Janus: un Generador de la Vista de Roma Framework basado en Plantillas

Pablo Martín, Guillermo Hernández
Division I+D+i
Informática Gesfor (Grupo Gesfor)
Avda Manoteras, 32 28050 Madrid
{pmartinb,ghernandezc}@grupogesfor.com

Carlos A. Iglesias
Division I+D+i
Germinus XXI (Grupo Gesfor)
Avda Manoteras, 32 28050 Madrid
cif@germinus.com

Luca Garulli, Giordano Maestro
Division I+D+i
Asset Data s.r.l.
Via Rhodesia, 34 00144 Roma
{luca.garulli, giordano.maestro}@assetdata.it

***Resumen*—Los enfoques tradicionales de desarrollo de aplicaciones web han mostrado serios problemas de productividad, lo que está dando lugar a nueva generación de frameworks web que automatizan en gran medida el desarrollo para aplicaciones principalmente de persistencia de datos. En este proyecto se presenta cómo la utlización de plantillas puede facilitar la generación de la capa de presentación en uno de estos nuevos frameworks ágiles, Roma Metaframework. Este trabajo ha sido desarrollado dentro del proyecto ICT Romulus. La solución, llamada Janus, utiliza un mecanismo de plantillas basado en FreeMarker para extraer información de los objetos que definen la vista de la aplicación generando una estructura de páginas JSP equivalente, y se comunica con el framework mediante una serie de etiquetas Java. Como resultado, Janus permite la generación automática de una interfaz gráfica personalizable, implementada con JSP, CSS y Javascript, con un tiempo de desarrollo reducido.**

***Palabras Clave*—Janus, JSP, Roma, Metaframework, Java, Javascript, FreeMarker, Romulus**

I. Introducción

El Desarrollo Web es una de las áreas más activas, tanto a nivel nacional, como europeo, pero también es un área poco madura debido a la proliferación de frameworks y componentes. En concreto, Java Enterprise Edition es la opción preferida por los europeos con más del 38.6 % de desarrolladores usuarios de esta tenología según los datos de Evans [5]. Este hecho ha provocado que las habilidades requeridas por los desarrolladores web para llevar a cabo un desarrollo software, aumenten, y con ello baje su productividad y la fiabilidad de sus desarrollos. Este problema es particularmente acentuado en el entorno java, en el que el desarrollo de aplicaciones exige multitud de conocimientos de frameworks y tecnologías (Struts, Hibernate, Java Server Pages, XSLT, JUnit, ...). En este contexto surge el proyecto Romulus [18], cuyo principal objetivo es investigar y desarrollar nuevos métodos para aumentar la productividad y la fiabilidad del desarrollo de aplicaciones web en Java.

El resto del artículo se estructura como sigue. La sección II hace una breve introducción al proyecto Romulus donde se encuadra Janus. La sección III presenta las necesidades que se pretenden cubrir y las deficiencias que presenta Echo2, seguida de la sección IV donde se explica la implementación de Janiculum como mecanismo actual para mostrar la vista de las aplicaciones en Roma. A continuación, en la sección V, se exponen las herramientas de generación de plantillas más populares y se define la arquitectura de la solución propuesta en la sección VI. La sección VII ilustra con un ejemplo el funcionamiento de esta solución y, por último, se recogen las conclusiones y el trabajo futuro en la sección VIII.

II. El proyecto Romulus

El proyecto ROMULUS [18] tiene el objetivo de contribuir en el desarrollo ágil de aplicaciones web en Java. El proyecto investiga en diseño dirigido por el dominio (DDD, *Domain Driven Design*) para aplicaciones web basado en el metaframework Roma, así como en la integración de la tecnología de mashups en el ciclo de vida de desarrollo software.

En el proyecto participa la gran empresa, **Informática Gesfor**, como coordinadora, las PYMEs **Asset Data**, **Liferay**, **IMola** e **ICI** y los centros de investigación: **Universidad Politécnica de Madrid** y el instituto **DERI**.

Romulus propone la mejora de la productividad del desarrollo de las aplicaciones web, así como de su calidad, mediante cuatro objetivos:

- Diseño Dirigido en el Dominio basado en un Metaframework. Como se presenta en la sección II-A, el proyecto investiga en el uso de un metaframework y en centrar los esfuerzos en el modelado del dominio, empleando técnicas de generación automática de código para los frameworks Java más populares.
- Desarrollo Orientado a Mashups. Reutilización de componentes y servicios disponibles para el desarrollo y extensión de aplicaciones.
- Introducción de objetivos no funcionales (seguridad, fiabilidad, análisis de prestaciones) en todo el ciclo de vida.
- Investigar en el balance adecuado entre tecnologías cliente, tecnologías servidor, y tecnologías de scripting. El proyecto investiga en la definición de buenas prácticas y patrones de diseño para repartir adecuadamente la carga entre estos puntos.

Estos objetivos suponen, en términos de productividad, una **reducción del 20 % en tiempos de desarrollo** de aplicaciones web estándar y un **30 % menos de conocimientos requeridos**, basándose en el número de librerías necesarias y su complejidad para implementar una serie de tareas estándar [17].

II-A. Roma Metaframework

Roma [6] propone un nuevo paradigma para el desarrollo de aplicaciones web tomando ventajas de las nuevas tendencias en ingeniería de software tales como el diseño

dirigido por dominio (*Domain Driven Design*) combinado con metodologías de desarrollo ágil y algunos de los principios de Ruby on Rails [16]. Para lograr este objetivo funciona como un meta-framework que permite desarrollar aplicaciones utilizando cualquier framework como una pieza intercambiable, desacoplada de la aplicación en sí. Esto permite la integración de las nuevas tecnologías sin modificar el dominio ni la lógica de la aplicación y también aumentar las tasas de productividad en el desarrollo de software.

Roma propone tres niveles de abstracción:

- El *Dominio*, definido por el desarrollador y que tiene que ver con el modelo de negocio de la aplicación.
- Los *Aspectos*, implementados por el framework (persistencia, vista, internacionalización, sesión, monitoreo, flujo, etc).
- Los *Módulos*, que proporcionan facilidades para acelerar el desarrollo, promoviendo la reutilización *de* código (persistencia, login, gestión de usuarios, integración con IDEs, etc.).

Las características principales de Roma son:

- Roma está basada totalmente en POJOs (Plain Old Java Objects)
- Todos los aspectos son orientados a objetos: desde el modelo a la vista.
- Promueve el uso del diseño dirigido por dominio (Domain Driven Design).
- Funciona con convenciones del estilo de Ruby On Rails: mucho menos código que escribir y mantener y mayor uniformidad de los proyectos.
- Las aplicaciones son portables a otros frameworks gracias a la orientación a POJOs.
- Permite modificar directamente el framework, se pueden desarrollar módulos adicionales o extender los ya existentes.
- Está basado en Spring Framework pero se pueden extender otros frameworks.

Roma proporciona al usuario soporte automático en cada capa y aspecto de su aplicación, desde interfaces web de usuario dinámicas y persistencia, hasta funcionalidad de informes, desarrollo de portlets y tecnologías semánticas. Proporciona interfaces de comportamiento muy genérico llamadas "Aspectos". Los Aspectos incluyen los casos de uso más comunes, y permiten hacer independiente el código de la tecnología, con lo que en caso de futuras migraciones de tecnología, el proceso será más rápido y sencillo. La figura 1 muestra los aspectos implementados por Roma. En este artículo se describirá el módulo Janiculum, que implementa el aspecto View, y se presentará Janus como una alternativa para la vista de las aplicaciones.

III. Necesidades

La interfaz gráfica de las aplicaciones es un aspecto complejo que, siguiendo la filosofía de Roma, debe ser independiente de la lógica de la aplicación y, además, debe permitir el uso de diferentes tecnologías subyacentes. Roma proporciona por defecto una interfaz gráfica para las aplicaciones generada con la tecnología de Echo2. Echo2 es una plataforma basada en AJAX que ofrece una capa de Java que genera dinámicamente Javascript. Con esto permite la construcción de aplicaciones

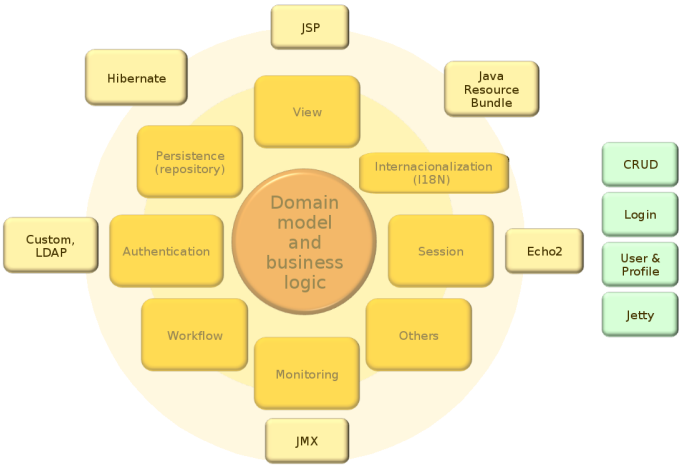

Figura 1. Aspectos de Roma

web aprovechando las capacidades de los clientes ricos. Las aplicaciones se realizan enteramente en Java mediante una API orientada a componentes y dirigida por eventos [13]. La ventaja que proporciona esta solución es su potencia y la total abstracción de la capa web. Sin embargo tiene una serie de deficiencias que motivan el desarrollo de otra solución para generar la vista de las aplicaciones web en Roma. La primera de sus limitaciones es la gestión de eventos. Al ejecutarse sobre un servidor, los eventos sólo se envían desde un cliente bajo ciertas circunstancias, por ejemplo, al hacer clic sobre un botón. Esto hace que para ciertas operaciones, como por ejemplo autocompletar o desplazar el ratón sobre un elemento, sea necesario crear componentes personalizados, lo cual influye negativamente en la productividad de la aplicación. Además Echo2 genera identificadores, dentro del código Javascript, que cambian con cada sesión lo que hace que dicho código generado no se pueda modificar. Otro problema de Echo2 es la generación de URLs ya que sus aplicaciones sólo cuentan con una URL global lo que resulta negativo para posicionarse en buscadores. Finalmente, el uso de esta plataforma requiere de una formación adicional ya que no se trata de un framework muy extendido y, además, su comunidad está decreciendo año tras año. Todas estas limitaciones hacen necesaria la utilización de otra tecnología más flexible en la personalización y que esté ampliamente difundida para mejorar la presentación y aumentar la productividad. La solución que presenta Janus soluciona los problemas de Echo2 ya que utiliza tecnologías de CSS y JSP, muy extendidas en el desarrollo de interfaces web, con una gran cantidad de librerías disponibles e independientes de servidor y plataforma. Janus genera una interfaz web en JSP que el desarrollador podrá utilizar directamente o modificar según sus necesidades. Esta solución permite que la aplicación presente diferentes URLs ya que se genera una estructura de páginas JSP similar a la que se genera en frameworks semejantes como Ruby on Rails o Grails donde se utilizan mecanismos de plantillas para generar la interfaz gráfica de las aplicaciones.

IV. Modulo Janiculum de Roma

Para solucionar algunos de los problemas de Echo2, dentro de Roma se ha desarrollado un aspecto llamado Janiculum

que traslada los conceptos de Roma a páginas dinámicas utilizando standards XHTML y CSS2. Para ello tiene que realizar una correspondencia entre los objetos de Java y las áreas definidas en el layout de Roma. Posteriormente se realiza una asociación entre los elementos de la vista y los objetos. Para ello Janiculum realiza las siguientes operaciones:

- Generar la estructura de la página a partir de la definición de las áreas de pantalla
- Generar fragmentos HTML para cada Objeto/atributo/método
- Asociar identificadores únicos para cada Objeto/atributo/método
- Realizar la conversión entre los parámetros de la petición (Strings) y los valores del atributo del objeto
- Identificar qué acción se ha realizado en el cliente e invocar el método correspondiente en el objeto ubicado en el servidor

IV-A. Definición de áreas en Roma

Roma permite definir áreas donde colocar los diferentes elementos de los POJOs, independientemente de la implementación utilizada para la vista. Estas áreas se definen en ficheros XML que se colocan por defecto dentro del paquete view.domain.screen, donde se encuentra la definición por defecto **main-screen.xml** (figura 2):

Este archivo permite definir las áreas y su colocación dentro de la pantalla [12]. La colocación de los elementos dentro de dichas áreas se realiza mediante anotaciones con el parámetro *layout*. Por ejemplo la anotación @ViewClass(layout = ”screen://body”) para una clase colocará dicha clase dentro del area llamada *body*. Esto permite definir la colocación de los elementos desde las clases java, sin hacer referencia a ninguna tecnología, lo que permite cambiar de Echo2 a JSP u otra tecnología sin necesidad de modificar la aplicación.

```
<?xml version="1.0" encoding="UTF-8"?>
<screen xmlns="http://www.romaframework.org
/xml/roma"
        xmlns:xsd="http://www.w3.org/2001
/XMLSchema-instance"
        xsd:schemaLocation="http://www
.romaframework.org/xml/roma http://www
.romaframework.org/schema/roma-view-screen
.xsd">

        <area name="main" type="grid" size="1">
        <area name="main" type="column">
                <area name="header" />
                <area name="menu" />
                <area name="body" />
                <area name="footer" />
        </area>
</screen>
```

Figura 2. Definición por defecto de las areas en main-screen.xml

IV-B. Taglib de Roma

Como se ha descrito anteriormente Roma proporciona una librería de etiquetas o taglib (**roma.tld**) que permite la comunicación entre la vista y el controlador generando código con los identificadores unívocos que utilizará internamente Roma. Esta taglib dispone de tres etiquetas básicas:

- *<roma:class>* - Muestra el objeto completo utilizando la renderización por defecto.
- *<roma:field name=”fieldName”part=”partName”>* - Muestra un atributo del objeto. El campo name (obligatorio) debe contener el nombre de dicho atributo mientras que part (opcional) permite especificar cómo se mostrará el atributo: entero, sólo la etiqueta o sólo el contenido.
- *<roma:action name=.ªctionName”>* - Muestra un método del objeto. El campo name (obligatorio) debe contener el nombre de dicho método. Por ejemplo, con la etiqueta: *<roma:field name=.ªddress/>*, el taglib generará el código HTML correspondiente al atributo address del objeto que se esté mostrando.

Con esta implementación Roma proporciona una interfaz gráfica basada en HTML que genera páginas dinámicamente a partir de los POJOs. Si un usuario quiere modificar el aspecto de un determinado POJO (Ejemplo.java) puede crear el archivo Ejemplo.jsp y personalizar su vista utilizando las etiquetas proporcionadas por la taglib de Roma. Esto resulta poco productivo porque para personalizar el aspecto de la aplicación sería necesario crear manualmente el archivo correspondientes a cada POJO. Por ello se ha implementado un mecanismo de plantillas llamado **Janus** como una alternativa que genere todos estos archivos automáticamente y que, además, permita personalizar fácilmente y desde un único punto la forma en que los archivos se generan.

V. HERRAMIENTAS DE GENERACIÓN DE PLANTILLAS

En esta sección se presenta un estudio de las distintas herramientas existentes para la generación de plantillas. Una plantilla [23] es una forma de dispositivo que proporciona una separación entre la forma o estructura y el contenido. Es un instrumento que permite guiar, portar o construir un diseño o esquema predefinido. Hay una gran variedad de herramientas para generar plantillas, como por ejemplo: Beilpuz [3], eRuby [7], Dwoo [19], Evoque Templating [20], GvTags [8], h2o [9], etc.

Velocity [1] es un generador de plantillas basado en Java y de sintaxis similar a Javascript. Permite a los diseñadores web referenciar métodos definidos en código Java. Esto proporciona a la página web mayor mantenibilidad, ya que separa el código del diseño, y es una alternativa viable a JSPs o PHP.

FreeMarker [15] es una librería Java que proporciona un generador de plantillas para generar texto. Ha sido diseñado para su uso por aplicaciones basadas en servlets y en el modelo MVC (Modelo Vista Controlador) para la generación de páginas web HTML. FreeMarker no es un lenguaje de programación, simplemente genera texto a partir de programas java y lo muestra usando plantillas.

Si comparamos ambas soluciones, la primera conclusión que se puede extraer es que Velocity es una herramienta *más simple* y ligera pero con un lenguaje de plantillas menos potente que no permite tantas operaciones como FreeMarker [14]. Además Velocity hace un uso directo de los métodos de los objetos java y mueve tareas de presentación al código del controlador lo que contradice los principios del patrón MVC. FreeMarker es más estricto con esto, lo que puede generar problemas con algunas operaciones, pero el html

generado es más correcto. La mayor ventaja de Velocity frente a FreeMarker es su comunidad mucho mayor y el soporte de Apache. Además, al ser más simple, es también más rápido que FreeMarker en algunas operaciones.

Sin embargo FreeMarker ofrece una serie de características que no pueden ser realizadas con Velocity de forma trivial. Las más importantes de estas características son las mejores macros, la capacidad de utilizar las bibliotecas de etiquetas JSP personalizadas en plantillas o trabajar directamente sobre objetos de Python y el uso de namespaces.

Con FreeMarker se puede convertir también plantillas de Velocity a plantillas de FreeMarker gracias a una herramienta propia [22]. Ambos mecanismos de plantillas son similares para las operaciones básicas y tienen buen rendimiento con plantillas pequeñas, si bien un factor a tener en cuenta es que FreeMarker es más rápido a la hora de procesar plantillas grandes. Por tanto, la potencia de FreeMarker y su elegancia lo convierten en una mejor opción para su integración con Roma.

VI. Definición de la arquitectura y solución

El componente Janus genera páginas JSP a partir de los objetos java de la aplicación y utiliza tecnología de CSS2 para la apariencia. Para la integración con el metaframework de Roma se utilizan anotaciones java [21] y las etiquetas proporcionadas por el taglib de Roma (**roma.tld**). Para esta generación de código, Janus utiliza **FreeMarker** como mecanismo de plantillas e integra tecnología de Javascript para mejorar la presentación, mediante la utilización de **JQuery UI** [2]. La Figura 3 muestra la estructura básica del módulo y el flujo de información para generar el código jsp final.

VI-A. Definición de Plantillas Dinámicas para Roma

Este apartado recoge la definición de plantillas dinámicas para Roma basadas en FreeMarker que facilitan la adaptación de las vistas de una aplicación basada en el metaframework Roma. Gracias a la integración de FreeMarker con Java, se ha realizado una biblioteca de etiquetas o taglib de Roma que permite la comunicación de la vista con el modelo, lo que posibilita la generación de diferentes vistas cambiando únicamente estas plantillas de FreeMarker.

La implementación por defecto de la plantilla de Janus para Romulus organiza el código de los archivos JSP generados colocando en una sección los atributos del objeto, cada uno dentro de un . A continuación, en una nueva sección, se colocan los métodos del objeto. La figura 4 muestra un fragmento de esta plantilla con la generación de etiquetas para los métodos y atributos del objeto utilizando la taglib de Roma.

```
...
<!-- fields of the pojo -->
<div class="${classname}_fields">
<#list fields as fieldname>
   <div>
        <roma:field
        name="${fieldname}" />
   </div>
</#list>
</div>

...

<!-- actions of the pojo -->
<h2>${classname} actions</h2>
<div class="buttonBar">
        <#list actions as actionname>
                <roma:action
                name="${actionname}"/>
        </#list>
</div>
...
```

Figura 4. Fragmento de la plantilla por defecto de Janus para Romulus

El código resultante generado con esta plantilla presentará los atributos y los métodos en diferentes columnas como muestra la figura 5.

VI-B. Configuración de Plantillas Dinámicas para Roma

Una plantilla de Romulus permite los siguientes constructores:

- *Date*. Las fechas se muestran con el calendario JQuery UI Datepicker.
- *ContainerPage*. Esta clase de Roma se muestra utilizando los tabs de JQuery UI.

Esto se puede configurar mediante el archivo *constructors.xml* que permite definir las asociaciones entre constructores y elementos de JQuery UI.

Además de esta estructura de los JSPs generados, válida para cualquier objeto, Janus tiene un comportamiento específico para las clases que Roma genera automáticamente

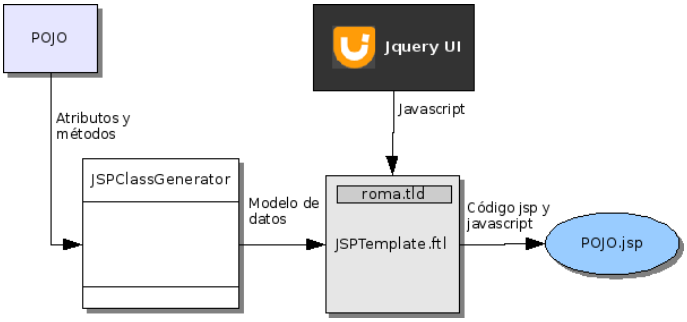

Figura 3. Flujo de información de Janus

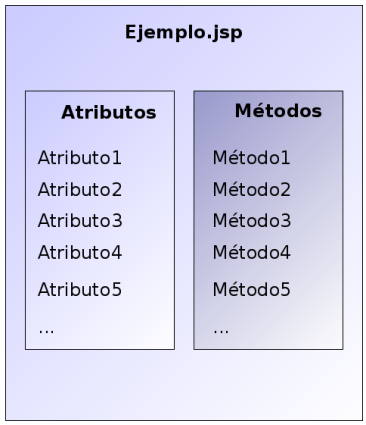

Figura 5. Estructura de los JSPs generados con la plantilla de Romulus

para facilitar las tareas de creación, lectura, actualización y borrado (CRUD). Para estas clases la generación de código está optimizado para una mejor usabilidad mostrando un filtro de búsqueda en la columna izquierda y los resultados, junto con las operaciones CRUD en la de la derecha como muestra la figura 6.

En el futuro se podrá definir una biblioteca de elementos para el cliente (Javascript, CSS, ...) de forma que se puedan establecer fácilmente correspondencias entre estos elementos de la biblioteca y los elementos del POJO. Estas correspondencias servirán para definir perfiles que permitirán a la aplicación ofrecer diferentes vistas dependiendo del perfil del usuario.

VI-C. Estructura y funcionamiento

Para conseguir la independencia de la vista en las aplicaciones en Roma se define la interfaz a través de objetos Java (POJOs) con anotaciones localizados dentro de un paquete específico llamado *view* [11]. El componente Janus crea una estructura de archivos JSP paralela a la existente en dicho paquete de forma que cada objeto java de ese paquete (o bien aquellos que el usuario elija) tengan un archivo JSP correspondiente. Esta estructura de archivos JSP tiene una jerarquía al igual que los objetos java de forma que el nivel más alto está representado por la clase Object.jsp que Roma provee por defecto. Estos archivos jsp se colocan en un directorio dentro de la estructura de directorios que Roma crea para cada aplicación. La Figura 7 muestra esta estructura.

Dentro del directorio *dynamic* se encuentran las hojas de estilo y los archivos jsp correspondientes a los objetos Java. En el directorio *static* se encuentran los archivos jsp, hojas de estilo y librerías correspondientes a las tecnologías externas que se quieran integrar en el proyecto. En este caso se trata de la librería de Javascript JQuery UI [2] y la librería de CSS Yaml [10].

El comportamiento de Roma para mostrar un objeto consiste en buscar un archivo jsp cuyo nombre sea igual al nombre de la clase de dicho objeto acabado en .jsp. Si existe, Roma utilizará este archivo para mostrar el objeto y, si no, irá ascendiendo por la estructura jerárquica hasta hallar el jsp adecuado o llegar a Object.jsp quien mostrará el objeto de la forma definida por defecto.

El componente Janus crea archivos jsp a partir de objetos java de forma que se puedan modificar fácilmente sin necesidad de compilar el código fuente de la aplicación. Para ello hace uso del taglib de Roma que genera el código html adecuado para los atributos y métodos del objeto que se va a mostrar, con los identificadores unívocos que Roma podrá utilizar para su funcionamiento interno. Este taglib se incluye en los archivos JSP que se generan con Janus de forma que para generar el código correspondiente a los atributos y los métodos basta con poner la etiqueta correspondiente del taglib dentro del archivo JSP generado.

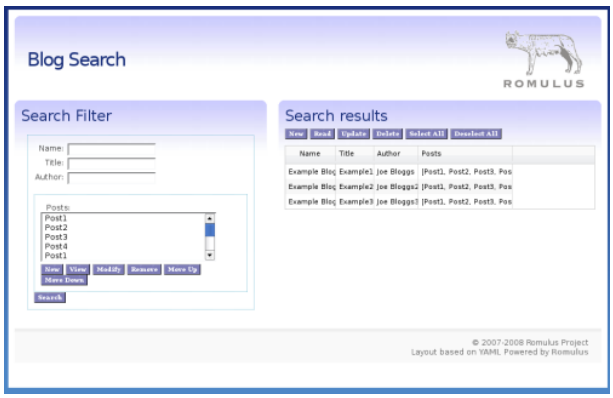

Figura 6. Estructura de los JSPs para las clases CRUD de Roma

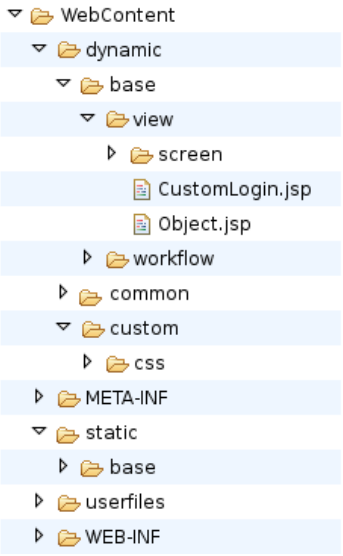

Figura 7. Estructura de directorios para los archivos JSP

El funcionamiento de Janus para generar código JSP es el siguiente: Mediante introspección se recorre el objeto que se desea mostrar añadiendo al modelo de datos de FreeMarker todos sus métodos y atributos. Posteriormente la plantilla JSPTemplate.ftl accede a este modelo de datos organizando los atributos y los métodos con las etiquetas correspondientes del taglib de Roma: **roma.tld**. La ventaja de este comportamiento es que el usuario puede modificar la plantilla consiguiendo cambiar la forma en que se generan los archivos JSP finales. Concretamente, se pueden integrar librerías de Javascript para mejorar la apariencia de los JSPs generados.

Para la implementación del Janus se ha incluido tecnología de JQuery UI [2] para organizar los atributos y los métodos en diferentes pestañas, así como para el renderizado de algunos tipos de datos como por ejemplo los atributos de tipo Date que son mostrados utilizando el calendario JQuery UI Datepicker. Además se ha incluido tecnología de Yaml [10] para el diseño de las columnas.

Esta solución proporciona flexibilidad para la implementación de la interfaz gráfica por parte de los desarrolladores ya que pueden retocar individualmente los jsp's generados mientras que se reduce el tiempo de desarrollo al permitir modificar el código de todos los archivos jsp desde un único punto y sin necesidad de compilar. La siguiente sección muestra un ejemplo del uso de Janus para crear la interfaz de una aplicación.

VII. Ejemplo de aplicación o futuras aplicaciones

El proyecto Scrooge [4] es un proyecto que se utilizará como demostrador del proyecto Romulus y que esta desarrollado en parte con Romaframework.

El esqueleto de la aplicación está creado con Romaframework. Como se ha explicado anteriormente, la tecnología Echo2 es muy poco flexible para realizar cualquier cambio en el estilo de las aplicaciones, al estar escrita la interfaz directamente en código java, de modo que se ha utilizado el módulo Janus para generar una vista en JSP, a partir de cada objeto, mucho más flexible, con lo que se permite mayor personalización a los diseñadores web, ya que pueden utilizar JQuery para conseguir aplicaciones web más vistosas. La figura 8 muestra el aspecto de la interfaz de Scrooge.

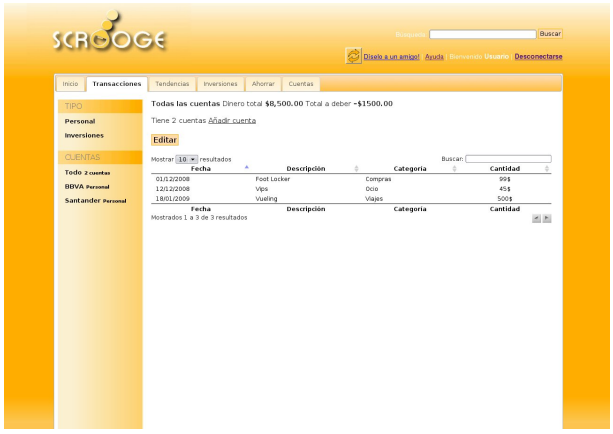

Figura 8. Interfaz de Scrooge, desarrollada con Janus

El proyecto Scrooge, aun en fase de desarrollo, hace uso de tecnologías web 2.0, AJAX y JQuery por lo que utilizará todas las facilidades aportadas por Janus.

La solución Janus ha sido diseñada para su uso con el metaframework Roma pero también puede ser utilizada en otras soluciones que utilicen objetos java para definir la interfaz gráfica o aquellos frameworks que generen automáticamente la interfaz gráfica a partir del modelo como por ejemplo Ruby on Rails o Grails. Un mecanismo de plantillas como Janus adaptado a estos entornos permitiría la personalización de la interfaz por defecto proporcionada por estos frameworks, así como la integración de nuevas tenologías de presentación.

VIII. Conclusiones y Trabajos Futuros

En este artículo se ha presentado el generador de plantillas Janus, desarrollado para Romaframework, dentro del proyecto europeo Romulus, aun en fase de desarrollo.

El trabajo ha expuesto los diferentes motores de generación de plantillas existentes en la actualidad, centrándose en los dos más interesantes (Velocity y FreeMarker), comentando sus similitudes y diferencias. Posteriormente se ha definido la arquitectura del módulo Janus y se ha mostrado un caso de uso del mismo.

En el futuro se aplicará el módulo Janus a otros demostradores y se desarrollará una API Javascript para una mayor facilidad en el desarrollo. Además se implementarán varios estilos mediante plantillas que permitirán la creación de temas personalizados según el perfil del usuario.

Agradecimientos

El módulo Janus ha sido desarrollado para Romaframework, dentro del proyecto europeo Romulus ICT-2007.1.2, financiado por la Comunidad Europea dentro del marco Seventh Framework Programme y en fase de desarrollo desde enero de 2008.

Referencias

[1] Apache. The apache velocity project, 2009. Disponible en http://velocity.apache.org/.
[2] P. Bakaus and the jQuery UI Team. Jqueri ui project, 2009. Disponible en http://jqueryui.com/.
[3] Beilpuz. Herramienta beilpuz, 2009. Disponible en http://beilpuz.getmike.de/doku.php.
[4] S. Consortium. Scrooge, 2009. Disponible en http://scrooge.germinus.com.
[5] E. D. Corporation. Evans reports, official site, online. available:, 2007.
[6] A. creativity Solutions. Roma framework, the new way to conceive web applications, 2009. Disponible en http://www.romaframework.org/.
[7] eRuby. Herramienta eruby, 2009. Disponible en http://en.wikipedia.org/wiki/ERuby.
[8] GvTags. Herramienta gvtags, 2009. Disponible en http://www.gvtags.org/.
[9] h2o. Herramienta h2o, an elegant php template engine, 2009. Disponible en http://www.h2o-template.org/.
[10] D. Jesse. Yet another multicolumn layout — an (x)html/css framework, 2009. Disponible en http://www.yaml.de/en/.
[11] G. M. Luigi Dell'Aquila, Luca Garulli. Roma <meta> framework handbook, 2009. Disponible en http://www.romaframework.org/doc/RomaHandbook.pdf.
[12] T. Meeks. Comparing the google web toolkit to echo2, 2006. Disponible en http://www.theserverside.com/news/thread.tss?thread_id=40804.
[13] NextApp. Echo2 web framework, 2009. Disponible en http://echo.nextapp.com/site/echo2.
[14] F. project. Freemarker versus velocity, 2009. Disponible en http://freemarker.org/fmVsVel.html.
[15] F. project. Fremarker: Java template engine library - overview, 2009. Disponible en http://freemarker.org/.
[16] A. Protopsaltou. Model driven development with ruby on rails, it university of göteborg, 2004.
[17] Romulus. Annex i, description of work, 2007. From the Seventh Framework Programme Theme ICT-2007.1.2, p11.
[18] Romulus Consortium. Service and software architectures, infrastructures and engineering.romulus - domain driven design and mashup oriented development, 2009. Disponible en http://www.ict-romulus.eu/.
[19] D. P. Templates. Herramienta dwoo, 2009. Disponible en http://dwoo.org/.
[20] E. Templating. Herramienta evoque templating, 2009. Disponible en http://evoque.gizmojo.org/.
[21] I. The java Comunity Process (SM) Program: Alex Buckley, Sun Microsystems. Jsr 175: A metadata facility for the javatm programming language, 2009. Disponible en http://www.jcp.org/en/jsr/detail?id=175.
[22] J. van Bergen. Velocity or freemarker?, 2009. Disponible en http://www.javaworld.com/javaworld/jw-11-2007/jw-11-java-template-engines.html?page=3.
[23] Wikipedia. Entrada template engine (web), 2009. Disponible en http://en.wikipedia.org/wiki/Template_engine_(web).