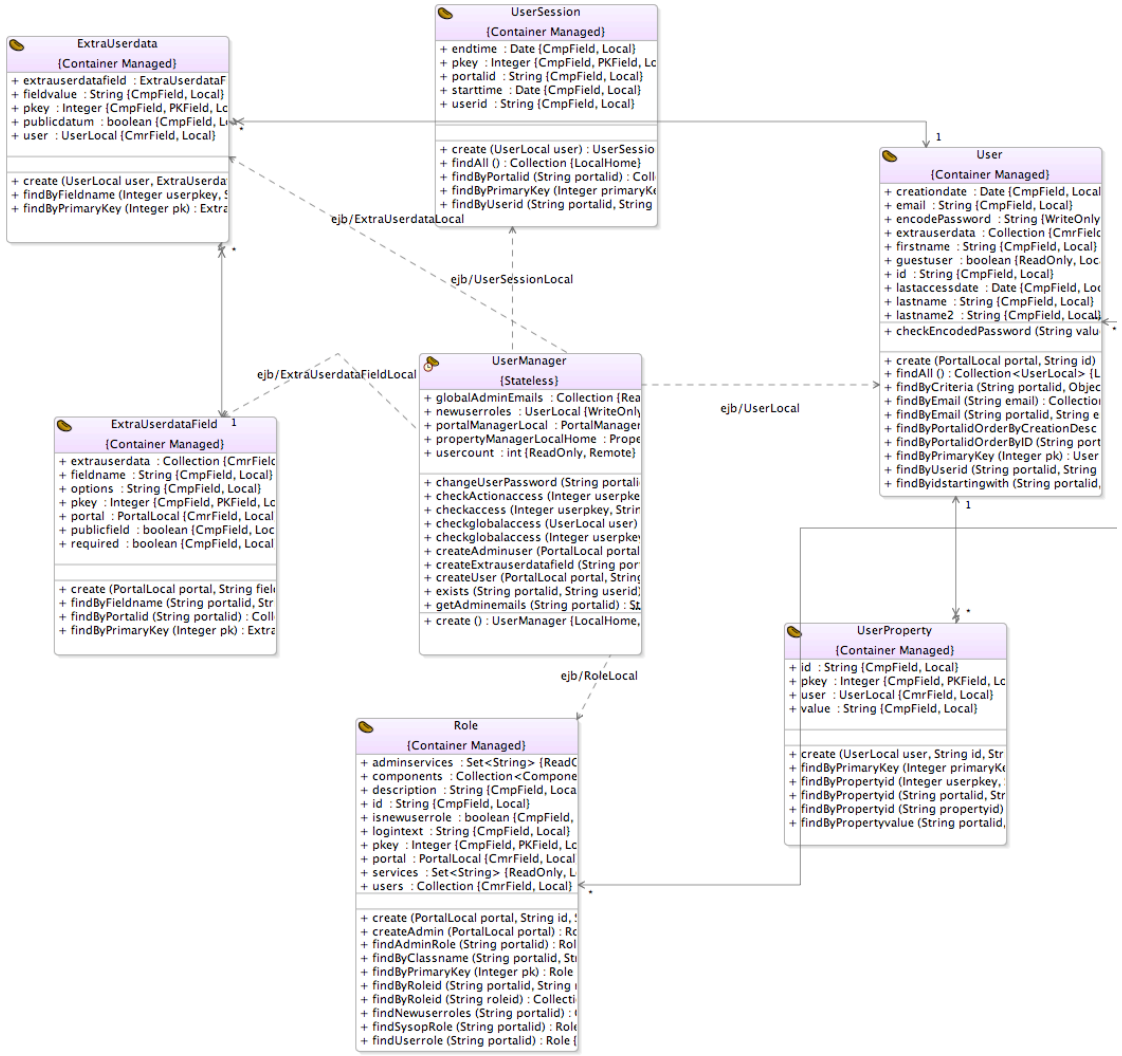
[16] http://www.onjava.com/pub/a/onjava/2004/02/18/strutssecurity.html

La utilización de un modelo basado en JAAS [17] [18] tampoco me ha parecido idónea, sobre todo por la tremenda complejidad de configuración (los archivos de configuración no son fácilmente substituibles por un modelo basado en persistencia). Además, no aporta más que complejidad adicional, cuando no necesitamos limitar a nivel de "método" el acceso a determinadas acciones. El resto de métodos de la capa modelo ya quedan aislados por la seguridad en la capa de *controller*, por lo que no necesitamos JAAS tampoco por ello.

6.5.- Paquete com.servengine.user

Aunque hay información ya presentada en los esquemas anteriores, los siguientes esquemas aclaran la relación entre EJBs de sesión, entidad y clases para la lógica de negocio y presentación de la entidad User.

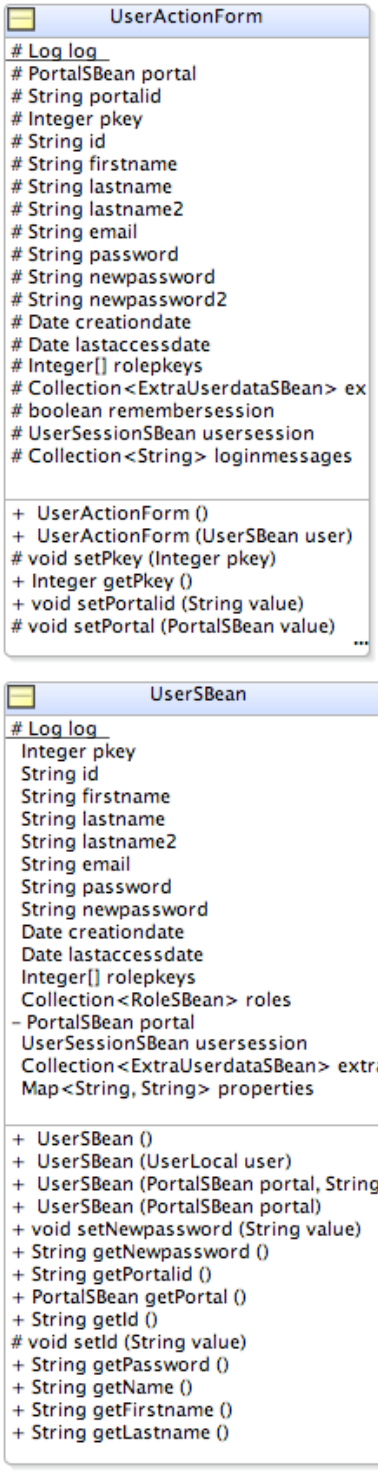
Esquema de EJBs



[17] http://www.mooreds.com/jaas.html#AEN162

[18] http://technology.amis.nl/blog/index.php?p=259

Esquema de clases



6.6.- La capa de presentación (View)

Originalmente analicé la opción de dotar al framework con tags para la generación fácil de código CSS y, especialmente, AJAX. En el proceso de análisis encontré varios productos de código abierto que resuelven este aspecto y pueden ser integrados directamente en una aplicación desarrollada con ServEngine. Por otra parte, Struts 2.0 (actualmente en fase alfa y en proceso de integración con el framework WebWorks) ofrecerá generación de código AJAX. Structs Shale (que no ha sido el usado para este desarrollo) también resuelve este aspecto con AJAX/JSF.

El equipo de Struts, sorprendentemente, recomienda la utilización de librerías del estándar JSTL en la medida de lo posible, en lugar de las propias de Struts, cuando se solapen funcionalidades.

6.7.- com.servengine.Cleaner

Cleaner resuelve dos problemas. La limpieza de datos y archivos ante la eliminación de un usuario o portal, y el calculo de espacio total también por usuario o portal.

Esta clase se sitúa en la capa de presentación, si bien puede parece que debería ejecutarse en la capa modelo. Inicialmente se diseñó así pero al estructurar extensiones de esta librería se observa la necesidad de formar parte de la anterior.

Cada vez que el sistema borra un usuario o portal, examina los paquetes instalados buscando clases que hereden de Cleaner mediante introspección. Hecho esto, las instancia y ejecuta el correspondiente método de limpieza. El por qué debe ejecutarse en la capa de presentación es sencillo. Algunos módulos no solo necesitan eliminar datos de portales y usuarios, sino también archivos. El respositorio de archivos se suele situar en la capa de presentación, incluso dentro de la aplicación web (webapp). De hecho la librería estándar sitúa ahí las fotografías de usuarios. La capa modelo no puede nunca tomar control de esta tarea.

Los módulos desarrollados para ServEngine deben también, mediante esta clase, informar al motor principal del espacio que cada portal y usuario esta utilizando. Por ejemplo, un módulo de correo web deberá devolver el espacio de los buzones; un módulo de hospedaje web deberá devolver el espacio de los objetos web guardados por cada usuario y/o portal. Es ya decisión del integrador de la librería si quiere hacer uso de esta información para establecer límites.

6.8.- Consideraciones adicionales

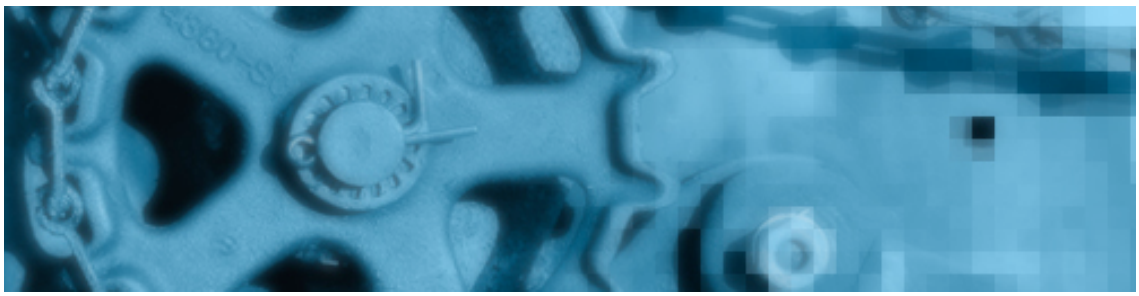
6.8.1.- La sesión de trabajo en ServEngine

Cuando un usuario entra por primera vez en un portal desarrollado con ServEngine, al ejecutar la primera acción, el sistema examina el portal. Si existe un usuario invitado (con identificador guest) y la clase no requiere ningún role para su ejecución, el comando de verificación de permisos permitirá su ejecución. Si el portal no dispone de usuario invitado o si la clase requiere algún role para su ejecución, se lanzará la excepción UnidentifiedUserException.

Esta excepción provocará la redirección a la página de identificación (y/o alta en el sistema si procede). Cuando un usuario se identifica o registra en el sistema, el objeto UserSBean quedará vinculado a su sesión, pudiendo utilizarse en cualquier página JSP para situar los datos del usuario actual.

7.- Codificación del sistema

Consideraciones sobre el código fuente



7.1.- Comentarios y Documentación JavaDoc

Puesto que decidí la publicación del código de la librería he traducido a inglés todos los comentarios que aparecen en la documentación JavaDoc generada. He dejado algunos comentarios en castellano si bien es mi intención comentar todo el código en inglés.

En algún caso puntual, para la elaboración de las clases actuales he tomado notas y código fuente de clases que no seguían los patrones MVC ni estaban integradas con ningún framework. He mantenido dichas clases añadiendo el comentario @deprecated, de forma que queda indicado su desaparición en la próxima versión de la librería. He preferido mantenerlas en la distribución por motivos históricos, y por si se diera el caso de necesitar consultar algún código fuente antiguo.

7.2.- Java 5

La librería hace un uso intensivo de algunas de las nuevas características de la revisión 5 del lenguaje Java. En especial las colecciones “tipadas” y el nuevo bucle for, no presente en las versiones anteriores de Java. Esto obliga al desarrollador o integrador a instalar una maquina virtual versión 5 en detrimento de la 1.4 y la ya vetusta 1.3 . Considero que la base de máquinas virtuales 1.4 será mínima en muy poco tiempo, sobre todo viendo la gran apuesta de Sun por el nuevo JEE 5. Incluso el servidor de aplicaciones web más difundido, Tomcat, requiere la instalación de un paquete especial de compatibilidad si quiere ejecutarse en Java 1.4 .

El uso de colecciones “tipadas” reduce consideráblemente la posibilidad de errores en tiempo de ejecución que eran muy comunes por errores de programación al usar colecciones y conjuntos genéricos. Además, el nuevo bucle for es extremadamente sencillo y claro de usar en detrimento de los antiguos iteradores.

7.3.- JEE 5

No hago uso de anotaciones más que para la generación de documentación JavaDoc. Tampoco he entrado en conceptos presentes en JEE 5, como las anotaciones para simplificar el desarrollo de Enterprise Java Beans . Personalmente y a riesgo de ir completamente en contra de la actual tendencia, me encuentro a favor del uso de descriptores XML para la configuración de la puesta en explotación de EJBs. El uso de anotaciones asocia la puesta en explotación de objetos con el código fuente de los mismos. Un integrador de un componente de librería no debe necesitar examinar el código fuente y, en general, modificarlo y compilar para modificar un parámetro de puesta en explotación.

El uso de anotaciones sin duda acerca JEE5 al mundo de los pequeños proyectos, actualmente dominado por PHP y Rails, pero en proyectos medios y grandes apuesto por la continuidad de los descriptores XML.

7.4.- Librerías utilizadas en el desarrollo

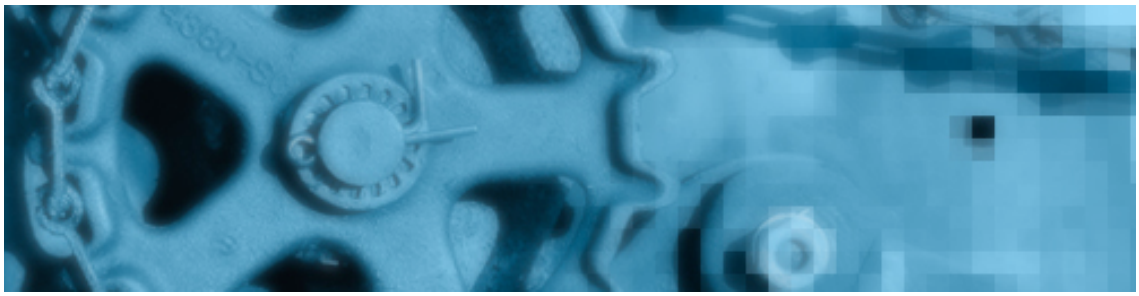
Además de las obviamente necesarias por las tecnologías usadas (J2EE, JSP, JSTL, Struts, Log4Java, ...) he utilizado la librería de envío de SMTP **Aspirin** [19]. Aspirin elimina la necesidad de contar con un *relay* de salida SMTP para el envío de correo. El sistema puede configurarse en cualquier caso para el uso de *relays* de salida.

Es necesario también añadir la librería Commons Utils, del proyecto Jakarta. La copia de datos entre ActionForms y objetos DTO se efectúa por introspección gracias a esta librería sin necesidad de copiar propiedad a propiedad.

[19] https://aspirin.dev.java.net/

8 Ejemplos de integración de la librería

Casos prácticos para proyectos de distinta envergadura



El nivel de complejidad en la implementación y desarrollo de un proyecto con ServEngine es extremadamente flexible, pudiéndose enfocar desde varias aproximaciones

8.1.- Integración sencilla

Incorporación de servicios dinámicos estándar sin necesidad de funcionalidades especiales. El sistema utiliza el control de usuarios y permisos de ServEngine, bien porque no tiene su propio sistema o por que su desarrollo parte de cero y adopta en ese momento inicial ServEngine como base de desarrollo.

Procedimiento

El equipo de diseño no tiene nada más que utilizar las paginas JSP de la aplicación ejemplo suministrada con ServEngine, elegir qué servicios quieren implementarse y editar con total libertad esas páginas. Una vez listas, se archivan en formato WAR (ZIP con descriptores XML) incluyendo la librería de componentes de ServEngine. En el contenedor de EJBs no es necesario más que poner en explotación (*deployment*) los paquetes de la librería elegidos y poner en explotación el cliente .war con las páginas JSP. Más adelante y en caso de necesidad se podrá, por supuesto, modificar la funcionalidad de los servicios puesto que se tiene control completo sobre el interfaz (JSP) y el corazón de los componentes (EJBs).

Requerimientos técnicos

- WEBmaster de nivel medio
- Equipo de diseño sin conocimientos de JSP
- No se necesita programación Java/JEE de ningún tipo.

Ejemplo práctico

SoloModa.com [20] es un portal que ofrece la comunidad virtual para interesados en el mundo de la moda líder en el Internet de habla castellana. Con una sencilla integración del componente ServEngine de correo WEB en modalidad ASP, puede ofrecer este servicio sin necesidad de instalar ningún componente en su propio servidor. No es necesario editar el interfaz gráfico de ServEngine más que cambiando los colores y logotipos para integrarlo en el resto del portal.

8.2.- Integración media

Incorporación de servicios dinámicos con programación específica (extensión de los componentes) pero adoptando el entorno de usuarios y permisos estándar de ServEngine.

Procedimiento

ServEngine se suministra con un paquete de componentes para gestión de usuarios y permisos que comparten el resto de componentes. Al desarrollar un nuevo servicio no hay más que referenciar las clases JNDI de gestión de usuarios y permisos desde las nuevas EJBs. De esta manera se dispone de un control de usuarios, permisos y sesiones centralizado y común a todos los servicios a desarrollar, y a los servicios que se ofrecen con los componentes estándar de ServEngine.

Puesto que en la mayoría de los casos, ServEngine se utiliza para desarrollar aplicaciones Internet, junto con la librería de EJBs se incluyen Actions y ActionForms basados en Struts Action 1.3

[20] http://www.solomoda.com

genéricos para desarrollar de forma sencilla los correspondientes clientes WEB/WAP basados en páginas JSP.

Requerimientos técnicos:

- WEBMaster de nivel medio.
- Equipo de diseño con conocimientos básicos de JSP.
- Equipo de programación Java/JEE con nivel básico/medio.

Ejemplo práctico:

Hoymesiento.com es un portal que ofrece una aproximación muy especial a la presentación de contenidos y servicios, basada en el estado anímico que especifica el usuario. Necesita ofrecer servicios de WEBMail, Chat y foros, con una autenticación única y, lo más importante, con la dirección del usuario que cambie dependiendo de su estado anímico. Para ello, se utiliza ServEngine com herramienta de desarrollo sobrecargando algunos métodos de la clase Usuario. Los 3 servicios quedan así totalmente integrados en Hoymesiento con los anteriores requerimientos sin más que desarrollando unos pocos cientos de lineas de código.

8.3.- Integración avanzada:

Incorporación de servicios dinámicos con programación específica e integración del sistema de autentificación de usuarios y permisos en uno ya existente.

Requerimientos técnicos:

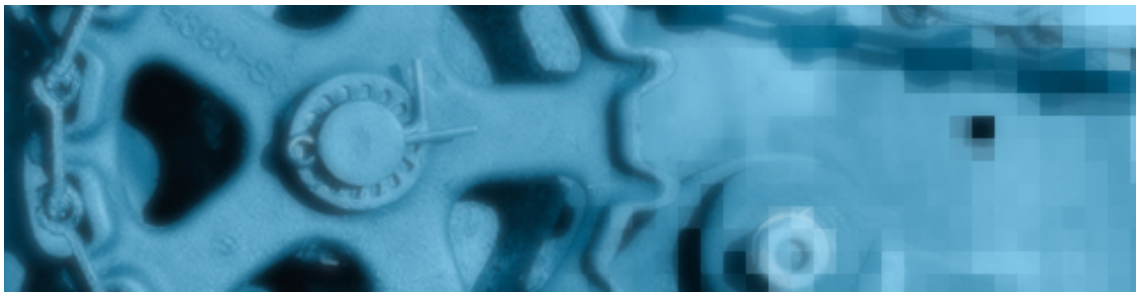
- WEBMaster avanzado.
- Equipo de diseño con experiencia en desarrollo JSP
- Equipo de programación con conocimientos avanzados de Java.

Ejemplo práctico:

Una universidad privada necesita un sistema de WEBMail para sus alumnos, pero integrado con su propia base de datos de usuarios. También precisa ofrecer datos de las calificaciones de los alumnos e incluso que estos puedan efectuar la reserva de matrícula a través de Internet. Para ello se desarrollan EJBs integradas con la base de datos de alumnos del centro (al utilizar persistencia CMP 2.0 en la mayoría de ocasiones ni siquiera es necesario reescribir estas EJBs, sino solo remapearlas a los nuevos campos de la base de datos). En lo que respecta al cliente WEB/WAP, se sobrecarga el componente de autenticación de ServEngine, permitiendo que el módulo de correo se integre directamente sin efectuar ningún cambio. Partiendo de los componentes heredables, se desarrollan dos componentes a medida que heredan todo el sistema de autenticación y seguridad incorporado a ServEngine.

9 Publicación de la librería en código abierto

Casos prácticos para proyectos de distinta envergadura



Una vez tomada la decisión de la publicación del código fuente de esta librería como código abierto [21], fue necesario elegir por un lado la licencia más adecuada, de entre las más de 30 licencias comunes, así como un servicio online de gestión de proyectos abiertos

9.1.- Licencia LGPL, *GNU Library GPL*

He elegido la “políticamente no tan correcta” [22] licencia GNU Library GPL en detrimento de la licencia GNU GPL. El motivo es sencillo. Al publicar un código fuente con licencia GPL obligamos a cualquier utilización de ese código a ser distribuida y hacerse accesible con la misma licencia. Esto hubiera hecho que ServEngine no pueda ser utilizado para algunas aplicaciones empresariales en las que el cliente no desea publicar el código fuente de su desarrollo.

Según mi experiencia en desarrollo de aplicaciones empresariales, esta situación es bastante común y creo en cualquier caso que se puede recibir el mismo nivel de aceptación y colaboración de la comunidad de desarrolladores Java / JEE publicando una librería con licencia LGPL.

9.2.- Hospedaje del proyecto ServEngine

Para la publicación de ServEngine ante la comunidad de desarrolladores java he elegido SourceForge.net, la comunidad de proyectos en código abierto más importante. He analizado también la oferta de java.net [23] si bien está orientada a Java me ha parecido menos flexible y con menos capacidad de generar colaboración que SourceForge.org

ServEngine está disponible desde la fecha de publicación de esta memoria en:

http://sourceforge.net/projects/servengine

---

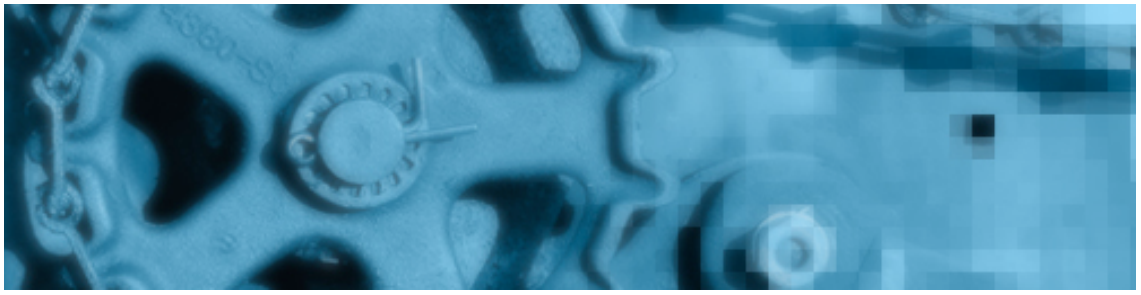
[21] http://opensource.org/docs/definition.php

[22] http://www.gnu.org/licenses/why-not-lgpl.html

[23] http://www.java.net

10 Aplicación ejemplo

Consideraciones sobre puesta en explotación



10.1.- Arquitectura de puesta en explotación

Para la puesta en explotación de la aplicación ejemplo objeto de esta memoria he utilizado el contenedor J2EE más difundido en la actualidad, JBoss. En el desarrollo de la librería he intentado evitar cualquier dependencia de un contenedor, pero a causa de las limitaciones actuales del lenguaje EJBQL (lenguaje query para EJBs de entidad) hay un método en la entidad de usuario, que permite una búsqueda por un número de parámetros arbitrarios, que esta resuelto usando JBoss Dynamic QL en el archivo jbosscmp-jdbc.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jbosscmp-jdbc PUBLIC "-//JBoss//DTD JBOSSCMP-JDBC 3.0//EN"
"http://www.jboss.org/j2ee/dtd/jbosscmp-jdbc_3_0.dtd">
<jbosscmp-jdbc>
   <defaults>
      <datasource>java:/servenginedemoDS</datasource>
      <datasource-mapping>mySQL</datasource-mapping>
      <create-table>true</create-table>
   </defaults>
   <enterprise-beans>
      <entity>
         <ejb-name>User</ejb-name>
           <query>
           <query-method>
                <method-name>findByCriteria</method-name>
                <method-params>
                     <method-param>java.lang.String</method-param>
                     <method-param>java.lang.Object[]</method-param>
                </method-params>
           </query-method>
           <dynamic-ql/>
           </query>
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

No soy partidario de utilizar Jaws o Hibernate, especialmente en el periodo de desarrollo y pruebas. La ventaja de utilizar una base de datos como MySQL es que tenemos la posibilidad de acceder a los datos de forma independiente y rápida mediante el cliente gráfico o de texto proporcionado por el desarrollador de la propia base de datos. Además, en algunos desarrollos he necesitado usar la base de datos forzosamente por que otras aplicaciones accedían a datos de la misma [24].

El uso de MySQL con JBoss es muy sencillo. Por un lado no hay más que indicar que se usará ese tipo de mapeo de datos en el archivo jboss.xml [25]:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS//EN"
"http://www.jboss.org/j2ee/dtd/jboss.dtd">
<jboss>
    <container-configurations>
        <container-configuration>
            <container-name>Standard CMP 2.x EntityBean</container-name>
            <commit-option>A</commit-option>
            <!--optiond-refresh-rate>300</optiond-refresh-rate-->
        </container-configuration>
    </container-configurations>
```

[24] Por ejemplo en un DNS dinámico basado en EJBs

[25] Si se pone en explotación la aplicación en un cluster de JBoss no debe usarse la opción de commit A (solo actualiza a base de datos cuando es imprescindible, ahorrando carga en la misma)

```
</jboss>
```

Por otra parte, es necesario situar un “datasource” JDBC independiente del archivo .jar o .ear que contenga las EJBs de entidad (y, lógicamente, debe ser puesto en explotación antes que nuestra aplicación). El nombre de este datasource se especifica en el archivo jbosscmp-jdbc.xml como puede comprobarse en el listado anterior. El contenido del archivo XML con la configuración del datasource en servenginedemo-mysql-ds.xml es:

```
<datasources>
  <local-tx-datasource>
    <jndi-name>servenginedemoDS</jndi-name>
    <connection-url>jdbc:mysql://localhost/servenginedemo</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>servengine</user-name>
    <password>supersql</password>
  </local-tx-datasource>
</datasources>
```

10.2.- La aplicación web

La aplicación web puede ser empaquetada junto con los JARs que contienen las EJBs. En mi servidor de pruebas pongo en explotación ambas capas de esta manera, pero en producción distribuyo en dos servidores las capas, conectadas por *lookups*/RMI. Así pues, el JBoss exclusivamente es utilizado como un contenedor de EJBs y las aplicaciones web las pongo en explotación en un Tomcat independiente.

10.3.- Servidor web

Para evitar la carga innecesario de servidor objetos estáticos por el Tomcat, y para envitar tener que ejecutar esa máquina virtual con privilegios de administrador y poder así servir el protocolo http en el puerto 80, se debe instalar un servidor web Apache para este propósito, enlazado mediante un módulo con el servidor de aplicaciones web (Tomcat en nuestro caso).

10.4.- Aplicación ejemplo

He situado la aplicación ejemplo de la librería en la siguiente dirección web:

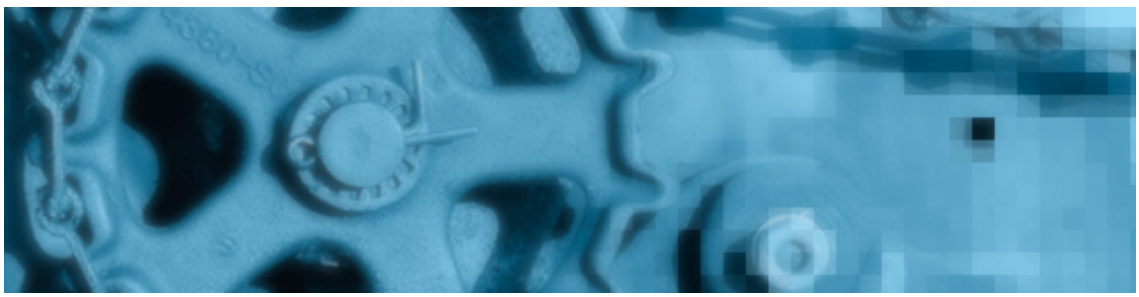
http://pfc.ignacio.decordoba.com

Dicha aplicación WEB hace uso del framework en una pequeña demo gráfica de portal. He implementado el proceso de login (usuario “**admin**”, sin contraseña) y la edición de preferencias para ese usuario. He dejado preparada la presentación gráfica para futuros módulos, que se irían incluyendo en esta aplicación ejemplo.

La aplicación ejemplo se entrega en el archivo servenginedemo.ear, el cual debe ser situado en el fichero $JBOSSHOME/server/default/deploy/

La aplicación utiliza Hypersonic como base de datos, la cual se encuentra embarcada en la distribución estándar de JBoss. He utilizado la última versión de JBoss, 4.0.4, para probar la aplicación si bien debe funcionar en cualquier versión de JBoss o en cualquier servidor de aplicaciones J2EE 1.4 si se editan los correspondientes ficheros de puesta en explotación (deployment).

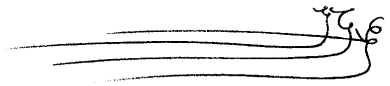
11 Conclusiones



En este proyecto se ha analizado las necesidades comunes en el desarrollo de portales, comunidades virtuales y sistemas de comercio electrónico. Hasta ahora, no existía una base de desarrollo, en forma de framework-librería, que siguiera un patrón de diseño como MVC, tuviera su código publicado en modo *Open Source* y permitiera a desarrolladores e integradores web efectuar un desarrollo rápido de aplicaciones siguiendo las anteriores directrices.

ServEngine no solo ofrece todo lo anterior, sino que permite el desarrollo de módulos integrados con el principal gracias a la preparación de una serie de clases abstractas y servicios genéricos para la configuración y uso por parte de dichos módulos.

Estoy convencido de que, habiendo dado el importante paso de publicación del código de la librería-framework que he desarrollado en este proyecto, se formará en torno a ServEngine una comunidad de desarrolladores JEE.

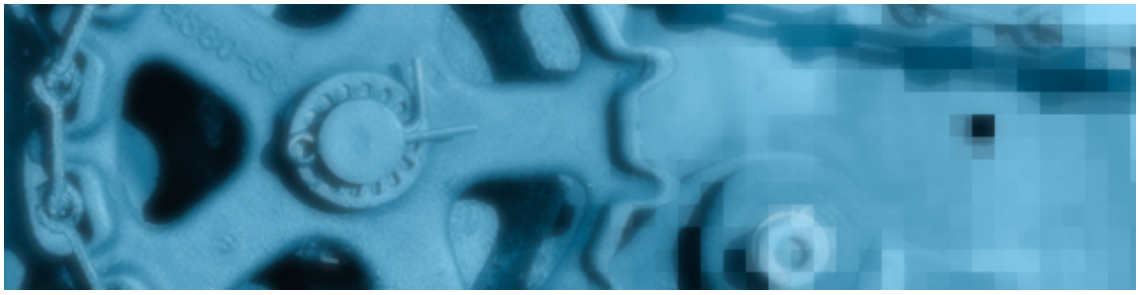


Ignacio de Córdoba

ignacio@decordoba.com

Junio - 2006

12 Enlaces de referencia



Librería y aplicación ejemplo

- http://sourceforge.net/projects/servengine
- http://pfc.ignacio.decordoba.com
- http://www.servengine.com
- http://www.tuportal.com

Licencia LGPL

- http://www.gnu.org/licenses/licenses.html#LGPL
- http://opensource.org/docs/definition.php

JEE

- http://java.sun.com/javaee
- http://www.jcp.org/en/jsr/detail?id=168

Frameworks

- http://struts.apache.org
- http://www.springframework.org
- http://www.opensymphony.com/webwork
- http://jakarta.apache.org/tapestry
- http://java.sun.com/javaee/javaserverfaces
- http://www.rubyonrails.org