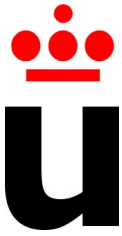

UNIVERSIDAD
REY JUAN CARLOS

MÁSTER OFICIAL

EN SISTEMAS TELEMÁTICOS E INFORMÁTICOS

Curso Académico 2009/2010

Trabajo de Fin de Máster

THEMEWEBSETTINGS, PERSONALIZACIÓN WEB DE TEMAS DE APARIENCIA EN LIFERAY

Autor : José Ignacio Huertas Fernández

Tutor : Gregorio Robles Martínez

Co-Tutor : Jorge Ferrer

Resumen

Liferay es una plataforma Web que permite la creación de portales, gestores de contenido y entornos colaborativos de una manera rápida y sencilla. Está basada en Java, es Open Source y en estos últimos años ha experimentado un crecimiento casi exponencial, lo que la ha llevado a ser una alternativa muy interesante en la construcción de portales Web colaborativos.

Entre sus características, podemos destacar que proporciona unos 60 portlets listos para usar tras su instalación (incluyendo blog, foro, wiki, correo, contenido web, ...), páginas personalizadas para todos los usuarios, gestión de usuarios y permisos a través de roles, un Panel de Control (que permite una administración centralizada para todo el contenido, usuarios, organizaciones, comunidades, roles, recursos de los servidores), drag-and-drop para la ubicación de los portlets en las páginas, capacidad de búsqueda y etiquetado, CMS, WCM, Software colaborativo, software social, etc.

Con este proyecto intento contribuir al crecimiento de Liferay, añadiendo una nueva funcionalidad a la personalización de la apariencia de las páginas mediante la creación de un nuevo concepto: los "ThemeWebSettings".

Basándose en los "Settings", que es una característica de los temas de apariencia (themes) ya implementada, esta nueva funcionalidad añade la posibilidad de controlar el valor de estos a través del entorno Web.

Constituye, por tanto, una evolución que simplifica y aumenta la personalización de la apariencia de los portales y páginas de Liferay a través de un nuevo apartado de configuración en su entorno Web.

Índice general

1. Introducción **3**

1.1. Liferay 3

1.1.1. Características principales 4

1.1.2. Comunidad Liferay 5

1.1.3. Evolución 7

1.1.4. Arquitectura o patrón de diseño 7

1.1.5. Análisis mediante la herramienta SLOCCount 9

1.2. Experiencias con Liferay 10

1.2.1. Personalización de la apariencia 10

1.2.2. Desarrollo de temas para Liferay. El Plugin-SDK 12

1.2.3. Conclusiones y problemas detectados 14

1.3. Software libre 15

1.3.1. Características generales 15

1.3.2. Gestión de proyectos de software libre 16

2. Objetivos **18**

2.1. Descripción del problema 18

2.2. Estudio de alternativas 19

2.3. Metodología empleada 20

3. Descripción Informática **23**

3.1. Fase 1: Análisis del código fuente e integración en la comunidad de Liferay . . 23

3.1.1. Integración dentro de la comunidad de Liferay 24

3.1.2. Estructuración del código y la Base de Datos 25

3.1.3. Funcionamiento de los Settings 27
3.2. Fase 2: Implementación de una versión funcional 28
3.2.1. Iteración 1 29
3.2.2. Iteración 2 30
3.2.3. Iteración 3 32
3.2.4. Iteración 4 35
3.2.5. Iteración 5 38
3.2.6. Iteración 6 39
3.3. Fase 3: Contribución al proyecto 40

4. Conclusiones **43**
4.1. Aportación de los ThemeWebSettings a Liferay 43
4.2. Dificultades encontradas 45
4.3. Logros alcanzados 46
4.4. Posibles trabajos futuros 47

A. Manual de utilización de los ThemeWebSettings **48**
A.1. Declaración y utilización de los ThemeWebSettings 48
A.2. Personalización de los ThemeWebSettings desde la Web 49

B. Ejemplo de email tras la iteración 3 **51**

Bibliografía **55**

Capítulo 1

Introducción

En este primer capítulo se realizará una introducción a Liferay, plataforma en la que se ha realizado el proyecto. Se expondrán sus características principales y se analizará su potencial en el ámbito de las plataformas para portales Web, gestores CMS y entornos colaborativos.

A continuación se expondrán experiencias desarrolladas con esta tecnología, en concreto en la personalización de la apariencia y en el desarrollo de nuevos temas de apariencias, para terminar con unas conclusiones y una exposición de los problemas detectados y posibles mejoras.

Por último, dado que el proyecto en el que me baso se distribuye con una licencia libre, se realizará una breve introducción al software libre, explicando sus conceptos básicos.

1.1. Liferay

Este apartado pretente presentar el proyecto Liferay. Para ello mostraremos sus características principales, los aspectos fundamentales de su comunidad, la evolución que ha tenido en estos últimos años y la arquitectura en la que se basa. Terminaremos de presentar Liferay con los resultados obtenidos tras la realización de un análisis inicial del código de Liferay. Para este análisis se hizo uso de una herramienta libre [9].

1.1.1. Características principales

Liferay es una plataforma web corporativa basada en Java y de código abierto para la creación de portales Web, gestores CMS y entornos colaborativos.

Dentro de sus carácterísticas, podemos destacar las siguientes [1]:

- Puede ejecutarse sobre la mayoría de servidores de aplicaciones y contenedores de Servlets, Bases de Datos, Sistemas Operativos y sobre 700 combinaciones de implantación.
- Usa Java, J2EE y las últimas tecnologías web 2.0.
- Usa un framework abierto SOA[1].
- Cumplimiento de *JSR-168*, que permite la interoperabilidad de los portlets entre portales web diferentes y *JSR-286*, similar al anterior pero para portlet 2.0.
- Incorpora 60 portlets[2] listos para usar tras su instalación.
- Páginas personalizadas para todos los usuarios.
- Interfaz de usuario AJAX[3].
- Soporte multilenguaje (más de 22 idiomas).
- Sicronización completa con servicios de directorios como LDAP y soporte seguro Single Sign On.
- Autorizaciones granulares basadas en roles.
- Panel de Control: administración centralizada para todo el contenido, usuarios, organizaciones, comunidades, roles, recursos de los servidores; personalización completa con la posibilidad de esconder diferentes partes o añadir otras con nuevos portlets.
- Configuración single-click, drag-and-drop dinámico, capacidad de búsqueda y etiquetado, y trabajo desde el escritorio (WebDAV).

[1] Arquitectura Orientada a Servicios

[2] componentes modulares de las interfaces de usuario gestionadas y visualizadas en un portal web

[3] Asynchronous JavaScript And XML:técnica de desarrollo web que mantiene una comunicación asíncrona con el servidor

- Integra CMS, WCM, Software colaborativo y software social.

En cuanto a la interfaz podemos destacar que es [1]:

- **Intuitiva**: los usuarios pueden arrastrar y pegar portlets para personalizar las preferencias de un usuario o comunidad.
- **Rica**: los usuarios pueden usar tanto los plugins de temas que vienen incorporados en la instalación, como los que desarrolla la comunidad, para cambiar el aspecto del portal sin tener que tocar el código.
- **Amigable**: los miembros de la comunidad pueden tener sus páginas con una URL definida por el usuario.
- **Colaborativa**: los usuarios pueden crear verdaderas comunidades de usuarios a traves de herramientas colaborativas como mi, foros, blogs, wikis, etc.

1.1.2. Comunidad Liferay

Como veremos un poco más adelante, dentro del apartado del Software libre, el concepto de comunidad en un proyecto de este tipo es muy importante.

El objetivo de todo proyecto es crear una comunidad en torno él para que en algún momento el nivel de actividad llegue a ser tal que su desarrollo sea autocatalítico, es decir, que la propia comunidad resuelva las necesidades que se plantea por sí misma [7].

Alrededor de Liferay existe una comunidad en constante crecimiento que cuenta, en el momento de la redacción de este proyecto, con más de 34.000 usuarios registrados.

Dentro de su Web [5] proporcionan una serie de herramientas que posibilitan la colaboración entre sus miembros. Estas son las siguientes:

- **Chat:** permite la comunicación instantánea entre sus miembros conectados.
- **Foros:** divididos en varios idiomas y temáticas principales. Constituyen un elemento esencial en toda comunidad. A través de ellos es posible proponer nuevas funcionalidades, comentar aspectos de la herramienta, así como la resolución de dudas y problemas.

- **Wiki:** con artículos escritos por la propia comunidad y estructurados en diferentes bloques: para iniciarse, para la instalación, para los desarrolladores, etc. Estos permiten, compartir el conocimiento de unos y el aprendizaje de otros.
- **Issues tracking** o seguimiento de problemas: esta herramienta, basada en JIRA[4], permite la comunicación de errores y nuevas funcionalidade así como su gestión.

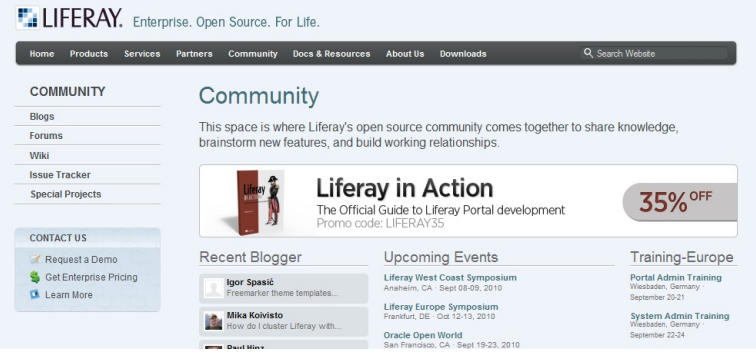

Figura 1.1: Apartado Community de la Web de Liferay

También podemos destacar que, tal y como puede verse en la siguiente figura, ofrece una serie de documentos (entradas de blogs, pdfs, Wikis, etc) sobre distintos temas agrupados en: Getting started, Administration y Development.

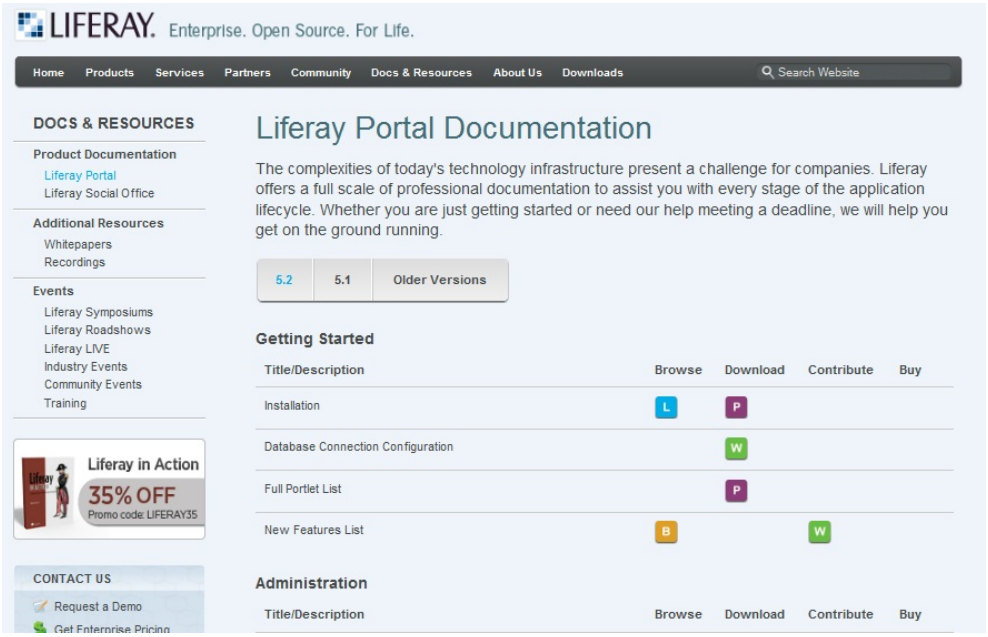

Figura 1.2: Apartado Docs & Resources de la Web de Liferay

[4]JIRA: aplicación web para el seguimiento de errores, de incidentes y para la gestión operativa de proyectos

Constituye un producto con una comunidad consolidada con más de 13.000 desarrolladores, más de 2,7 millones de descargas y 250.000 instalaciones en todo el mundo [5].

1.1.3. Evolución

Curiosamente, Liferay nació en el año 2000 como un proyecto personal de Brian Chan para hacer la web de su iglesia. Después de uno a dos años de desarrollo, Brian lo puso en sourceforge y tuvo un gran éxito. Le empezaron a pedir servicios como si fuera una empresa, por lo que finalmente optó por dejar su trabajo y fundó Liferay Inc. (entonces Liferay LLC).

La empresa fue fundada en Los Angeles y hoy tiene presencia también en América del Sur, Europa y Asia.

Desde entonces su evolución hasta la nueva versión 6 que conocemos hoy en día ha sido enorme.

Podemos destacar algunos datos actuales:

- Mantienen unas 70.000 descargas mensuales, con más de 3 millones en total[5].
- Existen 34.160 usuarios registrados en liferay.com, de los cuales 13.627 han participado alguna vez en los foros[6] (en 2006 eran unos 2.500).
- Cuenta con unos 50 colaboradores activos que envían parches.

Todo esto nos da una idea de la magnitud que tiene el proyecto y la evolución que ha experimentado en pocos años.

1.1.4. Arquitectura o patrón de diseño

Es importante reconocer la estructura en la que se basa el software de Liferay. Para ello veremos qué patrón de diseño sigue.

Un patrón de diseño en ingeniería del software es un modelo formal que es aplicable a diferentes dominios. Existen diferentes problemas que, aunque pertenecen a distintos ámbitos, son semejantes desde el punto de vista de la estructura lógica de la solución. El patrón de diseño es una estructura común que tiene diversas aplicaciones.

[5]Estadísticas generadas por sourceforge

[6]http://www.liferay.com/community/forums/-/message_boards/statistics

El patrón que sigue Liferay es el patrón Modelo Vista Controlador, conocido como patrón MVC. Este patrón divide el proyecto en los siguientes componentes:

- **Modelo**. Es el elemento encargado del manejo de los datos y de la lógica de negocio.
- **Controlador**. Se encarga de redirigir un procesamiento determinado por cada petición recibida. Gestiona la lógica de la aplicación.
- **Vista**. Maneja los objetos gráficos de la interfaz de usuario y se encarga de la lógica de la presentación.

Esta separación implica que una petición de un usuario sea procesada como sigue:

1. El navegador, desde el lado del cliente, envía la petición de una página al controlador del servidor.
2. El controlador recupera los datos necesarios del modelo para responder la petición.
3. El controlador facilita dichos datos a la vista.
4. Se genera la vista correspondiente y se envía al cliente para que se muestre en el navegador.

El proceso se ilustra en la siguiente figura:

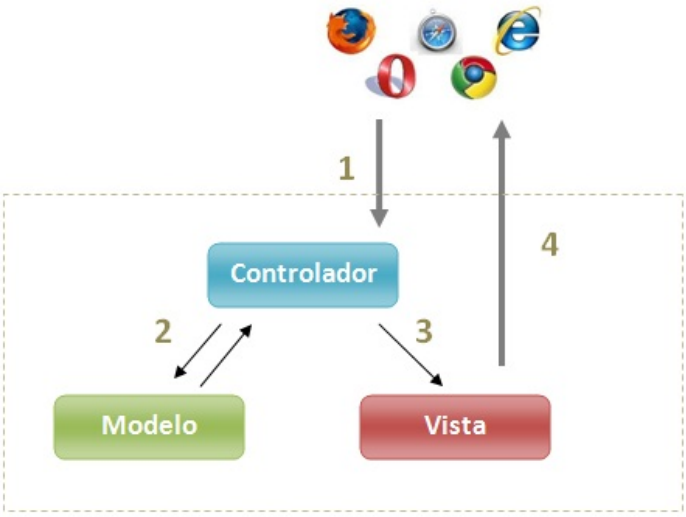

Figura 1.3: Procesando una petición de una página en la arquitectura MVC

Dividir una aplicación software en estos tres componentes implica varias ventajas:

- Mejora la escalabilidad.
- El mantenimiento es más sencillo
- Promueve la reutilización

1.1.5. Análisis mediante la herramienta SLOCCount

Gracias a la utilización de SLOCCount [9], una aplicación libre que permite analizar proyectos software, podemos sacar conclusiones concretas del portal de Liferay.

El programa cuenta las líneas físicas de código, separándolas por lenguajes, y realiza una estimación del coste y del esfuerzo necesario para su desarrollo basándose para ello en el modelo COCOMO (Modelo Constructivo de Costes).

El resultado podemos verlo en la siguiente tabla:

Número total de líneas de código:	**1.795.362**
Lenguajes más utilizados:	Java (92,85%) Jsp (4.46%) Xml (2,65%)
Estimación de años/meses de desarrollo:	5,78 / 69,31
Número de desarrolladores estimados:	90,42
Coste final Estimado:	**$ 70,550,688**

Figura 1.4: Datos obtenidos con SLOCCount

Los datos que se muestran en la tabla anterior son muy significativos. El primero nos da una idea de la magnitud que tiene este proyecto, con más de un millón setecientas mil líneas de código. También podemos constatar que está programado casi en su totalidad en **java** (92,85 %), utilizando **jsp** para la presentación y finalmente **xml** para los ficheros de configuración.

Los siguientes datos también nos dan un contexto en el que ubicar el proyecto. Para realizar un proyecto de este tipo se estima que se necesitarían más 5 años y más de 90 desarrolladores, teniendo un coste final (estimado) de unos $ 70.550.688, alrededor de 55.586.887 euros.

Esta es una de las grandes ventajas del software libre. Aplicaciones de este tipo solo serían accesibles a grandes empresas u organismos, pero en cambio está abierta a todo el mundo, permitiendo su utilización, el estudio de su código e incluso la participación en el proyecto.

Por último, mostramos el detalle del número de líneas de código por directorios, obtenidos de la herramienta anterior:

Líneas de código	Directorio	Nº de líneas de código por lenguaje
759.196	portal-impl	java=734824,xml=23644,sh=555,jsp=89,perl=51,exp=33
747.447	portal-web	java=658193,jsp=80033,xml=9221
253.220	portal-service	java=253149,xml=71
9.970	util-taglib	java=9897,xml=73
8.040	util-java	java=8032,xml=8

Figura 1.5: SLOCCount: Líneas de código por directorios

En esta tabla podemos comprobar cómo el grueso principal del código se concentra en tres directorios:

- portal-impl
- portal-web
- portal-service

Más adelante se explicará qué función tienen cada uno de ellos.

1.2. Experiencias con Liferay

Dentro de este apartado vamos a tratar el aspecto sobre el que va destinado este proyecto, la apariencia. Para ello, mostraremos cómo funciona la personalización de la apariencia dentro de Liferay, qué herramientas tenemos para el desarrollo de nuevos temas y finalmente unas conclusiones con un planteamiento de problemas detectados y posibles soluciones.

1.2.1. Personalización de la apariencia

En la actualidad, dentro de las herramientas para la construcción de entornos web, como Joomla, Drupal, Wordpress, etc., es habitual encontrar la posibilidad de personalizar su apariencia con lo que se denominan *temas* o *skins*.

En la mayoría de los casos, esta personalización de la apariencia consiste únicamente en la aplicación de una serie de estilos CSS.

En Liferay se utilizan, para tal efecto, los *temas de apariencia* o *themes*. Estos permiten, no solo la aplicación de una serie de estilos CSS, sino además personalizar la construcción de las páginas (templates) e incluir código javascript para que sea utilizado por las mismas.

Estos temas de apariencia pueden ser seleccionados desde la Web, a través de la barra superior, botón Administrar - Página, dentro de una pestaña denomina *Apariencia* (*look-and-feel*). Para poder visualizar esta barra superior es preciso haber iniciado sesión con un usuario que tenga los permisos adecuados. Así se mostrará la siguiente página de administración:

Figura 1.6: Ubicación del tema de apariencia

Tal y como puede verse en la imagen anterior, la apariencia se puede configurar a varios niveles: a nivel global y/o a nivel de página.

Esta separación de configuración de la apariencia en dos niveles nos va a permitir configurar un tema de apariencia para todo el sitio Web y dentro de éste establecer un tema de apariencia diferente para una o varias páginas en concreto. Igual ocurre con la personalización de los CSS.

El funcionamiento podría resumirse de la siguiente forma: Si existe un tema de apariencia o un CSS definido a nivel de página: se toma ese valor. En otro caso: se aplica el establecido a nivel de comunidad.

Desde el interfaz Web, por tanto, se podrá seleccionar el tema de apariencia deseado y además introducir sentencias CSS que modifiquen el estilo del tema sin necesidad de volver a general el tema.

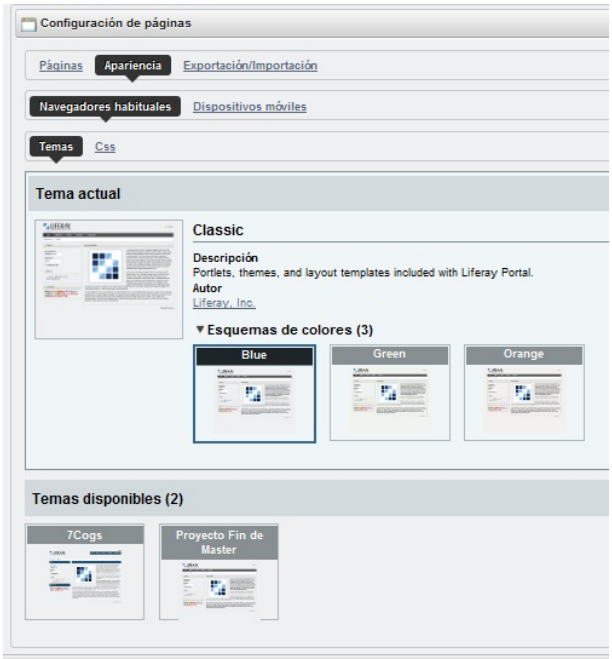

Figura 1.7: Pestaña de Apariencia

También hay que comentar que existe un último nivel en la personalización de la apariencia. Este nivel de personalización consiste en la configuración de la apariencia de cada uno de los portlets contenidos en las páginas.

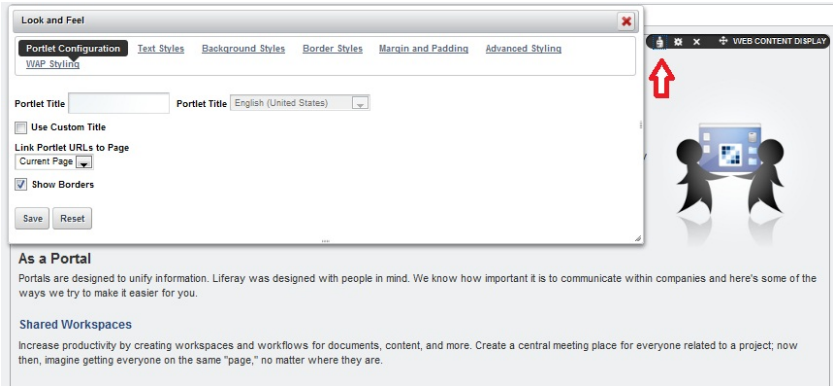

Figura 1.8: Configuración de la Apariencia de los portlets

1.2.2. Desarrollo de temas para Liferay. El Plugin-SDK

Liferay ofrece a los desarrolladores un plugin para la creación de temas, portlets, hooks y, desde la nueva versión 6, para implementar nuevas funcionalidades del portal (ext) [4].

Mediante este plugin, podremos desarrollar nuevos temas de apariencia, desde cero o bien

basándonos en otro tema ya existente. Realmente nunca se desarrolla un tema desde cero, sino que se basa sobre uno que se denomina unstyled.

Veamos cómo es la estructura del tema y qué permite personalizar.

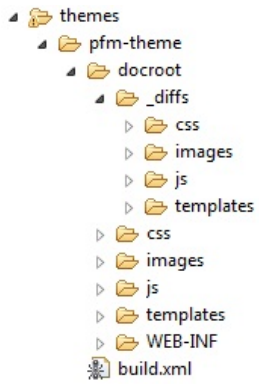

Figura 1.9: Estructura del tema de apariencia

Tal y como se puede observar en la imagen anterior, el esqueleto de cualquier tema de apariencia tiene la siguiente estructuración de carpetas:

- **css:** en esta carpeta estarán todos los archivos necesarios para establecer los estilos del tema de apariencia.
- **js:** contiene el código javascript que queramos utilizar en las páginas que tengan asignado este tema de apariencia.
- **templates:** aquí se encuentran los ficheros que contruyen cada una de las páginas Web. Para ello se hace uso de Velocity [8].
- **images:** Imágenes del tema.
- **_diffs:** Esta carpeta es muy interesante. Si observamos, dentro de ella vuelven a aparecer todas las carpetas anteriores. Ya hemos comentado que es posible crear un tema basado en otro. En Liferay, el desarrollo de temas de apariencia se realiza indicando las diferencias que queremos introducir en ese tema para crear el nuestro personalizado. Realmente nunca se modifican los ficheros contenidos en las carpetas anteriores, sino que para personalizar cualquiera de ellos se copia dentro de esta carpeta diff, en la subcarpeta correspondiente, y ahí se realizan las modificaciones. Al compilar y generar el tema, estas modificaciones se integran para obtener el tema personalizado.

- **WEB-INF:** contiene ficheros de configuración. Entre ellos podemos destacar el *liferay-look-and-feel.xml*, en el que se definen todas las propiedades del tema de apariencia.

1.2.3. Conclusiones y problemas detectados

Antes de la realización de este proyecto, mi experiencia con liferay se ha basado en la creación de un portal para un organismo público. Ello ha consistido por un lado, en la creación de un tema de apariencia personalizado, y por otro en la creación de páginas con los portlets necesarios para construir correctamente el portal.

En esta labor he podido comprobar el enorme potencial que tiene esta herramienta, pero también he echado en falta algunas funcionalidades, entre ellas la que he desarrollado en este proyecto.

A continuación voy a describir algunos de los problemas detectados en la generación del tema de apariencia, explicar cómo los he resuelto y qué funcionalidad debería existir para que pueda realizarse mejor.

Uno de los requisitos que tenía que implementar para la construcción del portal era que la barra de navegación tenía que ser distinta para la página de la portada y para las interiores.

Para resolver esta cuestión se tuvo que recurrir a una funcionalidad existente en los temas de apariencia (que se detalla más adelante), los Settings. Se creó un único tema sobre el que se definieron otros dos, cada uno con un valor de un Setting (que se denominó “navstyle”) distinto (“portada” e “interior”).

Hasta aquí todo iba correctamente. El problema apareció al aumentarse los requerimientos:

- Que algunos elementos, como por ejemplo el breadcrumb o el campo de búsqueda en la cabecera, fueran visibles en unas páginas, mientras que en otras no.
- Mostrar un slogan en distintas posiciones, dependiendo de la página.
- La posibilidad de cambiar el texto del slogan (bien introduciendo texto o cambiando la imagen).

Es evidente que todo puede ser resuelto mediante Settings. El problema reside en que para implementar todo esto tendríamos que generar distintos temas de apariencias (basados en uno solo) jugando con la asignación de valores a los distintos Settings que creemos. A esto se añade

que cada vez que se quiera cambiar el texto del slogan se tendría que volver a modificar y desplegar de nuevo el tema.

El resultado final es una gran cantidad de temas (uno por cada combinación de Settings) y la dependencia hacia el desarrollador del tema de apariencia, que tendrá que ir realizando cambios que podrían ser abordados, con una interfaz web adecuada, por el usuario.

Esto puede unirse a lo siguiente:

Un escenario típico con Liferay es la constitución de portales Web utilizando para ello el concepto de *comunidades*. De esta forma, existe un servidor que alberga más de un portal Web, asignando para cada uno a un administrador de la comunidad.

Este tipo de rol tiene sus inconvenientes, entre ellos la imposibilidad de poder instalar nuevos temas de apariencia, lo cual limita el margen de maniobra a la hora de desarrollar un tema. Cada vez que se concluye la actualización del tema de apariencia hay que solicitar que el administrador del portal lo integre en el servidor.

De esta forma, una mejora interesante sería aumentar el grado de personalización del tema de apariencia a través del entorno web.

Como conclusión, Liferay necesita una nueva funcionalidad: los ThemeWebSettings, que permitan la personalización de los temas de apariencia en tiempo real y a través de un entorno Web.

1.3. Software libre

Dado que el proyecto Liferay es de software libre, resulta conveniente describir qué implica esta afirmación.

A continuación vamos a describir brevemente en qué consiste una aplicación de software libre y cómo se gestiona un proyecto de este tipo.

1.3.1. Características generales

Todo programa que sea considerado software libre debe ofrecer una serie de libertades. Se resumen en [6]:

- Libertad de usar el programa con cualquier fin, sin necesidad de comunicarlo a los

desarrolladores.

- Libertad de estudiar el código fuente del programa y modificarlo adaptándolo a nuestras necesidades, sin necesidad de hacer públicas las modificaciones.
- Libertad de distribuir copias, tanto binarios como código fuente, modificadas o no, gratis o cobrando por su distribución.
- Libertad de modificar el programa y publicar las mejoras para beneficio de la comunidad.

Estas libertades conllevan una serie de factores que están cambiando el desarrollo de software tradicional. Por ejemplo, permiten que se pueda reutilizar gran cantidad de software que se encuentra disponible en repositorios públicos de Internet. Y no sólo eso, ya que también pueden consultarse las listas de correo, los foros y wikis utilizados por los desarrolladores del proyecto, donde puede encontrarse gran cantidad de información muy valiosa. De esta forma, el desarrollo de software se acerca a los procesos que se siguen en la investigación científica en general.

Liferay inicialmente nació como un proyecto privado, y en un momento dado, su desarrollador decidió dar un paso que fue decisivo para el proyecto: su liberación.

Esto implica que Liferay debe cumplir las libertades anteriores, permitiendo tanto el acceso y la modificación del código fuente como la publicación de mejoras para beneficio de la comunidad y por tanto para la mejora del proyecto. Todo esto supuso un crecimiento importante para Liferay.

1.3.2. Gestión de proyectos de software libre

Las libertades que ofrecen los programas libres hacen que a su alrededor se creen comunidades de usuarios y desarrolladores que colaboran en la detección y corrección de errores, en la solicitud y programación de nuevas funcionalidades y en otros muchos aspectos como la traducción de la interfaz o la documentación del proyecto a nuevos idiomas [7].

El objetivo de cualquier nuevo proyecto libre es conseguir una comunidad de usuarios y desarrolladores entorno al mismo que lo ayuden a crecer, a desarrollarse, de manera que el nivel de actividad llegue a ser tal que el desarrollo del proyecto sea autocatalítico, es decir, que la propia comunidad resuelva las necesidades que se plantean por sí misma.

Cómo conseguir esta comunidad no es un problema trivial y, aunque actualmente esta labor se lleva a cabo sin usar procesos ingenieriles, existen algunas buenas maneras de proceder para alcanzar este ansiado éxito.

Tal y como se ha comentado en el apartado de *Comunidad Liferay*, dentro de este capítulo de introducción, Liferay ha conseguido aumentar considerablemente el número de miembros de su comunidad y, aunque existe un equipo de personas trabajando en la resolución de dudas, postear en el blog, etc., cada vez más es la propia comunidad la que se encarga de la comunidad, es decir se está consiguiendo llegar a ser un proyecto autocatalítico.

Para la gestión de proyectos de software libre, también son importantes actitudes como [7]:

- Mantener un sitio Web bien estructurado y actualizado. En otro caso el usuario tendrá la sensación de que se corresponde con un proyecto muerto.
- Mantener mucha documentación sobre el proyecto y con una estructuración clara. Por ejemplo: para el usuario y para el desarrollador.
- Mantener las herramientas necesarias para la resolución de dudas, problemas, etc. y, en general, la interacción de los miembros de la comunidad.
- Contar con un sistema de control de versiones para el software y la documentación del proyecto.
- Que la instalación sea sencilla.
- Promocionar el proyecto para conseguir el *efecto en red*. Para ello podríamos dar de alta el proyecto en el portal Freshmeat, dedicado exclusivamente a informar de las últimas versiones de programas libres. Asi como llegar a sitios de noticias populares, como Barrapunto, Libertonia o Slashdot, acudir a conferencias y charlas de software libre, etc. Con ello nuestro proyecto ganará en audiencia considerablemente.

Realmente, Liferay mantiene las actitudes anteriormente descritas y sigue en constante evolución, por lo que mantiene una gestión activa del proyecto.

Capítulo 2

Objetivos

Una vez presentado el marco general en el que se encuadra este proyecto y la tecnología utilizada, en este capítulo abordaremos los objetivos que se han perseguido durante la elaboración del mismo. Se realizará una descripción detallada del problema, se resumirá, posteriormente, el estudio de las distintas alternativas planteadas para su resolución y se mostrará la metodología seguida en su desarrollo.

Los objetivos principales del proyecto son:

- Integrarme dentro de la comunidad de Liferay.
- El desarrollo de una nueva funcionalidad que permita la personalización de los temas de apariencia a través de propiedades que puedan ser modificadas en tiempo real y a través de la Web.
- Contribuir esta nueva funcionalidad a la comunidad de Liferay.

2.1. Descripción del problema

Como se ha visto en el capítulo de introducción, en el desarrollo de temas de apariencia existe la posibilidad de utilizar una característica muy interesante, que son los Settings, que actúan a modo de variables del tema. Con ellos es posible personalizar un tema de apariencia de forma que en función de los valores asignados a estos se construyan las páginas de una forma o de otra.

Veámoslo con un ejemplo: Imaginemos que tenemos que desarrollar dos temas de apariencia para una organización. Nos indican que ambas han de ser idénticas a excepción de la cabecera, que será distinta en cada caso: una para la portada y otra para el resto de páginas. En este caso, la utilización de los Settings en la construcción del tema de apariencia permitiría crear un único tema que, en función de un Setting que podríamos denominar “cabecera”, construyera la página utilizando una u otra cabecera, dependiendo del valor del Setting definido (que podrían ser: “portada” e “interiores”).

Finalmente tendremos que declarar dos temas de apariencia. Ambos se basarán en el mismo pero cada uno asignará un valor distinto al Setting declarado anteriormente (“cabecera“).

La sensación para el usuario es que existen dos temas de apariencia distintos, aunque realmente tan solo exista uno.

Esta utilidad resulta interesante, dado que no es necesario crear y mantener dos temas de apariencia casi idénticos, sino tan solo uno con los settings necesarios.

El problema que presenta el desarrollo de temas de apariencia con estos Settings radica en que los valores de los mismos deben establecerse en el momento de generar el tema y no pueden ser establecidos en tiempo real, a través de la web. De hecho es necesario definir un tema para cada conjunto de asignación de settings.

Sobre este escenario, supondría una mejora considerable el desarrollo de una nueva funcionalidad que permita la declaración y asignación de valores a los settings, en tiempo real a través de la web.

Siguiendo con el ejemplo anterior y con esta nueva funcionalidad, tan solo se tendría que desarrollar un tema donde el valor del Setting ”cabecera“ pueda seleccionarse por el administrador a través del interfaz web.

De forma que el administrador del portal podrá, sin tener que volver a modificar y desplegar el tema y seleccionando los valores correspondientes a través de la interfaz Web, personalizar al máximo la apariencia de su portal.

2.2. Estudio de alternativas

Un requisito indispensable con respecto a las contribuciones de nuevas funcionalidades, es que el desarrollador compruebe que no existe nada realizado o en vías de desarrollo igual o

similar a lo que va a realizar.

Estas nuevas funcionalidades, para que sean integradas en Liferay, deben ser supervisadas y aprobadas por los miembros responsables del proyecto.

Por ello, Liferay cuenta con una herramienta web (Liferay Issues, basada en JIRA), que permite tanto la notificación de bugs, como de nuevas funcionalidades que se desarrollen.

Tras comprobar que no existía nada, me puse en contacto con el Director General de Liferay España y Portugal, Jorge Ferrer, quien me constató que podría ser una buena contribución al proyecto.

Por tanto, el estudio de alternativas se ha basado en la comprobación de que no existía nada parecido a lo que iba desarrollar, mirando para ello en la herramienta anterior, en los foros y finalmente poniéndome en contacto con Jorge Ferrer.

2.3. Metodología empleada

El objetivo de aplicar una metodología o modelo de desarrollo a la construcción de una aplicación software es convertir el desarrollo en un proceso formal, con resultados predecibles, que permitan obtener un producto final de alta calidad, que satisfaga las necesidades y los requisitos identificados.

La metodología empleada para la realización de este proyecto se corresponde con el **modelo en espiral**. De esta forma se han establecido una serie iteraciones sobre el diseño e implementación.

La elección de este modelo viene motivada en la estructura sobre la que se implementa el proyecto, dividiendo el objetivo general en otros más pequeñas. De esta forma, con cada iteración se van consiguiendo subobjetivos sobre los que se irán basando los siguientes, incrementando la complejidad hasta llegar al resultado contenido en este documento.

En la siguiente figura pueden observarse las etapas contenidas en cada iteración: Determinar Objetivos, Análisis de Riesgos, Desarrollar y probar y Planificación.

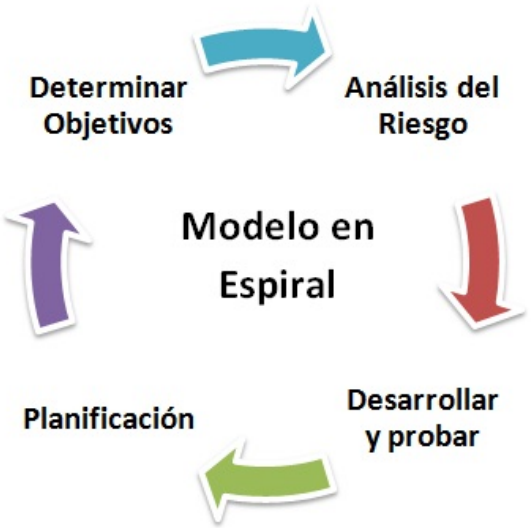

Figura 2.1: El modelo iterativo o espiral

En cada iteración, por tanto, realizaremos los siguientes pasos:

1. **Determinar Objetivos:** se tienen que definir los objetivos que se desean cumplir en dicha iteración. Estos, al basarse en una iteración anterior, cada vez serán mayores hasta llegar al objetivo final.

2. **Análisis de Riesgos:** es necesario, una vez definidos los objetivos, analizar la mejor forma de implementarlo en el código, minimizando el riesgo.

3. **Desarrollar y probar:** tras el análisis anterior, ya solo queda pasar a la implementación y la realización de pruebas posteriores. El final de esta fase será una implementación probada y ajustada al análisis anterior, que cumpla con todos los objetivos propuestos.

4. **Planificación:** se pasa a la planificación de la siguiente iteración.

La ventaja principal de este modelo es que podemos dividir el objetivo final en pequeños subobjetivos que pueden ser desarrollados en cada iteración. De esta forma, en cada una de ellas, tan solo tendremos que preocuparnos de una parte del problema general. Y hasta que no se verifica su correcto funcionamiento, no se pasa a la siguiente.

Por ello, tras cada iteración obtendremos un prototipo completamente funcional que sirve de base para el siguiente.

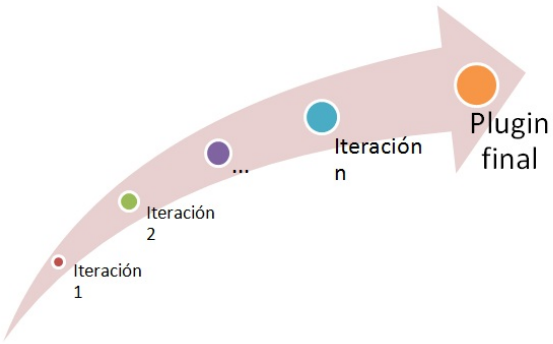

Figura 2.2: Iteraciones

Además es un modelo que ofrece flexibilidad en cuanto al cambio de requisitos iniciales, aspecto que hemos necesitado, dado que el proyecto ha ido cambiando conforme se ha ido madurando cada una de las iteraciones.

En el siguiente capítulo se definen las distintas iteraciones generadas.

Capítulo 3

Descripción Informática

En este capítulo se pretende describir, de una forma más técnica, cuál ha sido la evolución que ha tenido este proyecto.

Para ello, se ha realizado una división del ciclo de vida del proyecto en las siguientes etapas:

- Fase 1: Análisis del código fuente e integración en la comunidad de Liferay.
- Fase 2: Implementación de una versión funcional.
- Fase 3: Contribución al proyecto.

3.1. Fase 1: Análisis del código fuente e integración en la comunidad de Liferay

Esta primera fase se dedicó a la realización de un estudio del proyecto Liferay así como de su comunidad.

Inicialmente veremos cómo realicé mi integración dentro de la comunidad y cuáles son las vías de comunicación con los demás miembros, los procedimientos para el aprendizaje y las posibles contribuciones que existen al proyecto.

A continuación se mostrará una breve descripción acerca de la estructuración del código fuente de Liferay, así como de la base de datos.

Finalmente se describe el funcionamiento de una de las funcionalidades de los temas de apariencia de Liferay, los Settings.

Aunque no se incluye en ningún subapartado, podemos ubicar dentro de esta primera fase la preparación del entorno de trabajo, esto es, la instalación y configuración de las distintas herramientas necesarias para el desarrollo de temas y nuevas funcionalidades de Liferay, como Eclipse, Ant, Tomcat, Mysql, Toad, plugin-sdk, etc.

3.1.1. Integración dentro de la comunidad de Liferay

El primer objetivo planteado dentro de mi proyecto era integrarme dentro de la comunidad de Liferay.

Para conseguirlo realicé lo siguiente:

- Registrarme como usuario dentro de la página oficial del proyecto[1].
- Hacer intervenciones en el foro.
- Mantener una reunión y conversaciones, vía correos electrónicos, con el director general de Liferay.
- Darme de alta en la herramienta Issues tracking, descrita en la introducción, para proponer y gestionar la nueva funcionalidad.

De los pasos anteriores destacaría la reunión mantenida con Jorge Ferrer, Director General de Liferay España y Portugal, en su sede de Madrid. Las contribuciones en este tipo de proyectos se suelen realizar exclusivamente a través de Internet. Por ello supuso un paso importante para mi proyecto.

En dicha reunión acordamos qué tipo de contribución podría ser buena para la comunidad y Jorge Ferrer aceptó ser co-tutor de mi proyecto.

Esto supuso un cambio importante, dado que a raíz de dicha reunión hemos ido intercambiando correos electrónicos con la finalidad de ir afinando el resultado hasta obtener lo descrito en esta memoria.

Por tanto, el objetivo de integrarme dento de la comunidad de Liferay iba por buen camino. El proyecto a desarrollar había sido consensuado con una figura importante dentro de Liferay y además iba a integrarse en las futuras versiones del proyecto. Lo cual suponía una motivación importante.

[1] http://www.liferay.com

Con respecto a la vías de comunicación, dada la separación geográfica y la incompatibilidad de horarios laborales, optamos por utilizar los correos electrónicos. Además se perseguía el objetivo de presentar a la comunidad una nueva funcionalidad madurada. En caso contrario, por norma general, no solía tener mucho éxito.

Por último, comentar que en el transcurso del desarrollo de este proyecto, fui avisado por Jorge Ferrer de que en foro de Liferay había surgido una propuesta[2] relacionada con la nueva funcionalidad que estaba implementando. Era el momento, pues, de explicar lo que estaba desarrollando. Hubieron algunos más interesados en ver cómo se resolvía este tema.

Aunque lo que se proponía en el foro iba relacionado con lo que estaba desarrollando, iba más allá de los objetivos propuestos para este proyecto. Por ello lo he incluido dentro del apartado *Posibles trabajos futuros*, en el capítulo de Conclusiones.

3.1.2. Estructuración del código y la Base de Datos

En la siguiente figura [3] se muestra la arquitectura de capas de Liferay:

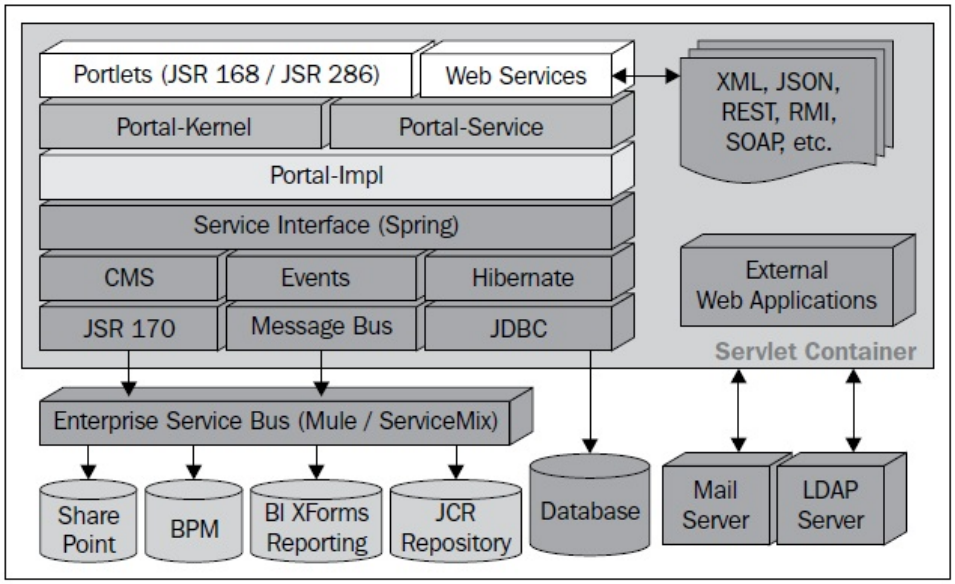

Figura 3.1: Arquitectura de Liferay

[2] http://www.liferay.com/community/forums/-/message_boards/message/4746928

[3] Gráfico tomado del libro de la bibliografia [1] y correspondiente a la versión 5.2 de Liferay. A partir de la versión 6 se ha unido la capa Portal-Kernel a la Portal-Service, por ello tan solo tendremos una capa encima de la Portal-Impl.

De forma muy resumida, puede verse que el acceso a los datos por parte de los portlets y Web Services se realiza a través del Portal-Service. Este a su vez se comunica con el Portal-Impl, que será el encargado de buscar los datos.

Tal y como se mostró en la introducción, el código de Liferay se estructura principalmente en tres directorios:

- **portal-impl**: Contiene todo el código fuente de las clases de objetos de Liferay y los ficheros de configuración (a excepción de los relacionados con el apartado web).
- **portal-service**: Contiene las declaraciones de las funciones y procedimientos del portal-impl para que puedan ser utilizadas por los portlets y Web Services.
- **portal-web**: Contiene los JSPs, HTMLs, imágenes, y todo lo que relacionado con el entorno Web del portal.

Además podemos destacar otros como:

- **portal-lib**: Contiene todas las librerías necesarias para el funcionamiento de Liferay.
- **portal-sql**: Contiene todos los scripts de bases de datos

Con respecto a la **base de datos**, la última versión 6 de Liferay cuenta con un total de 183 tablas. Entre ellas se encuentran las dos que vamos a utilizar dentro del proyecto:

- **layout**: Tabla que contiene información de cada página. Utilizaremos en concreto el campo typeSettings para almacenar las configuraciones asignadas a nivel de página.
- **layoutset**: Tabla que contiene información de un grupo de páginas. La utilizaremos para almacenar las configuraciones asignadas a nivel de comunidad u organizacion.

Esta estructura de base de datos es creada automáticamente, en el caso de que no lo esté, la primera vez que se inicie Liferay. Para ello se utilizarán los scripts de bases de datos mencionados anteriormente.

3.1.3. Funcionamiento de los Settings

En este apartado se describe, desde un punto de vista más técnico, una de las funcionalidades existentes en la creación de temas de apariencia de Liferay, ya comentada anteriormente: los Settings [4]. Ello consiste en la posibilidad de declarar propiedades que van a personalizar el tema.

Estos settings se definen en un archivo de nominado *liferay-look-and-feel.xml*, contenido en el directorio /docroot/WEB-INF del tema. La estructura a utilizar dentro del fichero xml es la siguiente:

```
 <settings>
<setting key="my-setting" value="el-valor-deseado">
...
</settings>
```

De esta forma, dentro de las plantillas del tema podremos acceder a las propiedades declaradas en este fichero utilizando la siguiente instrucción:

```
$theme.getSetting("my-setting")
```

Un ejemplo de la utilidad de los Settings podría ser la siguiente:

Imaginemos que tenemos que desarrollar dos temas de apariencia que serán exactamente iguales a excepción de la cabecera, que será distinta en cada caso. Es evidente que la solución idónea sería crear un único tema que pueda ser personalizado en cada caso, en vez de tener el tema por duplicado. La mayor ventaja de tener un único tema personalizable radica en la facilidad de mantenimiento del mismo. Ante una modificación no es necesario cambiar dos temas sino uno. Quizás con dos temas no se vea demasiado ventajoso utilizar los Settings pero, ¿qué ocurriría si se tuvieran que realizar cinco? ¿o bien ocho?.

Continuando con el ejemplo anterior, desarrollaríamos un único tema con una setting que podría denominarse *"tipo-de-cabecera"*. De esta forma en la plantilla del tema, en el fichero *portal_normal.vm* realizaremos lo siguiente:

```
if ($theme.getSetting("tipo-de-cabecera") == "portada" ) {
#parse("$full_templates_path/cabecera_portada.vm")
```

```
} else {
#parse("$full_templates_path/cabecera_interior.vm")
}
```

Igualmente será necesaria la configuración del setting en el fichero *liferay-look-and-feel.xml*, de forma que quedan definidos los dos temas que se corresponderán con el que hemos denominado TFM, teniendo cada uno un valor distinto en el setting "tipo-de-cabecera". Quedaría de la siguiente forma:

```
<theme id="tema_portada" name="tema_portada">
<root-path>/html/themes/TFM</root-path>
...
<settings>
<setting key="tipo-de-cabecera" value="portada" />
</settings>
...
</theme>
<theme id="tema_interiores" name=\tema_interiores">
<root-path>/html/themes/TFM</root-path>
...
<settings>
<setting key="tipo-de-cabecera" value="interiores" />
</settings>
...
</theme>
```

Esto hace que se generen dos temas de apariencia que estarán basados en uno pero con la cabecera distinta en cada caso.

3.2. Fase 2: Implementación de una versión funcional

Realmente la fase anterior no termina en este punto, dado que como es habitual, con cada nuevo requisito a implementar se ha necesitado invertir más tiempo al estudio del código y una

interacción más frecuente con la comunidad.

A continuación se pasan a describir cada una de las iteraciones generadas y los objetivos que se han ido cumpliendo para llegar a la última versión funcional.

Es importante destacar que en estas iteraciones tan solo mostramos el aspecto técnico de la programación del proyecto. Evidentemente, un proyecto de este tipo tiene un esfuerzo importante que no consiste solo en la programación.

3.2.1. Iteración 1

En esta primera iteración se plantearon los siguientes **objetivos**:

1. Construcción de un tema de apariencia nuevo para las pruebas que muestre el valor de una propiedad en la cabecera.

2. Estudiar en qué campo de la base de datos se deben almacenar los valores de las propiedades.

3. Modificación mínima del código de liferay para mostrar, permitir modificar y almacenar el valor de una propiedad.

El primer objetivo fue uno de los más sencillos en conseguir. Gracias a la extensa documentación que existe en la comunidad de Liferay, pude desarrollar un tema de apariencia nuevo de forma rápida.

Tras estudiar detenidamente el código, comprobé que dentro del apartado *Configuración de páginas* se realizaba algo similar a lo que necesitaba. Se mostraba el valor de tres propiedades (javascript-1, javascript-2 y javascript-3) y se permitía su modificación, tras lo cual se almacenaba en la base de datos, dentro de un campo denominado *typesettings* de la tabla *layout*.

Eso era justo lo que necesitaba, así que busqué el qué parte del código se implementaba esa página (en el fichero */portal-web/docroot/html/portal/layout/edit/common.jspf*) y lo modifiqué para añadir una nueva propiedad. Con ello conseguí aprovechar el código existente y gestionar, a través de la web, el valor de una nueva propiedad.

Lo habíamos conseguido. A través de la *Configuración de páginas* podíamos modificar el valor de una nueva propiedad, quedando registrado en la base de datos.

Tan solo quedaba modificar el tema creado anteriormente para que mostrase el contenido de la variable en la cabecera. Para ello había que modificar, dentro de la carpeta *templates*, el fichero *portal_normal.vm*. De esta forma, para recuperar el valor de la base de datos había que utilizar la siguiente instrucción:

```
$layout.getTypeSettingsProperies().getProperty("propiedad")
```

Realmente, esta primera iteración no era capaz de tomar los settings declarados en el tema, sino las propiedades que había programado de forma estática en la página de configuración, pero supuso un logro muy importante y una buena base para las siguientes iteraciones.

3.2.2. Iteración 2

Basándonos en la anterior iteración, se plantearon los siguientes **objetivos**:

1. Incluir la declaración de Settings en el tema creado en la iteración 1.
2. Modificar el funcionamiento de liferay para que lea y almacene correctamente estos Settings.
3. Modificar el funcionamiento de liferay para que se generen automáticamente estas propiedades en la web de configuración.

Para el primer objetivo decidí definir todos los settings que quisiera mostrar en la web en uno solo. A este setting lo denominé "custom-fields".

De esta forma, dentro del fichero liferay-look-and-feel.xml del tema creado en la iteración anterior, inserté la siguiente declaración del setting:

```
<theme id="TFM" name="Trabajo Fin de Máster">
<root-path>/html/themes/TFM</root-path>
...
<settings>
<setting key="custom-fields" value="cabecera;pie-de-pagina" />
</settings>
```

```
...
</theme>
```

El funcionamiento de este setting es distinto del resto. La diferencia está en el campo value, que consiste en una lista de nombres separados por ";". Cada uno de esos nombres se corresponderá con una propiedad. Por tanto en el código anterior existirán dos propiedades: cabecera y pie-de-pagina.

Dado que esto es nuevo para liferay, había que modificar su funcionamiento para que interpretara correctamente ese setting especial, de forma que para cada elemento de la lista introducida en el campo value se creara una nueva propiedad.

Una decisión importante fue la de almacenar estas propiedades de manera independiente al resto de Settings. Ello implicaba añadir a la clase Theme una variable, denominada *_customfields*, y los métodos necesarios para su manipulación (de forma similar a lo existente para los Settings):

```
public Properties getCustomFields();
public String getCustomField(String key);
public void setCustomField(String key, String value);

private Properties _customFields = new Properties();
```

A continuación, modifiqué la función *_readThemes* del fichero */portal-impl/src/com/liferay/portal/service/impl/ThemeLocalServiceImpl.java*, que es la encargada de cargar los datos del fichero *liferay-look-and-feel.xml* del tema, para que leyera y almacenara cada una de las propiedades declaradas en el setting *custom-fields*.

```
String key = settingEl.attributeValue("key");
String value = settingEl.attributeValue("value");

themeModel.setSetting(key, value);
if (key.equals("custom-fields")){
  StringTokenizer tokens=new StringTokenizer(value, ";");
```

```
  while(tokens.hasMoreTokens()){
    themeModel.setCustomField(tokens.nextToken(), "");
  }
}
else{
  themeModel.setSetting(key, value);
}
```

Finalmente había que modificar el fichero */portal-web/docroot/html/portal/layout/ edit/common.jspf* para que recorriera los customFields del tema y los mostrara dinámicamente. A continuación se muestra el código:

```
<%
  for (Enumeration e = selLayout.getTheme().getCustomFields().
keys(); e.hasMoreElements();) {
    key = (String)e.nextElement();
%>
    <textarea class="lfr-textarea" id='<%= portletDisplay.
getNamespace() + "CustomField_" + key %>' name='<%=
"TypeSettingsProperties(" + key + ")"%>' wrap="soft"
style="display: none; width: 300px;"> <bean:write
name="SEL_LAYOUT" property='<%= "typeSettingsProperties(" +
key + ")"%>' /></textarea>
<%
  }
%>
```

El resultado de esta iteración es un prototipo funcional que permite la declaración de una serie de propiedades, que hemos denominado customfields, y su gestión a través de la web.

Parece que se acerca al objetivo final, pero aún nos queda mucho camino.

3.2.3. Iteración 3

Se plantean los siguientes **objetivos**:

1. Insertar editores Wysiwyg asociados a los campos de las propiedades que facilite la introducción de código html.

2. Definir automáticamente las propiedades como variables de velocity para simplificar su utilización en los templates de los temas de apariencia.

Esta iteración persigue la simplicidad para los desarrolladores de temas de apariencia al utilizar esta nueva funcionalidad, así como para los usuarios, poniendo a su disposición un editor que permite generar código html sin necesidad de conocimientos de este lenguaje.

Para el primer objetivo, investigué cómo se utilizaban los editores en los blogs. Una vez indentificado y entendido el código lo apliqué a nuestras propiedades.

La dificultad estaba en que se tendrían que visualizar tantos editores como propiedades definidas en el tema, lo cual creaba un aspecto incómodo en la web. Además, la inserción de más de un editor en la misma página daba algún que otro problema.

Para solucionarlo creé una lista desplegable, cuyos valores se correspondían con los nombres de las propiedades definidas en el tema, y un único editor enlazado a la lista. De esta forma, al seleccionar una propiedad en la lista se cargaba su valor en el editor. A continuación se muestra el código que crea la lista dentro del fichero */portal-web/docroot/html/portal/layout/edit/common.jspf*:

```
<select id="<portlet:namespace />CustomFields"
  name="<portlet:namespace />CustomFields"
  onChange="<portlet:namespace />updateCustomFieldEditor();">
  <%
    String key = new String();
    for (Enumeration e = selLayout.getTheme().getCustomFields().keys();
e.hasMoreElements();) {
      key = (String)e.nextElement();
      CustomValue = (String)selLayout.getTypeSettingsProperties().
getProperty(key);
  %>
<option value="<%= CustomValue %>"><%= key%></option>
  <%
```

```
    }
  %>
</select>
```

Con respecto al segundo objetivo, modifiqué directamente el fichero init.vm del tema unstyled. Este tema viene por defecto en Liferay y se utiliza para crear nuevos temas desde cero.

Más adelante descubrí que ese fichero se generaba automáticamente y por tanto las modificaciones introducidas se perdían.

Finalmente tuve que modificar el procedimiento insertVariables del fichero /portal-impl/src/com/liferay/portal/velocity/VelocityVariables.java para incluir el siguiente código:

```
Properties customFields = new Properties();
customFields = theme.getCustomFields();

for (Enumeration e = customFields.keys(); e.hasMoreElements();)
{
            String key = (String)e.nextElement();
            velocityContext.put(key,
  layout.getTypeSettingsProperties().getProperty(key));
}
```

De ahora en adelante, el desarrollador del tema de apariencia podrá utilizar en los templates las propiedades definidas en el tema a través de su nombre (anteponiendo el carácter $).

El resultado de esta iteración es una gestión web de las propiedades más cómoda y simple para el usuario, además de facilitar a los desarrolladores de temas de apariencia la utilización de las propiedades definidas.

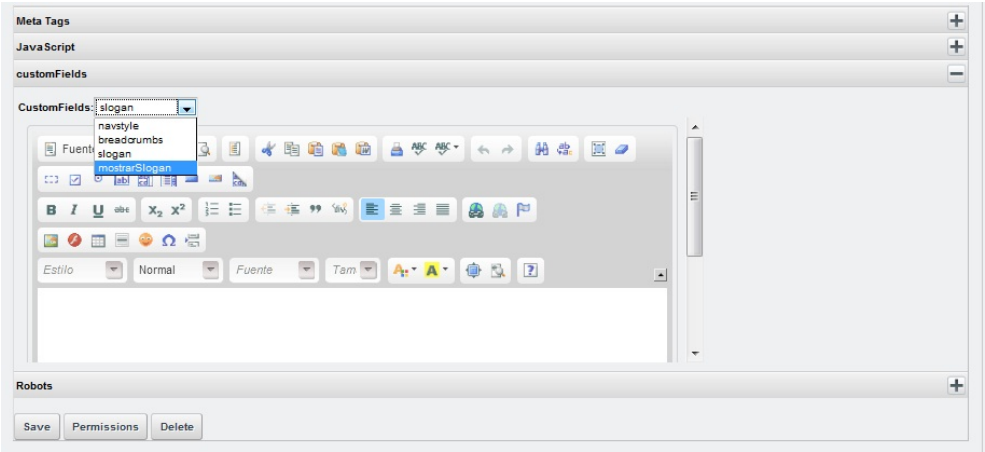

Figura 3.2: Resultado Iteración 3

Esta iteración 3 termina con la entrega de un parche que contiene el código de las diferencias entre el ultimo existente en el repositorio y la nueva funcionalidad.

3.2.4. Iteración 4

Se plantean los siguientes **objetivos**:

1. Ubicar correctamente en la web esta funcionalidad.
2. Dar coherencia a los nombres utilizados hasta ahora.
3. Actualizar a la nueva version 6 de Liferay.

Esta fue, sin lugar a dudas, la iteración más compleja y costosa de todo mi proyecto. En ella se replantean los objetivos de iteraciones anteriores y, en algunos casos, se da marcha atrás.

Tras el análisis, por parte de mis tutores, del parche generado en la iteración anterior, se tomaron algunas decisiones que modificaban sustancialmente el proyecto.

La primera decisión era la reubicación de esta funcionalidad dentro de la web de configuración de páginas. No tenía sentido mantenerlo a nivel de página. Por ello estudié elementos similares, como el que permite la selección de los temas de apariencia o bien la definición de nuevos estilos CSS. Ambos elementos se encuentran dentro de una pestaña denominada Apariencia (look-and-feel). Dado que esta nueva funcionalidad atañe a la configuración de los temas de apariencia, está claro que su ubicación correcta está dentro de esta pestaña. Por ello vamos a crear una nueva pestaña al lado de la denominada *CSS*.

Como se vió en la introducción, esta pestaña *Apariencia* se encuentra en dos niveles, para cada página y para el portal. En función de dónde se defina el tema y/o las sentencias CSS, afectará a una página en concreto o bien a todo el portal.

Hasta ahora, en las iteraciones anteriores habíamos conseguido definir settings para los temas de apariencia modificables por la web, pero solo a nivel de página. En esta iteración tenemos que añadir la posibilidad de que pueda ser asignado a nivel de portal.

Internamente, la diferencia entre estos dos niveles está en la tabla en la que se almacenan las propiedades y la gestión que se realiza para asignar unas u otras: primero se miran a nivel de página y si no hay ninguna apariencia definida para ella se toma la del portal.

En el caso del nivel de página, las propiedades se almacenan en el campo *typeSettings* de la tabla *layout*, mientras que a nivel de portal se tendrían que almacenar en la tabla *layoutset*. El problema estaba en que no existía ningún campo para almacenar los settings en esta tabla. Esto fue solucionado por el equipo de Liferay España y Portugal, quienes incluyeron el campo *setting_* y los procedimientos adecuados para su manejo.

Por tanto, ya teníamos claro qué es lo que queríamos conseguir, ahora nos quedaba implementarlo.

Como ya habíamos realizado en iteraciones anteriores, buscamos dónde se encontraba la codificación de la Apariencia y realizamos la modificación para incluir una nueva pestaña que contuviera la funcionalidad de los ThemeWebSettings.

En concreto la página de configuración se construye mediante el fichero */portal-web/docroot/ html/portlet/communities/edit_pages.jsp*, desde el que se llama al *edit_pages_look_and_feel.jsp*, ubicado junto al anterior.

En el primero es dónde se define la pestaña de *Apariencia* y en el segundo se desarrolla el contenido de ésta. Es en este último dónde se codifican las pestañas *Temas* y *Css*, y por tanto tendremos que modificarlo para ubicar la nueva pestaña.

Tras la codificacion de esta pestaña, este es el aspecto conseguido para un tema con distintos ThemeWebSettings configurados:

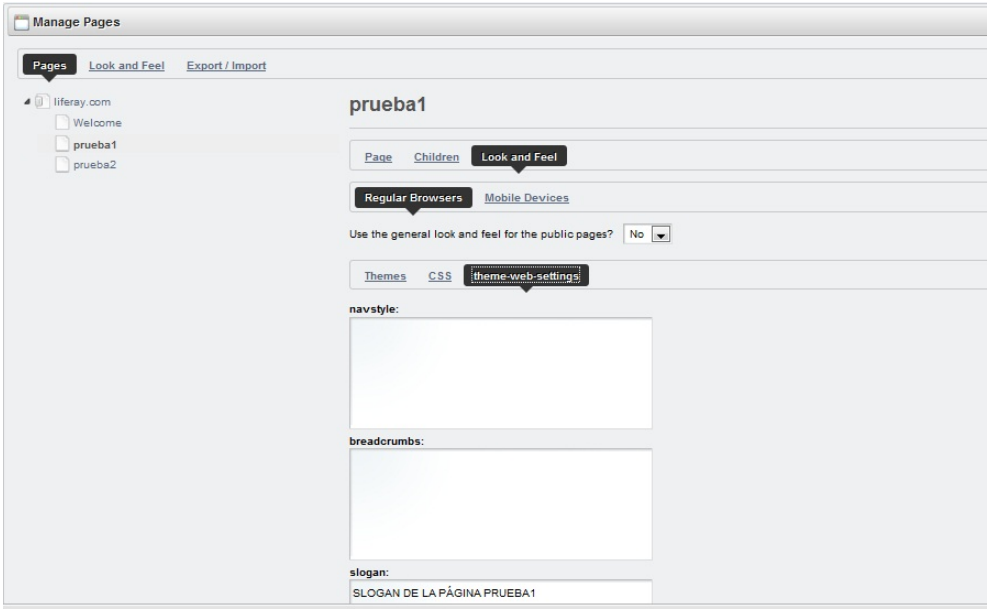

Figura 3.3: Resultado Iteración 4

Por último hubo que configurar una nueva acción para que, al pulsar el botón *salvar* desencadenara la lectura de los valores introducidos a través de la web y su almacenamiento en la tabla correcta.

Para ello se creó un nuevo procedimiento dentro del fichero *edit_pages.jsp* y una nueva acción dentro de */portal-impl/src/com/liferay/portlet/communities/action/EditPagesAction.java* que lo implementara.

De igual forma hubo que modificar la declaración automática de las variables de velocity para que al obtener los valores de los ThemeWebSettings se determinara cuando se debía leer de la tabla layout (página) o bien de layoutset (portal).

Con respecto al segundo objetivo, se tomó la siguiente decisión:

La denominación "custom-fields"pasa a ser ThemeWebSettings, dado que el concepto anterior se refería a otra funcionalidad dentro de liferay y por ello crearía confusión. Por fin teníamos un nombre para esta nueva funcionalidad. A partir de ahora podemos hablar de ThemeWebSettings, en vez de settings o propiedades.

Finalmente, dada la proximidad del lanzamiento de la nueva versión 6 de Liferay, no tenía sentido sacar una funcionalidad para una versión anterior. Por ello, en esta iteración rehicimos todas las modificaciones acumuladas hasta el momento sobre el código de esta nueva versión.

3.2.5. Iteración 5

Se plantean el siguiente **objetivo**:

1. Definir dos tipos de ThemeWebSettings: uno con valores cerrados y otro abierto a la introducción de cualquier texto.

Esta iteración va a constituir una mejora importante.

Las soluciones que aporta la utilización de los ThemeWebSettings pueden ser múltiples. Pero podríamos hacer una clasificación en dos tipos:

1. ThemeWebSettings que deben tomar valores entre algunos predefinidos. Por ejemplo: Si/No, portada/interiores, etc.

2. ThemeWebSettings que pueden tomar cualquier valor. Estos son los que hemos tenido hasta ahora.

De esta forma, sería interesante poder distinguir estos dos tipos de ThemeWebSettings. En el primer caso, se deberá mostrar en la web una lista desplegable con los posibles valores, mientras que en el segundo es necesario dejarlo abierto, utilizando para ello un textarea.

Figura 3.4: Distintos tipos de ThemeWebSettings

Por tanto, para conseguir el objetivo necesitamos poder distinguir de alguna forma la definición de los dos tipos de ThemeWebSettings.

La solución adoptada es la siguiente: se modifica el significado de los valores establecidos a los ThemeWebSettings a través del campo “value”de cada setting del fichero liferay-look-and-feel.xml del tema de apariencia. De forma que si el valor es nulo, en la web se mostraría como un textarea, mientras que si el valor se corresponde con una lista de opciones separadas por “;”se mostraría una lista de valores con las opciones introducidas.

Esta decisión establece una diferencia entre los settings y los ThemeWebSettings, dado que el valor establecido en el value tienen distintas funciones. En los Settings establecen el valor de la propiedad, mientras que en los ThemeWebSettings definen el tipo (cerrado o abierto) y, en su caso, los posibles valores de la propiedad.

Por ello se opta por separar la definición de los ThemeWebSettings a un bloque nuevo, dentro del fichero liferay-look-and-feel.xml, denominado *web-settings*. La estructura quedaría se la siguiente forma:

```
<theme id="id-el-tema" name="nombre-el-tema">
  <settings>
    <setting key="setting1" value="valor">
  </settings>
  <web-settings>
    <setting key="Websetting1" value="">
    <setting key="WebSetting2" value="option1;option2;option3">
    ...
  </web-settings>
</theme>
```

3.2.6. Iteración 6

Esta iteración tiene como **objetivo** la resolución de problemas encontrados y adición de mejoras.

Por ello, se solucionan los siguientes errores:

- Corrección del comportamiento anómalo con la opción staging.
- Corrección del comportamiento al establecer la opción “Use the general look and feel for the public (private) pages“ a ”yes“ en la configuración de la apariencia de cualquier

página. En este caso han de tomarse las propiedades establecidas para el portal en vez de las de la página.

Y finalmente se introducen las siguientes mejoras en el diseño:

- Se oculta la pestaña creada "ThemeWebSettings" en el caso de que el tema no tenga definido ningún ThemeWebSetting.
- Se configura el nombre de la pestaña para que pueda ser visualizado en inglés o en español.
- Se añade la "opción por defecto" a los WebSettings con valores cerrados. Esto es, en el caso de que exista una ThemeWebSetting con valores cerrados, si aún no se le ha asignado ningún valor (a nivel de portal o bien de página), es decir no existe en la base de datos ningún valor para ella, se establece con la primera opción que se haya introducido en el campo "value" al definirla.

Al finalizar la corrección de errores y la introducción de mejoras, se volvió a generar un parche con las modificaciones a partir de la última versión existente en el repositorio de trunk.

Como había realizado anteriormente, mandé el parche a Liferay España y Portugal, para que lo revisaran y probaran. La respuesta fue la siguiente:

Hola José Ignacio,

Ya hemos revisado el parche y hemos probado que funciona correctamente y según lo que habíamos acordado ¡Buen trabajo!

Por mi parte, creo que ya está listo para que lo presentes. El código todavía tiene que pasar por otra revisión de grano fino, donde se harán algunos cambios antes de meterlo en subversion. Probablemente llevará un par de semanas por lo que no creo que sea necesario que esperes para entregar el proyecto.

Un saludo, Jorge.

Por ello, con esta iteración terminamos la implementación del proyecto.

3.3. Fase 3: Contribución al proyecto

Esta última fase se describe el procedimiento seguido para realizar la contribución de la nueva funcionalidad a Liferay.

Para ello, Liferay dentro de su Web[4] establece el siguiente procedimiento:

- Asegurarse de que no exista la nueva funcionalidad.
- Comentar el objetivo en los foros para que la comunidad pueda opinar acerca de ello.
- Descargarse el último código fuente de Liferay y realizarle ahí los cambios.
- Abrir un nuevo ticket en JIRA explicando la nueva funcionalidad.
- Crear un parche (patch) con los cambios y adjuntarlo al ticket anterior con las siguientes recomendaciones:
    - Asegurarse que el código cumple las "Liferay's guidelines: Liferay Core Development Guidelines".
    - Realizar los cambios sobre la última version de trunk, especificando la revisión en el patch.
    - No incluir las diferencias de los archivos generados automáticamente por el ServiceBuilder.
    - Nombrar el patch de la siguiente forma: LEP-nnnn-build-mmmmmm.patch, donde nnnn es el número asociado al ticket de JIRA y mmmmmm es el número de revisión del código sobre el que se han realizado los cambios.
- Por último, hacer click en "Resolve issue". No se debe cerrar, solo resolver.

Todos los pasos fueron seguidos minuciosamente.

Para la creación del parche utilicé la aplicación TortoiseSVN[5]. Con ella descargué del repositorio de Liferay[6] la última versión del código fuente del portal (revisión 61649). En ese código tuve que reflejar, de nuevo, todas las modificaciones que había realizado para implementar mi proyecto. Finalmente, mediante una opción del programa anterior generé el parche.

Con el parche creado, esperé antes de abrir el ticket en Jira a recibir el visto bueno de Liferay España y Portugal, dado que el proyecto ha estado completamente ligado a ellos.

[4] http://www.liferay.com/community/wiki/-/wiki/Main/How+to+contribute+to+Liferay

[5] http://tortoisesvn.tigris.org/

[6] http://svn.liferay.com/repos/public/portal/trunk/

En la siguiente figura puede verse el principio de la contribución, la apertura del ticket.

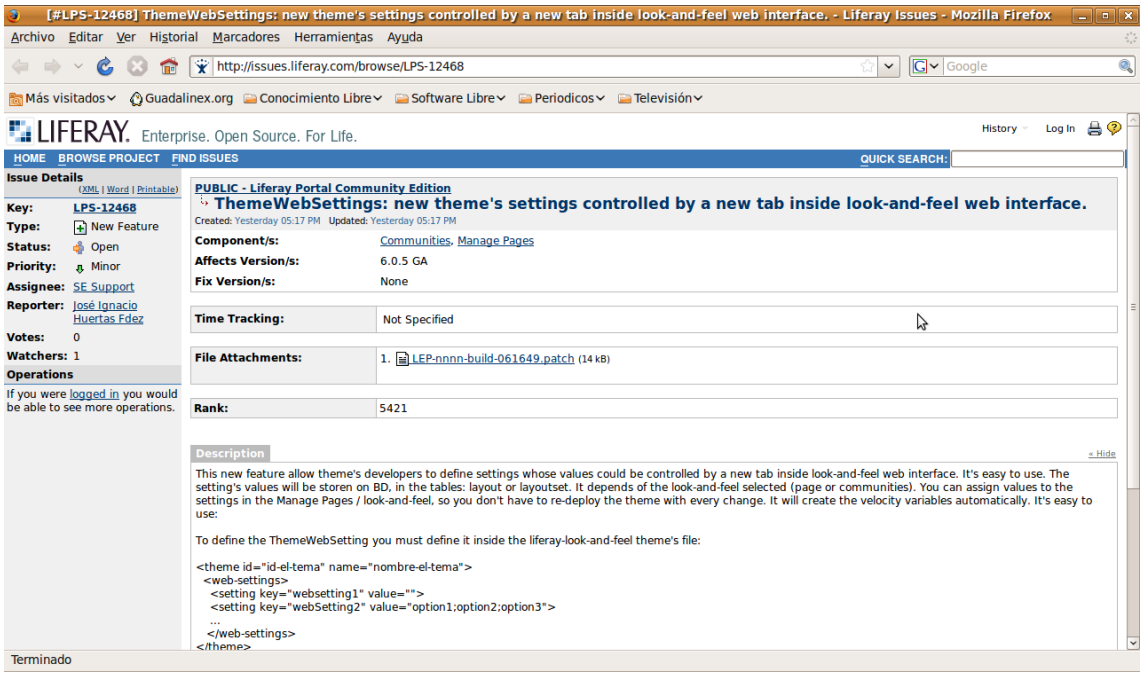

Figura 3.5: Ticket para la nueva funcionalidad en Liferay Issues

A partir de aquí, queda esperar a ver la aceptación que tiene esta nueva funcionalidad de Liferay a nivel internacional.

Capítulo 4

Conclusiones

Una vez descritos los objetivos del proyecto y analizada la solución desarrollada, esta memoria termina resumiendo las ventajas que este proyecto aporta a Liferay, las dificultades encontradas, los principales logros alcanzados y proponiendo posibles trabajos futuros.

4.1. Aportación de los ThemeWebSettings a Liferay

Las ventajas que aporta la utilización de los ThemeWebsettings, con respecto a los Settings mostrados anteriormente, son las siguientes:

1. **Simplicidad.**

    - Se simplifica la declaración en el fichero liferay-look-and-feel.xml. Tan solo hay que declarar los nombres de los ThemeWebSettings que se deseen utilizar. El valor se establecerá en el interfaz Web y a tiempo real.
    - Tan solo tendremos que definir un tema que podrá ser personalizado con los diferentes ThemeWebSettings que contenga, a diferencia de los settings, en los que era necesario definir distintos temas, uno por cada valor de los settings que se deseen utilizar.
    - Se simplifica su utilización dentro de las plantillas (templates). Para acceder a su valor tan solo será necesario poner su nombre (el declarado en el key del ThemeWebSetting) precedido del carácter “$”.

2. **Máxima personalización.** Se podrán personalizar los temas de apariencia tanto a nivel de comunidades como de páginas. Para cada comunidad, utilizando el mismo tema, se podrán personalizar los valores de los ThemeWebSettings.

3. **Seguridad.**

   - Los valores de las propiedades se almacenan dentro de la base de datos de Liferay. Realizando una copia de seguridad de la misma, ante cualquier fallo, podremos recuperar los valores establecidos restaurando la copia de la base de datos.
   - Es posible acotar el valor de las propiedades. De esta forma se mostrará una lista desplegable con los posibles valores del ThemeWebsetting.

4. **Versatilidad.** La utilización de los ThemeWebSettings pueden ser utilizados con múltiples aplicaciones:

   - Para establecer el nombre de una clase de un elemento html, como por ejemplo un div personalizable. En función del valor seleccionado se aplicarán unas u otras sentencias CSS.
   - Para mostrar u ocultar algún portlet o bloque html (p.e. un campo de búsqueda en la cabecera).
   - Para seleccionar entre distintas cabeceras, pie de página, ..., desarrolladas en el mismo tema.
   - Para agrupar múltiples temas en uno solo. Con la selección del ThemeWebSetting se mostraría uno u otro.
   - Para agrupar distintos diseños (a nivel de templates o bien de CSS) de un mismo bloque. Por ejemplo, es posible definir un ThemeWebSetting "NavStyle"que, mediante una lista acotada, permita seleccionar el estilo de la barra de navegación superior.
   - Para introducir texto o bien bloques html en alguna ubicación. Por ejemplo, se puede introducir una capa de publicidad en la cabecera cuyo contenido sea el valor de un ThemeWebSetting.
   - etc.

4.2. Dificultades encontradas

La mayor dificultad encontrada en la realización de este proyecto estaba en mi propia experiencia. Era la primera vez que me integraba en un proyecto tan grande y con estas tecnologías. Por lo cual tuve que dedicar bastante tiempo a ponerme al día. Aunque, realmente, ya contaba con esta dificultad antes de empezar.

Quitando la anterior, podemos destacar dos dificultades:

- Búsqueda de información compleja.
- Tiempo de aprendizaje inicial alto.

Con respecto a la primera, al tratarse de algo específico, en algunas cuestiones no conseguía encontrar mucha información al respecto.

Casi toda la documentación se encuentra dentro de los documentos, foros y wiki de la comunidad. Fuera, en la red, existía poca información específica referida a lo que estaba desarrollando.

Igualmente, existe aún poca bibliografía en el mercado y lo que hay normalmente va dirigido al desarrollo de portlets, temas de apariencia, construcción de intranets, etc.

Esta dificultad fue trasladada a Jorge Ferrer, quien me comentó que estaban trabajando en la redacción de más libros técnicos de Liferay.

Con respecto a la segunda dificultad, destacar que, tal y como se ha comentado en la fase 1 del capítulo *Descripción Informática*, el portal de Liferay consta de más de un millón y medio de líneas de código y, aunque está bien estructurado, al principio resultaba muy costoso encontrar cualquier cosa. Además, al ser un proyecto tan grande, para interpretar correctamente el código había que conocer muy bien todos los detalles de la herramienta para que las modificaciones no entraran en conflicto con otras funcionalidades existentes.

Por otro lado, una de las cosas que creo que se podría mejorar en la Web de Liferay es la ubicación de la descripción de las contribuciones. Realmente creo que está demasiado escondido. En otros proyectos está mucho más visible.

4.3. Logros alcanzados

A un nivel general, con el desarrollo de este proyecto se ha puesto a disposición de la comunidad de Liferay una nueva funcionalidad que mejora la personalización de los temas de apariencia.

Los desarrolladores tienen una nueva funcionalidad en la construcción de los temas de apariencia que permite aumentar la personalización de los mismos. De esta forma podrán decidir hasta qué grado quieren que el tema sea configurable por el usuario.

Los usuarios podrán experimentar un aumento del control sobre la personalización de la apariencia de su Web. Para ello podrán hacer uso de los ThemeWebSettings definidos por los desarrolladores del tema, sin necesitar para ello ningún tipo de conocimiento adicional y de una forma sencilla.

Por otro lado, a un nivel más personal, puedo destacar como logros los siguientes:

- La adquisición de conocimientos en:
  - Programación Web en Java en proyectos de gran magnitud.
  - Programación en Apache Velocity, para las plantillas de los temas de apariencia.
- Conseguir pertenecer a un proyecto de software libre de gran dimensión y poder contribuir a su desarrollo.
- Ser parte activa de una comunidad.
- Creación de temas de apariencia para Liferay.
- Creación de PDFs con Latex y Beamer.

Finalmente, añadir a modo de conclusión final, que este proyecto ha constituido una "excusa" para dar el paso de integrarme dentro de una comunidad de software libre y contribuir a su desarrollo.

Gracias a ello, he adquirido nuevos conocimientos técnicos, nuevos métodos de programación y también conocimientos acerca de cómo se gestionan este tipo de proyectos, cómo poder contribuirlos, etc.

Puedo concluir diciendo que, además de cumplir los objetivos descritos en esta memoria, ha contribuido muy positivamente en mi formación como informático.

4.4. Posibles trabajos futuros

1. Añadir la posibilidad de definir distintos tipos de ThemeWebSettings: listas, código HTML, etc.

2. Añadir un editor Wysiwyg para los ThemeWebSettings abiertos que faciliten la introducción de HTML.

3. Añadir la posibilidad de establecer un nombre para cada ThemeWebSetting distinto del de la variable, para mostrarlo en la web.

4. Desarrollar un editor del aspecto general de los temas de apariencia basado en estos ThemeWebSettings.

Apéndice A

Manual de utilización de los ThemeWebSettings

A.1. Declaración y utilización de los ThemeWebSettings

Los ThemeWebSettings se definen, al igual que los Settings en el fichero liferay-look-and-feel.xml, contenido en el directorio /docroot/WEB-INF del tema. La estructura a utilizar dentro del fichero xml es la siguiente:

```
<theme id="id-el-tema" name="nombre-el-tema">
  <web-settings>
    <setting key="my-Websetting1" value="">
    <setting key="my-WebSetting2" value="option1;option2;option3">
    ...
   </web-settings>
</theme>
```

Para cada ThemeWebSettings estableceremos dos valores:

- **key:** se corresponde con el nombre. Es importante destacar que deben ser distintos de nombres claves en Liferay, como: theme, layout, ...
- **Value:** contendrá los posibles valores del ThemeWebSetting. El formato se corresponde con una lista de valores separados por ";". Dichos valores son los que podrá tomar la

ThemeWebSetting. En el caso de que esté vacío se considerará que el valor a tomar puede ser cualquiera.

En la Web, para el ThemeWebSetting “my-Websetting1”se mostrará un textarea en el que se podrá introducir cualquier valor. Mientras que para el “my-Websetting2”se mostrará una lista desplegable con los valores: option1, option2, option3.

Una vez declarados podremos utilizarlos dentro de las plantillas del tema. Para ello tan solo es necesario anteponer el carácter “$.ªl key asignado al ThemeWebSetting. Ej.: $my-Websetting1 ó bien $my-Websetting2.

Al cargar por primera vez un tema, los ThemeWebSettings del mismo no tendrán ningún valor. Por ello, en el caso de que tenga los valores acotados se considerará que está establecido el primero de los declarados (option1, en el ejemplo anterior); si no tiene los valores acotados, el valor se corresponderá con una cadena vacía.

A.2. Personalización de los ThemeWebSettings desde la Web

Una vez que se han declarado los distintos ThemeWebSettings tenemos que pasar a establecerles un valor. Esto se realiza a través de la web, dentro del apartado de *Administración de la página*, en la pestaña *Apariencia*.

En esa ubicación, en el caso de que el tema de apariencia seleccionado contenga definidos ThemeWebSettings, se mostrará una nueva pestaña denominada “*Configuración del tema*“ en la que se encontrarán todas las propiedades definidas en el tema.

Tal y como se ha comentado anteriormente el establecimiento del valor de cada ThemeWebSetting podrá realizarse:

- **A nivel global:** afectará, por tanto, a todas las páginas de esa comunidad.
- **A nivel de página:** afectará sólo a esa página, independientemente de los valores establecidos en la comunidad a la que pertenezca.

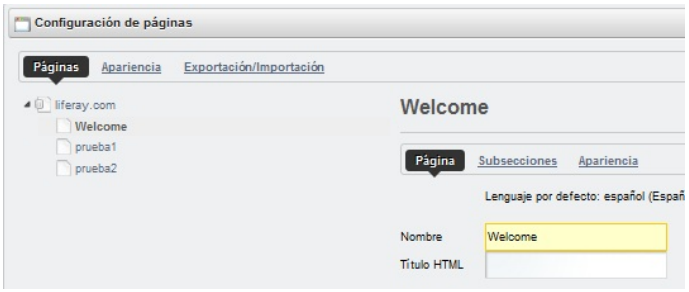

Figura A.1: Ubicación de la Apariencia

Dentro de esta pestaña, en ambos casos, se mostrarán todos los ThemeWebSettings del tema. Si se les establecieron posibles valores en el campo “values”, se mostrará una lista desplegable con los mismos. En caso contrario se mostrará un textarea para introducir el texto deseado.

Figura A.2: Ejemplo de ThemeWebSettings

Finalmente existe un botón para almacenar en la base de datos los valores introducidos.

Apéndice B

Ejemplo de email tras la iteración 3

Este email es una constestación de Jorge Ferrer a distintas cuestiones que planteo al finalizar de la Iteración 3, por lo que las recomendaciones que se incluyen van a ser implementadas en la iteración siguiente.

A continuación se muestra su contenido:

Jorge: Hola José Ignacio,

Yo: Buenas!

ya tengo una versión funcional con todos los bloques html que se deseen integrar. El html es interpretado correctamente, al igual que el css. De momento, en la Web, lo he colocado en un bloque desplegable debajo del de javascript en ”Administrar páginas - página“. Jorge, creo que tendría más sentido hacerlo directamente para el tema, en vez de para una página, ¿no?. (Tenemos que ver la inserción del campo en la tabla layoutSet, ...)

Jorge: Totalmente de acuerdo. Puedes contar con que ese campo acabará estando. Si no está a tiempo para que lo uses tu mismo no te preocupes porque lo haremos cuando apliquemos tu contribución.

Yo: Ahora estoy intentando implementar un procedimiento que facilite al diseñador la inserción de bloques html, pero no se muy bien cómo hacerlo. Os cuento:

El problema está en que el diseñador del tema tendrá que indicar para cada bloque el nombre que tiene que aparecer en la web, así como el de la variable que va a utilizar en velocity (y que se va a almacenar en la BD). Todo ello de una forma sencilla.

Ahora estoy utilizando entradas setting en el look-and-feel.xml, una por cada bloque html que se quiera introducir. En el ”key“ ponemos el nombre de la variable, mientras que en el value

ponemos el nombre para la web. Ej:

```
<setting key="html-footer" value="Pie de página"></setting>
```

Además es necesario definir cada variable en velocity y establecer su valor (si ya está en la base de datos) en el init_custom.vm con el siguiente set:

```
#set ($html-footer =
    $typeSettingsProperties.getProperty("html-footer"))
```

Con esto ya puede utilizarlo tan solo poniendo el nombre de la variable en el sitio que se desee del diseño. Por ejemplo, en mi caso lo he colocado en el portal_normal.vm, al final, dentro de un div:

```
<div id="footer">$html-footer</div>
```

¿Véis este procedimiento demasiado complejo para un diseñador?. Lo ideal sería que tan solo tuviera que definirlo en el look-and-feel.xml, ¿no?

Si es así, se me ocurren dos soluciones:

1. Crear una etiqueta propia para definir en el look-and-feel.xml la lista de bloques (al igual que settings). Sería algo así como ”htmlBlocks“. Esto implica tocar más código (replicar el mismo comportamiento que con los settings). Con estos definidos ya se perfectamente cuántos bloques existen.

2. Utilizar las entradas ”Settings“. En este caso tendremos que imponer la nomenclatura de las variables para saber qué setting se refiere a un bloque html. Por ejemplo, para que sea interpretada el nombre de la variable (el ”key“ del setting) debe comenzar por una cadena específica: ”html-“.

Jorge: Yo prefiero la opción 2. Aunque para que el portal pueda identificar cuales son los settings que el usuario final debe poder ”personalizarçreo que es mejor usar un setting con un nombre fijo que lista el resto de settings disponibles.

En cualquier caso yo no lo haría específico de HTML, porque lo que estás construyendo puede servir para que el usuario final introduzca otros tipos de personalizaciones (como un booleano para decidir si algo se muestra o no). Teniendo esto en cuenta podría quedaría algo así:

```
<setting key="customization-fields" value="header,footer"/>
```

Si más adelante queremos añadir información del tipo podríamos indicarlo después del nombre:

```
<setting key="customization-fields"
  value="header[html],footer[html]"/>
```

En cualquier caso esto lo dejaría para más adelante en función del feedback de los usuarios con lo que hagamos ahora.

Tanto en un caso como en otro, estoy investigando para intentar automatizar los ”set“ de las variables a utilizar en los velocities (vm), pero aún no tengo una solución... lo ideal es tocar el init.vm del tema unstyled para incluir un bloque foreach que recorra los bloques establecidos y que cree una línea set por cada variable (ayuda!!).

En principio esto no debería ser en el init.vm porque el tema de apariencia sobre escribe este fichero. Debería hacerse desde la clase Java que invoca el tema de apariencia. Probablemente puedas hacerlo desde VelocityVariables.java. En cualquier caso si no consigues hacer esto podemos dejarlo con que el usuario introduzca:

```
<div>$typeSettingsProperties.getProperty("html-footer"))</div>
```

Que no es tan bonito pero sigue siendo funcional y puede mejorarse más adelante.

Yo: Como véis esto marcha!. Está más avanzado pero estoy un poco atascado con este tema. ¿Aportaciones?

Jorge:

Vas fenomenal :)

Yo:

Espero haberme explicado bien y ...siento la parrafada de email

Por cierto Jorge, aunque pueda ceder todos los derechos de este proyecto a Liferay, ¿en ningún lado (a excepción de en nuestras mentes) constará mi nombre como desarrollador de dicha mejora?

Jorge: Por supuesto se mantendrá tu nombre. Lo unico que sería necesario ceder son los derechos de copyright.

Yo: Gracias y un saludo.

Jorge: Gracias a ti,

Jorge

–

Jorge Ferrer

Director General, Liferay España y Portugal

General Manager, Liferay Spain and Portugal

Enterprise. Open Source. For Life.

Bibliografía

[1] Jonas X. Yuan. *Liferay Portal 5.2 Systems Development*. PACKT Publishing, 2009

[2] Jonas X. Yuan. *Liferay Portal Enterprise Intranets*. PACKT Publishing, 2008

[3] Richard L. Sezov, Jr. *Liferay Portal Administrator's Guide*. Liferay Inc., 2009

[4] Samuel Kong. *Liferay Development Documentation*. Liferay Inc., 2008

[5] Web de la comunidad del proyecto Liferay.
`http://www.liferay.com/es/community`

[6] González Barahona, Jesús, Senoane Pascual, Joaquín y Robles Martínez, Gregorio. *Introducción al Software Libre*. UOC, 2007.

[7] Amor, Juan José, Herraiz, Israel y Robles, Gregorio. *Desarrollo de proyectos de software libre (segunda edición)*. UOC, 2007.

[8] Web de Apache Velocity Proyect.
`http://http://velocity.apache.org/`

[9] Web del proyecto SLOCCount.
`http://www.dwheeler.com/sloccount/`